import pandas as pd

**#Downloading CSV’s**

orders = pd.read\_csv('orders.csv')

order\_products\_prior = pd.read\_csv('order\_products\_\_prior.csv')

products = pd.read\_csv('products.csv')

departments = pd.read\_csv('departments.csv')

aisles = pd.read\_csv('aisles.csv')

**#Merging CSVs**

products = pd.merge(products, aisles, on='aisle\_id', how='left')

products = pd.merge(products, departments, on='department\_id', how='left')

orders\_prior = pd.merge(order\_products\_prior, orders, on='order\_id', how='left')

orders\_full = pd.merge(orders\_prior, products, on='product\_id', how='left')

df = orders\_full.copy()

**#Replacing NaN values**

df['days\_since\_prior\_order'] = df['days\_since\_prior\_order'].fillna(0)

**#Feature Engineering**  
user\_features = df.groupby('user\_id').agg(

    user\_total\_orders=('order\_number', 'max'),

    user\_avg\_days\_between\_orders=('days\_since\_prior\_order', 'mean'),

    user\_days\_since\_last\_order=('days\_since\_prior\_order', 'last')

).reset\_index()

product\_features = df.groupby('product\_id').agg(

    prod\_total\_orders=('reordered', 'count'),

    prod\_reorders=('reordered', 'sum'),

    prod\_reorder\_ratio=('reordered', 'mean')

).reset\_index()

user\_product\_features = df.groupby(['user\_id', 'product\_id']).agg(

    up\_orders=('order\_number', 'count'),

    up\_first\_order=('order\_number', 'min'),

    up\_last\_order=('order\_number', 'max'),

    up\_reorders=('reordered', 'sum')

).reset\_index()

final\_features = user\_product\_features.merge(user\_features, on='user\_id', how='left').merge(product\_features, on='product\_id', how='left')

final\_features['up\_order\_rate'] = final\_features['up\_orders'] / final\_features['user\_total\_orders']

product\_info = df[['product\_id', 'product\_name', 'aisle', 'department']].drop\_duplicates()

final\_features\_new = final\_features.merge(product\_info, on='product\_id', how='left')

**#Checking skewness and kurtosis**

import pandas as pd

num\_cols = final\_features\_new.select\_dtypes(include=['int64', 'float64']).columns

skewness = final\_features\_new[num\_cols].skew()

kurtosis = final\_features\_new[num\_cols].kurtosis()

print("Skewness of numerical columns:")

print(skewness)

print("\nKurtosis of numerical columns:")

print(kurtosis)

**#Log transforming**

import numpy as np

log\_transform\_cols = [

    'up\_orders', 'up\_first\_order', 'up\_last\_order',

    'up\_reorders', 'up\_order\_rate','user\_total\_orders', 'prod\_total\_orders', 'prod\_reorders'

]

for col in log\_transform\_cols:

    final\_features\_new[col] = np.log1p(final\_features\_new[col])

**#Joining train CSV**

order\_products\_train = pd.read\_csv('D:\Product\_Reordering\env\order\_products\_\_train.csv')

train\_labels = order\_products\_train.merge(orders[['order\_id', 'user\_id']], on='order\_id', how='left')

final\_df = final\_features\_new.merge(train\_labels[['user\_id', 'product\_id', 'reordered']],

    on=['user\_id', 'product\_id'],

    how='left'

)

**#Data preprocessing**

final\_df['reordered'] = final\_df['reordered'].fillna(0).astype(int)

id\_cols = ['user\_id', 'product\_id', 'product\_name', 'aisle', 'department']

from sklearn.preprocessing import LabelEncoder

string\_columns = final\_df.columns[final\_df.dtypes == 'object']

encoders = {}

for col in string\_columns:

    le = LabelEncoder()

    final\_df[col] = le.fit\_transform(final\_df[col])

    encoders[col] = le

**#Defining x and y**

x = final\_df.drop('reordered', axis=1)

y = final\_df['reordered']

**#Undersampling**

from imblearn.under\_sampling import RandomUnderSampler

undersampler = RandomUnderSampler(

    sampling\_strategy=0.666,

    random\_state=42

)

x\_under, y\_under = undersampler.fit\_resample(x, y)

**#Train test split**

from sklearn.model\_selection import train\_test\_split

x\_train\_full, x\_temp, y\_train\_full, y\_temp = train\_test\_split(

    x\_under, y\_under, test\_size=0.3, stratify=y\_under, random\_state=42

)

x\_val, x\_test, y\_val, y\_test = train\_test\_split(

    x\_temp, y\_temp, test\_size=0.67, stratify=y\_temp, random\_state=42

)

**#Dropping identifiers before training**

x\_train\_id = x\_train\_full[id\_cols]

x\_val\_id = x\_val[id\_cols]

x\_test\_id = x\_test[id\_cols]

x\_train\_numeric = x\_train\_full.drop(columns=id\_cols)

x\_val\_numeric = x\_val.drop(columns=id\_cols)

x\_test\_numeric = x\_test.drop(columns=id\_cols)

**#Feature scaling**

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

x\_train\_scaled = scaler.fit\_transform(x\_train\_numeric)

x\_val\_scaled = scaler.transform(x\_val\_numeric)

x\_test\_scaled = scaler.transform(x\_test\_numeric)

**#Defining the architecture**

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, BatchNormalization

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from tensorflow.keras.metrics import Precision, Recall, AUC

from tensorflow.keras.optimizers import Adam

model = Sequential()

model.add(Dense(256, input\_dim=x\_train\_scaled.shape[1], activation="relu"))

model.add(BatchNormalization())

model.add(Dropout(0.3))

model.add(Dense(128, activation="relu"))

model.add(BatchNormalization())

model.add(Dropout(0.25))

model.add(Dense(64, activation="relu"))

model.add(BatchNormalization())

model.add(Dropout(0.2))

model.add(Dense(32, activation="relu"))

model.add(BatchNormalization())

model.add(Dropout(0.15))

model.add(Dense(1, activation="sigmoid"))

model.compile(

    loss="binary\_crossentropy",

    optimizer=Adam(learning\_rate=0.001),

    metrics=["accuracy", Precision(name='precision'), Recall(name='recall'), AUC(name='auc')]

)

early\_stop = EarlyStopping(

    monitor='val\_loss',

    patience=10,

    restore\_best\_weights=True,

    verbose=1

)

**#Training the model**

result = model.fit(

    x\_train\_scaled, y\_train\_full,

    validation\_data=(x\_val\_scaled, y\_val),

    epochs=50,

    batch\_size=512,

    callbacks=[early\_stop],

    verbose=1

)

**#Saving the model**

model.save('instacart\_model1.keras')

**#Evaluating model’s performance on validation and test data**

from sklearn.metrics import classification\_report, roc\_auc\_score, confusion\_matrix

y\_val\_pred\_proba = model.predict(x\_val\_scaled)

y\_val\_pred = (y\_val\_pred\_proba > 0.5).astype(int)

y\_test\_pred\_proba = model.predict(x\_test\_scaled)

y\_test\_pred = (y\_test\_pred\_proba > 0.5).astype(int)

print("VALIDATION METRICS")

print("Validation Classification Report:")

print(classification\_report(y\_val, y\_val\_pred))

print("Validation AUC:", roc\_auc\_score(y\_val, y\_val\_pred\_proba))

print("Validation Confusion Matrix:")

print(confusion\_matrix(y\_val, y\_val\_pred))

print("TEST METRICS")

print("Test Classification Report:")

print(classification\_report(y\_test, y\_test\_pred))

print("Test AUC:", roc\_auc\_score(y\_test, y\_test\_pred\_proba))

print("Test Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_test\_pred))

**#Decoding the categorical columns for business use case purpose**

x\_test\_decoded = x\_test.copy()

for col in x\_test\_decoded.columns:

    if col in encoders:

        x\_test\_decoded[col] = encoders[col].inverse\_transform(x\_test[col].astype(int))

test\_results = x\_test\_decoded.copy()

test\_results['user\_id'] = x\_test\_id['user\_id']

test\_results['product\_id'] = x\_test\_id['product\_id']

test\_results['actual\_reordered'] = y\_test

test\_results['predicted\_reordered'] = y\_test\_pred.flatten()

test\_results['reorder\_probability'] = y\_test\_pred\_proba.flatten()