```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline

# Load dataset
data = pd.read_csv('heart.csv')

# Separate features and target
X = data.drop('HeartDisease', axis=1)
y = data['HeartDisease']

# Identify categorical and numerical columns
cat_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']
num_cols = [col for col in X.columns if col not in cat_cols]

# Label encode binary categorical columns with two categories (e.g. Sex, ExerciseAngina)
label_enc_cols = ['Sex', 'ExerciseAngina']
le = LabelEncoder()
for col in label_enc_cols:
    X[col] = le.fit_transform(X[col])

# For other categorical columns with more than two categories, apply OneHotEncoding
onehot_cols = list(set(cat_cols) - set(label_enc_cols))

# Preprocessing pipeline: OneHotEncoding + scaling numerical features
preprocessor = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(drop='first'), onehot_cols),
        ('scaler', StandardScaler(), num_cols)
    ],
    remainder='passthrough'  # To keep label encoded columns as is
)

# Split data into train/test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y)

# Helper function to train and evaluate a model
def train_evaluate_model(model, X_train, X_test, y_train, y_test):
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('classifier', model)])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    return acc, pipeline

# Train and evaluate SVM
svm = SVC(random_state=42)
svm_acc, svm_pipeline = train_evaluate_model(svm, X_train, X_test, y_train, y_test)

# Train and evaluate Logistic Regression
logreg = LogisticRegression(max_iter=1000, random_state=42)
logreg_acc, logreg_pipeline = train_evaluate_model(logreg, X_train, X_test, y_train, y_test)

# Train and evaluate Random Forest
rf = RandomForestClassifier(random_state=42)
rf_acc, rf_pipeline = train_evaluate_model(rf, X_train, X_test, y_train, y_test)

print(f"Accuracy Scores without PCA:\nSVM: {svm_acc:.4f}\nLogistic Regression: {logreg_acc:.4f}\nRandom Forest: {rf_acc:.4f}")

# Now apply PCA for dimensionality reduction after preprocessing (scaling + encoding)
# We modify the pipeline to include PCA before classification

def train_evaluate_model_pca(model, X_train, X_test, y_train, y_test, n_components):
    pca = PCA(n_components=n_components, random_state=42)
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('pca', pca),
        ('classifier', model)
    ])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    return acc, pipeline
```

```
# Choose number of components to keep 95% variance or fixed number (e.g. 5)
# Here let's pick 5 components arbitrarily
n_components = 5

svm_pca_acc, _ = train_evaluate_model_pca(svm, X_train, X_test, y_train, y_test, n_components)
logreg_pca_acc, _ = train_evaluate_model_pca(logreg, X_train, X_test, y_train, y_test, n_components)
rf_pca_acc, _ = train_evaluate_model_pca(rf, X_train, X_test, y_train, y_test, n_components)

print(f"\nAccuracy Scores with PCA (n_components={n_components}):")
print(f"SVM: {svm_pca_acc:.4f}")
print(f"Logistic Regression: {logreg_pca_acc:.4f}")
print(f"Random Forest: {rf_pca_acc:.4f}")
```

```
Accuracy Scores without PCA:
SVM: 0.9130
Logistic Regression: 0.8841
Random Forest: 0.9094

Accuracy Scores with PCA (n_components=5):
SVM: 0.8623
Logistic Regression: 0.8732
Random Forest: 0.8551
```