```
import pandas as pd
import numpy as np
from scipy import signal

from google.colab import drive
drive.mount('/content/drive')

# Define column names
column_names = [
    "Time", "Ankle_horizontal_forward",
"Ankle_vertical", "Ankle_horizontal_lateral",
    "Upper_leg_horizontal_forward",
"Upper_leg_vertical", "Upper_leg_horizontal_lateral",
    "Trunk_horizontal_forward", "Trunk_vertical",
"Trunk_horizontal_lateral","Annotation"
]

# Read the text file into a DataFrame
df1 =
pd.read_csv("/content/drive/MyDrive/dataset/S01R01.tx
t", delimiter=" ", names=column_names)
df2 =
pd.read_csv("/content/drive/MyDrive/dataset/S01R02.tx
t", delimiter=" ", names=column_names)
df3 =
pd.read_csv("/content/drive/MyDrive/dataset/S02R01.tx
t", delimiter=" ", names=column_names)
df4 =
pd.read_csv("/content/drive/MyDrive/dataset/S02R02.tx
t", delimiter=" ", names=column_names)
df5 =
pd.read_csv("/content/drive/MyDrive/dataset/S03R01.tx
t", delimiter=" ", names=column_names)
df6 =
pd.read_csv("/content/drive/MyDrive/dataset/S03R02.tx
t", delimiter=" ", names=column_names)
df7 =
pd.read_csv("/content/drive/MyDrive/dataset/S03R03.tx
t", delimiter=" ", names=column_names)
```

```
df8 =
pd.read_csv("/content/drive/MyDrive/dataset/S04R01.tx
t", delimiter=" ", names=column_names)
df9 =
pd.read_csv("/content/drive/MyDrive/dataset/S05R01.tx
t", delimiter=" ", names=column_names)
df10 =
pd.read_csv("/content/drive/MyDrive/dataset/S05R02.tx
t", delimiter=" ", names=column_names)
df11 =
pd.read_csv("/content/drive/MyDrive/dataset/S06R01.tx
t", delimiter=" ", names=column_names)
df12 =
pd.read_csv("/content/drive/MyDrive/dataset/S06R02.tx
t", delimiter=" ", names=column_names)
df13 =
pd.read_csv("/content/drive/MyDrive/dataset/S07R01.tx
t", delimiter=" ", names=column_names)
df14 =
pd.read_csv("/content/drive/MyDrive/dataset/S07R02.tx
t", delimiter=" ", names=column_names)
df15 =
pd.read_csv("/content/drive/MyDrive/dataset/S08R01.tx
t", delimiter=" ", names=column_names)
df16 =
pd.read_csv("/content/drive/MyDrive/dataset/S09R01.tx
t", delimiter=" ", names=column_names)
df17 =
pd.read_csv("/content/drive/MyDrive/dataset/S10R01.tx
t", delimiter=" ", names=column_names)

# Create a list to hold your dataframes
dataframes = [df1, df2, df3, df4, df5, df6, df7, df8,
df9, df10, df11, df12, df13, df14, df15, df16, df17]

# Concatenate dataframes vertically
data = pd.concat(dataframes, ignore_index=True)

# Print the first few rows of the combined dataframe
data.head(10)

acc_magnitude =
np.sqrt(data['Ankle_horizontal_forward']**2 +
```

```
data['Ankle_vertical']**2 +
data['Ankle_horizontal_lateral']**2)
data['Acc_magnitude'] = acc_magnitude

window_size = 100  # Choose an appropriate window
size

# Initialize empty arrays to store calculated
features
mean_values = []
variance_values = []
std_deviation_values = []

# Iterate through the data using the sliding window
for i in range(0, len(data) - window_size + 1, 1):
    window_data = data.iloc[i:i+window_size]

    # Calculate mean, variance, and standard
deviation for the window

mean_values.append(window_data['Ankle_horizontal_forw
ard'].mean())

variance_values.append(window_data['Ankle_horizontal_
forward'].var())

std_deviation_values.append(window_data['Ankle_horizo
ntal_forward'].std())

""" Add the calculated features to the DataFrame
data['Windowed_Mean'] = mean_values
data['Windowed_Variance'] = variance_values
data['Windowed_Std_Deviation'] =
std_deviation_values"""
len(mean_values)
len(variance_values)
len(std_deviation_values)

list1=[mean_values[len(mean_values)-1]]*(len(data)-
len(mean_values))

list2=[variance_values[len(variance_values)-
1]]*(len(data)-len(mean_values))
```

```
list3=[std_deviation_values[len(std_deviation_values)
-1]]*(len(data)-len(mean_values))


""" Add the calculated features to the DataFrame"""
data['Ankle_horizontal_forward_Windowed_Mean'] =
mean_values
data['Ankle_horizontal_forward_Windowed_Variance'] =
variance_values
data['Ankle_horizontal_forward_Windowed_Std_Deviation
'] = std_deviation_values

window_size = 100  # Choose an appropriate window
size

# Initialize empty arrays to store calculated
features
mean_values = []
variance_values = []
std_deviation_values = []

# Iterate through the data using the sliding window
for i in range(0, len(data) - window_size + 1, 1):
    window_data = data.iloc[i:i+window_size]

    # Calculate mean, variance, and standard
deviation for the window

mean_values.append(window_data['Ankle_vertical'].mean
())

variance_values.append(window_data['Ankle_vertical'].
var())

std_deviation_values.append(window_data['Ankle_vertic
al'].std())

list1=[mean_values[len(mean_values)-1]]*(len(data)-
len(mean_values))
list2=[variance_values[len(variance_values)-
1]]*(len(data)-len(mean_values))
list3=[std_deviation_values[len(std_deviation_values)
-1]]*(len(data)-len(mean_values))
mean_values.extend(list1)
```

```
variance_values.extend(list2)
std_deviation_values.extend(list3)
""" Add the calculated features to the DataFrame"""
data['Ankle_vertical_Windowed_Mean'] = mean_values
data['Ankle_vertical_Windowed_Variance'] =
variance_values
data['Ankle_vertical_Windowed_Std_Deviation'] =
std_deviation_values

window_size = 100  # Choose an appropriate window
size

# Initialize empty arrays to store calculated
features
mean_values = []
variance_values = []
std_deviation_values = []

# Iterate through the data using the sliding window
for i in range(0, len(data) - window_size + 1, 1):
    window_data = data.iloc[i:i+window_size]

    # Calculate mean, variance, and standard
deviation for the window

mean_values.append(window_data['Ankle_horizontal_late
ral'].mean())

variance_values.append(window_data['Ankle_horizontal_
lateral'].var())

std_deviation_values.append(window_data['Ankle_horizo
ntal_lateral'].std())

list1=[mean_values[len(mean_values)-1]]*(len(data)-
len(mean_values))
list2=[variance_values[len(variance_values)-
1]]*(len(data)-len(mean_values))
list3=[std_deviation_values[len(std_deviation_values)
-1]]*(len(data)-len(mean_values))
mean_values.extend(list1)
variance_values.extend(list2)
std_deviation_values.extend(list3)
""" Add the calculated features to the DataFrame"""
```

```
data['Ankle_horizontal_lateral_Windowed_Mean'] =
mean_values
data['Ankle_horizontal_lateral_Windowed_Variance'] =
variance_values
data['Ankle_horizontal_lateral_Windowed_Std_Deviation
'] = std_deviation_values

window_size = 100  # Choose an appropriate window
size

# Initialize empty arrays to store calculated
features
mean_values = []
variance_values = []
std_deviation_values = []

# Iterate through the data using the sliding window
for i in range(0, len(data) - window_size + 1, 1):
    window_data = data.iloc[i:i+window_size]

    # Calculate mean, variance, and standard
deviation for the window

mean_values.append(window_data['Upper_leg_horizontal_
forward'].mean())

variance_values.append(window_data['Upper_leg_horizon
tal_forward'].var())

std_deviation_values.append(window_data['Upper_leg_ho
rizontal_forward'].std())

list1=[mean_values[len(mean_values)-1]]*(len(data)-
len(mean_values))
list2=[variance_values[len(variance_values)-
1]]*(len(data)-len(mean_values))
list3=[std_deviation_values[len(std_deviation_values)
-1]]*(len(data)-len(mean_values))
mean_values.extend(list1)
variance_values.extend(list2)
std_deviation_values.extend(list3)
""" Add the calculated features to the DataFrame"""
data['Upper_leg_horizontal_forward_Windowed_Mean'] =
mean_values
```

```
data['Upper_leg_horizontal_forward_Windowed_Variance'
] = variance_values
data['Upper_leg_horizontal_forward_Windowed_Std_Devia
tion'] = std_deviation_values

window_size = 100  # Choose an appropriate window
size

# Initialize empty arrays to store calculated
features
mean_values = []
variance_values = []
std_deviation_values = []

# Iterate through the data using the sliding window
for i in range(0, len(data) - window_size + 1, 1):
    window_data = data.iloc[i:i+window_size]

    # Calculate mean, variance, and standard
deviation for the window

mean_values.append(window_data['Upper_leg_vertical'].
mean())

variance_values.append(window_data['Upper_leg_vertica
l'].var())

std_deviation_values.append(window_data['Upper_leg_ve
rtical'].std())

list1=[mean_values[len(mean_values)-1]]*(len(data)-
len(mean_values))
list2=[variance_values[len(variance_values)-
1]]*(len(data)-len(mean_values))
list3=[std_deviation_values[len(std_deviation_values)
-1]]*(len(data)-len(mean_values))
mean_values.extend(list1)
variance_values.extend(list2)
std_deviation_values.extend(list3)
""" Add the calculated features to the DataFrame"""
data['Upper_leg_vertical_Windowed_Mean'] =
mean_values
data['Upper_leg_vertical_Windowed_Variance'] =
variance_values
```

```
data['Upper_leg_vertical_Windowed_Std_Deviation'] =
std_deviation_values

window_size = 100  # Choose an appropriate window
size

# Initialize empty arrays to store calculated
features
mean_values = []
variance_values = []
std_deviation_values = []

# Iterate through the data using the sliding window
for i in range(0, len(data) - window_size + 1, 1):
    window_data = data.iloc[i:i+window_size]

    # Calculate mean, variance, and standard
deviation for the window

mean_values.append(window_data['Upper_leg_horizontal_
lateral'].mean())

variance_values.append(window_data['Upper_leg_horizon
tal_lateral'].var())

std_deviation_values.append(window_data['Upper_leg_ho
rizontal_lateral'].std())

list1=[mean_values[len(mean_values)-1]]*(len(data)-
len(mean_values))
list2=[variance_values[len(variance_values)-
1]]*(len(data)-len(mean_values))
list3=[std_deviation_values[len(std_deviation_values)
-1]]*(len(data)-len(mean_values))
mean_values.extend(list1)
variance_values.extend(list2)
std_deviation_values.extend(list3)
""" Add the calculated features to the DataFrame"""
data['Upper_leg_horizontal_lateral_Windowed_Mean'] =
mean_values
data['Upper_leg_horizontal_lateral_Windowed_Variance'
] = variance_values
data['Upper_leg_horizontal_lateral_Windowed_Std_Devia
tion'] = std_deviation_values
```

```
window_size = 100  # Choose an appropriate window
size

# Initialize empty arrays to store calculated
features
mean_values = []
variance_values = []
std_deviation_values = []

# Iterate through the data using the sliding window
for i in range(0, len(data) - window_size + 1, 1):
    window_data = data.iloc[i:i+window_size]

    # Calculate mean, variance, and standard
deviation for the window

mean_values.append(window_data['Trunk_horizontal_forw
ard'].mean())

variance_values.append(window_data['Trunk_horizontal_
forward'].var())

std_deviation_values.append(window_data['Trunk_horizo
ntal_forward'].std())

list1=[mean_values[len(mean_values)-1]]*(len(data)-
len(mean_values))
list2=[variance_values[len(variance_values)-
1]]*(len(data)-len(mean_values))
list3=[std_deviation_values[len(std_deviation_values)
-1]]*(len(data)-len(mean_values))
mean_values.extend(list1)
variance_values.extend(list2)
std_deviation_values.extend(list3)
""" Add the calculated features to the DataFrame"""
data['Trunk_horizontal_forward_Windowed_Mean'] =
mean_values
data['Trunk_horizontal_forward_Windowed_Variance'] =
variance_values
data['Trunk_horizontal_forward_Windowed_Std_Deviation
'] = std_deviation_values
```

```
window_size = 100  # Choose an appropriate window
size

# Initialize empty arrays to store calculated
features
mean_values = []
variance_values = []
std_deviation_values = []

# Iterate through the data using the sliding window
for i in range(0, len(data) - window_size + 1, 1):
    window_data = data.iloc[i:i+window_size]

    # Calculate mean, variance, and standard
deviation for the window

mean_values.append(window_data['Trunk_vertical'].mean
())

variance_values.append(window_data['Trunk_vertical'].
var())

std_deviation_values.append(window_data['Trunk_vertic
al'].std())

list1=[mean_values[len(mean_values)-1]]*(len(data)-
len(mean_values))
list2=[variance_values[len(variance_values)-
1]]*(len(data)-len(mean_values))
list3=[std_deviation_values[len(std_deviation_values)
-1]]*(len(data)-len(mean_values))
mean_values.extend(list1)
variance_values.extend(list2)
std_deviation_values.extend(list3)
""" Add the calculated features to the DataFrame"""
data['Trunk_vertical_Windowed_Mean'] = mean_values
data['Trunk_vertical_Windowed_Variance'] =
variance_values
data['Trunk_vertical_Windowed_Std_Deviation'] =
std_deviation_values

window_size = 100  # Choose an appropriate window
size
```

```
# Initialize empty arrays to store calculated
features
mean_values = []
variance_values = []
std_deviation_values = []

# Iterate through the data using the sliding window
for i in range(0, len(data) - window_size + 1, 1):
    window_data = data.iloc[i:i+window_size]

    # Calculate mean, variance, and standard
deviation for the window

mean_values.append(window_data['Trunk_horizontal_late
ral'].mean())

variance_values.append(window_data['Trunk_horizontal_
lateral'].var())

std_deviation_values.append(window_data['Trunk_horizo
ntal_lateral'].std())

list1=[mean_values[len(mean_values)-1]]*(len(data)-
len(mean_values))
list2=[variance_values[len(variance_values)-
1]]*(len(data)-len(mean_values))
list3=[std_deviation_values[len(std_deviation_values)
-1]]*(len(data)-len(mean_values))
mean_values.extend(list1)
variance_values.extend(list2)
std_deviation_values.extend(list3)
""" Add the calculated features to the DataFrame"""
data['Trunk_horizontal_lateral_Windowed_Mean'] =
mean_values
data['Trunk_horizontal_lateral_Windowed_Variance'] =
variance_values
data['Trunk_horizontal_lateral_Windowed_Std_Deviation
'] = std_deviation_values

# Save DataFrame as CSV on Google Drive
drive_path = '/content/drive/MyDrive/'  # Path to
your Google Drive root
csv_filename = 'parkinson_fog_dataframe_drive.csv'
data.to_csv(drive_path + csv_filename, index=False)
```

```
dataframe_from_drive =
pd.read_csv("/content/drive/MyDrive/parkinson_fog_dat
aframe_drive.csv")
print(dataframe_from_drive.shape)
## Read CSV file into DataFrame
#df = pd.read_csv(file_path)

acc_magnitude_upper_leg =
np.sqrt(dataframe_from_drive['Upper_leg_horizontal_fo
rward']**2 +
dataframe_from_drive['Upper_leg_vertical']**2 +
dataframe_from_drive['Upper_leg_horizontal_lateral']*
*2)

dataframe_from_drive["Acc_magnitude_upper_leg"]=acc_m
agnitude_upper_leg

acc_magnitude_trunk =
np.sqrt(dataframe_from_drive['Trunk_horizontal_forwar
d']**2 + dataframe_from_drive['Trunk_vertical']**2 +
dataframe_from_drive['Trunk_horizontal_lateral']**2)

dataframe_from_drive["Acc_magnitude_trunk"]=acc_magni
tude_trunk

# Count occurrences where the value in the specified
column is zero
count_zeros = (dataframe_from_drive['Acc_magnitude']
== 0).sum()
print(count_zeros)

# Filter the DataFrame based on the condition and
print 'Annotation' values
filtered_annotations =
dataframe_from_drive.loc[dataframe_from_drive['Acc_ma
gnitude'] == 0, 'Annotation']

print("Annotation values where Acc_magnitude is
zero:")
print(filtered_annotations)
WE CAN SAFELY DROP THE ROWS WHERE ANKLE ACCELERATION
MAGNITUDE IS ZERO
```

AS IT IS A PART OF DEBREIFING OF THE PATIENT I.E CLASS 0 (WHICH IS NOT STRICTLY A PART OF THE EXPERIMENT)

```
# Drop rows where 'Acc_magnitude' is zero
dataframe_from_drive =
dataframe_from_drive[dataframe_from_drive['Acc_magnit
ude'] != 0]

# Reset the index to consecutive values
dataframe_from_drive =
dataframe_from_drive.reset_index(drop=True)


# Count occurrences where the value in the specified
column is zero
count_zeros = (dataframe_from_drive['Acc_magnitude']
== 0).sum()
print(count_zeros)
```

Next feature : angle between two consecutive acceleration vectors (to see how acceleration vector is turning)
cosine theta = dot product of the acceleration vectors at i and i+1 time/ magnitude of acceleration at i * magnitude of acceleration at i+1

```
import math
list_ankle_theta=[]
list_upper_leg_theta=[]
list_trunk_theta=[]
for i in range(0,len(dataframe_from_drive)-1):
#for i in range(0,100):

x=((dataframe_from_drive['Ankle_horizontal_forward'][
i]*dataframe_from_drive['Ankle_horizontal_forward'][i
+1])+(dataframe_from_drive['Ankle_vertical'][i]*dataf
rame_from_drive['Ankle_vertical'][i+1])+(dataframe_fr
om_drive['Ankle_horizontal_lateral'][i]*dataframe_fro
m_drive['Ankle_horizontal_lateral'][i+1]))/(dataframe
_from_drive['Acc_magnitude'][i]*dataframe_from_drive[
'Acc_magnitude'][i+1])
```

```
  #now x represents cosine theta so taking inverse of
that to get theta
  #print(x)
  if(x>1.0):
    x=1.0
  if(x<-1.0):
    x=-1.0
  theta= math.acos(x)
  #insert this in dataframe -> represents angle
between i and i+1 acceleration vector
  list_ankle_theta.append(theta)
"""

x1=((dataframe_from_drive['Upper_leg_horizontal_forwa
rd'][i]*dataframe_from_drive['Upper_leg_horizontal_fo
rward'][i+1])+(dataframe_from_drive['Upper_leg_vertic
al'][i]*dataframe_from_drive['Upper_leg_vertical'][i+
1])+(dataframe_from_drive['Upper_leg_horizontal_later
al'][i]*dataframe_from_drive['Upper_leg_horizontal_la
teral'][i+1]))/(dataframe_from_drive['Acc_magnitude_u
pper_leg'][i]*dataframe_from_drive['Acc_magnitude_upp
er_leg'][i+1])
  #now x1 represents cosine theta so taking inverse
of that to get theta
  if(x1>1.0):
    x1=1.0
  if(x1<-1.0):
    x1=-1.0
  theta1= math.acos(x1)
  #insert this in dataframe -> represents angle
between i and i+1 acceleration vector
  list_upper_leg_theta.append(theta1)

x2=((dataframe_from_drive['Trunk_horizontal_forward']
[i]*dataframe_from_drive['Trunk_horizontal_forward'][
i+1])+(dataframe_from_drive['Trunk_vertical'][i]*data
frame_from_drive['Trunk_vertical'][i+1])+(dataframe_f
rom_drive['Trunk_horizontal_lateral'][i]*dataframe_fr
om_drive['Trunk_horizontal_lateral'][i+1]))/(datafram
e_from_drive['Acc_magnitude_trunk'][i]*dataframe_from
_drive['Acc_magnitude_trunk'][i+1])
  #now x2 represents cosine theta so taking inverse
of that to get theta
  if(x2>1.0):
```

```
    x2=1.0
  if(x2<-1.0):
    x2=-1.0
  theta2= math.acos(x2)
  #insert this in dataframe -> represents angle
between i and i+1 acceleration vector
  list_trunk_theta.append(theta2)
"""
```

Drop the last row and add the series/list_ankle_theta to the dataframe

```
# Drop the last row
dataframe_from_drive =
dataframe_from_drive.drop(dataframe_from_drive.index[
-1])


dataframe_from_drive['Ankle_Vector_Angle']=list_ankle
_theta

# Save DataFrame as CSV on Google Drive
drive_path = '/content/drive/MyDrive/'  # Path to
your Google Drive root
csv_filename =
'parkinson_fog_dataframe_saturday26august.csv'
dataframe_from_drive.to_csv(drive_path +
csv_filename, index=False)
```

SOURCE CODE (LSTM MODEL)

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.optimizers import Adam
from keras.utils import to_categorical
from scipy import signal
from keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')

dataframe_from_drive =
pd.read_csv("/content/drive/MyDrive/parkinson_fog_dat
aframe_saturday26august.csv")
print(dataframe_from_drive.shape)


# Drop specific columns
```

```
columns_to_drop = ["Time",
"Ankle_horizontal_forward", "Ankle_vertical",
"Ankle_horizontal_lateral",
"Upper_leg_horizontal_forward", "Upper_leg_vertical",
"Upper_leg_horizontal_lateral",
"Trunk_horizontal_forward", "Trunk_vertical",
"Trunk_horizontal_lateral"]
dataframe_from_drive =
dataframe_from_drive.drop(columns=columns_to_drop,
axis=1)
print(dataframe_from_drive.shape)

# Drop specific columns ----> Make X
columns_to_drop = ["Annotation"]
X =
dataframe_from_drive.drop(columns=columns_to_drop,
axis=1)
y=dataframe_from_drive[["Annotation"]]
print(type(y))
print(y.shape)
print(X.shape)




# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Convert annotations to one-hot encoding
y_encoded = to_categorical(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(X_scaled, y_encoded, test_size=0.3,
stratify=y, random_state=42)

# Reshape input data for LSTM (samples, time steps,
features)
n_timesteps=1
n_features = X_train.shape[1]  # Number of features
in your data
X_train_reshaped = X_train.reshape(X_train.shape[0],
n_timesteps, n_features)
```

```
X_test_reshaped = X_test.reshape(X_test.shape[0],
n_timesteps, n_features)

# Initialize the LSTM model
# Define the architecture
model = Sequential()

# Input layer
model.add(LSTM(128, input_shape=(n_timesteps,
n_features), activation='relu',
return_sequences=True))
model.add(Dropout(0.2))  # Adding dropout for
regularization

# Hidden layer
model.add(LSTM(64, activation='relu'))
model.add(Dropout(0.2))  # Adding dropout for
regularization

# Output layer
model.add(Dense(3, activation='softmax'))  # Assuming
3 classes (0, 1, 2)

# Compile the model
model.compile(loss='categorical_crossentropy',
optimizer=Adam(learning_rate=0.001),
metrics=['accuracy'])


# Define the early stopping callback
early_stopping = EarlyStopping(monitor='val_loss',
patience=5, restore_best_weights=True)

# Train the model
history=model.fit(X_train_reshaped, y_train,
epochs=25, batch_size=32,
validation_split=0.3,callbacks=[early_stopping])


# Assuming you've already trained the model and have
the 'history' object

# Plot training & validation loss values
plt.plot(history.history['loss'])
```

```
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper
right')
plt.show()

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='lower
right')
plt.show()


# Evaluate the model on the test data
loss, accuracy = model.evaluate(X_test_reshaped,
y_test)
print("Accuracy:", accuracy)
```