In [ ]:
```python
import pandas as pd

# Read a CSV file into a DataFrame
df = pd.read_csv('Train.csv')  # Read the CSV file named 'Train.csv' into a DataFrame
df = df.astype(str)  # Convert all data to string type

# Fill missing values with 'NA'
df.fillna('NA', inplace=True)  # Fill all NaN values in the DataFrame with the string 'NA'

# List of column names to be modified
cols_to_modify = [
    'Feature_1', 'Feature_2', 'Feature_3', 'Feature_4', 'Feature_5', 'Feature_6',
    'Feature_7', 'Feature_8', 'Feature_9', 'Feature_10', 'Feature_11', 'Feature_12',
    'Feature_13', 'Feature_14', 'Feature_17', 'Feature_19', 'Feature_20', 'Feature_23',
    'Feature_26', 'Feature_27', 'Feature_28', 'Feature_29', 'Feature_30', 'Feature_31',
    'Feature_32', 'Feature_33', 'Feature_35', 'Feature_36', 'Feature_37', 'Feature_38',
    'Feature_39', 'Feature_40', 'Feature_46', 'Feature_47', 'Feature_48', 'Feature_49',
    'Feature_50', 'Feature_53'
]

# Modify the column values based on conditions
for col in cols_to_modify:  # For each column in the list of columns to be modified
    df[col] = df[col].apply(  # Apply a lambda function to each value in the column
        lambda x: 0 if x == 0 else  # If the value is 0, keep it as 0
        2 if x == 'NA' else  # If the value is 'NA', change it to 2
        1  # Otherwise, change the value to 1
    )

# Save the modified data to a new CSV file
df.to_csv('Modified_Train.csv', index=False)  # Save the DataFrame to 'Modified_Train.csv' without row indices

# Read another CSV file into a DataFrame
df1 = pd.read_csv('Future.csv')  # Read the CSV file named 'Future.csv' into a DataFrame
df1 = df1.astype(str)  # Convert all data to string type

# Remove the 'target' column, which is not required in the feature data
df1 = df1.drop(columns=['target'])  # Drop the 'target' column

# Fill missing values with 'NA'
df1.fillna('NA', inplace=True)  # Fill all NaN values with the string 'NA'

# Modify the column values based on conditions
for col in cols_to_modify:  # For each column in the list of columns to be modified
    df1[col] = df1[col].apply(  # Apply a lambda function to each value in the column
        lambda x: 0 if x == 0 else  # If the value is 0, keep it as 0
        2 if x == 'NA' else  # If the value is 'NA', change it to 2
        1  # Otherwise, change the value to 1
    )

# Save the modified data to a new CSV file
df1.to_csv('Modified_Future.csv', index=False)  # Save the DataFrame to 'Modified_Future.csv' without row indices
```

In [ ]:
```python
from sklearn.model_selection import train_test_split  # For splitting the dataset
from sklearn.preprocessing import StandardScaler, OneHotEncoder  # For data preprocessing
from sklearn.compose import ColumnTransformer  # To combine different preprocessors
from sklearn.tree import DecisionTreeClassifier  # Decision tree classifier
from sklearn.metrics import accuracy_score, f1_score  # For calculating evaluation metrics
from imblearn.over_sampling import SMOTE  # For resampling data to address data imbalance
import joblib  # For saving and loading models

# 1. Load the dataset
data = pd.read_csv("Modified_Train.csv")  # Load the modified dataset from a CSV file

# 2. Extract features and target variable
X = data.drop(columns=['target', 'id'])  # Drop 'target' and 'id' columns from the dataset for feature data
y = data['target']  # 'target' column as the target variable

# 3. Identify categorical columns
categorical_columns = [col for col in X.columns if X[col].dtype == 'object']  # Find all categorical columns in the feature data

# 4. Apply one-hot encoding to categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_columns)  # Apply one-hot encoding to categorical columns
    ]
)

# 5. Preprocess the feature data
X = preprocessor.fit_transform(X)  # Apply preprocessing to the feature data

# 6. Use SMOTE to generate synthetic samples for the minority class
smote = SMOTE(random_state=42)  # Create a SMOTE object to address data imbalance
X, y = smote.fit_resample(X, y)  # Use SMOTE to generate new synthetic samples to balance the dataset

# 7. Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  # 80% training set, 20% testing set

# 8. Initialize and train the decision tree classifier
clf = DecisionTreeClassifier(random_state=42)  # Create a decision tree classifier with a fixed random seed
clf.fit(X_train, y_train)  # Train the classifier on the training set

# 9. Save the data processing model
joblib.dump(preprocessor, 'preprocessor_model.pkl')  # Save the preprocessing model to a file

# 10. Save the decision tree classifier
joblib.dump(clf, 'decision_tree_model.pkl')  # Save the decision tree classifier to a file

# 11. Predict on the testing set
y_pred = clf.predict(X_test)  # Use the trained classifier to predict on the testing set

# 12. Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)  # Calculate the accuracy of predictions on the testing set
print("Accuracy:", accuracy)  # Output the accuracy

# 13. Calculate the F1 score
f1 = f1_score(y_test, y_pred)  # Calculate the F1 score
print("F1 Score:", f1)  # Output the F1 score
```

In [ ]:
```python
# 1. Load the training dataset
data = pd.read_csv("Modified_Train.csv")  # Read the processed training dataset

# 2. Separate features and target variable
X_train = data.drop(columns=['target', 'id'])  # Remove 'target' and 'id' columns from the dataset, as features
y_train = data['target']  # Extract 'target' column as the target variable

# 3. Identify categorical columns
categorical_columns = [col for col in X_train.columns if X_train[col].dtype == 'object']  # Identify all categorical columns

# 4. Create a preprocessor
# Apply one-hot encoding to categorical columns, keeping the preprocessing consistent with the trained model
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_columns)  # Apply one-hot encoding to categorical columns
    ]
)

# 5. Apply preprocessing to the training data
X_train = preprocessor.fit_transform(X_train)  # Use the preprocessor to process the training data

# 6. Load the trained decision tree model
clf = joblib.load('decision_tree_model.pkl')  # Load the pre-trained decision tree classifier from a file

# 7. Load the future dataset
future_data = pd.read_csv("Modified_Future.csv")  # Read the future dataset

# 8. Separate features
X_future = future_data.drop(columns=['id'])  # Remove the 'id' column from future data, keeping the feature data

# 9. Apply the same preprocessing to the future data
X_future = preprocessor.transform(X_future)  # Use the previously created preprocessor to process the future data

# 10. Predict with the decision tree model
y_pred_future = clf.predict(X_future)  # Use the trained decision tree classifier to predict the future data

# 11. Save the prediction results with IDs to a new CSV file
result_df = pd.DataFrame({'id': future_data['id'], 'target': y_pred_future})  # Create a dataframe containing the prediction results and 'id'
```

```
result_df.to_csv('predicted_results.csv', index=False)  # Save the results to 'predicted_results.csv' without including an index
print("Predicted results saved")
```

In [ ]: