In [123]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pyreadstat

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
```

In [55]:
```
# calling rh and lh volumes
left_volume_file_path = r"Z:\Active-Diagnose_CTE\Fargol_Analysis\Volumetric_Analysis\lh_
left_volume = pd.read_csv(left_volume_file_path)
left_volume = pd.DataFrame(left_volume)

right_volume_file_path = r"Z:\Active-Diagnose_CTE\Fargol_Analysis\Volumetric_Analysis\rl
right_volume = pd.read_csv(right_volume_file_path)
right_volume = pd.DataFrame(right_volume)
```

In [56]:
```
left_volume.head()
right_volume.head()
```

Out[56]:

	subject_id	visit	checkin_bin	exposurebin	age_decade	racecat_combined	eduyears	totyr_foot	chiiseas
0	1001	1	2	1	1	5	16.0	7.0	43
1	1002	1	2	1	1	5	15.0	14.0	103
2	1003	1	2	1	1	5	18.0	12.0	66
3	1004	1	1	1	2	5	16.0	16.0	77
4	1005	1	3	0	2	5	21.0	NaN	N

5 rows × 51 columns

In [57]:
```
print("Column Names:")
print(right_volume.columns[2])
```

```
Column Names:
checkin_bin
```

In [74]:

```
#group them base on the value in the third column which indicates their level of playin
right_grouped = right_volume.groupby(right_volume.iloc[:, 2])
left_grouped = left_volume.groupby(left_volume.iloc[:, 2])

NFL_right_grouped = pd.DataFrame()
CP_right_grouped = pd.DataFrame()
HC_right_grouped = pd.DataFrame()


# group_name : 1, 2, 3   group_data:
for group_name, group_data in right_grouped:
    if group_name == 1:
        NFL_right_grouped = pd.concat([NFL_right_grouped,group_data], ignore_index = Tr
    if group_name == 2:
        CP_right_grouped = pd.concat([CP_right_grouped,group_data], ignore_index = True
    if group_name == 3:
        HC_right_grouped = pd.concat([HC_right_grouped,group_data], ignore_index = True

#print("DataFrame for NFL:")
#print(NFL_right_grouped.head())
```

In [75]:

```
NFL_right_grouped.head()
#print(NFL_right_grouped.columns)
```

Out[75]:

	subject_id	visit	checkin_bin	exposurebin	age_decade	racecat_combined	eduyears	totyr_foot	chiiseas
0	1004	1	1	1	2	5	16.0	16.0	77
1	1008	1	1	1	2	3	15.0	22.0	82
2	1011	1	1	1	2	5	16.0	20.0	93
3	1015	1	1	1	1	3	19.0	17.0	98
4	1018	1	1	1	1	3	16.0	23.0	106

5 rows × 51 columns

In [76]:

```
#NFL_right_grouped.columns[[1] +list(range(3,index_of_Atlas+1))]
index_of_Atlas = NFL_right_grouped.columns.get_loc("Atlas")
CP_right_grouped.columns[[1] +list(range(3,index_of_Atlas+1))]
```

Out[76]:

```
Index(['visit', 'exposurebin', 'age_decade', 'racecat_combined', 'eduyears',
       'totyr_foot', 'chiiseas_pf', 'chiiyrs_pf', 'chiiseas_pl', 'chiiyrs_pl',
       'chiiseas_pg', 'chiiyrs_pg', 'timepoint_aparc', 'FreeSurfer_Version',
       'Atlas'],
      dtype='object')
```

In [77]:

```
# Atlas is the last column that needs to be deleted
index_of_Atlas = NFL_right_grouped.columns.get_loc("Atlas")
NFL_right_grouped.drop(columns=NFL_right_grouped.columns[[1] +list(range(3,index_of_Atl
CP_right_grouped.drop(columns=CP_right_grouped.columns[[1] +list(range(3,index_of_Atlas
HC_right_grouped.drop(columns=HC_right_grouped.columns[[1] +list(range(3,index_of_Atlas
```

In [78]:
```
NFL_right_grouped.head()
```

Out[78]:

	subject_id	checkin_bin	rh_bankssts_volume	rh_caudalanteriorcingulate_volum	rh_caudalmiddlefrontal_vol
0	1004	1	2310.0	1647.0	46
1	1008	1	1946.0	1687.0	49
2	1011	1	1961.0	2483.0	60
3	1015	1	2092.0	2032.0	52
4	1018	1	2547.0	2028.0	59

5 rows × 36 columns

In [80]:
```
#combine all three classes
combined_right_volume = pd.concat([NFL_right_grouped, CP_right_grouped, HC_right_groupe
```

In [81]:
```
combined_right_volume.head()
```

Out[81]:

	subject_id	checkin_bin	rh_bankssts_volume	rh_caudalanteriorcingulate_volum	rh_caudalmiddlefrontal_vol
0	1004	1	2310.0	1647.0	46
1	1008	1	1946.0	1687.0	49
2	1011	1	1961.0	2483.0	60
3	1015	1	2092.0	2032.0	52
4	1018	1	2547.0	2028.0	59

5 rows × 36 columns

In [82]:
```
# Separate based on the level of professionalism
X = combined_right_volume.drop(columns='checkin_bin')  # Adjust 'Label' to the actual c
y = combined_right_volume['checkin_bin']
```

In [112]:
```
# Splitting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7, random_state=4
```

In [113]:
```
# Normalization
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [114]:
```
# Replace NaNs with means
imputer = SimpleImputer(strategy='mean')  # You can choose a different strategy
X_train_imputed = imputer.fit_transform(X_train_scaled)
X_test_imputed = imputer.transform(X_test_scaled)
```

In [115]:
```
# train the model
model = RandomForestClassifier()
model.fit(X_train_imputed, y_train)
```

Out[115]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

In [116]:
```
# prediction
y_pred = model.predict(X_test_imputed)
```

In [117]:
```
# Evaluation
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f"Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}, F1-Score: {f1}"
```

```
Accuracy: 0.4879518072289157, Precision: 0.47299196787148595, Recall: 0.48795180722891
57, F1-Score: 0.4743113258032832
```

In [ ]:

In [119]:
```
# Improve the model by changing the number of trees (n_estimators)

# Set different hyperparameters
n_estimators_list = [50, 100, 150]
max_depth_list = [None, 10, 20]

# Iterate over hyperparameters
for n_estimators in n_estimators_list:
    for max_depth in max_depth_list:
        # Create and train the model
        model = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth,
        model.fit(X_train_imputed, y_train)

        # Make predictions on the test set
        y_pred = model.predict(X_test_imputed)
```

In [111]:
```
# Evaluation
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f"Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}, F1-Score: {f1}"
```

```
Accuracy: 0.4578313253012048, Precision: 0.43334750413063666, Recall: 0.45783132530120
48, F1-Score: 0.4339850315435631
```

In [126]:

```
# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    # Add other hyperparameters you want to tune
}

rf_model = RandomForestClassifier(random_state=42)

grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='accuracy')

grid_search.fit(X_train_imputed, y_train)

best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

best_model = grid_search.best_estimator_
```

```
Best Hyperparameters: {'max_depth': None, 'n_estimators': 150}
```

In [122]:

```
# Evaluate the best model on the test set
y_pred = best_model.predict(X_test_imputed)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print(f"Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}, F1-Score: {f1}"
```

```
Accuracy: 0.4578313253012048, Precision: 0.43334750413063666, Recall: 0.45783132530120
48, F1-Score: 0.4339850315435631
```

In [127]:

```
# Define the parameter distributions
param_dist = {
    'n_estimators': randint(50, 200),
    'max_depth': [None, 10, 20],
    # Add other hyperparameters you want to tune
}

# Create the RandomizedSearchCV object
random_search = RandomizedSearchCV(rf_model, param_distributions=param_dist, n_iter=10,

# Fit the random search to the data
random_search.fit(X_train_imputed, y_train)

# Get the best hyperparameters
best_params_random = random_search.best_params_
print("Best Hyperparameters (Random Search):", best_params_random)

# Get the best model
best_model_random = random_search.best_estimator_
```

```
Best Hyperparameters (Random Search): {'max_depth': 20, 'n_estimators': 156}
```

In [128]:

```
# Evaluate the best model on the test set
y_pred_random  = best_model_random.predict(X_test_imputed)
accuracy = accuracy_score(y_test, y_pred_random )
precision = precision_score(y_test, y_pred_random , average='weighted')
recall = recall_score(y_test, y_pred_random , average='weighted')
f1 = f1_score(y_test, y_pred_random , average='weighted')
print(f"Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}, F1-Score: {f1}"
```

```
Accuracy: 0.45180722891566266, Precision: 0.43086339027428033, Recall: 0.4518072289156
6266, F1-Score: 0.43009275464871055
```

In [ ]: