Code with Comments:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
import tkinter as tk
from tkinter import ttk

# Assuming your dataset is in a DataFrame named 'df'
# Replace 'your_dataset.csv' with your actual dataset filepip install
scikit-learn

df = pd.read_csv('dataset2.csv')
# Columns: 'age_group', 'gender', 'PhAct', 'BMI', 'Glucose', 'Cholestrol',
'BP'

# Select relevant columns for standardization
columns_to_standardize = ['age', 'sex', 'cp', 'restbps', 'chol',
'fbsugar', 'restecg', 'maxhrate']

# Extract the selected columns from the DataFrame
data_to_standardize = df[columns_to_standardize]

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the data
standardized_data = scaler.fit_transform(data_to_standardize)

# Replace the original columns with the standardized values in the
DataFrame
df[columns_to_standardize] = standardized_data
```

Code with Comments:

```
# Now, 'age_group', 'BMI', 'blood_glucose_level', 'blood_insulin_level' are standardized in the DataFrame
# Print the DataFrame with standardized data
print(df)

# Assuming df is your DataFrame with features and 'target' is your target variable
X = df[['age', 'sex', 'cp', 'restbps', 'chol', 'fbsugar', 'restecg', 'maxhrate']]
y = df['target']

# Create a RandomForestClassifier
model = RandomForestClassifier()

# Fit the model to the data
model.fit(X, y)

# Access feature importances
feature_importances = model.feature_importances_

# Print or use feature importances as needed
print(feature_importances)

# Assuming X contains your features and y is your target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Logistic Regression
logreg_model = LogisticRegression()
logreg_model.fit(X_train, y_train)

# Decision Tree
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)

# Random Forest
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

# Support Vector Machine (SVM)
```

Code with Comments:

```
svm_model = SVC()
svm_model.fit(X_train, y_train)

# Evaluate Logistic Regression
logreg_pred = logreg_model.predict(X_test)
print("Logistic Regression:")
print("Accuracy:", accuracy_score(y_test, logreg_pred))
print("Precision:", precision_score(y_test, logreg_pred))
print("Recall:", recall_score(y_test, logreg_pred))
print("F1 Score:", f1_score(y_test, logreg_pred))
print("\n")

# Evaluate Decision Tree
dt_pred = dt_model.predict(X_test)
print("Decision Tree:")
print("Accuracy:", accuracy_score(y_test, dt_pred))
print("Precision:", precision_score(y_test, dt_pred))
print("Recall:", recall_score(y_test, dt_pred))
print("F1 Score:", f1_score(y_test, dt_pred))
print("\n")

# Evaluate Random Forest
rf_pred = rf_model.predict(X_test)
print("Random Forest:")
print("Accuracy:", accuracy_score(y_test, rf_pred))
print("Precision:", precision_score(y_test, rf_pred))
print("Recall:", recall_score(y_test, rf_pred))
print("F1 Score:", f1_score(y_test, rf_pred))
print("\n")

# Evaluate Support Vector Machine
svm_pred = svm_model.predict(X_test)
print("Support Vector Machine:")
print("Accuracy:", accuracy_score(y_test, svm_pred))
print("Precision:", precision_score(y_test, svm_pred))
print("Recall:", recall_score(y_test, svm_pred))
print("F1 Score:", f1_score(y_test, svm_pred))

#moving forward with random forest classifier
#X contains features and y is your target variable
```

Code with Comments:

```
#Perform K-Fold Cross-Validation (e.g., 5 folds)
cv_scores = cross_val_score(rf_model, X, y, cv=5, scoring='accuracy')

# Print cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean Accuracy:", cv_scores.mean())

# Define the hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

# Perform GridSearchCV with cross-validation
grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Use the best model for evaluation
best_rf_model = grid_search.best_estimator_
best_rf_pred = best_rf_model.predict(X_test)

# Evaluate the best model
accuracy = accuracy_score(y_test, best_rf_pred)
print("Accuracy of the Best Model:", accuracy)

def make_prediction():
    # Get user inputs from the entry fields
    age = int(age_entry.get())
    sex = int(sex_entry.get())
    cp = int(cp_entry.get())
    restbps = int(restbps_entry.get())
    chol = int(chol_entry.get())
    fbsugar = int(fbsugar_entry.get())
    restecg = int(restecg_entry.get())
    maxhrate = int(maxhrate_entry.get())
    # Include other features as needed
```

Code with Comments:

```
    # Preprocess the input data (standardization in this example)
    user_data = scaler.transform([[age, sex, cp, restbps, chol, fbsugar,
restecg, maxhrate]])

    # Make predictions using the trained model
    prediction = best_rf_model.predict(user_data)

    # Display the prediction in the result label
    result_label.config(text=f"Likelihood of having the disease:
{prediction[0]}")


# Create the main window
root = tk.Tk()
root.title("Disease Prediction Interface")

# Create and pack input fields
age_label = ttk.Label(root, text="Age:")
age_label.pack()

age_entry = ttk.Entry(root)
age_entry.pack()

# Create and pack input fields
sex_label = ttk.Label(root, text="Sex:")
sex_label.pack()

sex_entry = ttk.Entry(root)
sex_entry.pack()

# Create and pack input fields
cp_label = ttk.Label(root, text="Chest Pain:")
cp_label.pack()

cp_entry = ttk.Entry(root)
cp_entry.pack()

# Create and pack input fields
restbps_label = ttk.Label(root, text="Blood Press:")
```

Code with Comments:

```
restbps_label.pack()

restbps_entry = ttk.Entry(root)
restbps_entry.pack()

# Create and pack input fields
chol_label = ttk.Label(root, text="Cholestrol:")
chol_label.pack()

chol_entry = ttk.Entry(root)
chol_entry.pack()

# Create and pack input fields
fbsugar_label = ttk.Label(root, text="Blood Sugar:")
fbsugar_label.pack()

fbsugar_entry = ttk.Entry(root)
fbsugar_entry.pack()

# Create and pack input fields
restecg_label = ttk.Label(root, text="ECG:")
restecg_label.pack()

restecg_entry = ttk.Entry(root)
restecg_entry.pack()

# Create and pack input fields
maxhrate_label = ttk.Label(root, text="Max Heart Rate:")
maxhrate_label.pack()

maxhrate_entry = ttk.Entry(root)
maxhrate_entry.pack()

# Create prediction button
predict_button = ttk.Button(root, text="Predict", command=make_prediction)
predict_button.pack()

# Create a label to display the prediction result
result_label = ttk.Label(root, text="")
result_label.pack()
```

Code with Comments:

```
# Run the GUI
root.mainloop()
```