```
# Step 1: Import necessary libraries
import pandas as pd                    # For loading and handling the
dataset
import numpy as np                     # For numerical operations
from sklearn.model_selection import train_test_split   # To split the
data
from sklearn.ensemble import RandomForestClassifier    # Our base
model
from sklearn.metrics import accuracy_score             # To evaluate
model
from sklearn.preprocessing import LabelEncoder         # To convert
strings to numbers

# Step 2: Load the dataset
data = pd.read_csv('/content/healthcare-dataset-stroke-data.csv')

# Step 3: Drop rows with missing values
data = data.dropna()  # Removes rows that have missing (NaN) values

# Step 4: Encode categorical columns
label_encoders = {}
categorical_cols = ['gender', 'ever_married', 'work_type',
'Residence_type', 'smoking_status']

for col in categorical_cols:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])  # Convert text to numbers
    label_encoders[col] = le

# Step 5: Define features (X) and label (y)
X = data.drop(columns=['id', 'stroke'])  # Features: all columns
except 'id' and 'stroke'
y = data['stroke']                       # Target variable: stroke (0
or 1)

# Step 6: Split data into labeled and unlabeled
# Assume only 10% is labeled initially
X_labeled, X_unlabeled, y_labeled, y_unlabeled = train_test_split(
    X, y, test_size=0.90, stratify=y, random_state=42
)

# Step 7: Train model on labeled data
model = RandomForestClassifier(random_state=42)
model.fit(X_labeled, y_labeled)  # Train only on the small labeled
part

RandomForestClassifier(random_state=42)

# Step 8: Predict probabilities on the unlabeled data
probs = model.predict_proba(X_unlabeled)  # Predict class
probabilities
```

```
# Step 9: Select confident predictions (confidence > 0.9)
confident_indices = []
pseudo_labels = []

for i, prob in enumerate(probs):
    if max(prob) > 0.9:                          # If model is very
confident
        confident_indices.append(i)              # Save index
        pseudo_labels.append(np.argmax(prob))    # Save predicted label

# Step 10: Add confident pseudo-labeled data to the labeled set
X_confident = X_unlabeled.iloc[confident_indices]
y_confident = np.array(pseudo_labels)

# Combine with original labeled data
X_labeled = pd.concat([X_labeled, X_confident])
y_labeled = np.concatenate([y_labeled, y_confident])

# Remove used unlabeled samples
X_unlabeled = X_unlabeled.drop(X_unlabeled.index[confident_indices])

# Step 11: Retrain model on the updated labeled dataset
model.fit(X_labeled, y_labeled)
```

```
RandomForestClassifier(random_state=42)
```

```
# Step 12: Evaluate on a hold-out test set (optional, for accuracy
check)
# Let's split a separate test set from the original data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy on test data:", accuracy_score(y_test, y_pred))
```

```
Accuracy on test data: 0.9500978473581213
```

```
import matplotlib.pyplot as plt

# Prepare to repeat self-training for graph
X_labeled_iter = X_labeled.copy()
y_labeled_iter = y_labeled.copy()
X_unlabeled_iter = X_unlabeled.copy()

# Lists to store data for plotting
accuracy_list = []
pseudo_count_list = []
iterations = []
```

```
# Fixed test set for evaluation
X_train_eval, X_test_eval, y_train_eval, y_test_eval =
train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

# Self-training loop with graph data collection
for iteration in range(1, 11):  # up to 10 rounds
    model = RandomForestClassifier(random_state=42)
    model.fit(X_labeled_iter, y_labeled_iter)

    y_pred = model.predict(X_test_eval)
    accuracy = accuracy_score(y_test_eval, y_pred)

    # Predict on unlabeled
    probs = model.predict_proba(X_unlabeled_iter)

    confident_indices = []
    pseudo_labels = []

    for i, prob in enumerate(probs):
        if max(prob) > 0.9:  # confidence threshold
            confident_indices.append(i)
            pseudo_labels.append(np.argmax(prob))

    if not confident_indices:
        break  # Stop if no more confident predictions

    # Save graph info
    accuracy_list.append(accuracy)
    pseudo_count_list.append(len(confident_indices))
    iterations.append(iteration)

    # Add confident pseudo-labeled data
    X_confident = X_unlabeled_iter.iloc[confident_indices]
    y_confident = np.array(pseudo_labels)

    X_labeled_iter = pd.concat([X_labeled_iter, X_confident])
    y_labeled_iter = np.concatenate([y_labeled_iter, y_confident])
    X_unlabeled_iter =
X_unlabeled_iter.drop(X_unlabeled_iter.index[confident_indices])

# === □ Plotting ===
plt.figure(figsize=(10, 6))
plt.plot(iterations, accuracy_list, marker='o', color='blue',
label='Test Accuracy')
plt.bar(iterations, pseudo_count_list, alpha=0.3, color='orange',
label='Pseudo-labeled added')
```

```
plt.title("Self-Training Learning: Accuracy & Pseudo-Labels Over
Iterations")
plt.xlabel("Iteration")
plt.ylabel("Accuracy / Pseudo-Labels")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```