**1.First it mounts Google Drive to the Colab notebook using the Google Colab library:**

```
from google.colab import drive
drive.mount("/content/drive/")
```

**2.The code then imports necessary libraries for data preprocessing and model training:**

```
import numpy as np
import scipy
from sklearn.preprocessing import normalize
from sklearn import svm
from sklearn.model_selection import train_test_split

import keras
from keras.models import Sequential
from keras.layers import LSTM, Dense
```

**3.Next, the code loads various data files using np.load() function from NumPy:**

```
train_path = "/content/drive/MyDrive/Colab Notebooks/bbh/training/"
test_path = "/content/drive/MyDrive/Colab Notebooks/bbh/testing/"

train_labels = np.load(train_path + "trainLabels.npy")
train_ms_acc = np.load(train_path + "trainMSAccelerometer.npy")
train_ms_gyro = np.load(train_path + "trainMSGyroscope.npy")
train_jins_acc = np.load(train_path + "trainJinsAccelerometer.npy")
train_jins_gyro = np.load(train_path + "trainJinsGyroscope.npy")
train_acc = np.load(train_path + "trainAccelerometer.npy")
train_gravity = np.load(train_path + "trainGravity.npy")
train_gyro = np.load(train_path + "trainGyroscope.npy")
```

```
train_lin_acc = np.load(train_path + "trainLinearAcceleration.npy")
train_mag = np.load(train_path + "trainMagnetometer.npy")

test_labels = np.load(test_path + "testLabels.npy")
test_ms_acc = np.load(test_path + "testMSAccelerometer.npy")
test_ms_gyro = np.load(test_path + "testMSGyroscope.npy")
test_jins_acc = np.load(test_path + "testJinsAccelerometer.npy")
test_jins_gyro = np.load(test_path + "testJinsGyroscope.npy")
test_acc = np.load(test_path + "testAccelerometer.npy")
test_gravity = np.load(test_path + "testGravity.npy")
test_gyro = np.load(test_path + "testGyroscope.npy")
test_lin_acc = np.load(test_path + "testLinearAcceleration.npy")
test_mag = np.load(test_path + "testMagnetometer.npy")
```

**4.The code then defines functions for data normalization and feature extraction.**

**Next, the code performs normalization on the loaded data arrays:**

```
norm_train_ms_acc = normalization(train_ms_acc)
norm_train_ms_gyro = normalization(train_ms_gyro)
norm_train_jins_acc = normalization(train_jins_acc)
norm_train_jins_gyro = normalization(train_jins_gyro)
norm_train_acc = normalization(train_acc)
norm_train_gravity = normalization(train_gravity)
norm_train_gyro = normalization(train_gyro)
norm_train_lin_acc = normalization(train_lin_acc)
norm_train_mag = normalization(train_mag)

norm_test_ms_acc = normalization(test_ms_acc)
norm_test_ms_gyro = normalization(test_ms_gyro)
norm_test_jins_acc = normalization(test_jins_acc)
norm_test_jins_gyro = normalization(test_jins_gyro)
norm_test_acc = normalization(test_acc)
```

```
norm_test_gravity = normalization(test_gravity)
norm_test_gyro = normalization(test_gyro)
norm_test_lin_acc = normalization(test_lin_acc)
norm_test_mag = normalization(test_mag)
)
```

**5.The code defines functions for segmentation and feature extraction.**

**Segmentation is performed on the normalized data arrays:**

```
feature_train_ms_acc = get_features(norm_train_ms_acc, window_size, stride_size)
feature_train_ms_gyro = get_features(norm_train_ms_gyro, window_size, stride_size)
feature_train_jins_acc = get_features(norm_train_jins_acc, window_size, stride_size)
feature_train_jins_gyro = get_features(norm_train_jins_gyro, window_size, stride_size)
feature_train_acc = get_features(norm_train_acc, window_size, stride_size)
feature_train_gravity = get_features(norm_train_gravity, window_size, stride_size)
feature_train_gyro = get_features(norm_train_gyro, window_size, stride_size)
feature_train_lin_acc = get_features(norm_train_lin_acc, window_size, stride_size)
feature_train_mag = get_features(norm_train_mag, window_size, stride_size)

feature_test_ms_acc = get_features(norm_test_ms_acc, window_size, stride_size)
feature_test_ms_gyro = get_features(norm_test_ms_gyro, window_size, stride_size)
feature_test_jins_acc = get_features(norm_test_jins_acc, window_size, stride_size)
feature_test_jins_gyro = get_features(norm_test_jins_gyro, window_size, stride_size)
feature_test_acc = get_features(norm_test_acc, window_size, stride_size)
feature_test_gravity = get_features(norm_test_gravity, window_size, stride_size)
feature_test_gyro = get_features(norm_test_gyro, window_size, stride_size)
feature_test_lin_acc = get_features(norm_test_lin_acc, window_size, stride_size)
feature_test_mag = get_features(norm_test_mag, window_size, stride_size)
```

**6.Finally, the extracted features are combined into a single feature matrix:**

```
feature_train = feature_train_ms_acc
feature_train = np.hstack((feature_train, feature_train_ms_gyro))
feature_train = np.hstack((feature_train, feature_train_jins_acc))
feature_train = np.hstack((feature_train, feature_train_jins_gyro))
feature_train = np.hstack((feature_train, feature_train_acc))
feature_train = np.hstack((feature_train, feature_train_gravity))
feature_train = np.hstack((feature_train, feature_train_gyro))
feature_train = np.hstack((feature_train, feature_train_lin_acc))
feature_train = np.hstack((feature_train, feature_train_mag))

feature_test = feature_test_ms_acc
feature_test = np.hstack((feature_test, feature_test_ms_gyro))
feature_test = np.hstack((feature_test, feature_test_jins_acc))
feature_test = np.hstack((feature_test, feature_test_jins_gyro))
feature_test = np.hstack((feature_test, feature_test_acc))
feature_test = np.hstack((feature_test, feature_test_gravity))
feature_test = np.hstack((feature_test, feature_test_gyro))
feature_test = np.hstack((feature_test, feature_test_lin_acc))
feature_test = np.hstack((feature_test, feature_test_mag))

print(feature_train.shape)
print(feature_test.shape)
```

**7.Then if performs classification on it**

```
from sklearn import svm
from sklearn.impute import SimpleImputer

# Create an imputer object
imputer = SimpleImputer(strategy='mean')

# Fit the imputer on the training data
imputer.fit(feature_train)
```

```
# Transform the training data
feature_train_imputed = imputer.transform(feature_train)

# Create an SVM classifier with linear kernel
classification = svm.SVC(kernel='linear')

# Fit the classifier on the imputed training data
classification.fit(feature_train_imputed, trainLabels)
```

**8.and at the end it checks the model how accurate and good it is working**

```
# Transform the testing data using the imputer
feature_test_imputed = imputer.transform(feature_test)

# Evaluate the classifier on the imputed testing data
measurement_of_standard = classification.score(feature_test_imputed, testLabels)
print("How far it is good?:", measurement_of_standard)
```