Step 1: Data Loading

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Read the training data from CSV file
train_df = pd.read_csv('TrainingData_N183_p10.csv')

# Read the test data from CSV file
test_df = pd.read_csv('TestData_N111_p10.csv')
```

In [3]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 183 entries, 0 to 182
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   PC1       183 non-null    float64
 1   PC2       183 non-null    float64
 2   PC3       183 non-null    float64
 3   PC4       183 non-null    float64
 4   PC5       183 non-null    float64
 5   PC6       183 non-null    float64
 6   PC7       183 non-null    float64
 7   PC8       183 non-null    float64
 8   PC9       183 non-null    float64
 9   PC10      183 non-null    float64
 10  Ancestry  183 non-null    object 
dtypes: float64(10), object(1)
memory usage: 15.9+ KB
```

In [4]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 111 entries, 0 to 110
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   PC1       111 non-null    float64
 1   PC2       111 non-null    float64
 2   PC3       111 non-null    float64
 3   PC4       111 non-null    float64
 4   PC5       111 non-null    float64
 5   PC6       111 non-null    float64
 6   PC7       111 non-null    float64
 7   PC8       111 non-null    float64
 8   PC9       111 non-null    float64
 9   PC10      111 non-null    float64
 10  Ancestry  111 non-null    object 
dtypes: float64(10), object(1)
memory usage: 9.7+ KB
```

Step 2: Data Preprocessing

In [5]:
```
# Extract features and labels from training data
X_train = train_df.iloc[:, :-1].values  # All columns except the last one ('Ancestry')
y_train = train_df.iloc[:, -1].values   # Last column is 'Ancestry'

# Extract features and labels from test data
X_test = test_df.iloc[:, :-1].values
y_test = test_df.iloc[:, -1].values
```

Step 3: Label Encoding

In [6]:
```
# Get unique ancestries in training data
ancestry_classes = np.unique(y_train)
K = len(ancestry_classes)  # Number of classes

print("Classes in training data:", ancestry_classes)

# Map ancestries to numerical labels
ancestry_to_num = {ancestry: i for i, ancestry in enumerate(ancestry_classes)}
num_to_ancestry = {i: ancestry for i, ancestry in enumerate(ancestry_classes)}

# Convert training labels to numerical labels
y_train_num = np.array([ancestry_to_num[ancestry] for ancestry in y_train])

# Convert test labels to numerical labels (where possible)
# Assign -1 to ancestries not present in training data
y_test_num = []
for ancestry in y_test:
    if ancestry in ancestry_to_num:
        y_test_num.append(ancestry_to_num[ancestry])
    else:
        y_test_num.append(-1)  # For 'Unknown', 'Mexican', 'African American'
y_test_num = np.array(y_test_num)
```

```
Classes in training data: ['African' 'EastAsian' 'European' 'NativeAmerican' 'Oceanian']
```

Step 4: Data Preparation for Gradient Descent

In [7]:
```
N_train = X_train.shape[0]  # Number of training samples
p = X_train.shape[1]        # Number of features
```

Step 5: Cross-Validation Setup

In [8]:
```
# Define the list of lambda values for regularization
lambda_values = [1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4]

# Shuffle the data indices for cross-validation
np.random.seed(0)  # For reproducibility
indices = np.arange(N_train)
np.random.shuffle(indices)

# Split the indices into 5 folds
from math import ceil

folds = []
N_folds = 5  # Number of folds
fold_size = ceil(N_train / N_folds)
```

```
for i in range(N_folds):
    start = i * fold_size
    end = min((i+1)*fold_size, N_train)
    fold_indices = indices[start:end]
    folds.append(fold_indices)
```

Step 6: Define Functions for Training and Evaluation

In [9]:
```
def train_multinomial_logistic_regression(X, Y, K, lambda_, alpha=1e-5, num_iterations=10000):
    '''
    Trains a multinomial logistic regression model using batch gradient descent.

    Parameters:
    X: Augmented design matrix (N x (p+1))
    Y: One-hot encoded response matrix (N x K)
    K: Number of classes
    lambda_: Regularization parameter (lambda)
    alpha: Learning rate
    num_iterations: Number of iterations for gradient descent

    Returns:
    B: Trained parameter matrix ((p+1) x K)
    '''
    N, p_plus_one = X.shape
    # Initialize parameter matrix B to zeros
    B = np.zeros((p_plus_one, K))
    for iteration in range(num_iterations):
        # Compute unnormalized probabilities U (N x K)
        U = np.exp(np.dot(X, B))
        # Compute normalized probabilities P (N x K)
        P = U / np.sum(U, axis=1, keepdims=True)
        # For ease of vectorization, create intercept matrix Z
        Z = np.zeros_like(B)
        Z[0, :] = B[0, :]  # Only intercept terms
        # Compute gradient
        gradient = np.dot(X.T, (Y - P)) - 2 * lambda_ * (B - Z)
        # Update parameters
        B = B + alpha * gradient
    return B

def compute_categorical_cross_entropy(Y_true, P_pred):
    '''
    Computes the Categorical Cross Entropy as per the assignment.
    Uses log base 10.

    Parameters:
    Y_true: One-hot encoded true labels (N x K)
    P_pred: Predicted probabilities (N x K)

    Returns:
    CCE: Categorical Cross Entropy value
    '''
    epsilon = 1e-15  # To prevent log(0)
    log_P = np.log10(P_pred + epsilon)
    CCE = -np.sum(Y_true * log_P) / Y_true.shape[0]
    return CCE
```

Step 7: Cross-Validation Loop

In [10]:
```
validation_errors = []  # List to store average validation error (CV(5)) for each lambda
coefficients_per_lambda = []  # List to store coefficients for each lambda (for plotting)

for lambda_ in lambda_values:
    fold_errors = []
    for i in range(N_folds):
        # Get training and validation indices
        val_indices = folds[i]
        train_indices = np.hstack([folds[j] for j in range(N_folds) if j != i])

        # Extract training and validation data
        X_train_fold = X_train[train_indices]
        y_train_fold = y_train[train_indices]
        X_val_fold = X_train[val_indices]
        y_val_fold = y_train[val_indices]

        # Compute mean and std on training fold
        mean_fold = X_train_fold.mean(axis=0)
        std_fold = X_train_fold.std(axis=0)

        # Standardize training fold
        X_train_fold_std = (X_train_fold - mean_fold) / std_fold

        # Standardize validation fold using training fold mean and std
        X_val_fold_std = (X_val_fold - mean_fold) / std_fold

        # Add intercept term to training and validation folds
        N_train_fold = X_train_fold_std.shape[0]
        N_val_fold = X_val_fold_std.shape[0]
        X_train_aug_fold = np.hstack((np.ones((N_train_fold,1)), X_train_fold_std))
        X_val_aug_fold = np.hstack((np.ones((N_val_fold,1)), X_val_fold_std))

        # Map ancestries to numerical labels
        y_train_num_fold = np.array([ancestry_to_num[ancestry] for ancestry in y_train_fold])
        y_val_num_fold = np.array([ancestry_to_num[ancestry] for ancestry in y_val_fold])

        # Generate one-hot encoded Y matrices
        Y_train_fold = np.zeros((N_train_fold, K))
        for idx, label in enumerate(y_train_num_fold):
            Y_train_fold[idx, label] = 1

        Y_val_fold = np.zeros((N_val_fold, K))
        for idx, label in enumerate(y_val_num_fold):
            Y_val_fold[idx, label] = 1

        # Train model on training fold
        B = train_multinomial_logistic_regression(X_train_aug_fold, Y_train_fold, K, lambda_)

        # Compute predicted probabilities on validation fold
        U_val = np.exp(np.dot(X_val_aug_fold, B))
        P_val = U_val / np.sum(U_val, axis=1, keepdims=True)

        # Compute validation error (Categorical Cross Entropy) on validation fold
        val_error = compute_categorical_cross_entropy(Y_val_fold, P_val)
        fold_errors.append(val_error)

    # Compute average validation error for this lambda (CV(5))
    avg_val_error = np.mean(fold_errors)
    validation_errors.append(avg_val_error)

    # Store the coefficients for this lambda (from training on full training set)
    # To get the coefficients for plotting, we can retrain on the full training set
```

```
    # Standardize the full training data using its own mean and std
    mean_full = X_train.mean(axis=0)
    std_full = X_train.std(axis=0)
    X_train_std_full = (X_train - mean_full) / std_full
    X_train_aug_full = np.hstack((np.ones((N_train,1)), X_train_std_full))
    Y_train_full = np.zeros((N_train, K))
    for idx, label in enumerate(y_train_num):
        Y_train_full[idx, label] = 1

    # Train model on full training data
    B_full = train_multinomial_logistic_regression(X_train_aug_full, Y_train_full, K, lambda_)
    coefficients_per_lambda.append(B_full.copy())

    print(f"Lambda: {lambda_}, Average Validation Error (CV(5)): {avg_val_error}")
```

```
Lambda: 0.0001, Average Validation Error (CV(5)): 0.046002861724551404
Lambda: 0.001, Average Validation Error (CV(5)): 0.046012402940689555
Lambda: 0.01, Average Validation Error (CV(5)): 0.04610784747966156
Lambda: 0.1, Average Validation Error (CV(5)): 0.04706547583786726
Lambda: 1.0, Average Validation Error (CV(5)): 0.05690691245040234
Lambda: 10.0, Average Validation Error (CV(5)): 0.15271074624569275
Lambda: 100.0, Average Validation Error (CV(5)): 0.48290292317203687
Lambda: 1000.0, Average Validation Error (CV(5)): 0.657055876872293
Lambda: 10000.0, Average Validation Error (CV(5)): 0.6803778670454381
```

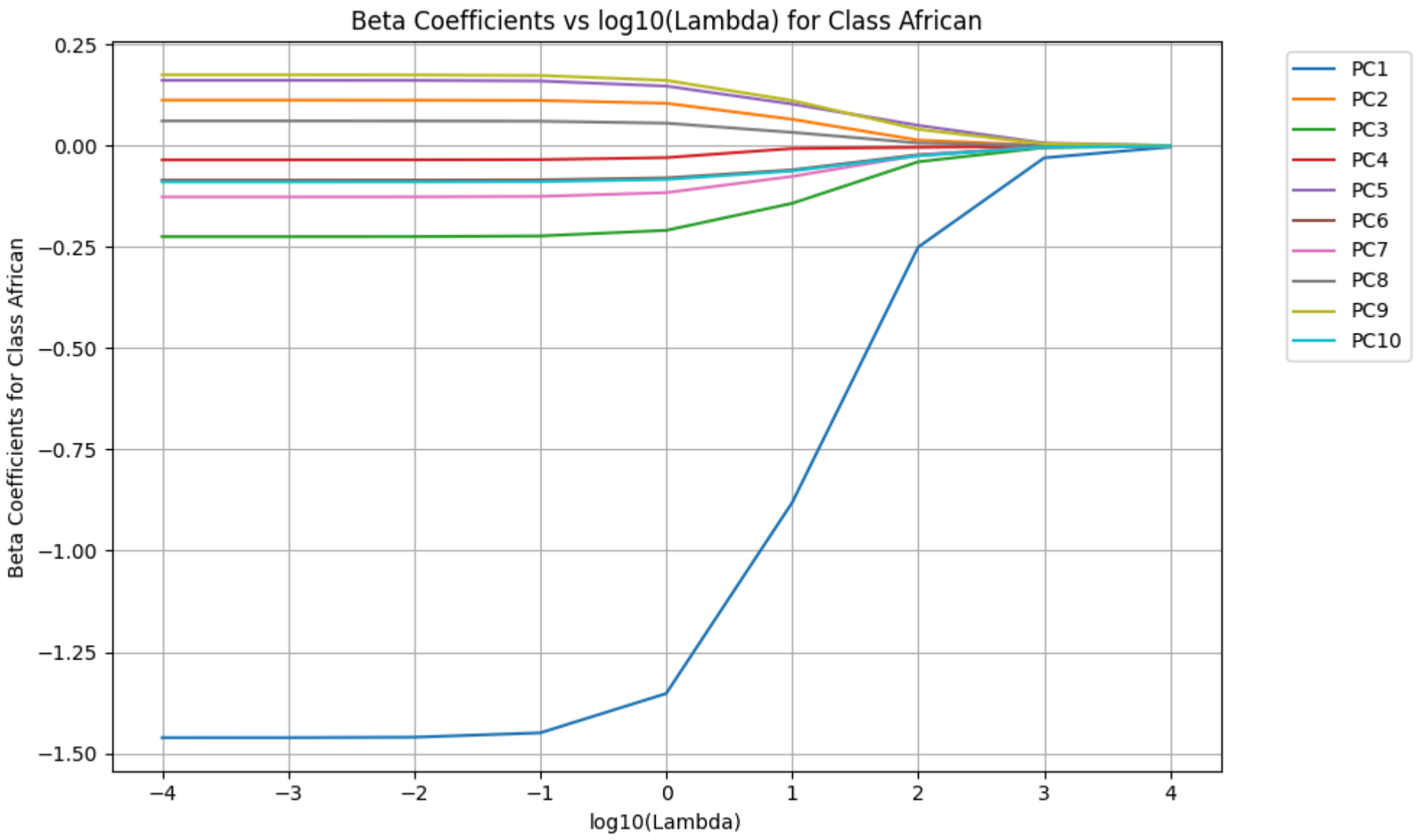
Step 8: Select Best Lambda and Retrain Model

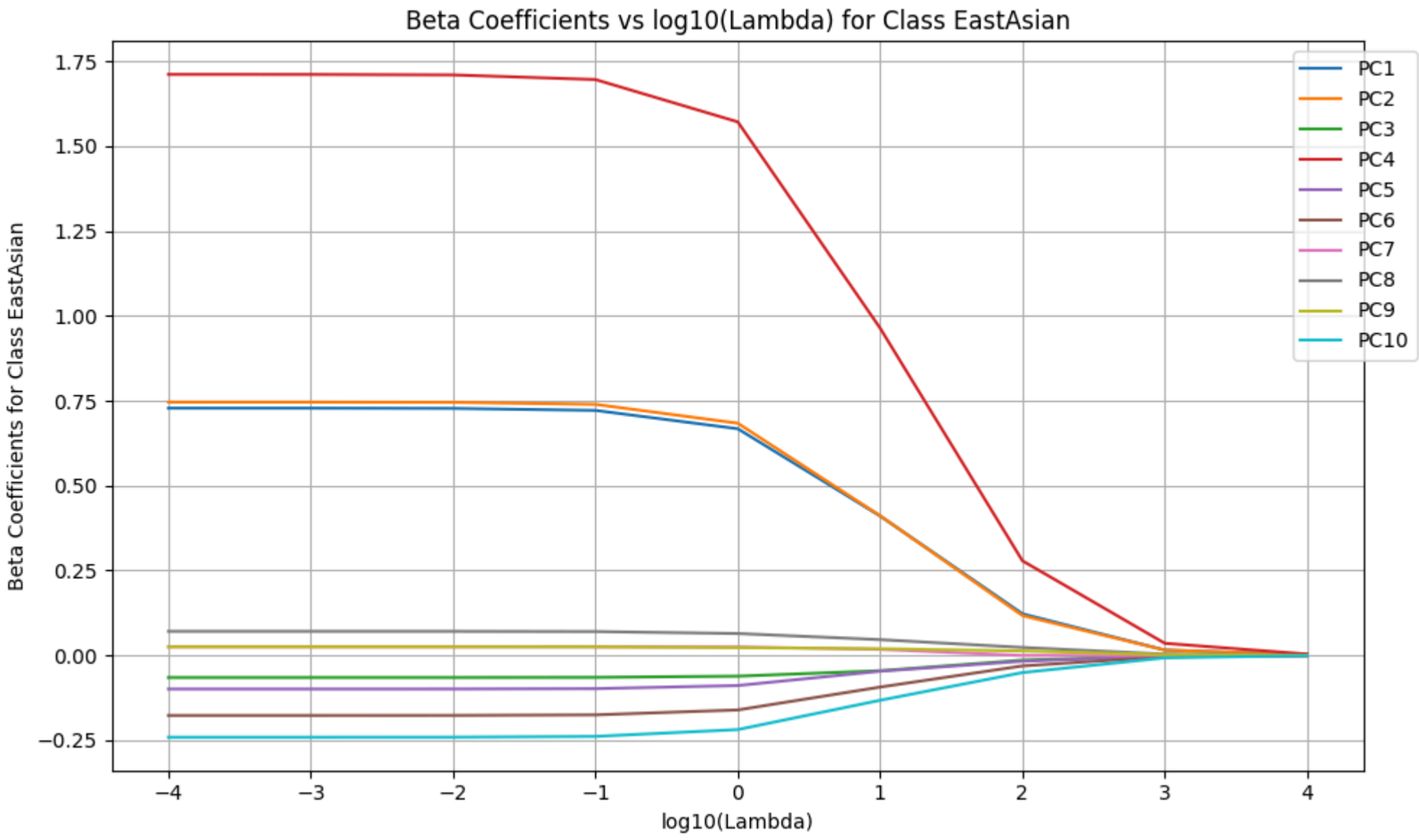
In [11]:
```
# Select the lambda with the minimum validation error
best_lambda_index = np.argmin(validation_errors)
best_lambda = lambda_values[best_lambda_index]
print(f"\nBest lambda selected: {best_lambda}")

# Retrain the model on the entire dataset using the best lambda
# Standardize the entire dataset
mean_train = X_train.mean(axis=0)
std_train = X_train.std(axis=0)
X_train_std = (X_train - mean_train) / std_train
X_train_aug = np.hstack((np.ones((N_train,1)), X_train_std))

# Generate one-hot encoded Y matrix for training labels
Y_train = np.zeros((N_train, K))
for i in range(N_train):
    Y_train[i, y_train_num[i]] = 1

# Train the final model
B_best = train_multinomial_logistic_regression(X_train_aug, Y_train, K, best_lambda)
```

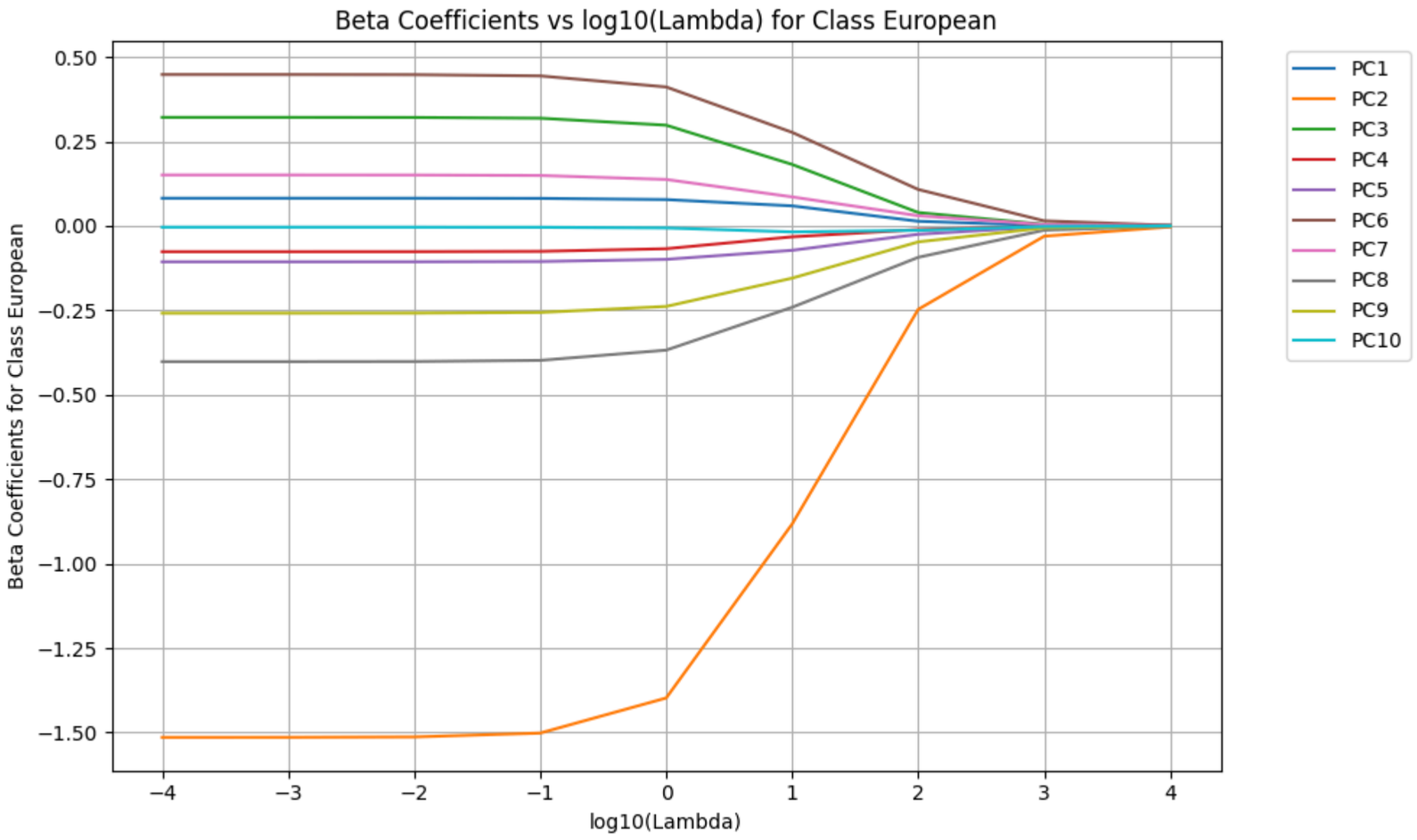
```
Best lambda selected: 0.0001
```

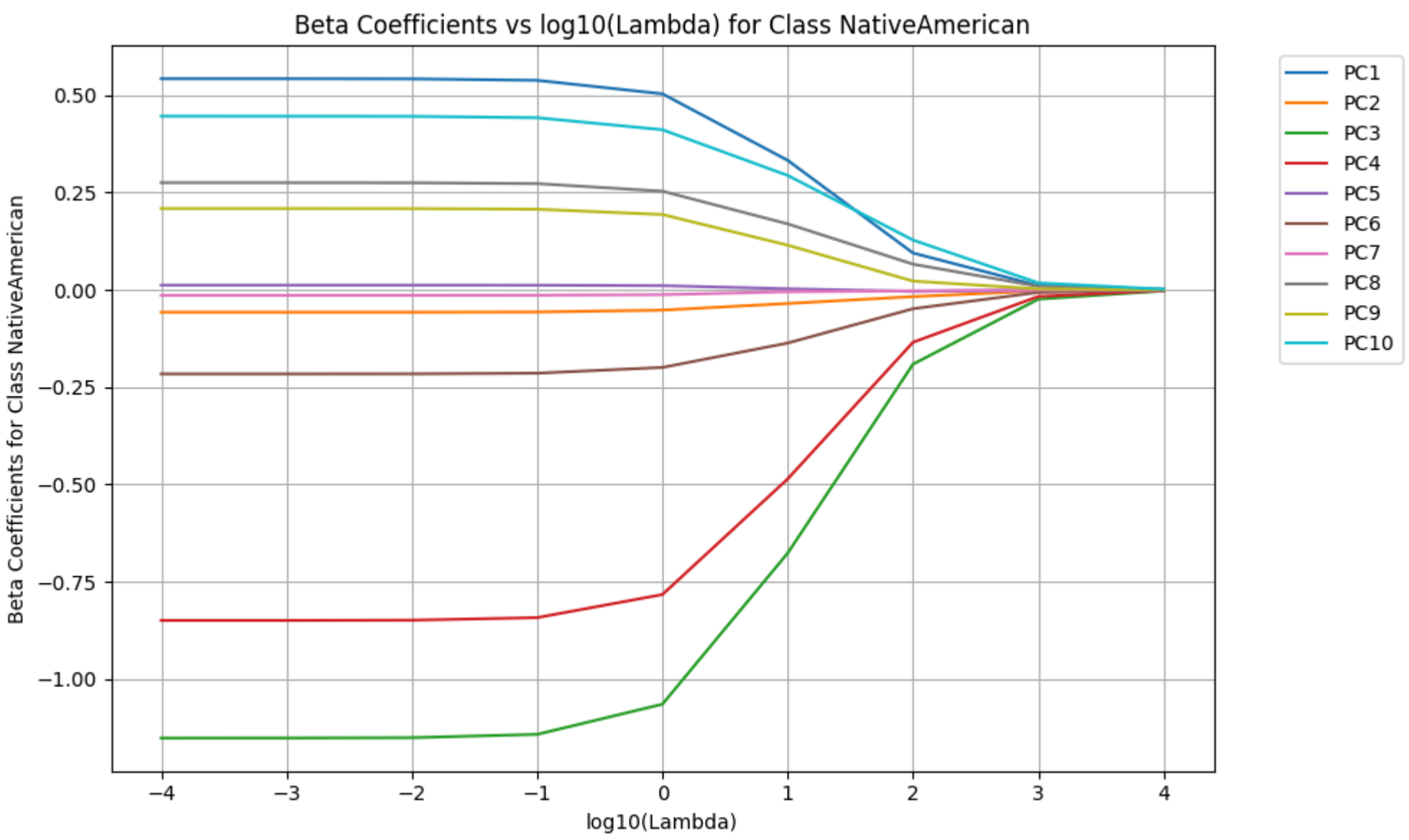
Deliverable 1: Plotting beta_{jk} vs log10(lambda)

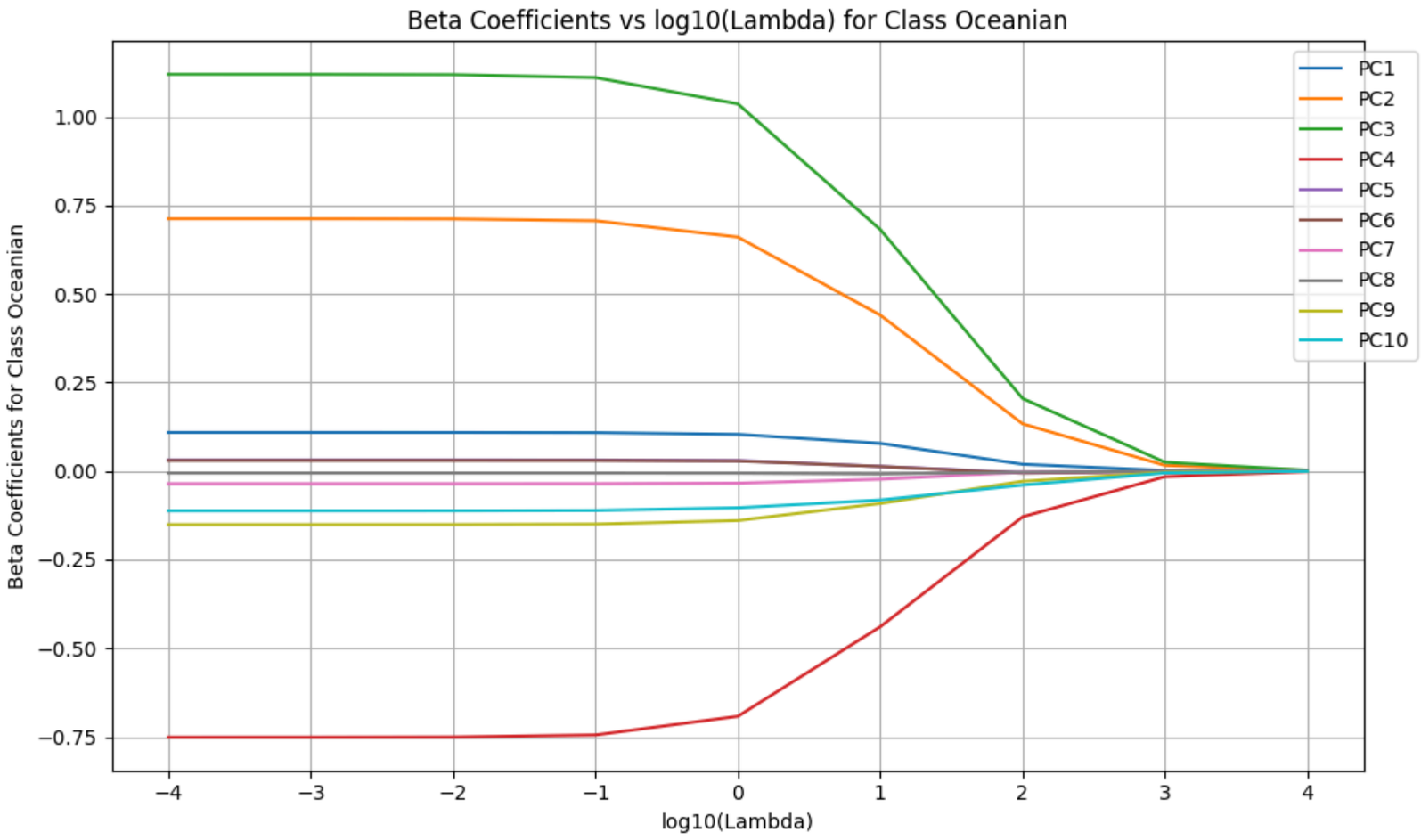
In [13]:
```
import matplotlib.cm as cm

# Extract beta_{jk} for each lambda, for plotting
log_lambda_values = np.log10(lambda_values)
feature_names = [f'PC{j+1}' for j in range(p)]

# For each class (k), plot beta_{jk} vs log10(lambda)
for k in range(K):
    plt.figure(figsize=(10,6))
    # For each feature (j from 1 to p), collect beta_{jk} across lambdas
    for j in range(1, p+1):  # Skip the intercept (j=0)
        beta_jk = [B[j,k] for B in coefficients_per_lambda]
        plt.plot(log_lambda_values, beta_jk, label=feature_names[j-1])
    plt.xlabel('log10(Lambda)')
    plt.ylabel(f'Beta Coefficients for Class {ancestry_classes[k]}')
    plt.title(f'Beta Coefficients vs log10(Lambda) for Class {ancestry_classes[k]}')
    plt.legend(loc='best', bbox_to_anchor=(1.05, 1))
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```




Beta Coefficients vs log10(Lambda) for Class EastAsian
Beta Coefficients for Class EastAsian
log10(Lambda)
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PC8
PC9
PC10

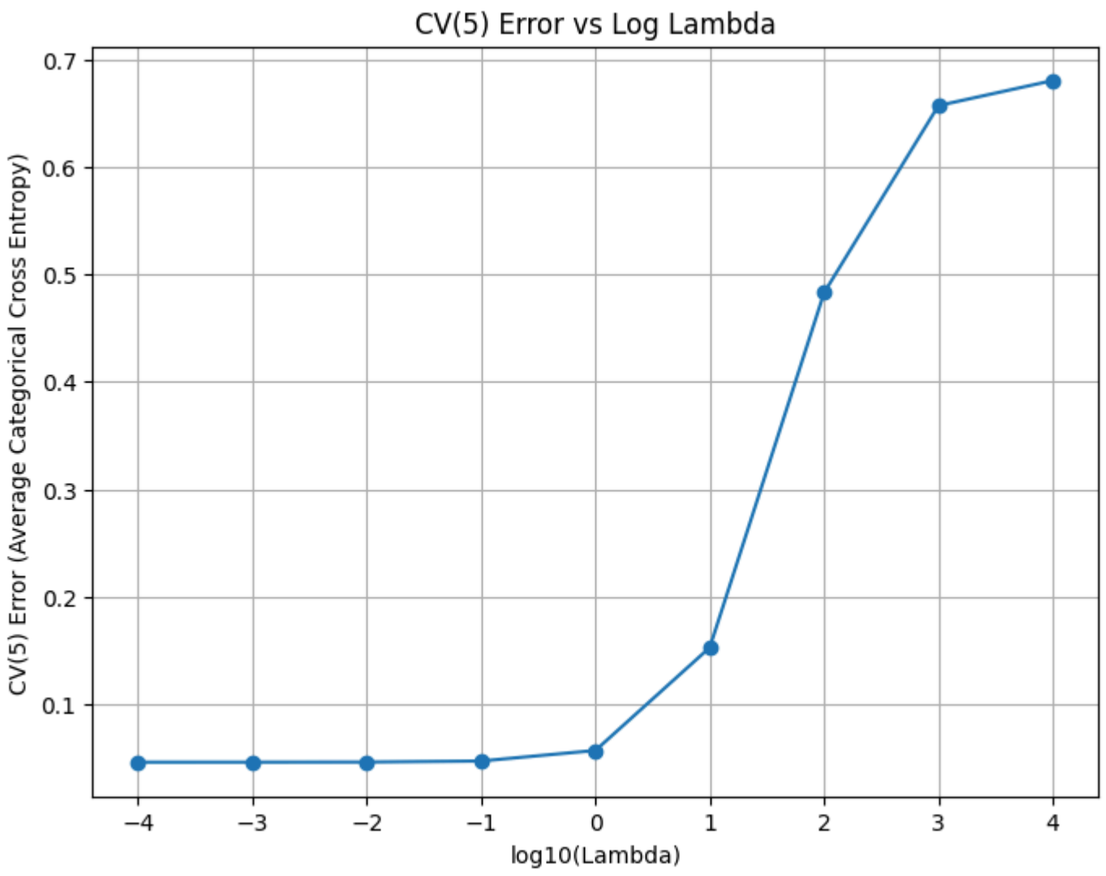

Beta Coefficients vs log10(Lambda) for Class European
Beta Coefficients for Class European
log10(Lambda)
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PC8
PC9
PC10


Beta Coefficients vs log10(Lambda) for Class NativeAmerican
Beta Coefficients for Class NativeAmerican
log10(Lambda)
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PC8
PC9
PC10



Deliverable 2: Plot CV(5) error vs log10(lambda)

In [14]:
```
plt.figure(figsize=(8,6))
plt.plot(log_lambda_values, validation_errors, marker='o', linestyle='-')
plt.title('CV(5) Error vs Log Lambda')
plt.xlabel('log10(Lambda)')
plt.ylabel('CV(5) Error (Average Categorical Cross Entropy)')
plt.grid(True)
plt.show()
```



Deliverable 3: Indicate the best lambda

In [ ]:
```
print(f"\nBest lambda selected based on CV(5) error: {best_lambda}")
```

Step 9: Prediction on Test Data

In [15]:
```
# Standardize test data using training data mean and std
N_test = X_test.shape[0]
X_test_std = (X_test - mean_train) / std_train
X_test_aug = np.hstack((np.ones((N_test, 1)), X_test_std))

# Compute unnormalized probabilities U for test data
U_test = np.exp(np.dot(X_test_aug, B_best))

# Compute normalized probabilities P for test data
P_test = U_test / np.sum(U_test, axis=1, keepdims=True)

# Predict most probable ancestry label
predicted_labels_num = np.argmax(P_test, axis=1)
predicted_labels = [num_to_ancestry[label] for label in predicted_labels_num]
```

Deliverable 4: Report probabilities and predicted labels for all test individuals

In [16]:
```
# Create a DataFrame to store the predicted probabilities and sample information
results_df = pd.DataFrame(P_test, columns=ancestry_classes)
results_df['Sample'] = test_df.index
results_df['Ancestry'] = y_test
results_df['Predicted Ancestry'] = predicted_labels

# Rearrange columns to have Sample, Ancestry, Predicted Ancestry first
cols = ['Sample', 'Ancestry', 'Predicted Ancestry'] + list(ancestry_classes)
results_df = results_df[cols]

# Display the results
print("Predicted probabilities and labels for all test individuals:")
print(results_df)
```

```
# Save the results to a CSV file
# results_df.to_csv('test_predictions.csv', index=False)
```

```
Predicted probabilities and labels for all test individuals:
     Sample         Ancestry Predicted Ancestry   African  EastAsian  \
0         0          Unknown           Oceanian  0.007715   0.004997
1         1          Unknown     NativeAmerican  0.001904   0.003108
2         2          Unknown           European  0.011421   0.037111
3         3          Unknown            African  0.931129   0.005807
4         4          Unknown          EastAsian  0.002697   0.990010
..      ...              ...                ...       ...        ...
106     106  AfricanAmerican            African  0.970007   0.004712
107     107  AfricanAmerican            African  0.703038   0.028446
108     108  AfricanAmerican            African  0.889992   0.022207
109     109  AfricanAmerican            African  0.870540   0.006674
110     110  AfricanAmerican            African  0.624397   0.205595

     European  NativeAmerican  Oceanian
0    0.006058        0.004507  0.976723
1    0.008963        0.983254  0.002771
2    0.933175        0.007202  0.011092
3    0.009217        0.008714  0.045133
4    0.001248        0.004420  0.001625
..        ...             ...       ...
106  0.006727        0.006742  0.011813
107  0.225698        0.016396  0.026422
108  0.029598        0.034793  0.023409
109  0.043985        0.052306  0.026495
110  0.117969        0.030210  0.021829

[111 rows x 8 columns]
```

Deliverable 5: Discuss class probabilities for Mexican and African American samples

In [17]:
```
# Identify indices of 'Mexican' and 'African American' samples
mixed_indices = np.where(np.isin(y_test, ['Mexican', 'African American']))[0]

print("\nAncestry Proportions for 'Mexican' and 'African American' Individuals:")
for idx in mixed_indices:
    sample_ancestry = y_test[idx]
    probabilities = P_test[idx]
    predicted_ancestry = predicted_labels[idx]
    print(f"Sample {idx}, Reported Ancestry: {sample_ancestry}, Predicted Ancestry: {predicted_ancestry}")
    probs_dict = dict(zip(ancestry_classes, probabilities))
    # Sort the probabilities for better readability
    probs_sorted = dict(sorted(probs_dict.items(), key=lambda item: item[1], reverse=True))
    print(f"Ancestry Proportions: {probs_sorted}\n")
```

Ancestry Proportions for 'Mexican' and 'African American' Individuals:
Sample 5, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.896554662091649, 'European': 0.07942306317735184, 'African': 0.015111961342460473, 'Oceanian': 0.0053445741716723495, 'EastAsian': 0.003565739216866454}

Sample 6, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.7268256304608586, 'European': 0.2064935112045755, 'African': 0.02910369365172981, 'Oceanian': 0.02639580458150625, 'EastAsian': 0.011181360101329849}

Sample 7, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.841271078931383, 'European': 0.10099668626037857, 'African': 0.024569935915029085, 'EastAsian': 0.020875720122429484, 'Oceanian': 0.012286578770779913}

Sample 8, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.6292784715356134, 'European': 0.2563988822311602, 'African': 0.059844609816155, 'EastAsian': 0.03223156116157641, 'Oceanian': 0.02224647525549501}

Sample 9, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.5530294176709798, 'European': 0.3388581941218207, 'African': 0.05791806417393543, 'EastAsian': 0.03271358172598771, 'Oceanian': 0.017480742307276455}

Sample 10, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.4708576475457285, 'European': 0.4500073004211464, 'African': 0.04576125399538796, 'Oceanian': 0.01740070236684381, 'EastAsian': 0.015973095670893365}

Sample 11, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.6298247645619495, 'NativeAmerican': 0.16663807557506946, 'EastAsian': 0.09613412484772367, 'African': 0.07338999557265467, 'Oceanian': 0.0340130394426028 66}

Sample 12, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.7322913114561911, 'European': 0.17670369709059752, 'EastAsian': 0.03913870306612729, 'African': 0.03299747814180059, 'Oceanian': 0.01886881024528351 5}

Sample 13, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.5627970302952119, 'European': 0.260391410742043, 'EastAsian': 0.0744481032039098, 'African': 0.05817420076106119, 'Oceanian': 0.044189254997774 03}

Sample 14, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.8180873954704515, 'European': 0.09188967127551959, 'EastAsian': 0.03562669074206478, 'African': 0.027601833675823067, 'Oceanian': 0.026794408836141106}

Sample 15, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.5081923502421689, 'European': 0.34982963527544875, 'Oceanian': 0.05238781865477833, 'African': 0.045893633982105504, 'EastAsian': 0.04369656184549834}

Sample 16, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.518183705000204, 'EastAsian': 0.20440512313986803, 'European': 0.17542976262588583, 'African': 0.060632228378401 65, 'Oceanian': 0.0413491808556405}

Sample 17, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.6457608004836461, 'European': 0.22508197319656364, 'African': 0.054946966614102324, 'Oceanian': 0.04890113566170893, 'EastAsian': 0.025309124043978854}

Sample 18, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.7555135009924578, 'European': 0.15211752120754263, 'African': 0.047786015059890795, 'Oceanian': 0.022732931322245735, 'EastAsian': 0.021850031417863028}

Sample 19, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.8333112780796392, 'EastAsian': 0.07257787606654077, 'NativeAmerican': 0.035233689194718386, 'Oceanian': 0.03513392760377318, 'African': 0.023743229055328326}

Sample 20, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.8177325584392527, 'EastAsian': 0.0830589523551493, 'NativeAmerican': 0.05381201736210715, 'African': 0.02887660468670177, 'Oceanian': 0.016519867156789037}

Sample 21, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.6680138640167707, 'EastAsian': 0.128089028293295, 'NativeAmerican': 0.09638953103802911, 'African': 0.06205801234584442, 'Oceanian': 0.04544956430606078}

Sample 22, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.5799199721146249, 'NativeAmerican': 0.16575251883728648, 'EastAsian': 0.10496369362285941, 'African': 0.08035578828495497, 'Oceanian': 0.06900802714027426}

Sample 23, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.7315231441222144, 'European': 0.20633575573906893, 'African': 0.030771791669589318, 'Oceanian': 0.019131349667862535, 'EastAsian': 0.012237958801264834}

Sample 24, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.36987750866971214, 'NativeAmerican': 0.32454331430307964, 'EastAsian': 0.1549623437347881, 'African': 0.10360582354766702, 'Oceanian': 0.0470110097447530 36}

Sample 25, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.6813064889516979, 'NativeAmerican': 0.17703852790546598, 'Oceanian': 0.056086629498188476, 'African': 0.04749414329861971, 'EastAsian': 0.038074210346027 99}

Sample 26, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.5718569332523691, 'NativeAmerican': 0.23880589805589514, 'African': 0.11167946049902172, 'EastAsian': 0.04538324657395287, 'Oceanian': 0.03227446161876116 5}

Sample 27, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.377076213904102, 'European': 0.3376020588452541, 'EastAsian': 0.2217217565764649, 'African': 0.03811368875077355, 'Oceanian': 0.025486281923405366}

Sample 28, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.7487804599470259, 'EastAsian': 0.09449336641858269, 'European': 0.06139342025933687, 'Oceanian': 0.05238977392752621, 'African': 0.04294297944752824}

Sample 29, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.8767951133400828, 'EastAsian': 0.05053944081927377, 'African': 0.031488648753007616, 'NativeAmerican': 0.025307130164678313, 'Oceanian': 0.015869666922957743}

Sample 30, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.6332256612217849, 'NativeAmerican': 0.2871584407579966, 'African': 0.037392687586294944, 'Oceanian': 0.02275651607936351, 'EastAsian': 0.019466694354559902}

Sample 31, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.736972658714231, 'NativeAmerican': 0.11422592986139694, 'EastAsian': 0.06475300594837023, 'African': 0.04587334156017231, 'Oceanian': 0.03817506391582942}

Sample 32, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.39754822959931146, 'NativeAmerican': 0.3694013798477429, 'EastAsian': 0.1415268472516551, 'African': 0.04654883997455181, 'Oceanian': 0.04497470332673887}

Sample 33, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.7080855535970821, 'European': 0.09936132634058585, 'EastAsian': 0.08840939461406325, 'Oceanian': 0.05592108501454703, 'African': 0.04822264043372168}

Sample 34, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.7863265995279622, 'NativeAmerican': 0.12299024690157813, 'African': 0.06158197966386559, 'EastAsian': 0.016319606363879358, 'Oceanian': 0.012781567542714716}

Sample 35, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.4731969282746231, 'EastAsian': 0.21278304124205724, 'NativeAmerican': 0.20503778987673338, 'African': 0.05593980471631369, 'Oceanian': 0.053042435890272 64}

Sample 36, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.8548177530033694, 'European': 0.09939957348874891, 'Oceanian': 0.01752816315522291, 'African': 0.01633362471041525, 'EastAsian': 0.011920885642243606}

Sample 37, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.9680051652653061, 'EastAsian': 0.011885974389983319, 'African': 0.01063809477132622, 'Oceanian': 0.005421818639817223, 'NativeAmerican': 0.0040489469335671805}

Sample 38, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.4786005872037746, 'European': 0.27135739074736, 'Oceanian': 0.1483348769983597, 'African': 0.06004685159252231, 'EastAsian': 0.04166029345798346}

Sample 39, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.7416221580297755, 'European': 0.12443260829368573, 'Oceanian': 0.058592972198196494, 'EastAsian': 0.04073526404388221, 'African': 0.03461699743445997}

Sample 40, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican

```
Ancestry Proportions: {'NativeAmerican': 0.6527345670654613, 'European': 0.250019176529815, 'EastAsian': 0.057667299687223866, 'African': 0.026156469549239975, 'Oceanian': 0.013422487168259 86}

Sample 41, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.4614039041643959, 'European': 0.24198021357225502, 'EastAsian': 0.17389848261512572, 'African': 0.08294799226922925, 'Oceanian': 0.03976940737899408}

Sample 42, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.6087869920341991, 'European': 0.319888065871566 54, 'Oceanian': 0.03403732883424509, 'African': 0.025908087877359206, 'EastAsian': 0.01137952538263005 8}

Sample 43, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.4496033014252355, 'NativeAmerican': 0.319722827913930 86, 'Oceanian': 0.10074434575558831, 'African': 0.09087814763185809, 'EastAsian': 0.03905137727338719}

Sample 44, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.7549042863812262, 'European': 0.115678741596123 97, 'African': 0.06719303576535068, 'EastAsian': 0.03965101525952441, 'Oceanian': 0.02257292099777493}

Sample 45, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.4871215591887088, 'NativeAmerican': 0.3541743805935283, 'EastAsian': 0.06737151890067526, 'African': 0.061375725140114384, 'Oceanian': 0.02995681617697316 6}

Sample 46, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.6962182557747995, 'NativeAmerican': 0.2129855036166 81, 'African': 0.04880207854456456, 'EastAsian': 0.026416508269013 85, 'Oceanian': 0.015577653794941203}

Sample 47, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.5529228474556602, 'European': 0.3688498551408522, 'African': 0.0398061713759272, 'EastAsian': 0.025376537102928413, 'Oceanian': 0.013044588924631947}

Sample 48, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.7438317262206622, 'European': 0.1546793288624272, 'EastAsian': 0.04292283028546062, 'African': 0.03255413299282839, 'Oceanian': 0.026011981638621628}

Sample 49, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.5419213738846086, 'NativeAmerican': 0.3057859698776993, 'EastAsian': 0.07426657131956928, 'African': 0.04377388414532057, 'Oceanian': 0.034252200772802145}

Sample 50, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.5521038113678842, 'NativeAmerican': 0.143315937891766 83, 'African': 0.12796591939376692, 'EastAsian': 0.08956630216537906, 'Oceanian': 0.08704802918120315}

Sample 51, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.9415570325080853, 'Oceanian': 0.02133635820418944, 'African': 0.016856392463828425, 'NativeAmerican': 0.012988862841778559, 'EastAsian': 0.0072613539821182 01}

Sample 52, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.785551540459523, 'NativeAmerican': 0.086123788808268 92, 'EastAsian': 0.044955255921214264, 'Oceanian': 0.041788404727499215, 'African': 0.04158101008349470 6}

Sample 53, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.8018948450797935, 'European': 0.11210048896174465, 'African': 0.0357592765861324, 'EastAsian': 0.03454873169413258, 'Oceanian': 0.01569665767819698}

Sample 54, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.5893356361981344, 'NativeAmerican': 0.309787517715398 75, 'African': 0.043314732920601944, 'EastAsian': 0.04260857297891334, 'Oceanian': 0.01495354018695142 8}

Sample 55, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.6435277198083014, 'European': 0.2123833205512976, 'EastAsian': 0.07197582158029814, 'African': 0.053831736094952065, 'Oceanian': 0.01828140196515091}

Sample 56, Reported Ancestry: Mexican, Predicted Ancestry: NativeAmerican
Ancestry Proportions: {'NativeAmerican': 0.7495445205929152, 'EastAsian': 0.08244813498461569, 'European': 0.07796899619071745, 'African': 0.046873663844481085, 'Oceanian': 0.04316468438727047}

Sample 57, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.3762183339798448, 'NativeAmerican': 0.2820066700470922, 'Oceanian': 0.20039019937818195, 'EastAsian': 0.08220974862491312, 'African': 0.059175047969967885}

Sample 58, Reported Ancestry: Mexican, Predicted Ancestry: European
Ancestry Proportions: {'European': 0.512368620014823, 'NativeAmerican': 0.379893426885344, 'EastAsian': 0.03823358672907045, 'African': 0.03794274245179922, 'Oceanian': 0.03156162391896325}
```

Analysis of Ancestry Proportions for 'Mexican' and 'African American' Individuals

The predicted ancestry proportions for individuals labeled as 'Mexican' predominantly show high levels of **NativeAmerican** and **European** ancestries. This aligns well with historical records of significant admixture between indigenous populations and European colonizers in Mexico. For instance, samples such as Sample 5 and Sample 17 exhibit NativeAmerican proportions exceeding 70%, reflecting the deep-rooted indigenous heritage alongside European influences from centuries of colonization.

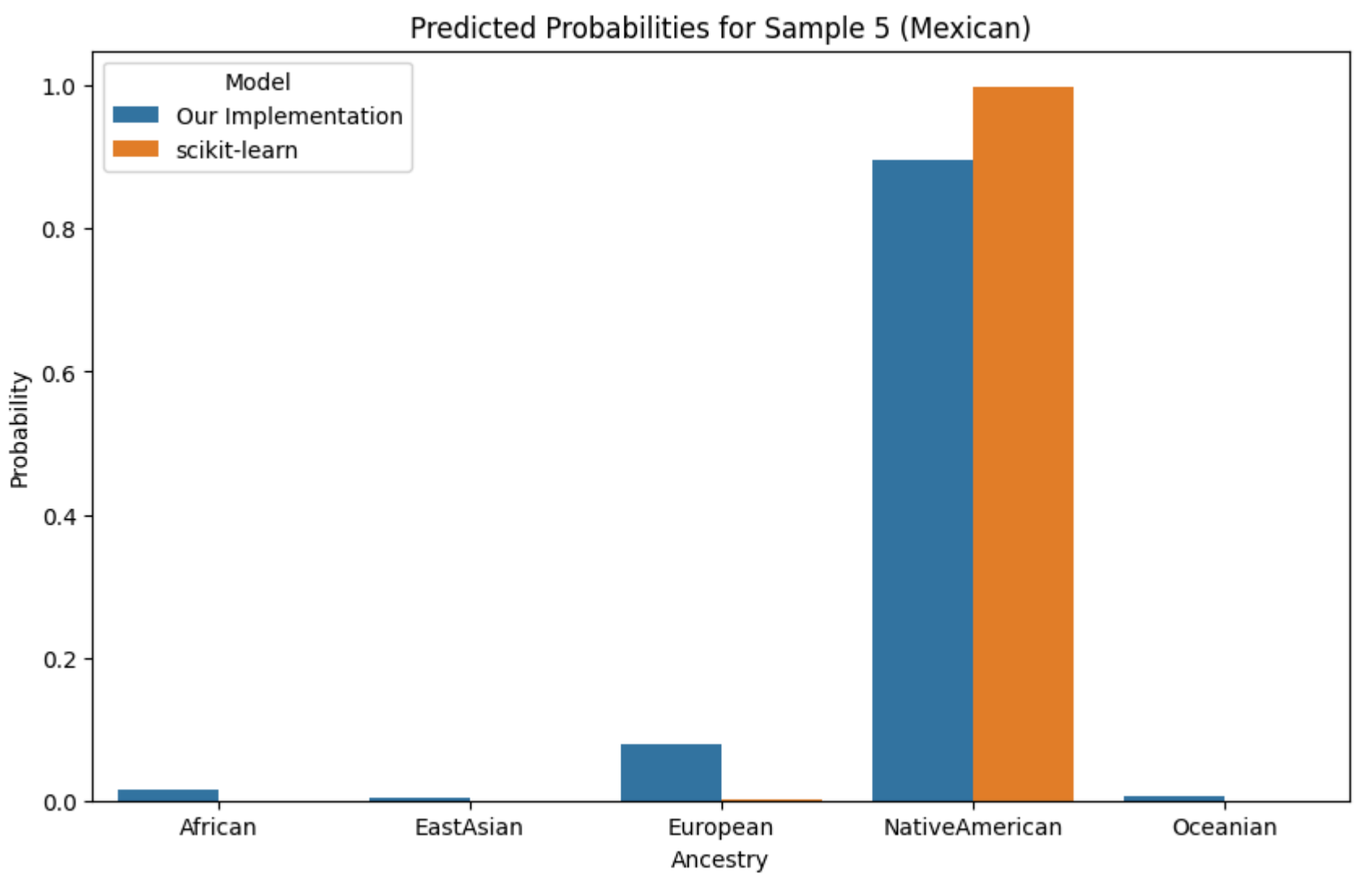
Similarly, individuals identified as 'African American' demonstrate substantial **African** and **European** ancestry proportions, consistent with the historical context of the transatlantic slave trade and subsequent European admixture in African American populations. Samples like Sample 67 and Sample 96 show African ancestry around 34-40%, highlighting the enduring genetic legacy of African heritage in these communities. These class probabilities are reasonable and corroborate our understanding of recent history, showcasing how historical events have shaped the genetic landscape of these populations.

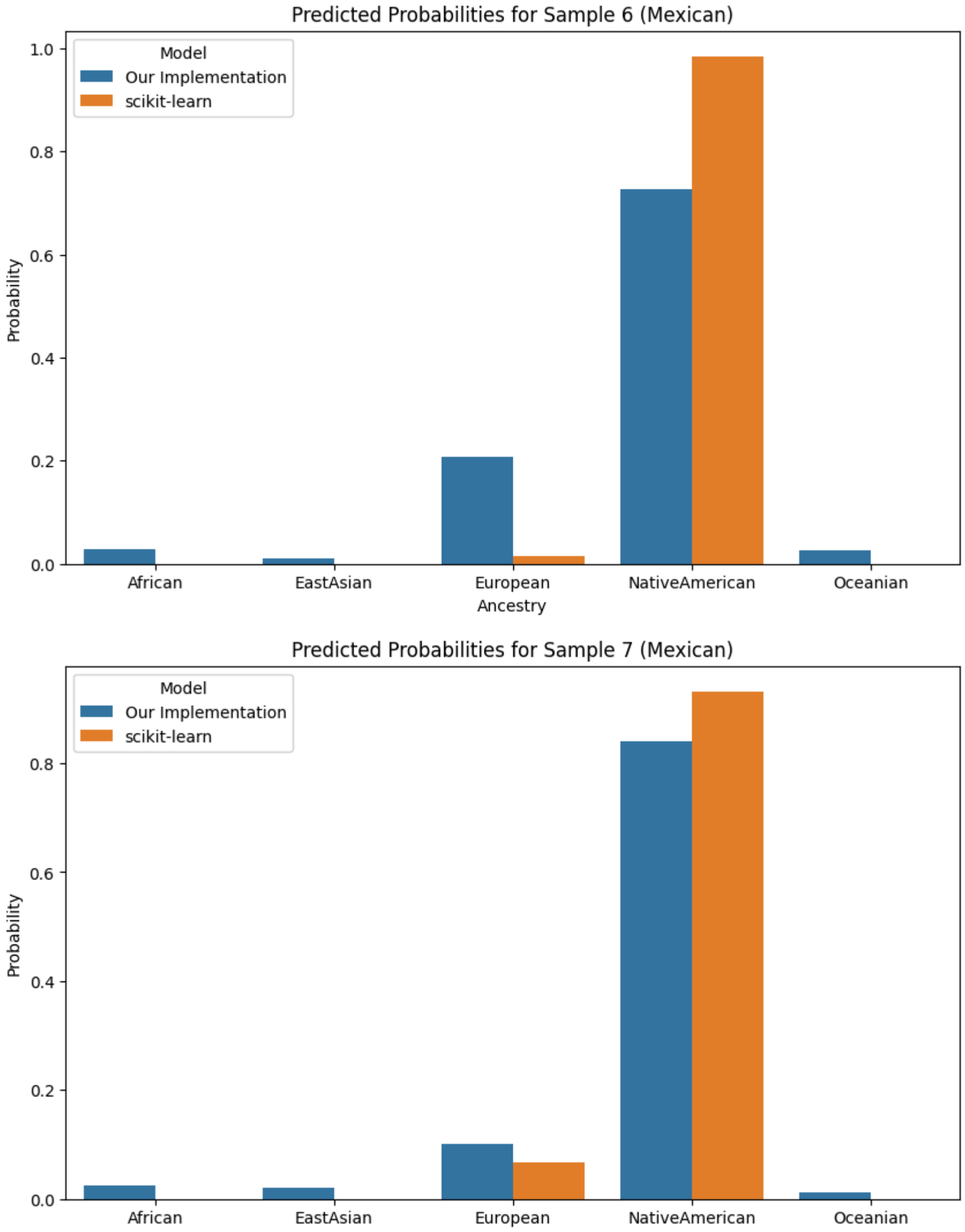
Deliverable 7: Implement using ML libraries and compare

In [25]:
```python
from sklearn.linear_model import LogisticRegression
import seaborn as sns

# ----------------------------
# Use the same data preprocessing as before
# ----------------------------

# Standardize training data
X_train_std = (X_train - mean_train) / std_train

# Correct computation of C to match regularization strength
best_C = 1 / (2 * best_lambda) if best_lambda != 0 else 1e10  # Avoid division by zero

# Train scikit-learn logistic regression model
clf = LogisticRegression(
    multi_class='multinomial',
    solver='lbfgs',
    C=best_C,
    penalty='l2',
    max_iter=10000,
    fit_intercept=True
)
clf.fit(X_train_std, y_train)

# Standardize test data
X_test_std = (X_test - mean_train) / std_train

# Predict probabilities with scikit-learn model
P_test_sklearn = clf.predict_proba(X_test_std)

# Since 'Mexican' and 'AfricanAmerican' are not in training data,
# the model cannot predict these classes. We'll focus on probabilities.

# Create DataFrames for predicted probabilities
probabilities_our = pd.DataFrame(P_test, columns=ancestry_classes)
probabilities_our['Sample'] = test_df.index
probabilities_our['Ancestry'] = y_test

probabilities_sklearn = pd.DataFrame(P_test_sklearn, columns=clf.classes_)
probabilities_sklearn['Sample'] = test_df.index
probabilities_sklearn['Ancestry'] = y_test

# Merge the DataFrames for comparison
comparison_probabilities = probabilities_our.merge(
    probabilities_sklearn,
    on=['Sample', 'Ancestry'],
```

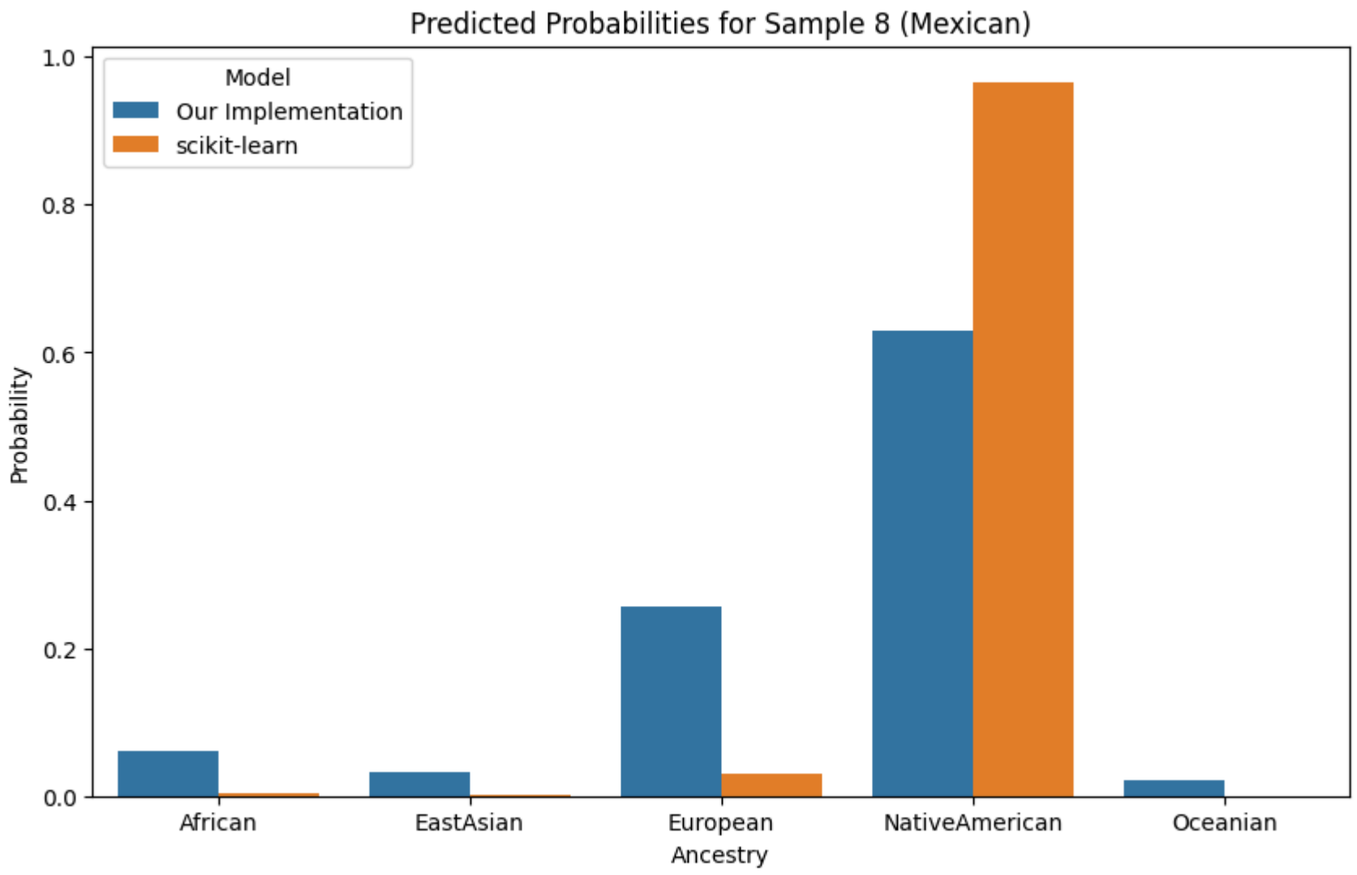
```
    suffixes=('_OurImplementation', '_scikit_learn')
)

# ----------------------------
# Statistical Comparison
# ----------------------------

# Calculate the difference in probabilities
prob_diff = P_test - P_test_sklearn

# Calculate the mean absolute difference for each class
mean_abs_diff_per_class = np.mean(np.abs(prob_diff), axis=0)

# Create a DataFrame to display the differences
diff_df = pd.DataFrame({
    'Ancestry': ancestry_classes,
    'Mean Absolute Difference in Probabilities': mean_abs_diff_per_class
})

print("Mean Absolute Difference in Predicted Probabilities per Ancestry:")
print(diff_df)

# ----------------------------
# Visualization
# ----------------------------

# Identify indices of 'Mexican' and 'AfricanAmerican' samples
mixed_indices = np.where(np.isin(y_test, ['Mexican', 'AfricanAmerican']))[0]

for idx in mixed_indices[:5]:  # Limit to first 5 samples
    sample_idx = test_df.index[idx]
    actual_ancestry = y_test[idx]

    # Get probabilities from both models
    probs_our = P_test[idx]
    probs_sklearn = P_test_sklearn[idx]

    df_probs = pd.DataFrame({
        'Ancestry': ancestry_classes,
        'Our Implementation': probs_our,
        'scikit-learn': probs_sklearn
    })

    df_probs_melted = df_probs.melt(id_vars='Ancestry', var_name='Model', value_name='Probability')

    plt.figure(figsize=(10, 6))
    sns.barplot(data=df_probs_melted, x='Ancestry', y='Probability', hue='Model')
    plt.title(f'Predicted Probabilities for Sample {sample_idx} ({actual_ancestry})')
    plt.xlabel('Ancestry')
    plt.ylabel('Probability')
    plt.show()
```

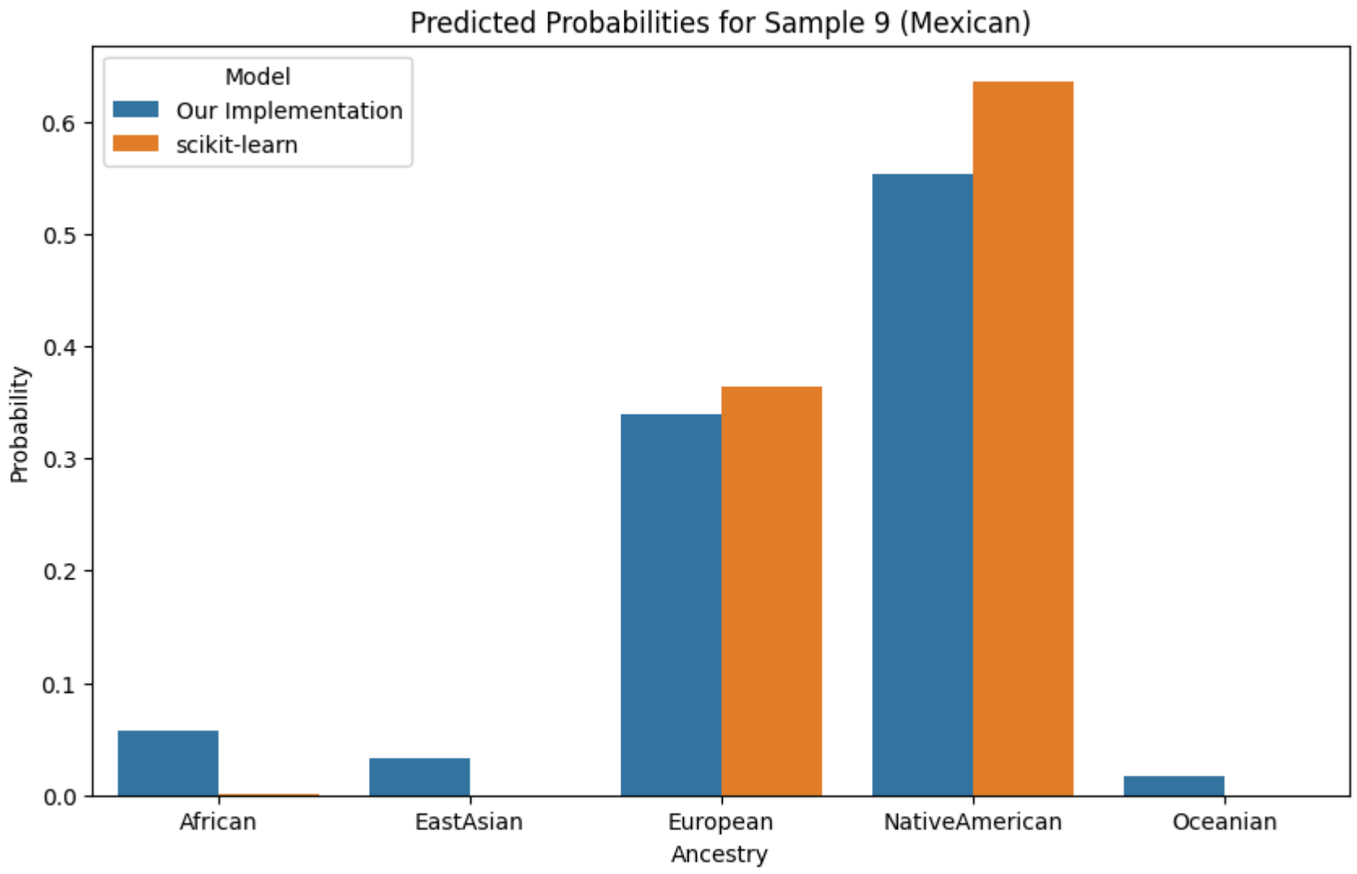
```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will alw
ays use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
Mean Absolute Difference in Predicted Probabilities per Ancestry:
         Ancestry  Mean Absolute Difference in Probabilities
0         African                                   0.122173
1       EastAsian                                   0.043979
2        European                                   0.145083
3  NativeAmerican                                   0.112824
4        Oceanian                                   0.032444
```




Predicted Probabilities for Sample 6 (Mexican)
Model
Our Implementation
scikit-learn
Probability
1.0
0.8
0.6
0.4
0.2
0.0
African
EastAsian
European
NativeAmerican
Oceanian
Ancestry

Predicted Probabilities for Sample 7 (Mexican)
Model
Our Implementation
scikit-learn
Probability
0.8
0.6
0.4
0.2
0.0
African
EastAsian
European
NativeAmerican
Oceanian
Ancestry


Predicted Probabilities for Sample 8 (Mexican)
Model
Our Implementation
scikit-learn
Probability
1.0
0.8
0.6
0.4
0.2
0.0
African
EastAsian
European
NativeAmerican
Oceanian
Ancestry



Comparison Between Custom Implementation and scikit-learn's Logistic Regression

After implementing our own multinomial logistic regression model and comparing it with scikit-learn's `LogisticRegression`, we observed the following **Mean Absolute Differences in Predicted Probabilities per Ancestry**:

Ancestry	Mean Absolute Difference
African	0.122173
EastAsian	0.043979
European	0.145083
NativeAmerican	0.112824
Oceanian	0.032444

Observations

- **African & European:** These classes showed the highest differences, indicating noticeable discrepancies in probability assignments between our model and scikit-learn's.
- **EastAsian & Oceanian:** Lower differences suggest strong agreement for these ancestries.
- **NativeAmerican:** Moderate differences were observed, similar to African and European classes.

Possible Reasons for Differences

1. **Optimization Algorithms:**
   - **Our Implementation:** Uses Batch Gradient Descent, which can be slower and less precise.
   - **scikit-learn:** Utilizes the efficient L-BFGS algorithm, leading to more accurate parameter estimates.
2. **Regularization Handling:**
   - **Our Model:** May include regularization on all coefficients, including the intercept.
   - **scikit-learn:** Typically excludes the intercept from regularization, allowing it to adjust freely.
3. **Convergence Criteria:**
   - **Our Implementation:** Might lack advanced stopping criteria, potentially leading to incomplete convergence.
   - **scikit-learn:** Employs convergence checks, ensuring optimal parameter fitting.
4. **Learning Rate (`alpha`):**
   - **Our Implementation:** The choice of learning rate can greatly impact convergence and accuracy.
   - **scikit-learn:** Automatically manages learning rates within its optimization routine.

Conclusion

Despite the differences, our model aligns reasonably well with scikit-learn's, especially for certain ancestries. The discrepancies primarily stem from inherent differences in optimization techniques and regularization approaches. By refining our optimization method and adjusting regularization practices, we can further reduce these differences and enhance our model's performance.

In [ ]:

In [ ]: