```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import xgboost as xgb
import shap
import warnings
warnings.filterwarnings("ignore")

# Step 1: Load the dataset
def load_data_in_chunks(file_path, nrows=10000):
    """Load a specific number of rows from a large CSV file and add a
Serial Number column."""
    chunk = pd.read_csv(file_path, nrows=nrows)
    chunk['Serial'] = range(1, len(chunk) + 1)  # Adding Serial Number
column starting from 1
    print(f"Loaded {nrows} rows from {file_path}")
    return chunk

train_data = load_data_in_chunks(r"E:\AYUSH\amex-default-prediction\
train_data.csv", nrows=10000)
test = load_data_in_chunks(r"E:\AYUSH\amex-default-prediction\
test_data.csv", nrows=10000)
train_labels = load_data_in_chunks(r"E:\AYUSH\amex-default-prediction\
train_labels.csv", nrows=10000)
sample_submission = load_data_in_chunks(r"E:\AYUSH\amex-default-
prediction\sample_submission.csv", nrows=10000)


print("\nTop 10 rows of Train Data with Serial Numbers:")
print(train_data.head(10))
print("\nTop 10 rows of Test Data with Serial Numbers:")
print(test.head(10))
print("\nTop 10 rows of Train Data with Serial Numbers:")
print(train_labels.head(10))
print("\nTop 10 rows of Train Data with Serial Numbers:")
print(sample_submission.head(10))
```

```
Loaded 10000 rows from E:\AYUSH\amex-default-prediction\train_data.csv
Loaded 10000 rows from E:\AYUSH\amex-default-prediction\test_data.csv
Loaded 10000 rows from E:\AYUSH\amex-default-prediction\
train_labels.csv
Loaded 10000 rows from E:\AYUSH\amex-default-prediction\
sample_submission.csv
```

```
Top 10 rows of Train Data with Serial Numbers:
                                         customer_ID         S_2
P_2  \
0  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-03-09
0.938469
1  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-04-07
0.936665
2  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-05-28
0.954180
3  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-06-13
0.960384
4  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-07-16
0.947248
5  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-08-04
0.945964
6  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-09-18
0.940705
7  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-10-08
0.914767
8  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-11-20
0.950845
9  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2017-12-04
0.868580

       D_39       B_1       B_2       R_1       S_3      D_41
B_3  ...  \
0  0.001733  0.008724  1.006838  0.009228  0.124035  0.008771
0.004709  ...
1  0.005775  0.004923  1.000653  0.006151  0.126750  0.000798
0.002714  ...
2  0.091505  0.021655  1.009672  0.006815  0.123977  0.007598
0.009423  ...
3  0.002455  0.013683  1.002700  0.001373  0.117169  0.000685
0.005531  ...
4  0.002483  0.015193  1.000727  0.007605  0.117325  0.004653
0.009312  ...
5  0.001746  0.007863  1.005006  0.004220  0.110946  0.009857
0.009866  ...
6  0.002183  0.018859  1.008024  0.004509  0.103329  0.006603
0.000783  ...
7  0.003029  0.014324  1.000242  0.000263  0.108115  0.009527
0.007836  ...
8  0.009896  0.016888  1.003995  0.001789  0.102792  0.002519
0.009817  ...
9  0.001082  0.001930  1.007504  0.001772  0.100470  0.004626
0.006073  ...

   D_137  D_138     D_139     D_140     D_141  D_142     D_143
D_144  \
```

```
0    NaN    NaN  0.002427  0.003706  0.003818    NaN  0.000569
0.000610
1    NaN    NaN  0.003954  0.003167  0.005032    NaN  0.009576
0.005492
2    NaN    NaN  0.003269  0.007329  0.000427    NaN  0.003429
0.006986
3    NaN    NaN  0.006117  0.004516  0.003200    NaN  0.008419
0.006527
4    NaN    NaN  0.003671  0.004946  0.008889    NaN  0.001670
0.008126
5    NaN    NaN  0.001924  0.008598  0.004529    NaN  0.000674
0.002223
6    NaN    NaN  0.001336  0.004361  0.009387    NaN  0.007727
0.007661
7    NaN    NaN  0.002397  0.008452  0.005553    NaN  0.001831
0.009616
8    NaN    NaN  0.009742  0.003968  0.007945    NaN  0.008722
0.004369
9    NaN    NaN  0.003611  0.009607  0.007266    NaN  0.008763
0.004753

      D_145  Serial
0  0.002674       1
1  0.009217       2
2  0.002603       3
3  0.009600       4
4  0.009827       5
5  0.002884       6
6  0.002225       7
7  0.007385       8
8  0.000995       9
9  0.009068      10

[10 rows x 191 columns]

Top 10 rows of Test Data with Serial Numbers:
                                         customer_ID         S_2
P_2  \
0  00000469ba478561f23a92a868bd366de6f6527a684c9a...  2019-02-19
0.631315
1  00000469ba478561f23a92a868bd366de6f6527a684c9a...  2019-03-25
0.587042
2  00000469ba478561f23a92a868bd366de6f6527a684c9a...  2019-04-25
0.609056
3  00000469ba478561f23a92a868bd366de6f6527a684c9a...  2019-05-20
0.614911
4  00000469ba478561f23a92a868bd366de6f6527a684c9a...  2019-06-15
0.591673
5  00000469ba478561f23a92a868bd366de6f6527a684c9a...  2019-07-13
0.587472
```

```
6  00000469ba478561f23a92a868bd366de6f6527a684c9a...  2019-08-16
0.625006
7  00000469ba478561f23a92a868bd366de6f6527a684c9a...  2019-09-29
0.597074
8  00000469ba478561f23a92a868bd366de6f6527a684c9a...  2019-10-12
0.568930
9  00001bf2e77ff879fab36aa4fac689b9ba411dae63ae39...  2018-04-22
0.894195

       D_39       B_1       B_2       R_1       S_3      D_41
B_3  ...  \
0  0.001912  0.010728  0.814497  0.007547  0.168651  0.009971
0.002347  ...
1  0.005275  0.011026  0.810848  0.001817  0.241389  0.000166
0.009132  ...
2  0.003326  0.016390  1.004620  0.000114  0.266976  0.004196
0.004192  ...
3  0.009065  0.021672  0.816549  0.009722  0.188947  0.004123
0.015325  ...
4  0.238794  0.015923  0.810456  0.002026  0.180035  0.000731
0.011281  ...
5  0.005827  0.007959  1.001009  0.008814  0.173701  0.000653
0.009779  ...
6  0.238704  0.013420  0.816605  0.009849  0.170644  0.002495
0.019999  ...
7  0.003537  0.017426  1.000670  0.005143  0.158006  0.000985
0.011962  ...
8  0.121385  0.010779  1.009347  0.006923  0.149413  0.000396
0.003576  ...
9  0.325657  0.020970  1.001803  0.005125  0.073243  0.005347
0.001597  ...

   D_137  D_138     D_139     D_140     D_141  D_142     D_143
D_144  \
0    NaN    NaN       NaN  0.004669       NaN    NaN       NaN
0.008281
1    NaN    NaN  0.000142  0.004940  0.009021    NaN  0.003695
0.003753
2    NaN    NaN  0.000074  0.002114  0.004656    NaN  0.003155
0.002156
3    NaN    NaN  0.004743  0.006392  0.002890    NaN  0.006044
0.005206
4    NaN    NaN  0.008133  0.004329  0.008384    NaN  0.001008
0.007421
5    NaN    NaN  0.007001  0.003962  0.005530    NaN  0.009870
0.009667
6    NaN    NaN  0.001132  0.007676  0.005410    NaN  0.006762
0.005664
7    NaN    NaN  0.008730  0.000628  0.002821    NaN  0.004141
```

```
0.005733
8    NaN    NaN  0.005912  0.001250  0.006543    NaN  0.009160
0.003690
9    NaN    NaN  0.008065  0.009861  0.009535    NaN  0.003964
0.008436

      D_145  Serial
0       NaN       1
1  0.001460       2
2  0.006482       3
3  0.007855       4
4  0.009471       5
5  0.005398       6
6  0.002627       7
7  0.005657       8
8  0.003219       9
9  0.008323      10

[10 rows x 191 columns]

Top 10 rows of Train Data with Serial Numbers:
                                         customer_ID  target  Serial
0  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...       0       1
1  00000fd6641609c6ece5454664794f0340ad84dddce9a2...       0       2
2  00001b22f846c82c51f6e3958ccd81970162bae8b007e8...       0       3
3  000041bdba6ecadd89a52d11886e8eaaec9325906c9723...       0       4
4  00007889e4fcd2614b6cbe7f8f3d2e5c728eca32d9eb8a...       0       5
5  000084e5023181993c2e1b665ac88dbb1ce9ef621ec537...       0       6
6  000098081fde4fd64bc4d503a5d6f86a0aedc425c96f52...       0       7
7  0000d17a1447b25a01e42e1ac56b091bb7cbb06317be4c...       0       8
8  0000f99513770170a1aba690daeeb8a96da4a39f11fc27...       1       9
9  00013181a0c5fc8f1ea38cd2b90fe8ad2fa8cad9d9f13e...       1      10

Top 10 rows of Train Data with Serial Numbers:
                                         customer_ID  prediction
Serial
0  00000469ba478561f23a92a868bd366de6f6527a684c9a...           0
1
1  00001bf2e77ff879fab36aa4fac689b9ba411dae63ae39...           0
2
2  0000210045da4f81e5f122c6bde5c2a617d03eef67f82c...           0
3
3  00003b41e58ede33b8daf61ab56d9952f17c9ad1c3976c...           0
4
4  00004b22eaeeeb0ec976890c1d9bfc14fd9427e98c4ee9...           0
5
5  00004ffe6e01e1b688170bbd108da8351bc4c316eacfef...           0
6
6  00007cfcce97abfa0b4fa0647986157281d01d3ab90de9...           0
7
```

```
7  000089cc2a30dad8e6ba39126f9d86df6088c9f975093a...           0
8
8  00008f50a1dd76fa211ba36a2b0d5a1b201e4134a5fd53...           0
9
9  0000b48a4f27dc1d61e78d081678e811620300b88eb3ab...           0
10
```

```python
# Step 2: Exploratory Data Analysis (EDA)
def explore_data(data):
    """Perform basic exploration to understand the data."""
    print("Basic Information of the Dataset:")
    print("-" * 50)
    print(data.info())
    print("\nBasic Statistics of Numerical Columns:")
    print("-" * 50)
    print(data.describe().T)

    print("\nMissing Values in Top Columns:")
    print("-" * 50)
    print(data.isnull().sum().sort_values(ascending=False).head(10))

    # Check if 'target' column exists before analyzing its
distribution
    if 'target' in data.columns:
        print("\nTarget Distribution:")
        print("-" * 50)
        print(data['target'].value_counts(normalize=True))

        # Visualizing target distribution
        sns.countplot(x='target', data=data)
        plt.title("Target Distribution")
        plt.xlabel("Target Class")
        plt.ylabel("Count")
        plt.show()
    else:
        print("\n'Target' column is not present in the dataset.")

# Call the function
explore_data(train_data)
```

```
Basic Information of the Dataset:
--------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 191 entries, customer_ID to Serial
dtypes: float64(185), int64(2), object(4)
memory usage: 14.6+ MB
None

Basic Statistics of Numerical Columns:
```

```
--------------------------------------------------
            count         mean          std           min
25%  \
P_2        9936.0     0.650498     0.252416 -2.569212e-01     0.471264

D_39      10000.0     0.157031     0.275827  8.701630e-07     0.004574

B_1       10000.0     0.126147     0.212428 -1.414690e-01     0.009126

B_2       10000.0     0.617122     0.403145  3.432000e-05     0.091656

R_1       10000.0     0.074035     0.219977  3.031180e-06     0.002863

...           ...          ...          ...           ...          ...

D_142      1572.0     0.370263     0.249771 -8.804185e-03     0.150429

D_143      9847.0     0.162935     0.364742  2.172557e-06     0.002939

D_144      9937.0     0.046364     0.174197  7.773110e-07     0.002674

D_145      9847.0     0.057846     0.226945  1.577788e-07     0.002966

Serial    10000.0  5000.500000  2886.895680  1.000000e+00  2500.750000


                50%          75%           max
P_2        0.690195     0.866060      1.009926
D_39       0.009296     0.238015      4.268383
B_1        0.032948     0.122675      1.323411
B_2        0.814048     1.002262      1.009999
R_1        0.005726     0.008528      2.259283
...             ...          ...           ...
D_142      0.352169     0.559880      1.185992
D_143      0.005944     0.008907      1.010000
D_144      0.005323     0.008105      1.342362
D_145      0.005932     0.008878      4.282032
Serial  5000.500000  7500.250000  10000.000000

[187 rows x 8 columns]

Missing Values in Top Columns:
--------------------------------------------------
D_87     9990
D_88     9980
D_111    9975
D_110    9975
B_39     9972
D_108    9944
D_73     9926
B_42     9831
```

```
D_138    9628
D_136    9628
dtype: int64

'Target' column is not present in the dataset.

# Step 3: Handling Missing Values
def handle_missing_values(train, test):
    """Fill missing values with a placeholder."""
    train.fillna(-999, inplace=True)
    test.fillna(-999, inplace=True)
    return train, test

train, test = handle_missing_values(train_data, test)

# Step 4: Encoding Categorical Variables
from sklearn.preprocessing import LabelEncoder

def encode_categorical(train, test):
    """Label encode categorical features."""
    categorical_cols = train.select_dtypes(include=['object']).columns
    encoder = LabelEncoder()

    # Apply encoding for each categorical column
    for col in categorical_cols:
        # Fit on training data and transform both train and test sets
        train[col] = encoder.fit_transform(train[col].astype(str))

        # Transform test data using the same encoder
        # We use 'fit_transform' only on training data to avoid test
data leakage
        test[col] = encoder.transform(test[col].astype(str))

    return train, test

# Apply the encoding function to train and test datasets
train, test = encode_categorical(train_data, test)

from sklearn.preprocessing import StandardScaler

def scale_features(train, test):
    """Standardize numerical features with aligned columns."""
    scaler = StandardScaler()

    # Drop non-numeric columns first
    train_numeric = train.select_dtypes(include=['int64', 'float64'])
    test_numeric = test.select_dtypes(include=['int64', 'float64'])

    # Align columns between train and test
    train_numeric, test_numeric = train_numeric.align(test_numeric,
join='inner', axis=1)
```

```
    # Fit scaler on train data and transform both train and test
    train_scaled = scaler.fit_transform(train_numeric)
    test_scaled = scaler.transform(test_numeric)

    # Replace the numerical columns in the original train and test
DataFrames
    train[train_numeric.columns] = train_scaled
    test[train_numeric.columns] = test_scaled

    return train, test

# Apply feature scaling
train, test = scale_features(train_data, test)

print("Feature scaling completed successfully!")
```

```
Feature scaling completed successfully!
```

```
# Step 6: Splitting Data
from sklearn.model_selection import train_test_split

def split_train_data(train):
    """Split data into training and validation sets."""
    if not isinstance(train, pd.DataFrame):
        raise ValueError("Input should be a DataFrame.")

    X = train.drop(columns=['target', 'id'], errors='ignore')
    y = train['target'] if 'target' in train.columns else None

    if y is None:
        raise ValueError("Column 'target' not found in the
DataFrame.")

    X_train, X_val, y_train, y_val = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)
    return X_train, X_val, y_train, y_val


# Step 7: Custom Evaluation Metric
def amex_metric(y_true, y_pred):
    """Define the AMEX evaluation metric."""
    def top_four_percent_captured(y_true, y_pred):
        df = pd.DataFrame({'y_true': y_true, 'y_pred': y_pred})
        df = df.sort_values('y_pred', ascending=False)
        df['weight'] = df['y_true'].apply(lambda x: 20 if x == 1 else
1)
        cutoff = int(0.04 * df['weight'].sum())
        return df.iloc[:cutoff]['y_true'].sum() / df['y_true'].sum()
```

```
    gini = 2 * roc_auc_score(y_true, y_pred) - 1
    return 0.5 * (gini + top_four_percent_captured(y_true, y_pred))

# Step 8: Train Baseline Logistic Regression
def split_train_data(train):
    """Split data into training and validation sets."""
    # Check for available columns
    print("Columns in train:", train.columns)

    # Drop 'target' or 'id' only if they exist
    drop_cols = [col for col in ['target', 'id'] if col in
train.columns]
    X = train.drop(columns=drop_cols)  # Drop valid columns only

    y = train['target'] if 'target' in train.columns else None  #
Ensure 'target' exists

    if y is None:
        raise ValueError("Column 'target' not found in the dataset!")

    X_train, X_val, y_train, y_val = train_test_split(
        X, y, test_size=0.2, stratify=y, random_state=42
    )
    return X_train, X_val, y_train, y_val



# Step 9: Train Advanced XGBoost Model
from sklearn.model_selection import train_test_split

def split_train_data(train):
    target_column = 'target'  # Update this to the correct column name
if needed
    if target_column not in train.columns:
        raise ValueError(f"Column '{target_column}' not found in the
dataset!")

    y = train[target_column]
    X = train.drop(columns=[target_column])

    X_train, X_val, y_train, y_val = train_test_split(
        X, y, test_size=0.2, stratify=y, random_state=42
    )
    return X_train, X_val, y_train, y_val



# Step 10: Analyze Feature Importance
# Ensure 'customer_ID' has the same data type in both datasets
train_data["customer_ID"] = train_data["customer_ID"].astype(str)
```

```
train_labels["customer_ID"] = train_labels["customer_ID"].astype(str)

# Check if 'customer_ID' is unique before setting it as the index
if train_data["customer_ID"].is_unique and
train_labels["customer_ID"].is_unique:
    # Set 'customer_ID' as the index for both DataFrames before
concatenation
    train = pd.concat([train_data.set_index("customer_ID"),
                       train_labels.set_index("customer_ID")],
axis=1).reset_index()
else:
    # If 'customer_ID' is not unique, merge based on 'customer_ID'
instead
    train = pd.merge(train_data, train_labels, on="customer_ID",
how="inner")

# Prepare data for training
X = train.drop(columns=["customer_ID", "target"], errors="ignore")  #
Features
y = train["target"]  # Target column

# Ensure that X and y are not empty
if len(X) > 0 and len(y) > 0:
    # Train-validation split with a valid test_size and train_size
    X_train, X_val, y_train, y_val = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Step 5: Train XGBoost Model
    xgb_model = xgb.XGBClassifier(
        objective="binary:logistic",
        eval_metric="auc",
        learning_rate=0.05,
        max_depth=6,
        subsample=0.8,
        colsample_bytree=0.8,
        random_state=42
    )
    xgb_model.fit(X_train, y_train, eval_set=[(X_val, y_val)],
early_stopping_rounds=50, verbose=True)

    # Step 6: Feature Importance Visualization
    def analyze_features(model, X_val):
        """
        Visualize feature importance using SHAP and XGBoost.
        """
        # XGBoost Feature Importance
        xgb.plot_importance(model)
        plt.title("XGBoost Feature Importance")
        plt.show()
```

```
        # SHAP Feature Importance
        explainer = shap.Explainer(model, X_val)
        shap_values = explainer(X_val)
        shap.summary_plot(shap_values, X_val)

    # Step 7: Call the function
    analyze_features(xgb_model, X_val)

else:
    print("Error: The input data is empty. Please check the data.")
```

```
Error: The input data is empty. Please check the data.
```

```
# Step 11: Generate Submission File
def encode_categorical(train, test):
    """Label encode categorical features."""
    categorical_cols = train.select_dtypes(include=['object']).columns
    encoder = LabelEncoder()
    for col in categorical_cols:
        train[col] = encoder.fit_transform(train[col].astype(str))
        test[col] = encoder.transform(test[col].astype(str))
    return train, test

# Step 2: Train XGBoost Model without early stopping
def train_xgb_model(train_data, train_labels):
    """Train XGBoost model on the training data."""
    # Prepare training data
    X = train_data.drop(columns=['target'], errors='ignore')  # Drop
'target' if it exists
    y = train_labels['target']

    # Split into train and validation sets
    X_train, X_val, y_train, y_val = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Initialize the XGBoost classifier
    xgb_model = xgb.XGBClassifier(
        objective="binary:logistic",
        learning_rate=0.05,
        max_depth=6,
        subsample=0.8,
        colsample_bytree=0.8,
        random_state=42
    )

    # Train the model without early stopping
    xgb_model.fit(X_train, y_train, verbose=True)

    return xgb_model
```

```python
# Step 3: Generate Submission File
def generate_submission(test, xgb_model):
    """Create the final submission file."""
    # Check if the 'id' column exists in the test DataFrame
    if 'id' not in test.columns:
        print("Error: 'id' column not found in the test DataFrame.")
        return

    # Prepare test data (drop 'id' column and other non-predictive
columns)
    X_test = test.drop(columns=['id'], errors='ignore')

    # Convert test data to DMatrix format for XGBoost
    dtest = xgb.DMatrix(X_test)

    # Make predictions (ensure binary classification, or adjust as
necessary)
    predictions = xgb_model.predict(dtest)

    # If the model is binary classification, convert probabilities to
0 or 1
    if predictions.shape[0] > 1:  # In case of output being
probabilities
        predictions = (predictions > 0.5).astype(int)  # Threshold at
0.5

    # Create the submission DataFrame
    submission = pd.DataFrame({
        'id': test['id'],
        'prediction': predictions
    })

    # Save the submission file to the specified path
    submission.to_csv(r'E:\AYUSH\amex-default-prediction\
sample_submission.csv', index=False)
    print("Submission file saved as sample_submission.csv")

# Example usage:
# Assuming 'train_data', 'train_labels', and 'test' are already loaded
with the respective data

# Step 4: Encode categorical features (if needed)
train_data, test = encode_categorical(train_data, test)

# Step 5: Train the XGBoost model
xgb_model = train_xgb_model(train_data, train_labels)  # Ensure
'train_data' and 'train_labels' are defined

# Step 6: Generate the submission file
```

```
generate_submission(test, xgb_model)
```

```
Error: 'id' column not found in the test DataFrame.
```