```
import pandas as pd
import numpy as np
import tkinter as tk
from tkinter import messagebox
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, Imputer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import shap

# Data Collection (Assuming a diverse dataset named "health_data.csv" is available)
data = pd.read_csv("health_data.csv")

# Data Preprocessing
# Handle missing values using imputation
imputer = Imputer(strategy='mean')
data[['age', 'BMI', 'blood_pressure', 'cholesterol']] = imputer.fit_transform(data[['age', 'BMI', 'blood_pressure', 'cholesterol']])

# Normalize features
scaler = StandardScaler()
data[['age', 'BMI', 'blood_pressure', 'cholesterol']] = scaler.fit_transform(data[['age', 'BMI', 'blood_pressure', 'cholesterol']])

# Handle outliers (if necessary)

# Split features and target
X = data[['age', 'gender', 'BMI', 'blood_pressure', 'cholesterol', 'family_history']]
```

```
y = data['disease']

# Feature Selection
selector = SelectKBest(chi2, k=5)
X_selected = selector.fit_transform(X, y)

# Model Development
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)

# Explore and implement various machine learning algorithms
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Support Vector Machine": SVC()
}

# Evaluate and compare the performance of different models
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    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"Model: {name}")
    print(f"Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}, F1-score: {f1}")

# Cross-Validation
cv_scores = {}
```

```
for name, model in models.items():
    scores = cross_val_score(model, X_selected, y, cv=5)
    cv_scores[name] = scores.mean()

print("Cross-validation scores:")
print(cv_scores)

# Hyperparameter Tuning
parameters = {
    "Logistic Regression": {'C': [0.1, 1, 10]},
    "Decision Tree": {'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10]},
    "Random Forest": {'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20],
'min_samples_split': [2, 5, 10]},
    "Support Vector Machine": {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
}

best_models = {}
for name, model in models.items():
    grid_search = GridSearchCV(model, parameters[name], cv=5)
    grid_search.fit(X_selected, y)
    best_models[name] = grid_search.best_estimator_

print("Best models after hyperparameter tuning:")
print(best_models)

# Model Interpretability
# SHAP values for all models
explainers = {name: shap.Explainer(model, X_train) for name, model in best_models.items()}
shap_values = {name: explainer.shap_values(X_test) for name, explainer in explainers.items()}

# User Interface
```

```python
def predict_disease():
    # Retrieve user input
    user_data = {
        'age': float(age_entry.get()),
        'gender': gender_var.get(),
        'BMI': float(BMI_entry.get()),
        'blood_pressure': float(blood_pressure_entry.get()),
        'cholesterol': float(cholesterol_entry.get()),
        'family_history': family_history_var.get()
    }
    user_input = pd.DataFrame([user_data])

    # Feature selection
    user_input_selected = selector.transform(user_input)

    # Predict disease using all models
    predictions = {name: model.predict(user_input_selected)[0] for name, model in best_models.items()}

    # Show predictions
    messagebox.showinfo("Predictions", "\n".join([f"{name}: {prediction}" for name, prediction in predictions.items()]))

# Placeholder function for EHR integration
def integrate_with_ehr():
    # Placeholder for EHR integration
    pass

# Create GUI
root = tk.Tk()
root.title("Disease Prediction")
root.geometry("400x400")
```

```
# User input fields
tk.Label(root, text="Age").grid(row=0, column=0)
age_entry = tk.Entry(root)
age_entry.grid(row=0, column=1)

tk.Label(root, text="Gender").grid(row=1, column=0)
gender_var = tk.StringVar(root)
gender_var.set("Male")
gender_menu = tk.OptionMenu(root, gender_var, "Male", "Female")
gender_menu.grid(row=1, column=1)

tk.Label(root, text="BMI").grid(row=2, column=0)
BMI_entry = tk.Entry(root)
BMI_entry.grid(row=2, column=1)

tk.Label(root, text="Blood Pressure").grid(row=3, column=0)
blood_pressure_entry = tk.Entry(root)
blood_pressure_entry.grid(row=3, column=1)

tk.Label(root, text="Cholesterol").grid(row=4, column=0)
cholesterol_entry = tk.Entry(root)
cholesterol_entry.grid(row=4, column=1)

family_history_var = tk.BooleanVar(root)
tk.Checkbutton(root, text="Family History", variable=family_history_var).grid(row=5, columnspan=2)

predict_button = tk.Button(root, text="Predict", command=predict_disease)
predict_button.grid(row=6, columnspan=2)

ehr_button = tk.Button(root, text="Integrate with EHR", command=integrate_with_ehr)
```

```
ehr_button.grid(row=7, columnspan=2)


root.mainloop()
```