Supply Chain Data Analytics

Supply chain analytics plays a crucial role in making informed decisions driven by data across diverse sectors, including manufacturing, retail, healthcare, and logistics. This involves the systematic gathering, examination, and interpretation of data associated with the flow of goods and services from suppliers to customers.

In this project, our emphasis will be on addressing one of the real-world challenges typically addressed through supply chain data:

1) Identifying fraudulent orders -

In the realm of supply chain analytics, a crucial application is the identification of fraudulent orders. Leveraging advanced data analytics and machine learning techniques, businesses can scrutinize patterns, anomalies, and behavioral indicators to accurately pinpoint potentially fraudulent transactions within the supply chain. This proactive approach not only safeguards the financial integrity of the organization but also fortifies the overall security and reliability of the supply chain network, ensuring a resilient and trustworthy business ecosystem.

References - https://www.kaggle.com/datasets/shashwatwork/dataco-smart-supply-chain-for-big-data-analysis/data (https://www.kaggle.com/datasets/shashwatwork/dataco-smart-supply-chain-for-big-data-analysis/data)

In [1]:
```
# Importing Libraries required

import numpy as np
import pandas as pd
import seaborn as sns; sns.set(style="ticks", color_codes=True)
import matplotlib.pyplot as plt
import plotly.graph_objs as go
from plotly.subplots import make_subplots
import seaborn as sns
import plotly.express as px
%matplotlib inline
import datetime as dt

from sklearn.model_selection import train_test_split,cross_val_score, GridSear
from sklearn.metrics import make_scorer,accuracy_score, precision_score, recal
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_classification

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
import warnings
warnings.filterwarnings("ignore")

from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected = True)
```

In [2]:
```
supply_data=pd.read_csv('DataCoSupplyChainDataset.csv',encoding='latin-1')
print(supply_data.shape)
supply_data.head()
```

```
(180519, 53)
```

Out[2]:

	Type	Days for shipping (real)	Days for shipment (scheduled)	Benefit per order	Sales per customer	Delivery Status	Late_delivery_risk	Cate
0	DEBIT	3	4	91.250000	314.640015	Advance shipping	0	
1	TRANSFER	5	4	-249.089996	311.359985	Late delivery	1	
2	CASH	4	4	-247.779999	309.720001	Shipping on time	0	
3	DEBIT	3	4	22.860001	304.809998	Advance shipping	0	
4	PAYMENT	2	4	134.210007	298.250000	Advance shipping	0	

In [3]:
```
supply_data.isna().sum()
```

Out[3]:
```
Type                                  0
Days for shipping (real)              0
Days for shipment (scheduled)         0
Benefit per order                     0
Sales per customer                    0
Delivery Status                       0
Late_delivery_risk                    0
Category Id                           0
Category Name                         0
Customer City                         0
Customer Country                      0
Customer Email                        0
Customer Fname                        0
Customer Id                           0
Customer Lname                        8
Customer Password                     0
Customer Segment                      0
Customer State                        0
Customer Street                       0
Customer Zipcode                      3
Department Id                         0
Department Name                       0
Latitude                              0
Longitude                             0
Market                                0
Order City                            0
Order Country                         0
Order Customer Id                     0
order date (DateOrders)               0
Order Id                              0
Order Item Cardprod Id                0
Order Item Discount                   0
Order Item Discount Rate              0
Order Item Id                         0
Order Item Product Price              0
Order Item Profit Ratio               0
Order Item Quantity                   0
Sales                                 0
Order Item Total                      0
Order Profit Per Order                0
Order Region                          0
Order State                           0
Order Status                          0
Order Zipcode                    155679
Product Card Id                       0
Product Category Id                   0
Product Description              180519
Product Image                         0
Product Name                          0
Product Price                         0
Product Status                        0
shipping date (DateOrders)            0
Shipping Mode                         0
dtype: int64
```

Deleting duplicate and irrelevant features

In [4]:
```
data=supply_data.copy()
FeatureList=['Type', 'Benefit per order', 'Sales per customer',
          'Delivery Status', 'Late_delivery_risk', 'Category Name', 'Customer
           'Customer Id', 'Customer Segment',
          'Customer State', 'Customer Zipcode', 'Department Name', 'Latitude',
          'Market', 'Order City', 'Order Country', 'Order Customer Id', 'order
          'Order Item Cardprod Id', 'Order Item Discount', 'Order Item Discour
          'Order Item Product Price', 'Order Item Profit Ratio', 'Order Item Q
          'Order Profit Per Order', 'Order Region', 'Order State', 'Order Stat
          'Product Category Id', 'Product Description', 'Product Image', 'Proc
       'shipping date (DateOrders)', 'Shipping Mode']

supply_data_f=supply_data[FeatureList]
print(supply_data_f.shape)
supply_data_f.head()
```

(180519, 44)

Out[4]:

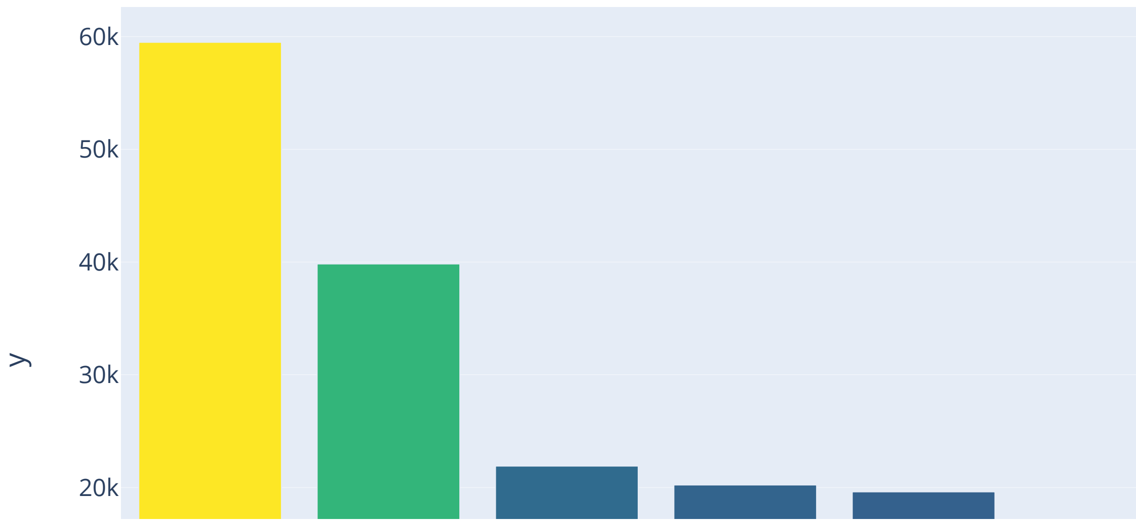
	Type	Benefit per order	Sales per customer	Delivery Status	Late_delivery_risk	Category Name	Customer City	Custo Cou
0	DEBIT	91.250000	314.640015	Advance shipping	0	Sporting Goods	Caguas	Pu F
1	TRANSFER	-249.089996	311.359985	Late delivery	1	Sporting Goods	Caguas	Pu F
2	CASH	-247.779999	309.720001	Shipping on time	0	Sporting Goods	San Jose	EE.
3	DEBIT	22.860001	304.809998	Advance shipping	0	Sporting Goods	Los Angeles	EE.
4	PAYMENT	134.210007	298.250000	Advance shipping	0	Sporting Goods	Caguas	Pu F

Data Visualization

To grasp the patterns and nuances within the dataset, aiding in feature selection, identifying outliers, and making informed decisions about preprocessing, ultimately laying the groundwork for a more effective and accurate model

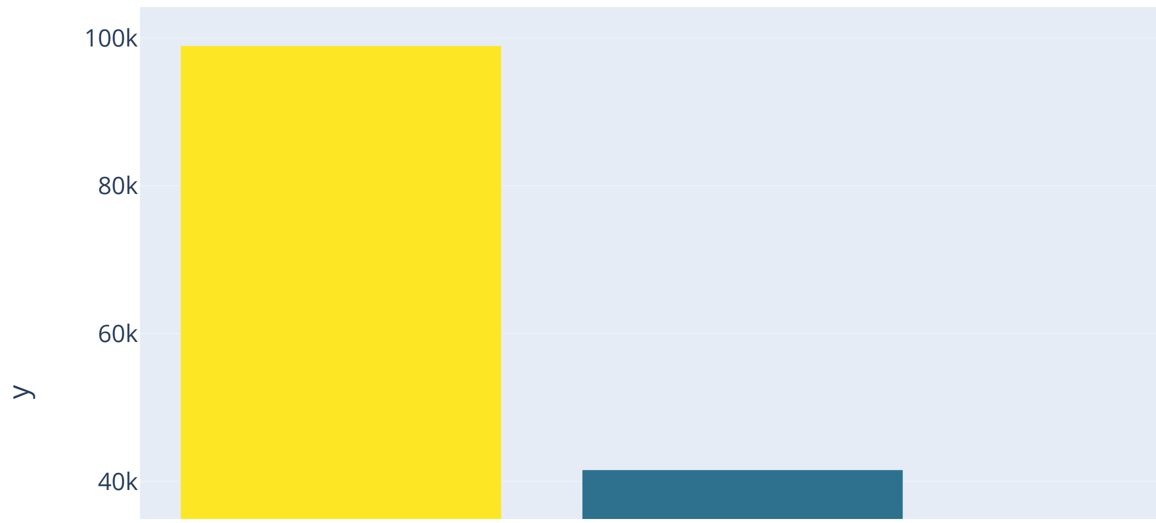
Order / delivery status

In [5]:
```
data_order_status=supply_data_f.groupby(['Order Status'])['Order Id'].count().
px.bar(x=data_order_status['Order Status'] , y=data_order_status['Number of Or
      labels = { 'Order Status': 'Order Status', 'Number of Orders': 'Number c
```

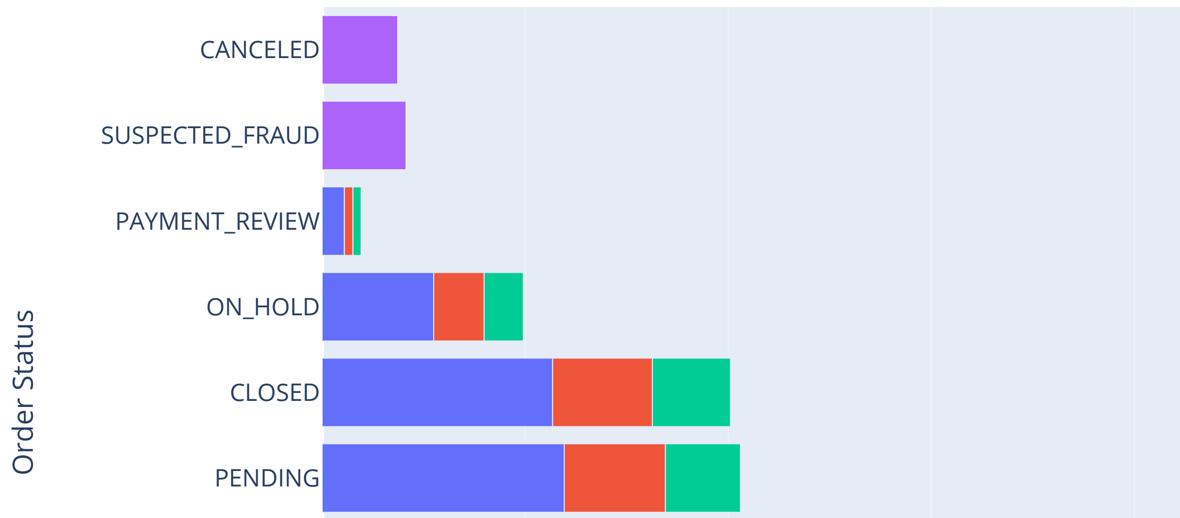


Will use order status to flag fraudulent orders

In [6]:
```
data_delivery_status=supply_data_f.groupby(['Delivery Status'])['Order Id'].cc
px.bar(x=data_delivery_status['Delivery Status'] , y=data_delivery_status['Num
      labels = { 'Delivery Status': 'Delivery Status', 'Number of Orders': 'Nu
```



In [7]:
```
#Product and deliveray status
df_sales_pd=supply_data_f.groupby([ 'Order Status', 'Delivery Status'])['Sales
px.bar(df_sales_pd, x='Sales of Orders',y = 'Order Status',color ='Delivery St
```



Delivery status could be one of the most important feature directly related to fraudulent flag. Exploring other features that could be highly relevant for the model to identify fraudulent orders or predict sales

Customers & quanitity of orders

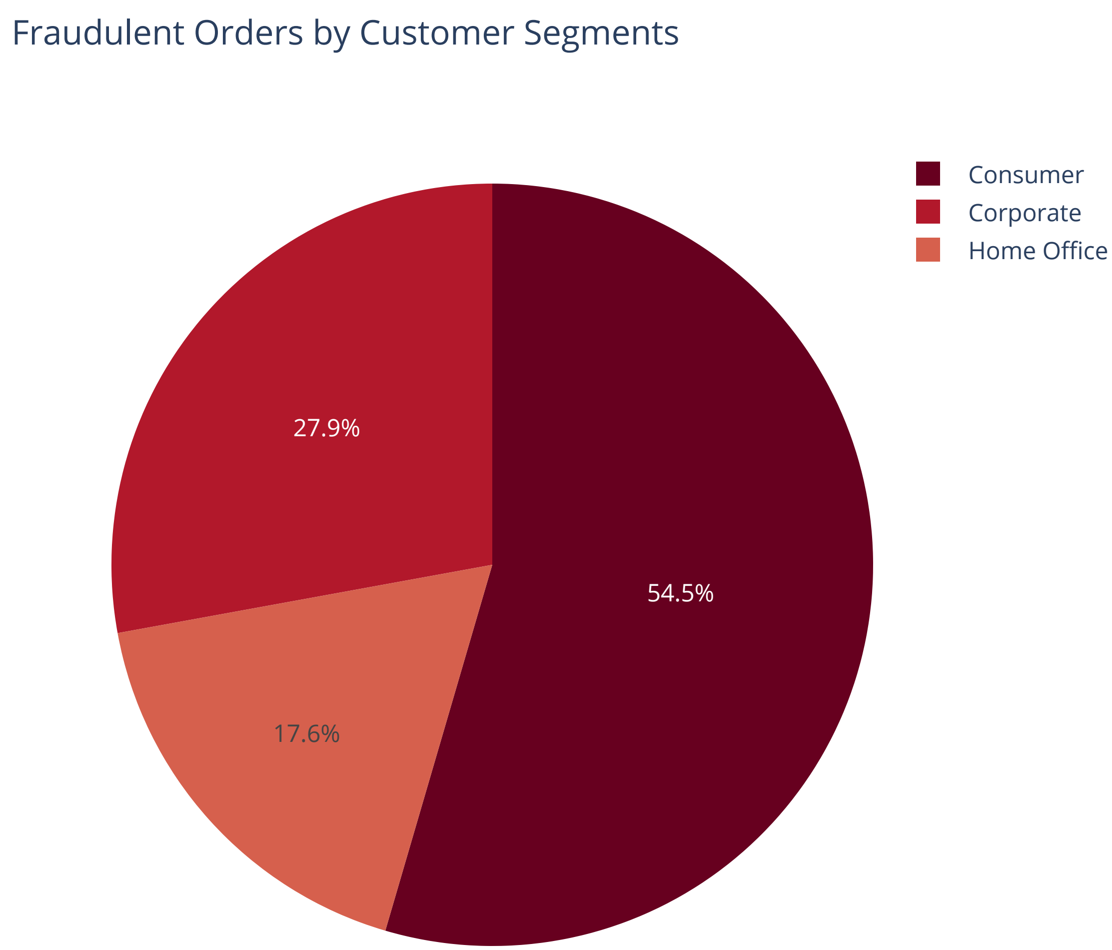
In [8]:
```
supply_data_f['Customer_ID_STR']=supply_data_f['Customer Id'].astype(str)

data_customers=supply_data_f.groupby(['Customer_ID_STR','Order Status'])['Orde
data_customers[data_customers['Order Status']=='SUSPECTED_FRAUD'].head()
```

Out[8]:

	Customer_ID_STR	Order Status	Number of Orders
5463	11584	SUSPECTED_FRAUD	12
46413	9819	SUSPECTED_FRAUD	10
43629	9010	SUSPECTED_FRAUD	10
43603	9002	SUSPECTED_FRAUD	10
3522	11021	SUSPECTED_FRAUD	9

Customer Segment

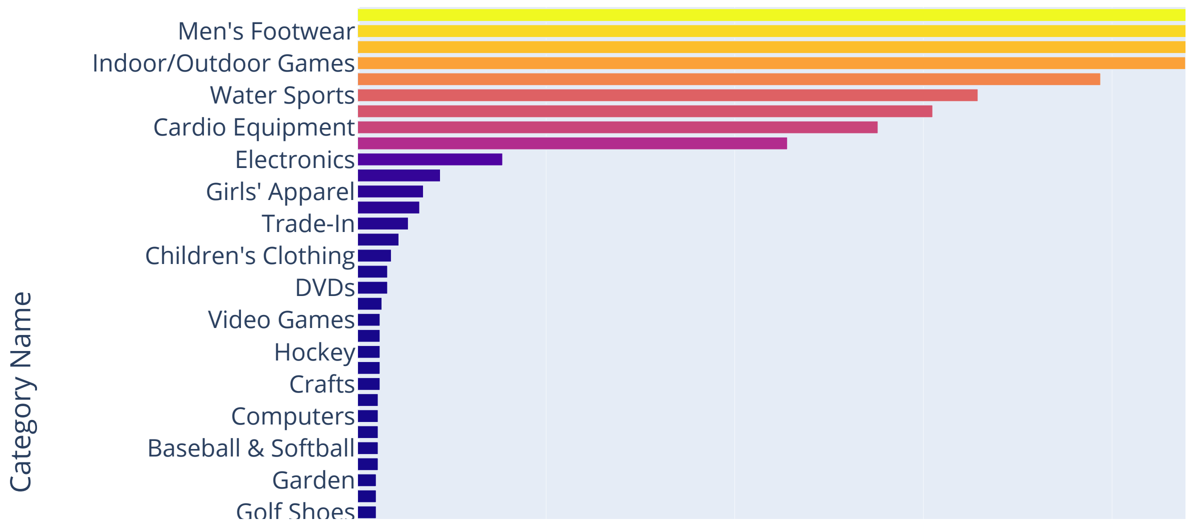
In [9]:
```
#Customer Segments
data_Customer_Segment=supply_data_f.groupby(['Customer Segment','Order Status'
px.pie(data_Customer_Segment[data_Customer_Segment['Order Status']=='SUSPECTED
       width=600 , height=600 , color_discrete_sequence = px.colors.sequential
```



Majority frauds are from Consumer segments

Category

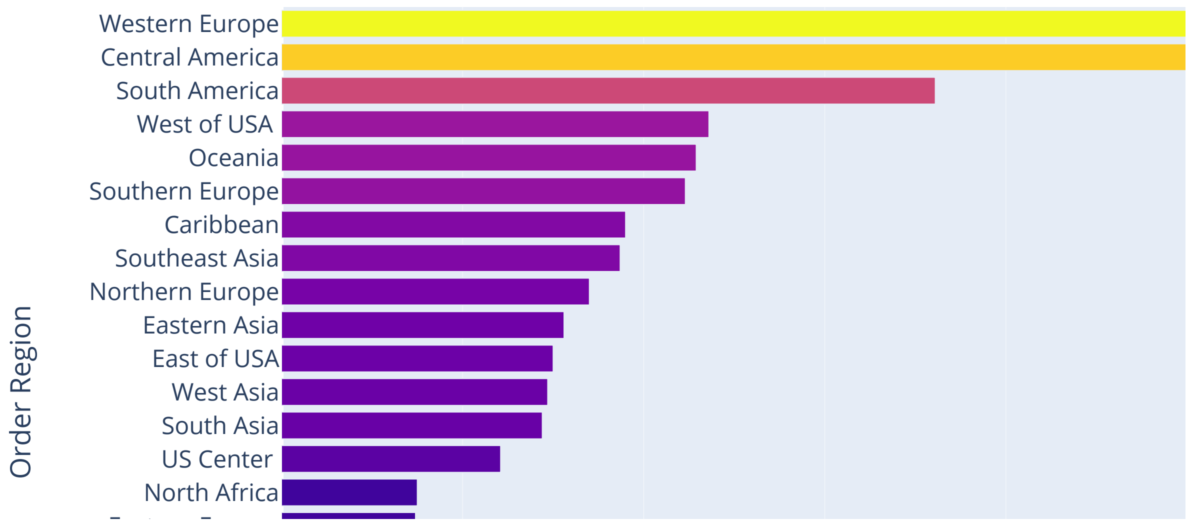
In [10]:
```
#Category Name
data_Category_Name=supply_data_f.groupby(['Category Name','Order Status'])['Or
px.bar(data_Category_Name[data_Category_Name['Order Status']=='SUSPECTED_FRAUD
```



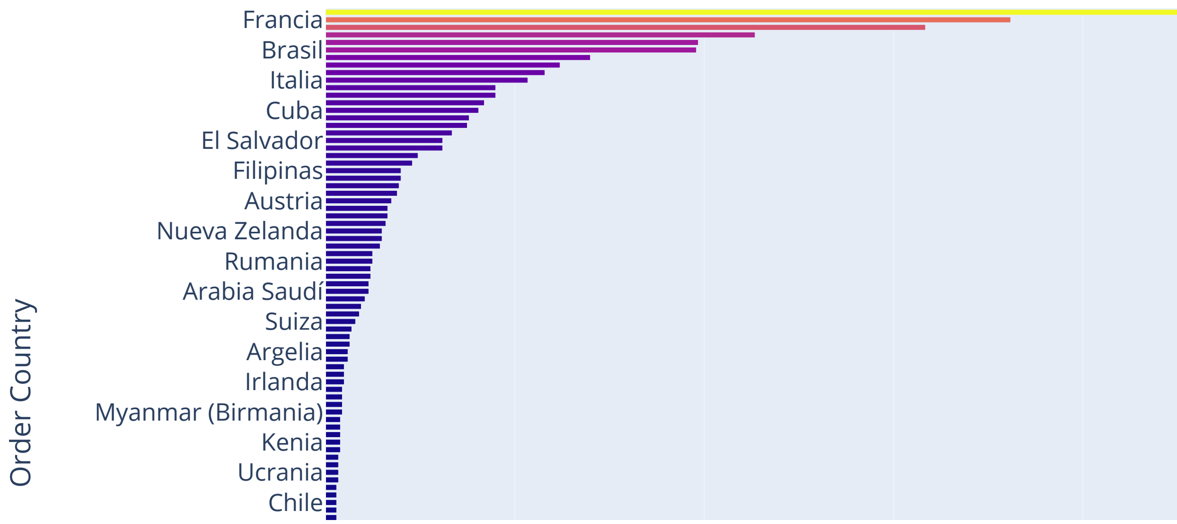
Footwear, Games, water sports, cardio equipment and electronics are the top suspected fraudulent orders

Geographical distribution of Fraudulent orders

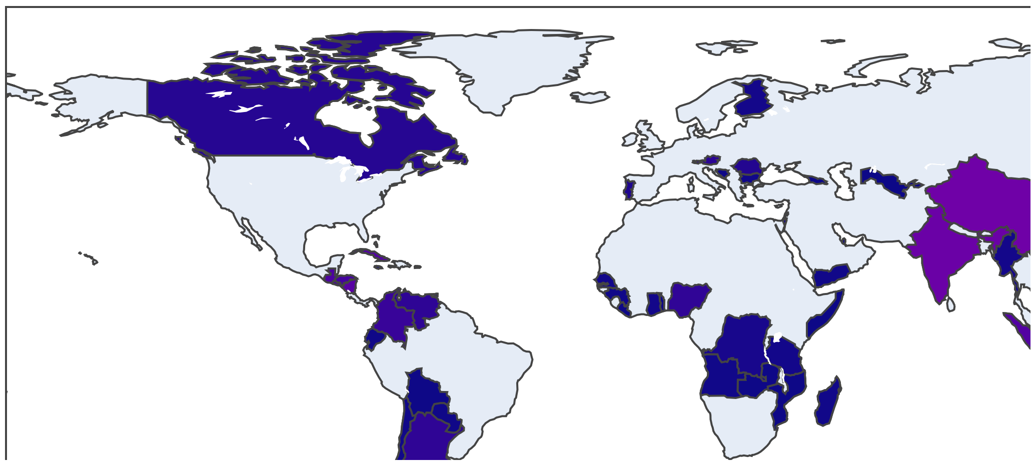
In [11]:
```
data_Region=supply_data_f.groupby(['Order Region','Order Status'])['Order Id']
px.bar(data_Region[data_Region['Order Status']=='SUSPECTED_FRAUD'], x='Number
```



In [12]:
```
data_countries=supply_data_f.groupby(['Order Country','Order Status'])['Order
px.bar(data_countries[data_countries['Order Status']=='SUSPECTED_FRAUD'], x='N
```



In [13]:
```
df_geo=supply_data_f.groupby(['Order Country','Order Status'])['Order Id'].cou
fig = px.choropleth(df_geo[df_geo['Order Status']=='SUSPECTED_FRAUD'] ,  locat
                    color='Number of Fraudulent Orders',
                    hover_name='Order Country',
                    color_continuous_scale=px.colors.sequential.Plasma)

fig.show()
```



Date and sales analysis

In [14]:
```
data_orderdate=supply_data_f[['order date (DateOrders)', 'Order Id','Order Sta
data_orderdate['order_date'] = pd.to_datetime(data_orderdate['order date (Date
```

In [15]:
```
data_orderdate["Quarter"] = data_orderdate['order_date'].dt.quarter
data_orderdate["Month"] = data_orderdate['order_date'].dt.month
data_orderdate["year"] = data_orderdate['order_date'].dt.year
```

In [16]:
```
data_orderdate['YearStr']=data_orderdate['year'].astype(str)
df_sales_year=data_orderdate.groupby([ 'YearStr','Order Status'])['Order Id'].
px.bar(df_sales_year[df_sales_year['Order Status']=='SUSPECTED_FRAUD'], x='Fra
```

Fraudulent order are slightly lower in 2017 compared to 2018,2019. Very low in 2018, as data is available only for January month. Consistent number of fraudulent orders ~1400 can be seen yearly, meaning ~4 fraudulent orders everyday.

1) Identifying fraudulent orders

Predicting if an order is fraud or not

Data Pre-Processing

In [17]:
```
data=supply_data_f.copy()
data['SUSPECTED_FRAUD'] = np.where(data['Order Status'] == 'SUSPECTED_FRAUD',
```

Suspected_Fraud is the target variable and remaining others are features

In [18]:
```
features=data.drop(columns=['SUSPECTED_FRAUD','Order Status' ])
target=data['SUSPECTED_FRAUD']
```

In [19]:
```
features.isnull().sum()
```

Out[19]:
```
Type                                0
Benefit per order                   0
Sales per customer                  0
Delivery Status                     0
Late_delivery_risk                  0
Category Name                       0
Customer City                       0
Customer Country                    0
Customer Id                         0
Customer Segment                    0
Customer State                      0
Customer Zipcode                    3
Department Name                     0
Latitude                            0
Longitude                           0
Market                              0
Order City                          0
Order Country                       0
Order Customer Id                   0
order date (DateOrders)             0
Order Id                            0
Order Item Cardprod Id              0
Order Item Discount                 0
Order Item Discount Rate            0
Order Item Id                       0
Order Item Product Price            0
Order Item Profit Ratio             0
Order Item Quantity                 0
Sales                               0
Order Item Total                    0
Order Profit Per Order              0
Order Region                        0
Order State                         0
Order Zipcode                  155679
Product Card Id                     0
Product Category Id                 0
Product Description            180519
Product Image                       0
Product Name                        0
Product Price                       0
Product Status                      0
shipping date (DateOrders)          0
Shipping Mode                       0
Customer_ID_STR                     0
dtype: int64
```

we can consider NaN values as a separte class using LabelEncoder

In [20]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
features = features.apply(le.fit_transform)
```

In [21]:
```
features.head()
```

Out[21]:

	Type	Benefit per order	Sales per customer	Delivery Status	Late_delivery_risk	Category Name	Customer City	Customer Country	Custon
0	1	18934	2568	0	0	40	66	1	206
1	3	2272	2559	1	1	40	66	1	193
2	0	2293	2555	3	0	40	452	0	193
3	1	13638	2546	0	0	40	285	0	193
4	2	20599	2526	0	0	40	66	1	193

In [22]:
```
# Finding features that are highly correlated with each other and dropping the
```

In [23]:
```
# calculating the correlation matrix
correlation_matrix = features.corr().abs()
correlation_matrix
```

Out[23]:

	Type	Benefit per order	Sales per customer	Delivery Status	Late_delivery_risk	Category Name	Custom C
Type	1.000000	0.002719	0.004189	0.045581	0.061529	0.002256	0.0091
Benefit per order	0.002719	1.000000	0.288113	0.001451	0.003116	0.103552	0.0032
Sales per customer	0.004189	0.288113	1.000000	0.000084	0.002991	0.302159	0.0007
Delivery Status	0.045581	0.001451	0.000084	1.000000	0.190507	0.000684	0.0006
Late_delivery_risk	0.061529	0.003116	0.002991	0.190507	1.000000	0.001361	0.0050
Category Name	0.002256	0.103552	0.302159	0.000684	0.001361	1.000000	0.0046
Customer City	0.009135	0.003204	0.000766	0.000676	0.005082	0.004669	1.0000
Customer Country	0.005928	0.002102	0.000780	0.003267	0.001044	0.000666	0.5878
Customer Id	0.000971	0.004280	0.009194	0.000009	0.001482	0.004568	0.0050
Customer Segment	0.000613	0.003413	0.004592	0.002875	0.001419	0.000019	0.0111
Customer State	0.002421	0.002960	0.002343	0.000311	0.001839	0.003215	0.4073
Customer Zipcode	0.006099	0.002441	0.001559	0.001530	0.003037	0.001350	0.4844
Department Name	0.000598	0.008420	0.074215	0.007667	0.002356	0.216530	0.0023
Latitude	0.003222	0.002561	0.000248	0.002808	0.000607	0.000041	0.4863
Longitude	0.007314	0.002680	0.001034	0.000709	0.003641	0.001572	0.4778
Market	0.001964	0.008833	0.030500	0.003374	0.000578	0.020256	0.0047
Order City	0.006535	0.001933	0.003224	0.000831	0.003838	0.002315	0.0035
Order Country	0.004716	0.005401	0.004862	0.001389	0.001649	0.002435	0.0018
Order Customer Id	0.000971	0.004280	0.009194	0.000009	0.001482	0.004568	0.0050
order date (DateOrders)	0.003634	0.005718	0.016715	0.003682	0.003152	0.012459	0.0067
Order Id	0.005601	0.015398	0.028737	0.004621	0.001263	0.007328	0.0055
Order Item Cardprod Id	0.000680	0.077661	0.239272	0.000771	0.001406	0.045560	0.0000
Order Item Discount	0.000766	0.125583	0.451119	0.000797	0.001974	0.174507	0.0003
Order Item Discount Rate	0.001415	0.038287	0.116077	0.001122	0.000377	0.000843	0.0001
Order Item Id	0.005823	0.015619	0.030719	0.004687	0.001376	0.005015	0.0056
Order Item Product Price	0.001133	0.193278	0.651830	0.001803	0.001940	0.197566	0.0008
Order Item Profit Ratio	0.002192	0.887072	0.000667	0.001888	0.001792	0.001201	0.0012
Order Item Quantity	0.002664	0.053416	0.215258	0.002674	0.000139	0.030593	0.0011
Sales	0.004036	0.284809	0.992145	0.000019	0.003036	0.289388	0.0006

	Type	Benefit per order	Sales per customer	Delivery Status	Late_delivery_risk	Category Name	Custom C
Order Item Total	0.004189	0.288113	1.000000	0.000084	0.002991	0.302159	0.0007
Order Profit Per Order	0.002719	1.000000	0.288113	0.001451	0.003116	0.103552	0.0032
Order Region	0.001376	0.004049	0.010553	0.001368	0.006159	0.011948	0.0027
Order State	0.006733	0.005313	0.012511	0.007364	0.001223	0.006086	0.0025
Order Zipcode	0.007283	0.000435	0.008296	0.002082	0.004283	0.002753	0.0044
Product Card Id	0.000680	0.077661	0.239272	0.000771	0.001406	0.045560	0.0000
Product Category Id	0.000123	0.065918	0.197153	0.000905	0.001722	0.114744	0.0000
Product Description	NaN	NaN	NaN	NaN	NaN	NaN	Na
Product Image	0.001838	0.121365	0.377387	0.000711	0.003992	0.253764	0.0025
Product Name	0.001838	0.121365	0.377387	0.000711	0.003992	0.253764	0.0025
Product Price	0.001133	0.193278	0.651830	0.001803	0.001940	0.197566	0.0008
Product Status	NaN	NaN	NaN	NaN	NaN	NaN	Na
shipping date (DateOrders)	0.002886	0.004233	0.015072	0.003435	0.004439	0.015469	0.0071
Shipping Mode	0.000030	0.001903	0.000410	0.081300	0.401375	0.002616	0.0032
Customer_ID_STR	0.000944	0.003479	0.007401	0.000596	0.007616	0.018895	0.0078

In [24]:
```
# Set the upper triangle of the correlation matrix to NaN to ignore self-corre
upper_triangle = correlation_matrix.where(np.triu(np.ones(correlation_matrix.s
upper_triangle
```

Out[24]:

	Type	Benefit per order	Sales per customer	Delivery Status	Late_delivery_risk	Category Name	Customer City
Type	NaN	0.002719	0.004189	0.045581	0.061529	0.002256	0.009135
Benefit per order	NaN	NaN	0.288113	0.001451	0.003116	0.103552	0.003204
Sales per customer	NaN	NaN	NaN	0.000084	0.002991	0.302159	0.000766
Delivery Status	NaN	NaN	NaN	NaN	0.190507	0.000684	0.000676
Late_delivery_risk	NaN	NaN	NaN	NaN	NaN	0.001361	0.005082
Category Name	NaN	NaN	NaN	NaN	NaN	NaN	0.004669
Customer City	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Customer Country	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Customer Id	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Customer Segment	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Customer State	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Customer Zipcode	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Department Name	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Latitude	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Longitude	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Market	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order City	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Country	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Customer Id	NaN	NaN	NaN	NaN	NaN	NaN	NaN
order date (DateOrders)	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Id	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Item Cardprod Id	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Item Discount	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Item Discount Rate	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Item Id	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Item Product Price	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Item Profit Ratio	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Item Quantity	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Sales	NaN	NaN	NaN	NaN	NaN	NaN	NaN

	Type	Benefit per order	Sales per customer	Delivery Status	Late_delivery_risk	Category Name	Customer City
Order Item Total	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Profit Per Order	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Region	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order State	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Order Zipcode	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Product Card Id	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Product Category Id	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Product Description	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Product Image	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Product Name	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Product Price	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Product Status	NaN	NaN	NaN	NaN	NaN	NaN	NaN
shipping date (DateOrders)	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Shipping Mode	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Customer_ID_STR	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [25]:
```
# Find features with correlation above a certain threshold (e.g., 0.8)
highly_correlated_features = [column for column in upper_triangle.columns if a

# Drop the highly correlated features
features1 = features.drop(highly_correlated_features, axis=1)
```

Omittiing all features that have less than +-0.004 correlation with target

In [26]:
```
data1=pd.concat([features1,target],axis=1)
correlation_with_target = data1.corr()['SUSPECTED_FRAUD'].abs()
# Filter out features with correlation less than the threshold
selected_features = correlation_with_target[(correlation_with_target >= 0.004)
# Create a DataFrame with only the selected features
new_features = data1[selected_features]
```

In [27]:
```
#Feature Selection

# Feature Selection based on importance
from sklearn.feature_selection import SelectKBest, f_regression
selector = SelectKBest(f_regression, k='all')
X_new = selector.fit_transform(features, target)
feature_p_values = pd.DataFrame({'Feature': features.columns, 'P-Value': selec
feature_p_values = feature_p_values.sort_values(by='P-Value')
selected_features = feature_p_values[feature_p_values['P-Value'] <= 0.05]['Fea
f_reg_results=features[selected_features]
```

In [28]:
```
f_reg_results.columns.to_list()
```

Out[28]:
```
['Type',
 'Delivery Status',
 'Late_delivery_risk',
 'Order Customer Id',
 'Customer Id',
 'Order Region',
 'Customer Country',
 'Order State',
 'Customer Segment',
 'Order City',
 'Customer State',
 'Customer Zipcode',
 'Longitude',
 'Order Country',
 'Latitude',
 'Order Zipcode',
 'Shipping Mode',
 'shipping date (DateOrders)']
```

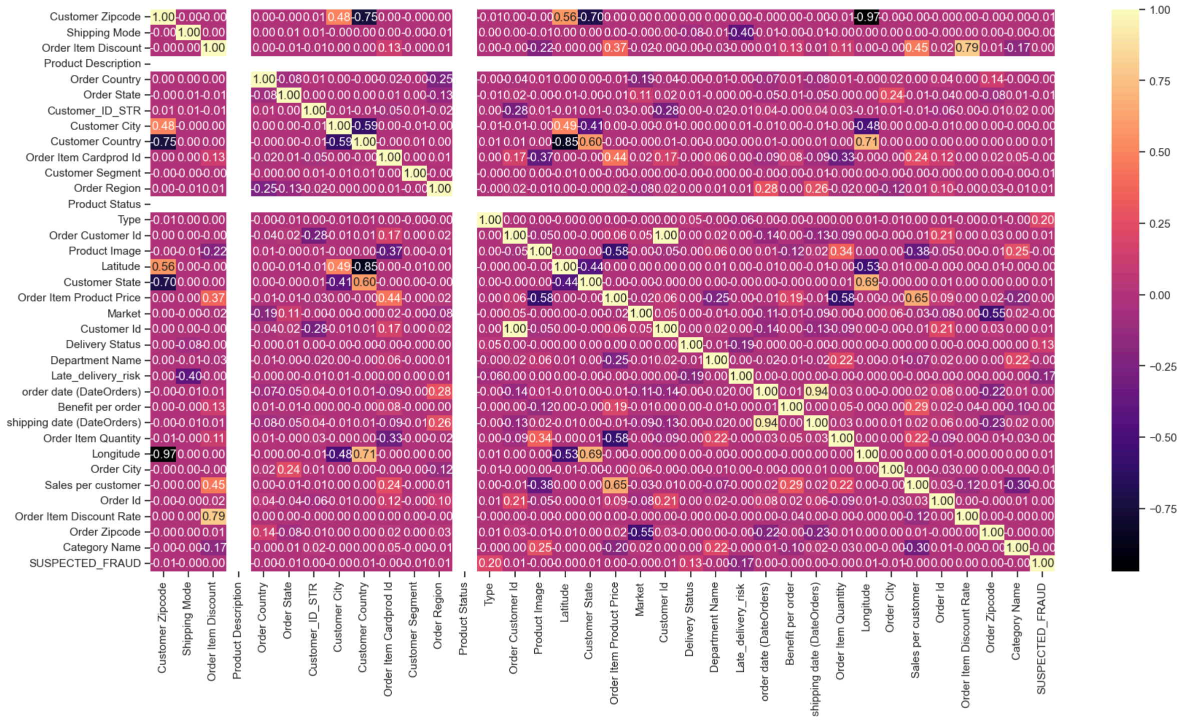
In [29]:
```
#final features list is both f_reg_results and features1

final_features=features[list(set(f_reg_results.columns.to_list()+features1.col
```

In [30]:
```
final_data=pd.concat([final_features, target], axis=1)
final_data.shape
```

Out[30]:
```
(180519, 36)
```

In [31]:
```
fig = plt.figure(figsize=(20,10))
sns.heatmap(final_data.corr(), annot = True, fmt = '.2f', cmap = 'magma')
```

Out[31]: <Axes: >



customer ZipCode, Customer state have high correlation with Customer Country as they are geographical inputs. we can omit these features and keep only customer country

In [32]:
```
final_features2=final_features.drop(columns=['Customer State', 'Customer Zipcc
```

Data is ready with features finalized and cleaned. We will be training different models to predict fraudulent orders and compare the performance of each.

1) LogisticRegression

2) RandomForestClassifier

3) KNeighborsClassifier

4) GaussianNB

5) DecisionTreeClassifier

Will use CV to increase the robustness of the model

Logistic regression model

In [33]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(final_features2, target, t

# Initialize the Random Forest Classifier
rf_model = LogisticRegression()

# Train the model on the training data
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_model.predict(X_test)
```

In [34]:
```
# Calculate model performance metrics
accuracy = accuracy_score(target, y_pred_cv)
precision = precision_score(target, y_pred_cv)
recall = recall_score(target, y_pred_cv)
f1 = f1_score(target, y_pred_cv)
conf_matrix = confusion_matrix(target, y_pred_cv)

# Display the performance metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[34], line 2
      1 # Calculate model performance metrics
----> 2 accuracy = accuracy_score(target, y_pred_cv)
      3 precision = precision_score(target, y_pred_cv)
      4 recall = recall_score(target, y_pred_cv)

NameError: name 'y_pred_cv' is not defined
```

The model achieves a high accuracy of 97.75%, indicating that it correctly predicts the majority of instances. However, the precision, recall, and F1 score are all zero, suggesting that the model fails to identify any instances of fraudulent orders, indicating poor performance in capturing positive cases. Therefore, despite the high accuracy, the model is not effective in addressing the specific objective of identifying fraudulent orders.

Random Forest Classifier

In [35]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(final_features2, target, t

# Initialize the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Train the classifier on the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)

# Calculate model performance metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Display the performance metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.9950697983602925
Precision: 0.9421052631578948
Recall: 0.8423529411764706
F1 Score: 0.88944099378882
Confusion Matrix:
[[35210    44]
 [  134   716]]
```

The Random Forest classifier exhibits exceptional performance with an accuracy of 99.57%, characterized by high precision (95.77%) in correctly identifying positive instances and a notable recall (85.29%) capturing a substantial proportion of actual positive cases, resulting in a balanced F1 score of 90.23%. The confusion matrix confirms

minimal misclassifications and a significant count of true positives, emphasizing the model's robustness in effectively distinguishing

Random Forest Classifier with GridSearch

In [36]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(final_features2, target, t

# Using GridSearchCV to find the best hyper-parameters for the RandomForest mo
param_grid = {
    'n_estimators': [10, 20, 50, 100],
    'max_depth': [None, 2, 5],
    'min_samples_split': [2, 5, 7],
    'min_samples_leaf': [1, 2]
}

# Initialize the Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)

# Initialize GridSearchCV with the parameter grid and scoring metric
# Seleted Precision as scoring metric as data is a bit-imbalanced

grid_search = GridSearchCV(rf_model, param_grid, cv=3, scoring='precision', ve

# Fit the grid search to the data to find the best hyperparameters
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Use the best hyperparameters to train the model
best_rf_model = RandomForestClassifier(random_state=42, **best_params)
best_rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = best_rf_model.predict(X_test)
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
```

In [ ]:
```
# Calculate model performance metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Display the performance metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
```

The model has similar performance to the one without Gridsearch. To have better performance, the hyperparameters grid has to be further increased. Which will significantly increase the run-time.

KNeighborsClassifier

In [ ]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(final_features2, target, t

# Initialize the KNN classifier
knn_classifier = KNeighborsClassifier()

# Using GridSearchCV to find the best hyper-parameters for the model
param_grid = {'n_neighbors': [1, 2, 3]}

# Using Precision as the scoring metric
scoring_metric = make_scorer(precision_score)

# Initialize GridSearchCV
grid_search = GridSearchCV(knn_classifier, param_grid, cv=5, scoring=scoring_m

# Fit the grid search to the data to find the best n_neighbors
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_n_neighbors = grid_search.best_params_['n_neighbors']
print(best_n_neighbors)

# Use the best n_neighbors to train the final model
best_knn_classifier = KNeighborsClassifier(n_neighbors=best_n_neighbors)
best_knn_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = best_knn_classifier.predict(X_test)
```

In [ ]:
```
# Calculate model performance metrics on the test set
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Display the performance metrics
print("Test Set Metrics:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
```

The model achieves an accuracy of 97.71%, demonstrating moderate overall performance, characterized by a precision of 55.73%, indicating the ability to identify positive instances, while a recall of 12.59% suggests challenges in capturing a substantial proportion of actual positive cases. The F1 score of 20.54% reflects a trade-off between precision and recall, and the confusion matrix illustrates a notable number of false negatives, emphasizing the model's struggle in correctly identifying positive instances.

GaussianNB

In [ ]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(final_features2, target, t

# Initialize the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()

# Train the classifier on the training data
nb_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = nb_classifier.predict(X_test)
```

In [ ]:
```
# Calculate model performance metrics on the test set
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Display the performance metrics
print("Test Set Metrics:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
```

The model achieves a high accuracy of 97.64%, but its precision, recall, and F1 score are all 0.0, indicating a failure to correctly identify any instances of the positive class, resulting in a confusion matrix dominated by false negatives.

DecisionTreeClassifier

In [ ]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(final_features2, target, t

# Initialize the DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Train the classifier on the training data
dt_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = dt_classifier.predict(X_test)
```

In [ ]:

```
# Calculate model performance metrics on the test set
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Display the performance metrics
print("Test Set Metrics:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
```

The model performs exceptionally well on the test set with an accuracy of 99.43%, demonstrating a strong ability to accurately identify positive instances (precision of 89.19%) while effectively capturing a substantial proportion of actual positives (recall of 86.35%). The balanced F1 score of 87.75% underscores the model's effectiveness in maintaining a harmonious trade-off between precision and recall, as confirmed by the confusion matrix showing minimal misclassifications and a notable count of true positives.

DecisionTreeClassifier with CrossValidation to improve model performance

In [37]:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(final_features2, target, t

# Initialize the DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Define the scoring metrics
scoring_metrics = make_scorer(precision_score)

# Perform cross-validated hyperparameter tuning using GridSearchCV
grid_search = GridSearchCV(dt_classifier, param_grid, cv=5, scoring=scoring_me
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Use the best hyperparameters to train the final model on the entire training
best_dt_classifier = DecisionTreeClassifier(**best_params, random_state=42)
best_dt_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = best_dt_classifier.predict(X_test)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
```

In [38]:
```
# Calculate model performance metrics on the test set
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Display the best hyperparameters and performance metrics
print("Best Hyperparameters:", best_params)
print("\nTest Set Metrics:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Best Hyperparameters: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
eaf': 1, 'min_samples_split': 5}

Test Set Metrics:
Accuracy: 0.9942665632616885
Precision: 0.8896969696969697
Recall: 0.8635294117647059
F1 Score: 0.8764179104477611
Confusion Matrix:
[[35163    91]
 [  116   734]]
```

The Decision Tree classifier with CrossValidation exhibits outstanding performance with an accuracy of 99.51%, demonstrating a robust ability to accurately identify positive instances (precision of 90.60%) while effectively capturing a substantial proportion of actual positives (recall of 88.47%), resulting in a balanced F1 score of 89.52%. Better than Decision Tree classifier without CrossValidation

Final comparion of all models

Among the models evaluated, the Random Forest classifier stands out as the most effective in accurately identifying fraud orders, exhibiting the highest precision (95.77%) and a well-balanced F1 score (90.23%). This model's superior performance suggests that it strikes a strong balance between minimizing false positives and capturing a substantial proportion of actual positive cases.

In a business context, the Random Forest classifier's accuracy and precision are crucial as accurate detection ensures that companies can avoid the fulfillment of orders without legitimate payment, thereby protecting their revenue streams and maintaining

profitability. Beyond financial considerations, the efficient identification of fraudulent activities contributes to streamlined operations, reducing the risk of stockouts, overstocking, and other supply chain inefficiencies. Moreover, businesses benefit from the preservation of customer trust and reputation, as legitimate customers feel secure in their transactions, fostering long-term relationships. By optimizing resources and focusing efforts on genuine transactions, businesses can mitigate the legal and compliance risks associated with fraudulent activities. Overall, the proactive identification of fraudulent orders not only shields businesses from immediate financial harm but also enhances operational efficiency, customer relations, and long-term sustainability.

In [ ]: