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

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

#Plotly Graphing Libraries
from plotly.offline import init_notebook_mode, iplot
import cufflinks
cufflinks.go_offline()
cufflinks.set_config_file(world_readable=True, theme='pearl')
import plotly.graph_objs as go
import plotly
import plotly.express as px

## Machine Learning Libaries
from sklearn import svm,metrics,tree,preprocessing,linear_model
from sklearn.preprocessing import MinMaxScaler,StandardScaler
import tensorflow as tf
import keras
from sklearn.model_selection import train_test_split,cross_val_score, cross_val_predict
from sklearn import svm,metrics,tree,preprocessing,linear_model
from sklearn.preprocessing import MinMaxScaler,StandardScaler
import statsmodels.api as sm
from sklearn.metrics import accuracy_score,mean_squared_error,recall_score,confusion_matrix,f1_score,roc_curve, auc
from keras import Sequential
from keras.layers import Dense, Activation
from keras.callbacks import CSVLogger
from sklearn.neural_network import MLPClassifier
from dmba import classificationSummary

# from tensorflow_core.estimator import inputs
```

In [2]:
```
# Importing the dataset
df = pd.read_csv('DataCoSupplyChainDataset.csv',header= 0,encoding= 'unicode_escape')
```

In [3]:
```
df.head()
```

Out[3]:

	Type	Days for shipping (real)	Days for shipment (scheduled)	Benefit per order	Sales per customer	Delivery Status	Late_delivery_risk	Category Id	Category Name	Customer City	...	Order Zipcode	Product Card Id	Product Category Id	Des
0	DEBIT	3	4	91.250000	314.640015	Advance shipping	0	73	Sporting Goods	Caguas	...	NaN	1360	73	
1	TRANSFER	5	4	-249.089996	311.359985	Late delivery	1	73	Sporting Goods	Caguas	...	NaN	1360	73	
2	CASH	4	4	-247.779999	309.720001	Shipping on time	0	73	Sporting Goods	San Jose	...	NaN	1360	73	
3	DEBIT	3	4	22.860001	304.809998	Advance shipping	0	73	Sporting Goods	Los Angeles	...	NaN	1360	73	
4	PAYMENT	2	4	134.210007	298.250000	Advance shipping	0	73	Sporting Goods	Caguas	...	NaN	1360	73	

5 rows × 53 columns

Data Description

In [4]:
```
df.shape
```

Out[4]: (180519, 53)

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180519 entries, 0 to 180518
Data columns (total 53 columns):
 #   Column                         Non-Null Count   Dtype  
---  ------                         --------------   -----  
 0   Type                           180519 non-null  object 
 1   Days for shipping (real)       180519 non-null  int64  
 2   Days for shipment (scheduled)  180519 non-null  int64  
 3   Benefit per order              180519 non-null  float64
 4   Sales per customer             180519 non-null  float64
 5   Delivery Status                180519 non-null  object 
 6   Late_delivery_risk             180519 non-null  int64  
 7   Category Id                    180519 non-null  int64  
 8   Category Name                  180519 non-null  object 
 9   Customer City                  180519 non-null  object 
 10  Customer Country               180519 non-null  object 
 11  Customer Email                 180519 non-null  object 
 12  Customer Fname                 180519 non-null  object 
 13  Customer Id                    180519 non-null  int64  
 14  Customer Lname                 180511 non-null  object 
 15  Customer Password              180519 non-null  object 
 16  Customer Segment               180519 non-null  object 
 17  Customer State                 180519 non-null  object 
 18  Customer Street                180519 non-null  object 
 19  Customer Zipcode               180516 non-null  float64
 20  Department Id                  180519 non-null  int64  
 21  Department Name                180519 non-null  object 
 22  Latitude                       180519 non-null  float64
 23  Longitude                      180519 non-null  float64
 24  Market                         180519 non-null  object 
 25  Order City                     180519 non-null  object 
 26  Order Country                  180519 non-null  object 
 27  Order Customer Id              180519 non-null  int64  
 28  order date (DateOrders)        180519 non-null  object 
 29  Order Id                       180519 non-null  int64  
 30  Order Item Cardprod Id         180519 non-null  int64  
 31  Order Item Discount            180519 non-null  float64
 32  Order Item Discount Rate       180519 non-null  float64
 33  Order Item Id                  180519 non-null  int64  
 34  Order Item Product Price       180519 non-null  float64
 35  Order Item Profit Ratio        180519 non-null  float64
 36  Order Item Quantity            180519 non-null  int64  
 37  Sales                          180519 non-null  float64
 38  Order Item Total               180519 non-null  float64
 39  Order Profit Per Order         180519 non-null  float64
 40  Order Region                   180519 non-null  object 
 41  Order State                    180519 non-null  object 
 42  Order Status                   180519 non-null  object 
 43  Order Zipcode                  24840 non-null   float64
 44  Product Card Id                180519 non-null  int64  
 45  Product Category Id            180519 non-null  int64  
 46  Product Description            0 non-null       float64
 47  Product Image                  180519 non-null  object 
 48  Product Name                   180519 non-null  object 
 49  Product Price                  180519 non-null  float64
 50  Product Status                 180519 non-null  int64  
 51  shipping date (DateOrders)     180519 non-null  object 
 52  Shipping Mode                  180519 non-null  object 
dtypes: float64(15), int64(14), object(24)
memory usage: 73.0+ MB
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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
```

Combining the Last Name and First names to identify unique customers

In [7]:
```
df['Cust_Full_Name'] = df['Customer Fname'].astype(str) + df['Customer Lname'].astype(str)
```

Data Cleaning

Dropping unimporant columns

In [8]:
```
df.drop(['Customer Email','Product Status','Customer Password','Customer Street','Customer Fname',
         'Customer Lname','Latitude','Longitude','Product Description','Product Image','Order Zipcode',
         'shipping date (DateOrders)'],axis=1, inplace = True)
```

In [9]:
```
df.shape
```

Out[9]:
```
(180519, 42)
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
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 Id                      0
Customer Segment                 0
Customer State                   0
Customer Zipcode                 3
Department Id                    0
Department Name                  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
Product Card Id                  0
Product Category Id              0
Product Name                     0
Product Price                    0
Shipping Mode                    0
Cust_Full_Name                   0
dtype: int64
```

Customer Zipcode has 3 null values which we will fill with 0 as we cannot be sure of the zip code of the customers

In [11]:
```
df['Customer Zipcode'] = df['Customer Zipcode'].fillna(0)
```

In [ ]:

Creating new column using the Order Date Column

In [12]:
```
df['order date (DateOrders)'].head()
```

Out[12]:
```
0    1/31/2018 22:56
1    1/13/2018 12:27
2    1/13/2018 12:06
3    1/13/2018 11:45
4    1/13/2018 11:24
Name: order date (DateOrders), dtype: object
```

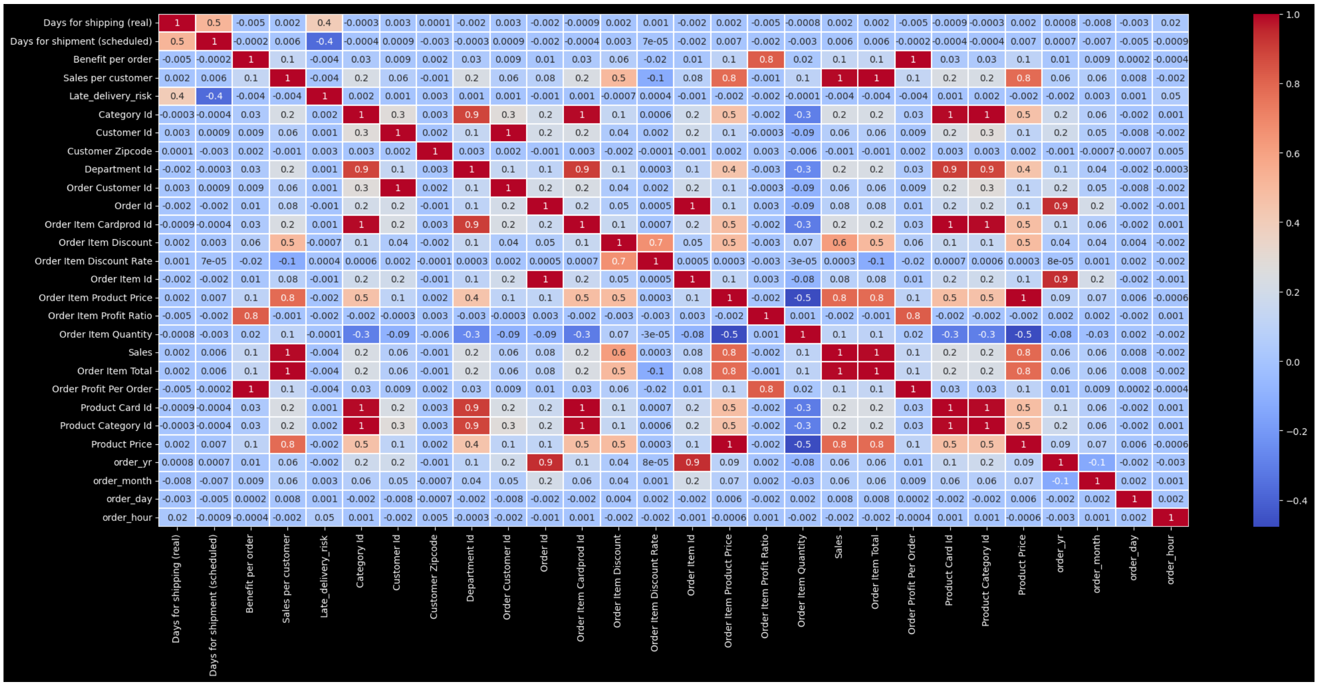
In [13]:
```
## Splitting Order dates and creating new columns
df['order_yr']= pd.DatetimeIndex(df['order date (DateOrders)']).year
df['order_month'] = pd.DatetimeIndex(df['order date (DateOrders)']).month
df['order_day'] = pd.DatetimeIndex(df['order date (DateOrders)']).weekday
df['order_hour'] = pd.DatetimeIndex(df['order date (DateOrders)']).hour
```

In [ ]:

Data Visuzalization

In [14]:
```
plt.style.use("dark_background")
fig, ax = plt.subplots(figsize=(25,10))          # figsize
sns.heatmap(df.corr(),annot=True, linewidths=.3 ,fmt='.1g', cmap= 'coolwarm')
```

Out[14]:
```
<AxesSubplot:>
```

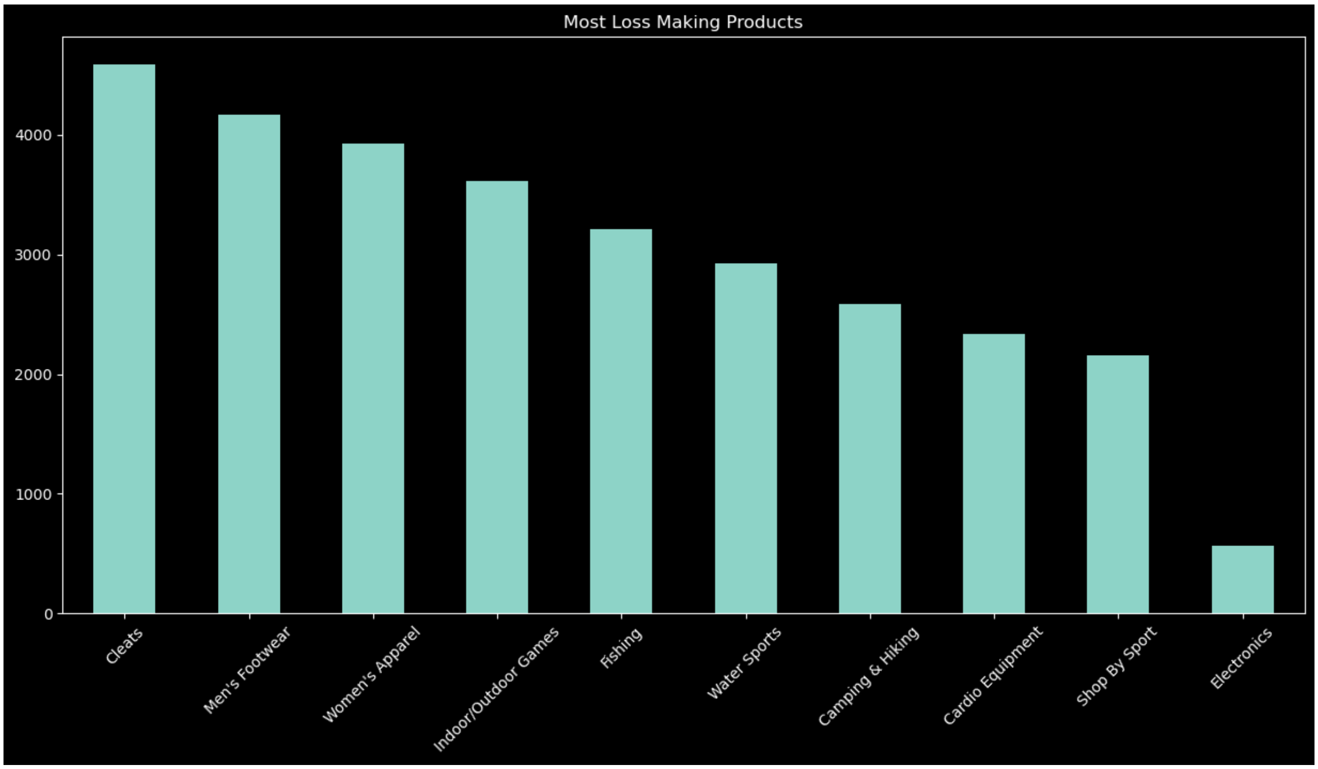


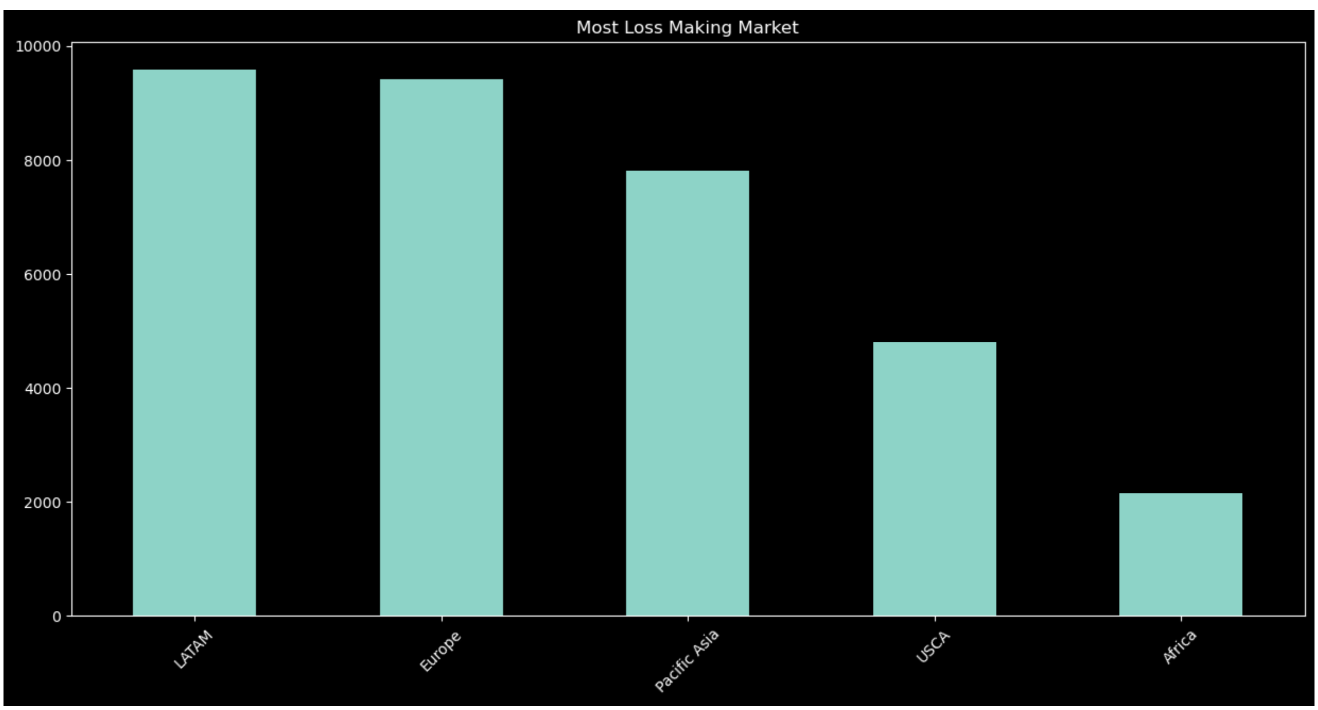
Some products have a negative benefit per order, indicating that the orders are costing the company money.

In [15]:
```
loss = df[(df['Benefit per order']<0)]
```

In [16]:
```
plt.figure(1)
loss['Category Name'].value_counts().nlargest(10).plot.bar(figsize=(15,7), title="Most Loss Making Products")
plt.xticks(rotation = 45)
plt.figure(2)
loss['Market'].value_counts().nlargest(10).plot.bar(figsize=(15,7), title="Most Loss Making Market")
plt.xticks(rotation = 45)
```

Out[16]:
```
(array([0, 1, 2, 3, 4]),
 [Text(0, 0, 'LATAM'),
  Text(1, 0, 'Europe'),
  Text(2, 0, 'Pacific Asia'),
  Text(3, 0, 'USCA'),
  Text(4, 0, 'Africa')])
```





Order Status as per the payment types

In [17]:
```
status = df.groupby('Type')['Order Status'].value_counts()
status_df = status.to_frame()
```

In [18]:
```
status_df
```

Out[18]:

Type	Order Status	Order Status
CASH	CLOSED	19616
DEBIT	COMPLETE	59491
	ON_HOLD	9804
PAYMENT	PENDING_PAYMENT	39832
	PAYMENT_REVIEW	1893
TRANSFER	PROCESSING	21902
	PENDING	20227
	SUSPECTED_FRAUD	4062
	CANCELED	3692

As we can see that only Transfer payments have a possible suspected fraud situation, what products have the most fraud?

In [19]:
```
fraud_region = df[(df['Order Status'] == 'SUSPECTED_FRAUD')]
```

In [20]:
```
plt.style.use("dark_background")
temp = fraud_region['Category Name'].value_counts().nlargest(10)
temp.iplot(kind='bar', xTitle = 'Category', yTitle = "Count", title = 'Count of Products with suspected fraud', color = '#FF0000'
```

Data Modelling and Neural Networks to predict possible fraud

In [21]:
```
# Creating a copy of the datafram
train_df = df.copy()
```

In [22]:
```
# Creating Binary enocode for Suspected Fraud and Late delivery

train_df['fraud'] = np.where(train_df['Order Status'] == 'SUSPECTED_FRAUD', 1, 0)
train_df['late_delivery']=np.where(train_df['Delivery Status'] == 'Late delivery', 1, 0)
```

In [23]:
```
train_df.columns
```

Out[23]:
```
Index(['Type', 'Days for shipping (real)', 'Days for shipment (scheduled)',
       'Benefit per order', 'Sales per customer', 'Delivery Status',
       'Late_delivery_risk', 'Category Id', 'Category Name', 'Customer City',
       'Customer Country', 'Customer Id', 'Customer Segment', 'Customer State',
       'Customer Zipcode', 'Department Id', 'Department Name', 'Market',
       'Order City', 'Order Country', 'Order Customer Id',
       'order date (DateOrders)', 'Order Id', 'Order Item Cardprod Id',
       'Order Item Discount', 'Order Item Discount Rate', 'Order Item Id',
       'Order Item Product Price', 'Order Item Profit Ratio',
       'Order Item Quantity', 'Sales', 'Order Item Total',
       'Order Profit Per Order', 'Order Region', 'Order State', 'Order Status',
       'Product Card Id', 'Product Category Id', 'Product Name',
       'Product Price', 'Shipping Mode', 'Cust_Full_Name', 'order_yr',
       'order_month', 'order_day', 'order_hour', 'fraud', 'late_delivery'],
      dtype='object')
```

In [24]:
```
## Removing Identical columns after creating new columns

train_df.drop(['Delivery Status','Late_delivery_risk','Order Status','order date (DateOrders)'], axis=1, inplace=True)
```

In [25]:
```
## Final dimensions of the dataset after wrangling and cleaning

train_df.shape
```

Out[25]:
```
(180519, 44)
```

In [26]:
```
train_df.dtypes
```

Out[26]:
```
Type                              object
Days for shipping (real)           int64
Days for shipment (scheduled)      int64
Benefit per order                float64
Sales per customer               float64
Category Id                        int64
Category Name                     object
Customer City                     object
Customer Country                  object
Customer Id                        int64
Customer Segment                  object
Customer State                    object
Customer Zipcode                 float64
Department Id                      int64
Department Name                   object
Market                            object
Order City                        object
Order Country                     object
Order Customer Id                  int64
Order Id                           int64
Order Item Cardprod Id             int64
Order Item Discount              float64
Order Item Discount Rate         float64
Order Item Id                      int64
Order Item Product Price         float64
Order Item Profit Ratio          float64
Order Item Quantity                int64
Sales                            float64
Order Item Total                 float64
Order Profit Per Order           float64
Order Region                      object
Order State                       object
Product Card Id                    int64
Product Category Id                int64
Product Name                      object
Product Price                    float64
Shipping Mode                     object
Cust_Full_Name                    object
order_yr                           int64
order_month                        int64
order_day                          int64
order_hour                         int64
fraud                              int32
late_delivery                      int32
dtype: object
```

Encoding all Object type variables

In [27]:
```
le = preprocessing.LabelEncoder()
#convert the categorical columns into numeric
train_df['Customer Country']  = le.fit_transform(train_df['Customer Country'])
train_df['Market']            = le.fit_transform(train_df['Market'])
train_df['Type']              = le.fit_transform(train_df['Type'])
train_df['Product Name']      = le.fit_transform(train_df['Product Name'])
train_df['Customer Segment']  = le.fit_transform(train_df['Customer Segment'])
train_df['Customer State']    = le.fit_transform(train_df['Customer State'])
train_df['Order Region']      = le.fit_transform(train_df['Order Region'])
train_df['Order City']        = le.fit_transform(train_df['Order City'])
train_df['Category Name']     = le.fit_transform(train_df['Category Name'])
train_df['Customer City']     = le.fit_transform(train_df['Customer City'])
train_df['Department Name']   = le.fit_transform(train_df['Department Name'])
train_df['Order State']       = le.fit_transform(train_df['Order State'])
train_df['Shipping Mode']     = le.fit_transform(train_df['Shipping Mode'])
train_df['Order Country']     = le.fit_transform(train_df['Order Country'])
train_df['Cust_Full_Name']    = le.fit_transform(train_df['Cust_Full_Name'])
```

In [28]:
```
train_df.head()
```

Out[28]:

	Type	Days for shipping (real)	Days for shipment (scheduled)	Benefit per order	Sales per customer	Category Id	Category Name	Customer City	Customer Country	Customer Id	...	Product Name	Product Price	Shipping Mode	Cust_Full_Name
0	1	3	4	91.250000	314.640015	73	40	66	1	20755	...	78	327.75	3	1876
1	3	5	4	-249.089996	311.359985	73	40	66	1	19492	...	78	327.75	3	5378
2	0	4	4	-247.779999	309.720001	73	40	452	0	19491	...	78	327.75	3	4429
3	1	3	4	22.860001	304.809998	73	40	285	0	19490	...	78	327.75	3	12929
4	2	2	4	134.210007	298.250000	73	40	66	1	19489	...	78	327.75	3	10638

5 rows × 44 columns

Preparing Data for Neural Networks

Creating a Validation Set from the orignal data

In [29]:
```
# rows = int(train_df.shape[0] * 0.1)

# randomly select the specified number of rows
# random_rows = np.random.choice(train_df.index, rows, replace=False)

# create a new dataframe from the randomly selected rows
# validation_df = df.loc[random_rows]

# Dropping those rows from the original dataset
# train_df.drop(random_rows,inplace = True)
```

In [30]:
```
## Selecting 10% of the dataset for validation

fraction_of_rows = train_df.sample(frac=0.1, random_state = 1)
fraction_of_rows
```

Out[30]:

	Type	Days for shipping (real)	Days for shipment (scheduled)	Benefit per order	Sales per customer	Category Id	Category Name	Customer City	Customer Country	Customer Id	...	Product Name	Product Price	Shipping Mode	Cust_Ful
101369	3	2	4	71.849998	197.919998	46	30	529	0	6862	...	67	49.980000	3	
3026	2	5	4	57.529999	164.380005	67	16	66	1	15052	...	17	164.380005	3	
57549	1	5	4	22.680000	226.759995	17	12	66	1	7391	...	71	59.990002	3	
127144	2	6	4	-3.020000	103.989998	18	34	66	1	2588	...	56	129.990005	3	
160375	3	2	4	11.760000	37.560001	75	45	66	1	19769	...	25	39.750000	3	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
22603	1	5	4	34.849998	99.580002	17	12	66	1	4792	...	71	59.990002	3	
152893	3	2	1	135.190002	399.980011	45	18	271	0	4055	...	24	399.980011	0	
172935	0	2	2	-39.320000	347.980011	45	18	66	1	148	...	24	399.980011	2	
79244	1	2	1	40.000000	159.990005	48	46	66	1	8580	...	70	199.990005	0	
62015	1	2	4	50.959999	195.990005	48	46	261	0	6574	...	70	199.990005	3	

18052 rows × 44 columns

In [31]:
```
## Creating a list of index of the samples to be dropped from the main dataset

index = fraction_of_rows.index.values.tolist()
index
```

Out[31]:
```
[101369,
 3026,
 57549,
 127144,
 160375,
 21822,
 87233,
 12914,
 51333,
 18605,
 115091,
 125669,
 22070,
 11734,
 58228,
 138444,
 85450,
 73875,
 61313,
```

In [32]:
```
## Dropping rows with the index numbers

train_df.drop(index = index, inplace = True)
```

In [33]:
```
train_df.shape
```

Out[33]:
```
(162467, 44)
```

In [34]:
```
## Resetting index for the validation dataset

fraction_of_rows.reset_index(drop = True, inplace=True)
```

In [35]:
```
fraction_of_rows.shape
```

Out[35]:
```
(18052, 44)
```

In [36]:
```
## Creating X and y for dependant and independent variables

X = train_df.loc[:,train_df.columns != 'fraud']
y = train_df['fraud']
```

In [37]:
```
## Splitting the dataset into training and test data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state=42)
```

In [38]:
```
## Creating dataframe for X and y validation dataset

X_valid = fraction_of_rows.loc[:,fraction_of_rows.columns != 'fraud']
y_valid = fraction_of_rows['fraud']
```

In [39]:
```
X_train.shape
```

Out[39]:
```
(113726, 43)
```

In [40]:
```
# Defining the classes
classes = sorted(y_train.unique())
```

Standardizing the data

In [41]:
```
## Standardizing the X dataset

ss = StandardScaler()
X_train=ss.fit_transform(X_train)
X_test=ss.transform(X_test)
X_valid=ss.transform(X_valid)
```

Creating MLPClassifier Model

In [42]:
```
clf = MLPClassifier(hidden_layer_sizes=(6), activation='logistic', solver='lbfgs',random_state=1)
```

In [43]:
```
## Fitting the data using MLPClassifier

clf.fit(X_train, y_train)
```

Out[43]:
```
MLPClassifier(activation='logistic', hidden_layer_sizes=6, random_state=1,
              solver='lbfgs')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [44]:
```
## Predicting the trained model

clf.predict(X_test)
```

Out[44]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [45]:
```
## Creating a Classification matrix for MLPClassifier

classificationSummary(y_test, clf.predict(X_test), class_names=classes)
```

```
Confusion Matrix (Accuracy 0.9789)

       Prediction
Actual     0     1
     0 47280   330
     1   697   434
```

In [ ]:

Creating a Custom Neural Network Model

In [46]:
```
train_df.shape
```

Out[46]:
```
(162467, 44)
```

In [47]:
```
keras.layers.BatchNormalization()
model = Sequential()
#First Hidden Layer
model.add(Dense(1024, activation='relu', kernel_initializer='random_normal', input_dim=43)) #As we have 43 columns

#All other hidden layers in a for loop with nodes reducing in each loop
nodes = 1024
for i in range(9):
    nodes = nodes // 2
    model.add(Dense(nodes, activation='relu', kernel_initializer='random_normal'))

#Output Layer
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 1024)              45056     
                                                                 
 dense_1 (Dense)             (None, 512)               524800    
                                                                 
 dense_2 (Dense)             (None, 256)               131328    
                                                                 
 dense_3 (Dense)             (None, 128)               32896     
                                                                 
 dense_4 (Dense)             (None, 64)                8256      
                                                                 
 dense_5 (Dense)             (None, 32)                2080      
                                                                 
 dense_6 (Dense)             (None, 16)                528       
                                                                 
 dense_7 (Dense)             (None, 8)                 136       
                                                                 
 dense_8 (Dense)             (None, 4)                 36        
                                                                 
 dense_9 (Dense)             (None, 2)                 10        
                                                                 
 dense_10 (Dense)            (None, 1)                 3         
                                                                 
 activation (Activation)     (None, 1)                 0         
                                                                 
=================================================================
Total params: 745,129
Trainable params: 745,129
Non-trainable params: 0
_________________________________________________________________
```

As F1 score is not accessible in Keras, binary crossentropy is used to measure loss and accuracy because the output data is binary classification.

In [48]:
```
## Compiling the model created

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

Fitting the model using custom neural network

Intitally we'll tried with 10 epochs which had a good accuracy but the loss was high, then we increased it to 15 which reduced the loss significantly and we stopped there to avoid overfitting of the data.

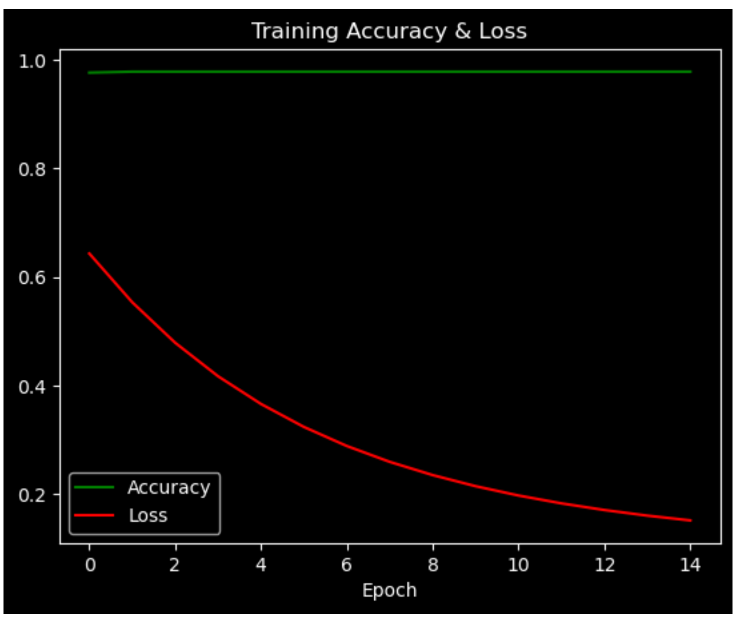
In [49]:
```
csvlogger = CSVLogger('training.log',separator=',',append=False)

result = model.fit(X_train, y_train, batch_size = 512, epochs = 15, callbacks=[csvlogger])
```

```
Epoch 1/15
223/223 [==============================] - 8s 26ms/step - loss: 0.6431 - accuracy: 0.9759
Epoch 2/15
223/223 [==============================] - 5s 24ms/step - loss: 0.5534 - accuracy: 0.9777
Epoch 3/15
223/223 [==============================] - 6s 25ms/step - loss: 0.4789 - accuracy: 0.9777
Epoch 4/15
223/223 [==============================] - 6s 25ms/step - loss: 0.4172 - accuracy: 0.9777
Epoch 5/15
223/223 [==============================] - 5s 24ms/step - loss: 0.3662 - accuracy: 0.9777
Epoch 6/15
223/223 [==============================] - 5s 24ms/step - loss: 0.3240 - accuracy: 0.9777
Epoch 7/15
223/223 [==============================] - 6s 25ms/step - loss: 0.2889 - accuracy: 0.9777
Epoch 8/15
223/223 [==============================] - 6s 25ms/step - loss: 0.2597 - accuracy: 0.9777
Epoch 9/15
223/223 [==============================] - 6s 27ms/step - loss: 0.2354 - accuracy: 0.9777
Epoch 10/15
223/223 [==============================] - 7s 31ms/step - loss: 0.2150 - accuracy: 0.9777
Epoch 11/15
223/223 [==============================] - 6s 29ms/step - loss: 0.1978 - accuracy: 0.9777
Epoch 12/15
223/223 [==============================] - 7s 31ms/step - loss: 0.1834 - accuracy: 0.9777
Epoch 13/15
223/223 [==============================] - 7s 30ms/step - loss: 0.1712 - accuracy: 0.9777
Epoch 14/15
223/223 [==============================] - 7s 31ms/step - loss: 0.1608 - accuracy: 0.9777
Epoch 15/15
223/223 [==============================] - 7s 33ms/step - loss: 0.1521 - accuracy: 0.9777
```

In [50]:
```
plt.plot(result.history['accuracy'],'green',label='Accuracy')
plt.plot(result.history['loss'],'red',label='Loss')
plt.title('Training Accuracy & Loss')
plt.xlabel('Epoch')
plt.legend(loc=0)
```

Out[50]:
```
<matplotlib.legend.Legend at 0x171efd29f70>
```



In [51]:
```
# Predicting the custom model model

train_evaluate=model.evaluate(X_train, y_train)
test_evaluate=model.evaluate(X_test, y_test)
print('accuracy for Train set is',train_evaluate)
print('accuracy for Test set is',test_evaluate) # evaluation of model.
yf_pred1=model.predict(X_test,batch_size=512,verbose=1)
yf_pred=np.argmax(yf_pred1,axis=1)
print(f1_score(y_test,yf_pred,average="weighted"))
```

```
3554/3554 [==============================] - 12s 3ms/step - loss: 0.1481 - accuracy: 0.9777
1524/1524 [==============================] - 5s 3ms/step - loss: 0.1501 - accuracy: 0.9768
accuracy for Train set is [0.14811132848262787, 0.9776568412780762]
accuracy for Test set is [0.15007199347019196, 0.9767957329750061]
96/96 [==============================] - 1s 9ms/step
0.9653297639890498
```

As we can see that the Train and test accuracy is very high at ~99% and loss is at ~8.88%.
Along with that the F1 Score calculated is 96.58%.

In [ ]:

Validating the Models

MLP Classifier

In [52]:
```
## Predicting the validation dataset

clf_pred = clf.predict(X_valid)
clf_pred
```

Out[52]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [53]:
```
## Classification Matrix for Validation Dataset

classificationSummary(y_valid, clf.predict(X_valid), class_names=classes)
```

```
Confusion Matrix (Accuracy 0.9787)

       Prediction
Actual     0     1
     0 17532   130
     1   254   136
```

In [54]:
```
accuracy_score(y_valid,clf_pred)
```

Out[54]:
```
0.9787281187680036
```

Type *Markdown* and LaTeX: α^2

Custom NN

In [55]:
```
predictions = model.predict(X_valid)
```

```
565/565 [==============================] - 2s 3ms/step
```

In [57]:
```
predictions = (predictions > 0.5).astype(np.float32)
```

In [58]:
```
predictions
```

Out[58]:
```
array([[0.],
       [0.],
       [0.],
       ...,
       [0.],
       [0.],
       [0.]], dtype=float32)
```

In [59]:
```
accuracy_score(y_valid,predictions)
```

Out[59]:
```
0.9783957456237536
```

In [ ]:
```
keras.backend.clear_session()
```