In [1]:

```
import pickle
import time
import numpy as np
import pandas as pd
import numpy as np
from sklearn.metrics import f1_score
```

In [2]:

```
file ="KNNimputer.pkl"
with open(file,'rb') as file:
    imputer = pickle.load(file)
```

In [3]:

```
file ="SVD.pkl"
with open(file,'rb') as file:
    SVD = pickle.load(file)
```

In [4]:

```
file ="PCA.pkl"
with open(file,'rb') as file:
    PCA = pickle.load(file)
```

In [5]:

```
file ="scaler.pkl"
with open(file,'rb') as file:
    Minmax = pickle.load(file)
```

In [6]:

```
file ="Tree_sale.pkl"
with open(file,'rb') as file:
    tree_sale = pickle.load(file)
```

In [7]:

```
file ="Tree_fore.pkl"
with open(file,'rb') as file:
    tree_forecast = pickle.load(file)
```

In [8]:

```
file = "bestmodel.pkl"
with open(file,'rb') as file:
    bestmodel = pickle.load(file)
```

In [9]:

```
df=pd.read_csv('Kaggle_Training_Dataset_v2.csv')
```

```
C:\Users\shubh\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3146: DtypeWarning: Columns (
0) 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,
```

In [10]:

```
df.head()
```

Out[10]:

	sku	national_inv	lead_time	in_transit_qty	forecast_3_month	forecast_6_month	forecast_9_month	sales_1_month	sales_3_month	sal
0	1026827	0.0	NaN	0.0	0.0	0.0	0.0	0.0	0.0	
1	1043384	2.0	9.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	1043696	2.0	NaN	0.0	0.0	0.0	0.0	0.0	0.0	
3	1043852	7.0	8.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	1044048	8.0	NaN	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 23 columns

In [13]:

```
def final_function_1(X):

    # replacing -99 by Nan in performance column
    X.perf_6_month_avg.replace({-99.0 : np.nan},inplace=True)
    X.perf_12_month_avg.replace({-99.0 : np.nan},inplace=True)

    # Converting categories like Yes and No to 0s and 1s
    categorical_columns = ['rev_stop','stop_auto_buy','ppap_risk','oe_constraint','deck_risk','potential_
    for col in categorical_columns:
        X[col].replace({'Yes':1,'No':0},inplace=True)
        X[col]=X[col].astype(int)
    # Removing outliers points by taking only values below 99 percentile
    X = X[(X.national_inv >= 0.000) & (X.national_inv <= 5487.000) & (X.in_transit_qty <= 5510.000 ) &\
        (X.forecast_3_month <= 2280.000) & (X.forecast_6_month <= 4335.659999999916) &\
        (X.forecast_9_month <= 6316.000) & (X.sales_1_month <= 693.000) & (X.sales_3_month <= 2229.000) &\
        (X.sales_6_month <= 4410.000) & (X.sales_9_month <= 6698.000) & (X.min_bank <= 679.6599999999162)]


    # KNN Imputation
    cols = X.columns
    X = pd.DataFrame(imputer.transform(X),columns = cols)

    # Getting PCA Features
    X_pca = PCA.transform(X)
    #Adding PCA  features in the main dataframe
    for i in range(2):
        X['PCA'+str(i)] = X_pca[:,i]
    # Getting SVD Features
    X_svd = SVD.transform(X)

    # Adding SVD  features in the main dataframe
    for i in range(2):
        X['SVD'+str(i)] = X_svd[:,i]

    # Dicretisation using Decision Tree
    X['sales_9_tree'] = tree_sale.predict_proba(X.sales_9_month.to_frame())[:,1]

    # For forecast columns
    X['forecast_9_tree'] = tree_forecast.predict_proba(X.forecast_9_month.to_frame())[:,1]


    # Performing MinMaxScaler on Data
    cols = X.columns
    X = pd.DataFrame(Minmax.transform(X),columns = cols)

    opt = bestmodel.predict(X)

    return opt
```

In [14]:

```
data_temp = df.head()
target_data = df['went_on_backorder']
data_temp = data_temp.drop(['sku','went_on_backorder'],axis=1)

start_time = time.time()
output = final_function_1(data_temp)
print("Output : ",output)
print("Time Taken for execution is {}".format((time.time() - start_time)))
```

```
Output :  [0 0 0 0 0]
Time Taken for execution is 1.0967717170715332
```

In [17]:

```
def final_function_2(X,Y):

    # replacing -99 by Nan in performance column
    X.perf_6_month_avg.replace({-99.0 : np.nan},inplace=True)
    X.perf_12_month_avg.replace({-99.0 : np.nan},inplace=True)

    # Converting the Target variable
    Y.replace({'Yes':1,'No':0},inplace=True)
    Y.astype(int)

    # Converting categories like Yes and No to 0s and 1s
    categorical_columns = ['rev_stop','stop_auto_buy','ppap_risk','oe_constraint','deck_risk','potential_
    for col in categorical_columns:
        X[col].replace({'Yes':1,'No':0},inplace=True)
        X[col]=X[col].astype(int)

    X['went_on_backorder'] = Y

    # Removing outliers points by taking only values below 99 percentile
    X = X[(X.national_inv >= 0.000) & (X.national_inv <= 5487.000) & (X.in_transit_qty <= 5510.000 ) &\
        (X.forecast_3_month <= 2280.000) & (X.forecast_6_month <= 4335.659999999916) &\
        (X.forecast_9_month <= 6316.000) & (X.sales_1_month <= 693.000) & (X.sales_3_month <= 2229.000) &\
        (X.sales_6_month <= 4410.000) & (X.sales_9_month <= 6698.000) & (X.min_bank <= 679.6599999999162)]
    Y = X['went_on_backorder']
    print("Shape of outlier free target :",Y.shape)

    X = X.drop(columns='went_on_backorder',axis=1)

    print("Shape of outlier free dataframe :",X.shape)


    # KNN Imputation
    cols = X.columns
    X = pd.DataFrame(imputer.transform(X),columns = cols)


    # Getting PCA Features
    X_pca = PCA.transform(X)
    #Adding PCA  features in the main dataframe
    for i in range(2):
        X['PCA'+str(i)] = X_pca[:,i]
    # Getting SVD Features
    X_svd = SVD.transform(X)

    # Adding SVD  features in the main dataframe
    for i in range(2):
        X['SVD'+str(i)] = X_svd[:,i]

    # Dicretisation using Decision Tree
    X['sales_9_tree'] = tree_sale.predict_proba(X.sales_9_month.to_frame())[:,1]

    # For forecast columns
    X['forecast_9_tree'] = tree_forecast.predict_proba(X.forecast_9_month.to_frame())[:,1]


    # Performing MinMaxScaler on Data
    cols = X.columns
    X = pd.DataFrame(Minmax.transform(X),columns = cols)

    opt = bestmodel.predict(X)

    f1 = f1_score(Y,opt,average="macro")

    return f1
```

In [18]:

```
data_temp = df.head(2000)
target_data = data_temp['went_on_backorder']
data_temp = data_temp.drop(['sku','went_on_backorder'],axis=1)

start_time = time.time()
output = final_function_2(data_temp,target_data)
print("F1_Score : ",output)
print("Time Taken for execution is {}".format((time.time() - start_time)))
```

```
C:\Users\shubh\anaconda3\lib\site-packages\pandas\core\series.py:4563: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().replace(
Shape of outlier free target : (1949,)
Shape of outlier free dataframe : (1949, 21)
F1_Score :  0.82197661673365
Time Taken for execution is 28.098380088806152
```

In [ ]: