1. Data Preprocessing

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

```
Mounted at /content/drive
```

1.1 Import Data

```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
```

```
Mounted at /content/gdrive
```

```
!ls /content/gdrive/MyDrive/data_preprocessed_python
```

```
s01.dat  s05.dat  s09.dat  s13.dat  s17.dat  s21.dat  s25.dat  s29.dat
s02.dat  s06.dat  s10.dat  s14.dat  s18.dat  s22.dat  s26.dat  s30.dat
s03.dat  s07.dat  s11.dat  s15.dat  s19.dat  s23.dat  s27.dat  s31.dat
s04.dat  s08.dat  s12.dat  s16.dat  s20.dat  s24.dat  s28.dat  s32.dat
```

1.2 Reading Data

```
import os
import time
import pickle
import pandas as pd
import numpy as np

from scipy import signal
from scipy.signal import welch
from scipy.integrate import simps
from scipy.stats import f_oneway

from tqdm import tqdm

#!pip install scikit-learn==0.20.3
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn import model_selection
from sklearn.metrics import classification_report, confusion_matrix
import itertools
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import f1_score
from sklearn.metrics import precision_recall_curve

!pip install mne==0.22.0
import mne
from mne.preprocessing import (ICA, create_eog_epochs, create_ecg_epochs, corrmap)
from mne.time_frequency import psd_welch
from mne.decoding import cross_val_multiscore

!pip install fooof
from fooof import FOOOFGroup
from fooof.bands import Bands
from fooof.analysis import get_band_peak_fg
from fooof.plts.spectra import plot_spectrum

import matplotlib.pyplot as plt
from matplotlib import cm
import seaborn as sns
%matplotlib inline
```

```
Requirement already satisfied: mne==0.22.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: scipy>=0.17.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: fooof in /usr/local/lib/python3.7/dist-packages (1.0.
Requirement already satisfied: scipy>=0.19.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
```

```
with open('/content/gdrive/MyDrive/data_preprocessed_python/s01.dat', 'rb') as f:
 raw_data = pickle.load(f, encoding='latin1')
 print(raw_data)
```

```
          [ 5.98429831e-03,  5.98429831e-03,  5.98429831e-03, ...,
           -6.20148303e-02, -6.20148303e-02, -6.20148303e-02]],

         [[ 1.01304956e+00, -1.06783230e+00,  3.90824949e+00, ...,
            2.18687657e+00,  2.66767712e-02, -7.51193325e+00],
          [ 3.31725539e-02, -1.51860504e+00,  4.81957628e+00, ...,
            3.98227619e-01, -1.06766739e+00, -5.90302866e+00],
          [ 7.22816236e-01,  5.12160665e-01,  8.04440282e+00, ...,
            9.66963103e-02,  2.20454025e-01, -4.36041903e+00],
          ...,
          [-2.14800222e+01, -2.61792078e+02, -1.01386125e+02, ...,
            2.58992140e+03,  1.61176696e+03,  7.66674772e+02],
          [ 1.16210735e+04,  1.14355295e+04,  1.12646336e+04, ...,
            2.76134490e+03,  2.67687353e+03,  2.60314418e+03],
          [ 2.90621276e-03,  2.90621276e-03,  2.90621276e-03, ...,
           -1.51091814e-01, -1.51091814e-01, -1.51091814e-01]],

         ...,

         [[-1.13944132e+01, -1.34502274e+01, -9.66299865e+00, ...,
           -2.43900889e+00, -2.33095827e+00, -3.36814485e+00],
          [-9.19759768e+00, -1.04656392e+01, -6.56847604e+00, ...,

           -3.51219647e+00, -2.91042815e+00, -2.82260677e+00],
          [-8.61617511e+00, -1.00405069e+01, -6.40832041e+00, ...,
           -4.15937090e+00, -4.30358260e+00, -2.69228055e+00],
```

```
        ...,
        [ 1.33536977e+03,  1.18468254e+03,  9.23995526e+02, ...,
          9.61057957e+02,  5.48652469e+02,  1.16462008e+03],
        [-3.84721885e+03, -3.90385861e+03, -3.96391629e+03, ...,
          2.21035586e+04,  2.19683067e+04,  2.18359845e+04],
        [ 1.74997757e-03,  1.74997757e-03,  1.74997757e-03, ...,
          4.67494009e-02,  4.67494009e-02,  4.67494009e-02]],

       [[ 2.53187763e+00,  1.58111089e-01, -4.67449746e+00, ...,
          3.13209243e+00,  4.65647663e-01,  1.85004049e+00],
        [ 6.21303506e+00,  1.60022210e+00, -4.68115214e+00, ...,
          3.09468204e+00, -4.47081977e-01,  7.07338119e-01],
        [ 7.13694330e+00,  4.83688756e+00, -2.90448250e+00, ...,
          8.47561300e-02, -7.15406930e-02, -8.34269599e-01],
        ...,
        [-1.72646702e+03, -1.77037319e+03, -1.74206075e+03, ...,
         -1.37621767e+03, -1.46368626e+03, -1.39456139e+03],
        [ 7.78738138e+03,  7.87087621e+03,  7.88894234e+03, ...,
          1.77575401e+03,  1.77087127e+03,  1.76891817e+03],
        [ 1.30206665e-03,  2.30205383e-03,  2.30205383e-03, ...,
         -1.06977796e-02, -1.06977796e-02, -1.06977796e-02]],

       [[ 3.08300605e+00,  6.27214597e-01, -3.40256017e+00, ...,
         -4.43802092e+00, -3.27103236e+00,  2.13500861e+00],
        [-2.73052705e-01, -2.20023500e-01, -3.81054293e+00, ...,
         -3.75791147e+00, -3.31224301e+00,  1.15426073e+00],
        [-1.72910530e+00,  1.73630179e+00, -1.14991197e-01, ...,
         -2.94468621e+00, -5.71142735e+00, -1.69792350e+00],
        ...,
        [-1.17067329e+02,  2.14775808e+02,  6.61994220e+00, ...,
         -1.10328426e+03, -1.11556548e+03, -1.07090932e+03],
        [-7.16243920e+03, -7.22005551e+03, -7.17659914e+03, ...,
          4.80999311e+02,  3.31587520e+02,  1.82175728e+02],
```

```
data = raw_data['data']
labels = raw_data['labels']

print("Labels: ", labels.shape) # trial x label
print("Data: ", data.shape) # trial x channel x data
```

```
Labels:  (40, 4)
Data:  (40, 40, 8064)
```

```
em_labels = []
for i in range(0, labels.shape[0]):
  if (labels[i][0]>5): #high valence
    if(labels[i][1]>5): # high arousal
      em_labels.append(1) # HVHA
    else:
      em_labels.append(0) # HVLA
  else: # low valence
    if(labels[i][1]>5): # high arousal
      em_labels.append(2) #LVHA
    else:
      em_labels.append(3) # LVLA


print(em_labels)
```

```
[1, 1, 1, 2, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 2, 1, 2, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 3,
```

1.3 Reducing the Raw Data upto 40 32 7680

```
reduced_eeg_data  = data[0:40, 0:32, 384:8064]
reduced_eeg_data.shape
#print(type(reduced_eeg_data))
print(reduced_eeg_data)
```

```
  -5.42113073e+00 -9.12074717e+00]]

 [[-1.51764991e+00  3.33769448e+00  5.33411583e+00 ...  2.18687657e+00
    2.66767712e-02 -7.51193325e+00]
  [-1.22945968e+00  3.94136879e+00  6.03334581e+00 ...  3.98227619e-01
   -1.06766739e+00 -5.90302866e+00]
  [ 2.44943001e+00  5.19001842e+00  7.20289597e+00 ...  9.66963103e-02
    2.20454025e-01 -4.36041903e+00]
  ...
  [-1.04758258e+01 -9.93038403e+00 -7.82450628e+00 ...  2.35138961e+00
    8.75618636e-01  4.95425282e+00]
  [-6.77643788e+00 -9.13021385e+00 -9.21009118e+00 ...  1.35514060e+00
    3.46652206e+00  5.29448848e+00]
  [-9.79636464e+00 -1.15126955e+01 -1.09249503e+01 ...  6.34267637e+00
    7.34487869e+00  8.89437967e+00]]

 ...


 [[ 3.14642879e+00  7.93115700e-01  2.16631010e+00 ... -2.43900889e+00
   -2.33095827e+00 -3.36814485e+00]
  [ 9.64812279e-01  1.87187192e+00  2.38790148e+00 ... -3.51219647e+00
   -2.91042815e+00 -2.82260677e+00]
  [ 2.73065885e+00  2.68167511e+00  5.13581833e-01 ... -4.15937090e+00
   -4.30358260e+00 -2.69228055e+00]
  ...
  [ 1.87543230e+00 -5.41164025e-01 -2.30159121e+00 ...  1.06559658e+00
    1.83206361e+00  5.57754002e-01]
  [-3.11105144e+00 -5.12884654e+00 -9.04235539e+00 ...  1.33300259e+00
   -9.52507204e-01 -2.58667480e+00]
  [-4.38154899e+00 -6.05651571e+00 -9.98596478e+00 ...  1.30680417e+00
   -8.32571575e-01 -1.14356339e+00]]

 [[ 7.76239493e+00  7.36342358e+00  5.90754815e+00 ...  3.13209243e+00
    4.65647663e-01  1.85004049e+00]
  [ 9.09619591e+00  9.23566658e+00  6.64979102e+00 ...  3.09468204e+00
   -4.47081977e-01  7.07338119e-01]
  [ 8.67286805e+00  8.00817148e+00  8.39455326e+00 ...  8.47561300e-02
   -7.15406930e-02 -8.34269599e-01]
  ...
  [ 2.47438436e+00 -6.93148928e+00 -1.31281076e+01 ...  6.89017349e+00
    6.08268226e+00  1.47206475e+00]
  [-5.70594038e+00 -1.27253802e+01 -1.96668036e+01 ...  1.55477019e+00
    3.10542161e+00  5.35199233e+00]
```

```
  [-3.63494363e+00 -9.10573738e+00 -1.33531587e+01 ...  9.94502073e-01
    4.14792269e+00  6.03725304e+00]]

 [[-1.87877447e-01 -1.49636478e+00  1.38064172e-02 ... -4.43802092e+00
   -3.27103236e+00  2.13500861e+00]
  [ 1.04575048e+00  5.34701060e-02 -6.64393801e-02 ... -3.75791147e+00
   -3.31224301e+00  1.15426073e+00]
  [-1.63974327e-02 -1.04686938e+00  2.03125832e-01 ... -2.94468621e+00
   -5.71142735e+00 -1.69792350e+00]
  ...
  [-1.44771172e+00  1.01967246e+00  2.61434503e+00 ...  2.44461576e+00
    3.67915943e-01 -3.40510722e+00]
  [ 2.56835707e-01  1.72619127e+00 -1.50182478e-01 ...  3.34613039e+00
    2.29806139e+00 -9.18429156e-01]
  [-2.21610149e+00 -7.27286055e-01 -2.37324433e+00 ...  1.96733159e+00
    2.57801114e+00 -6.09354349e-01]]]
```

1.4 Discretize the labels Rating 1-5: as low valence /low arousal Rating 6-9: high valence/high arousal

```
# Only extract Valence and Arousal ratings
df_label_ratings = pd.DataFrame({'Valence': labels[:,0], 'Arousal': labels[:,1]})
print(df_label_ratings.describe())
```

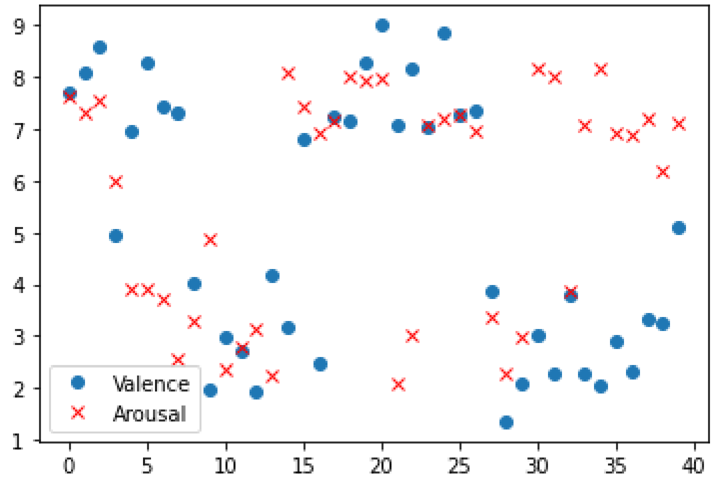
```
        Valence    Arousal
count  40.00000  40.000000
mean    5.11775   5.659750
std     2.51712   2.179549
min     1.36000   2.080000
25%     2.85250   3.335000
50%     4.56000   6.915000
75%     7.32750   7.342500
max     9.00000   8.150000
```

```
print(df_label_ratings.head(15))
```

```
    Valence  Arousal
0      7.71     7.60
1      8.10     7.31
2      8.58     7.54
3      4.94     6.01
4      6.96     3.92
5      8.27     3.92
6      7.44     3.73
7      7.32     2.55
8      4.04     3.29
9      1.99     4.86
10     2.99     2.36
11     2.71     2.77
12     1.95     3.12
13     4.18     2.24
14     3.17     8.08
```

```
# Plot the first 40 data rows (first participant)
df_label_ratings.iloc[0:40].plot(style=['o','rx'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbda1e0b510>
```




```
# High Arousal Positive Valence dataset
df_hahv = df_label_ratings[(df_label_ratings['Valence'] >= np.median(labels[:,0])) & (df_l
# Low Arousal Positive Valence dataset
df_lahv = df_label_ratings[(df_label_ratings['Valence'] >= np.median(labels[:,0])) & (df_l
# High Arousal Negative Valence dataset
df_halv = df_label_ratings[(df_label_ratings['Valence'] < np.median(labels[:,0])) & (df_la
# Low Arousal Negative Valence dataset
df_lalv = df_label_ratings[(df_label_ratings['Valence'] < np.median(labels[:,0])) & (df_la
```

```
# Check nummber of trials per each group
print("Positive Valence:", str(len(df_hahv) + len(df_lahv)))
print("Negative Valence:", str(len(df_halv) + len(df_lalv)))
print("High Arousal:", str(len(df_hahv) + len(df_halv)))
print("Low Arousal:", str(len(df_lahv) + len(df_lalv)))
```

```
Positive Valence: 20
Negative Valence: 20
High Arousal: 20
Low Arousal: 20
```

```
# Check nummber of trials per each group
print("High Arousal Positive Valence:", str(len(df_hahv)))
print("Low Arousal Positive Valence:", str(len(df_lahv)))
print("High Arousal Negative Valence:", str(len(df_halv)))
print("Low Arousal Negative Valence:", str(len(df_lalv)))
```

```
High Arousal Positive Valence: 13
Low Arousal Positive Valence: 7
High Arousal Negative Valence: 7
Low Arousal Negative Valence: 13
```

```
# Get mean and std of each group
print("HAHV")
print("Valence:", "Mean", np.round(df_hahv['Valence'].mean(),2), "STD", np.round(df_hahv['
print("Arousal:", "Mean", np.round(df_hahv['Arousal'].mean(),2), "STD", np.round(df_hahv['
print()
print("LAHV:")
print("Valence:", "Mean", np.round(df_lahv['Valence'].mean(),2), "STD", np.round(df_lahv['
```

```
print("Arousal:", "Mean", np.round(df_lahv['Arousal'].mean(),2), "STD", np.round(df_lahv['
print()
print("HALV:")
print("Valence:", "Mean", np.round(df_halv['Valence'].mean(),2), "STD", np.round(df_halv['
print("Arousal:", "Mean", np.round(df_halv['Arousal'].mean(),2), "STD", np.round(df_halv['
print()
print("LALV:")
print("Valence:", "Mean", np.round(df_lalv['Valence'].mean(),2), "STD", np.round(df_lalv['
print("Arousal:", "Mean", np.round(df_lalv['Arousal'].mean(),2), "STD", np.round(df_lalv['
```

```
HAHV
Valence: Mean 7.58 STD 1.04
Arousal: Mean 7.43 STD 0.35

LAHV:
Valence: Mean 7.17 STD 1.1
Arousal: Mean 3.6 STD 1.28

HALV:
Valence: Mean 2.72 STD 0.5
Arousal: Mean 7.65 STD 0.56

LALV:
Valence: Mean 2.85 STD 0.92
Arousal: Mean 3.93 STD 1.71
```

```
# Function to check if each trial has positive or negative valence
def positive_valence(trial):
    return 1 if labels[trial,0] >= np.median(labels[:,0]) else 0
# Function to check if each trial has high or low arousal
def high_arousal(trial):
    return 1 if labels[trial,1] >= np.median(labels[:,1]) else 0


# Convert all ratings to boolean values
labels_encoded = []
for i in range (len(labels)):
  labels_encoded.append([positive_valence(i), high_arousal(i)])
labels_encoded = np.reshape(labels_encoded, (40, 2))
df_labels = pd.DataFrame(data=labels_encoded, columns=["Positive Valence", "High Arousal"]
print(df_labels.describe())
```

```
       Positive Valence  High Arousal
count          40.00000      40.00000
mean            0.50000       0.50000
std             0.50637       0.50637
min             0.00000       0.00000
25%             0.00000       0.00000
50%             0.50000       0.50000
75%             1.00000       1.00000
max             1.00000       1.00000
```

```
# Dataset with only Valence column
df_valence = df_labels['Positive Valence']
# Dataset with only Arousal column
df_arousal = df_labels['High Arousal']
```

1.5 Filtering Data Through Butterworth Bandpass Filter

1.6 Removal of artifacts using Independent Component Analysis(ICA)

```
#!/usr/bin/env python
import pickle
import numpy as np
from scipy.signal import butter, lfilter, sosfilt, sosfreqz, freqz
from sklearn.decomposition import FastICA

def butter_bandpass(lowcut, highcut, fs, order = 3):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band', analog=False)
    return b, a
def butter_bandpass_filter(data, lowcut, highcut, fs, order = 5):
    b, a = butter_bandpass(lowcut, highcut, fs, order = order)
    y = lfilter(b, a, data)
    return y
def eye_movement_artifact(shyam): # parameter must be an 2D array like 32_channels*7860_da
    # Inverse that 2D array
    shyam = shyam.transpose()
    ica = FastICA(n_components = 32, random_state = 0, tol = 0.05)
    comps = ica.fit_transform(shyam)
    # invert the array
    data_after = comps.transpose()
    return data_after
def signal_pro(data):
    fs=128
    lowcut=0.5
    highcut=45
    mean_value = 0
    # do the bandpass filter
    for i in range(40):
        for j in range(32):
            data[i][j] = butter_bandpass_filter(data[i][j], lowcut, highcut, fs, order=5)
    # creating dummy variable which contains same data information
    error_eye =  np.zeros((40,32,7680))
    new_data =  np.zeros((40,32,7680))
    for i in range(40):
        for j in range(32):
            for k in range(7680):
                #print(data[i][j][k])
                error_eye[i][j][k] = data[i][j][k]
                new_data[i][j][k] = data[i][j][k]
    for i in range(40):
        error_eye[i] = eye_movement_artifact(error_eye[i])
    for i in range(40):
        for j in range(32):
```

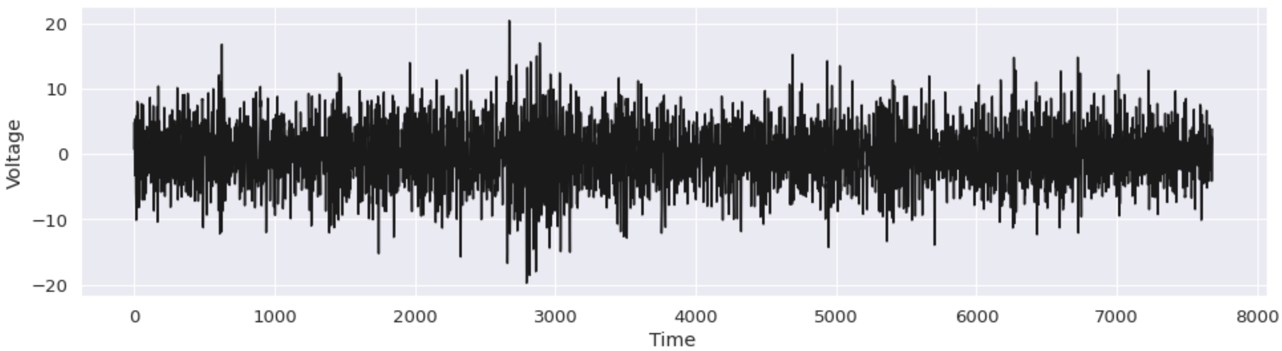
```
            mean_value = np.mean(data[i][j])
            for k in range(7680):
                if(data[i][j][k]>0.0): # data is positive
                    if(mean_value>0.0): # error is positive
                        new_data[i][j][k] = data[i][j][k] - mean_value
                    elif(mean_value<0.0): # error is negative
                        new_data[i][j][k] = data[i][j][k] - abs(mean_value)
                elif(data[i][j][k]<0.0): # data is negative
                    if(mean_value>0.0): # error is positive
                        new_data[i][j][k] = data[i][j][k] + mean_value
                    elif(mean_value<0.0): # error is negative
                        new_data[i][j][k] = data[i][j][k] - mean_value
    return new_data

fname = '/content/gdrive/MyDrive/data_preprocessed_python/s01.dat'
x = pickle.load(open(fname, "rb"), encoding="latin1")
labels = x['labels']
data = x['data']
data = data[0:40 , 0:32 , 384:8064]
filter_data = signal_pro(data)
print(np.shape(filter_data))
print(filter_data)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/decomposition/_fastica.py:119: Conv
  ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/decomposition/_fastica.py:119: Conv
  ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/decomposition/_fastica.py:119: Conv
  ConvergenceWarning)
(40, 32, 7680)
[[[ 7.95648528e-01  2.89501356e+00  2.61935760e+00 ...  1.39535332e+00
   -3.99159442e+00 -5.33963190e+00]
  [ 5.99469265e-01  2.56429761e+00  3.35377849e+00 ...  3.13934281e+00
   -4.01302308e+00 -7.55416031e+00]
  [ 2.32549631e-01  1.58746963e+00  3.19184223e+00 ...  3.15431765e+00
   -4.35828486e+00 -8.80256423e+00]
  ...
  [ 5.38170240e-01  2.19210252e+00  3.58815943e+00 ...  2.77831240e+00
    7.62529741e+00  7.26733037e+00]
  [ 5.52265355e-01  2.24286773e+00  3.78882258e+00 ... -1.94641285e+00
    3.79207073e+00  7.29873267e+00]
  [ 7.19570848e-01  3.02262303e+00  4.84982576e+00 ... -1.53293283e+00
    3.46835442e+00  3.81903014e+00]]

 [[-8.88176185e-01 -4.95794362e+00 -1.08714171e+01 ...  3.54493852e+00
    8.13599910e+00  8.80617996e+00]
  [-1.24602436e-01 -1.28229620e+00 -4.31482625e+00 ...  4.09423238e+00
    7.43010495e+00  9.32669398e+00]
  [ 2.89096823e-01  5.01637472e-01 -1.52059829e+00 ...  4.51104536e+00
    7.50199258e+00  7.39482773e+00]
  ...
  [ 5.55133036e-01  2.31396661e+00  4.04759845e+00 ...  3.01578755e-01
   -1.62942297e+00  2.18471910e-01]
  [-4.46262033e-01 -1.09192152e+00  1.01075961e+00 ...  3.16788338e-02
   -6.40617911e+00 -1.02001412e+01]
  [-7.11053581e-02  7.32993959e-01  4.27759995e+00 ...  1.47149985e+00
   -3.95870724e+00 -7.41675373e+00]]
```

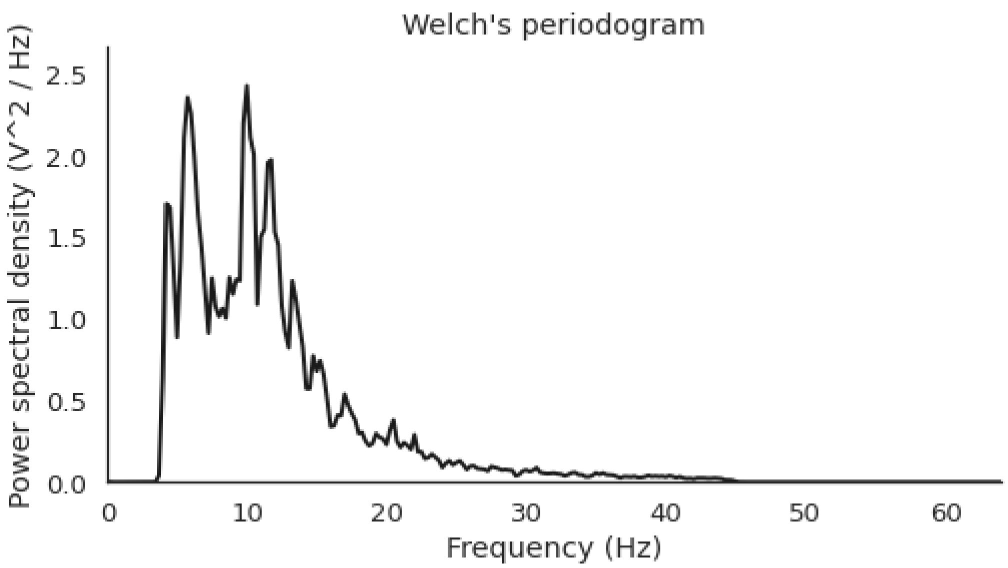
```
[[-2.96784316e-01 -2.14905693e-01  2.47327829e+00 ...  4.15073831e+00
   2.52150209e-02 -2.77493066e+00]
 [-2.40650274e-01  6.71000290e-02  3.05084454e+00 ...  2.88273757e+00
  -7.21795353e-01 -3.63007808e+00]
 [ 4.80269913e-01  2.42533944e+00  5.18563922e+00 ...  1.11801197e+00
  -7.57386588e-01 -1.91287958e+00]
 ...
 [-2.05678537e+00 -7.96650455e+00 -1.06222838e+01 ...  7.29953317e-01
   3.73816677e+00  1.65137802e+00]
 [-1.33060508e+00 -5.68494505e+00 -9.24063874e+00 ... -8.82285611e-01
   1.02542049e+00  5.04191558e+00]
 [-1.92255548e+00 -7.88621569e+00 -1.19198055e+01 ...  2.06323457e+00
   6.53925159e+00  8.68652354e+00]]

...

[[ 6.16925983e-01  1.96192152e+00  1.89594716e+00 ... -2.47892225e+00
  -3.45968776e+00 -1.52648586e+00]
 [ 1.88193078e-01  9.20476214e-01  1.85433072e+00 ... -3.15085076e+00
  -4.14932877e+00 -1.94315392e+00]
 [ 5.34244224e-01  2.09299084e+00  2.52078864e+00 ... -2.85599452e+00
  -4.40596060e+00 -2.58720358e+00]
```

2. Feature Extraction

```
sns.set(font_scale=1.2)

# Define sampling frequency and time vector
sf = 128.
time = np.arange(filter_data.size) / sf

# Plot the signal of first trial, last channel
fig, ax = plt.subplots(1, 1, figsize=(16, 4))
plt.plot(filter_data[0,31], lw=1.5, color='k')
plt.xlabel('Time')
plt.ylabel('Voltage')
sns.despine()
```

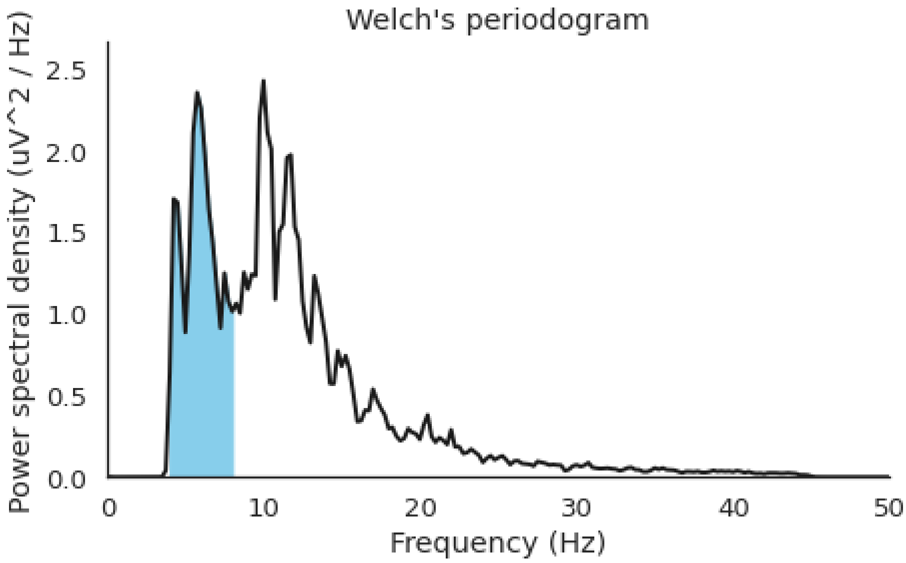


```
# Define window length (4 seconds)
win = 4 * sf
freqs, psd = signal.welch(filter_data[0,31], sf, nperseg=win)

#print(freqs.shape, psd.shape)  # psd has shape (n_channels, n_frequencies)
# Plot the power spectrum
sns.set(font_scale=1.2, style='white')
plt.figure(figsize=(8, 4))
plt.plot(freqs, psd, color='k', lw=2)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power spectral density (V^2 / Hz)')
plt.ylim([0, psd.max() * 1.1])
plt.title("Welch's periodogram")
plt.xlim([0, freqs.max()])
sns.despine()
```



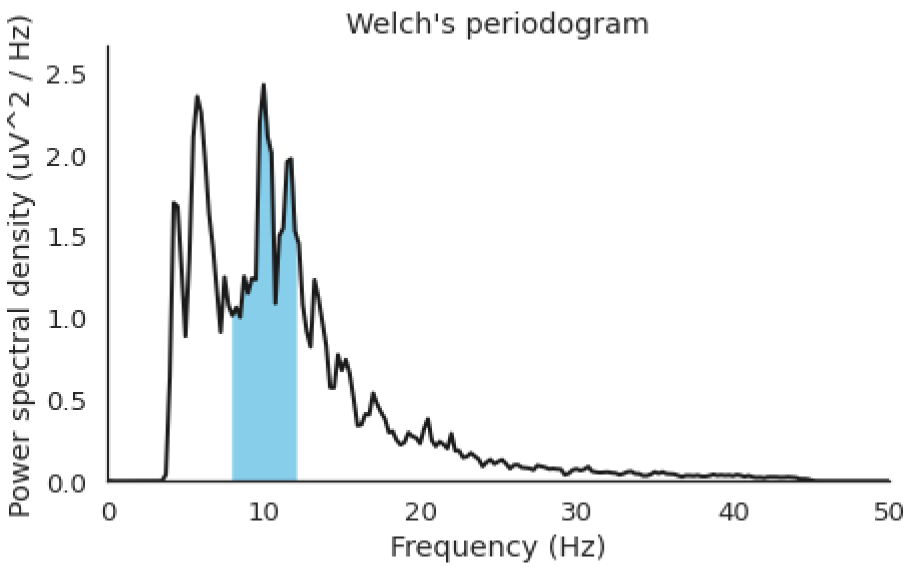
2.1 Theta Band Power(4-8)Hz

```
# Define delta lower and upper limits
low, high = 4, 8

# Find intersecting values in frequency vector
idx_theta = np.logical_and(freqs >= low, freqs <= high)

# Plot the power spectral density and fill the delta area
plt.figure(figsize=(7, 4))
plt.plot(freqs, psd, lw=2, color='k')
plt.fill_between(freqs, psd, where=idx_theta, color='skyblue')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power spectral density (uV^2 / Hz)')
plt.xlim([0, 50])
plt.ylim([0, psd.max() * 1.1])
plt.title("Welch's periodogram")
sns.despine()
#print(freqs.shape, psd.shape)  # psd has shape (n_channels, n_frequencies)
```



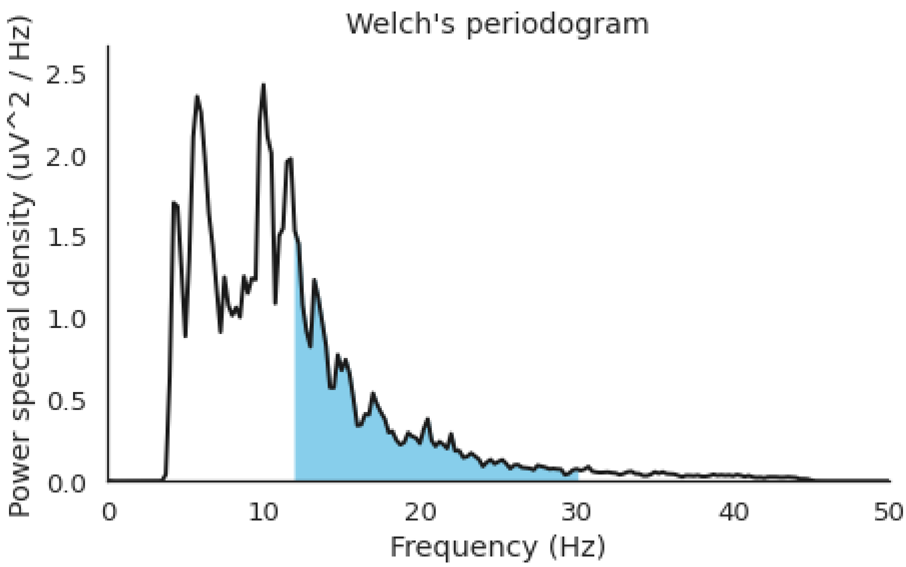
2.3 Alpha Band Power(8-12)Hz

```
# Define delta lower and upper limits
low, high = 8, 12

# Find intersecting values in frequency vector
idx_alpha = np.logical_and(freqs >= low, freqs <= high)

# Plot the power spectral density and fill the alpha area
plt.figure(figsize=(7, 4))
plt.plot(freqs, psd, lw=2, color='k')
plt.fill_between(freqs, psd, where=idx_alpha, color='skyblue')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power spectral density (uV^2 / Hz)')
plt.xlim([0, 50])
plt.ylim([0, psd.max() * 1.1])
plt.title("Welch's periodogram")
sns.despine()
```



2.4 Beta Band Power(12-30)Hz

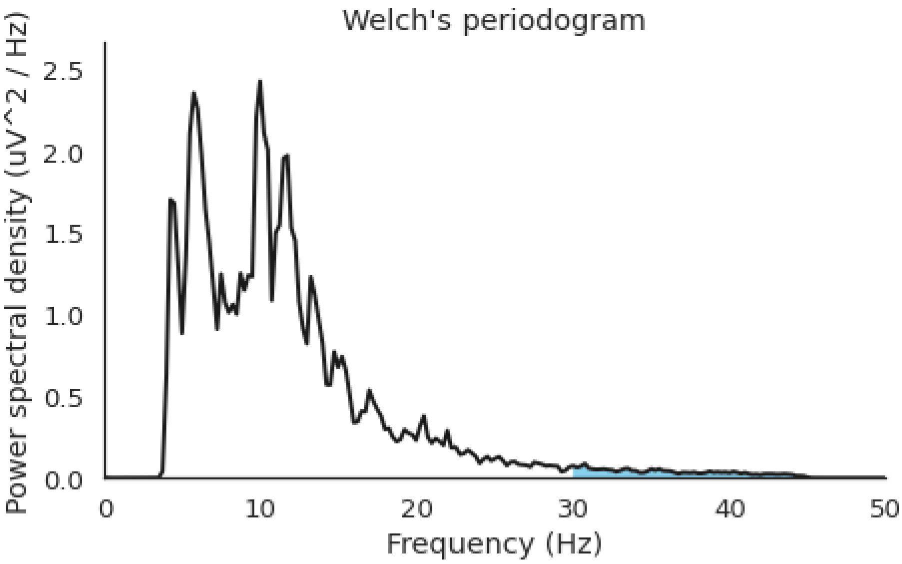
```
# Define beta lower and upper limits
low, high = 12, 30

# Find intersecting values in frequency vector
idx_beta = np.logical_and(freqs >= low, freqs <= high)

# Plot the power spectral density and fill the beta area
plt.figure(figsize=(7, 4))
plt.plot(freqs, psd, lw=2, color='k')
plt.fill_between(freqs, psd, where=idx_beta, color='skyblue')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power spectral density (uV^2 / Hz)')
plt.xlim([0, 50])
plt.ylim([0, psd.max() * 1.1])
plt.title("Welch's periodogram")
sns.despine()
```



2.5 Gamma Band Power(30-45)Hz

```
# Define gamma lower and upper limits
low, high = 30, 45

# Find intersecting values in frequency vector
idx_gamma = np.logical_and(freqs >= low, freqs <= high)

# Plot the power spectral density and fill the gamma area
plt.figure(figsize=(7, 4))
plt.plot(freqs, psd, lw=2, color='k')
plt.fill_between(freqs, psd, where=idx_gamma, color='skyblue')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power spectral density (uV^2 / Hz)')
plt.xlim([0,50])
plt.ylim([0, psd.max() * 1.1])
```

```
plt.title("Welch's periodogram")
sns.despine()
```




```
"""Compute the average power of the signal x in a specific frequency band.

    Parameters
    ----------
    data : 1d-array
        Input signal in the time-domain.
    sf : float
        Sampling frequency of the data.
    band : list
        Lower and upper frequencies of the band of interest.
    window_sec : float
        Length of each window in seconds.
        If None, window_sec = (1 / min(band)) * 2
    relative : boolean
        If True, return the relative power (= divided by the total power of the signal).
        If False (default), return the absolute power.

    Return
    ------
    bp : float
        Absolute or relative band power.
    """
def bandpower(data, sf, band, window_sec=None, relative=False):
    band = np.asarray(band)
    low, high = band

    # Define window length
    if window_sec is not None:
        nperseg = window_sec * sf
    else:
        nperseg = (2 / low) * sf

    # Compute the modified periodogram (Welch)
    freqs, psd = welch(data, sf, nperseg=nperseg)

    # Frequency resolution
```

```
        freq_res = freqs[1] - freqs[0]

        # Find closest indices of band in frequency vector
        idx_band = np.logical_and(freqs >= low, freqs <= high)

        # Integral approximation of the spectrum using Simpson's rule.
        bp = simps(psd[idx_band], dx=freq_res)

        if relative:
            bp /= simps(psd, dx=freq_res)
        return bp

    def get_band_power(trial, channel, band):
      bd = (0,0)

      if (band == "theta"): # drownsiness, emotional connection, intuition, creativity
        bd = (4,8)
      elif (band == "alpha"): # reflection, relaxation
        bd = (8,12)
      elif (band == "beta"): # concentration, problem solving, memory
        bd = (12,30)
      elif (band == "gamma"): # cognition, perception, learning, multi-tasking
        bd = (30,48)

      return bandpower(filter_data[trial,channel], 128, bd)

    print(get_band_power(0,31,"theta"))
    print(get_band_power(0,31,"alpha"))
    print(get_band_power(0,31,"beta"))
    print(get_band_power(0,31,"gamma"))
```

```
5.521876938791159
5.3704511273706395
6.202063672937247
1.0958763000706127
```

```
    eeg_channels = np.array(["Fp1", "AF3", "F3", "F7", "FC5", "FC1", "C3", "T7", "CP5", "CP1",
    peripheral_channels = np.array(["hEOG", "vEOG", "zEMG", "tEMG", "GSR", "Respiration belt",


    # Transform 40 x 32 x 7680 => 40 x 128
    eeg_band_arr = []
    for i in range (len(filter_data)):
      for j in range (len(filter_data[0])):
        eeg_band_arr.append(get_band_power(i,j,"theta"))
        eeg_band_arr.append(get_band_power(i,j,"alpha"))
        eeg_band_arr.append(get_band_power(i,j,"beta"))
        eeg_band_arr.append(get_band_power(i,j,"gamma"))
    eeg_band_arr = np.reshape(eeg_band_arr, (40, 128))


    left = np.array(["Fp1", "AF3", "F7", "FC5", "T7"])
    right = np.array(["Fp2", "AF4", "F8", "FC6", "T8"])
    frontal = np.array(["F3", "FC1", "Fz", "F4", "FC2"])
    parietal = np.array(["P3", "P7", "Pz", "P4", "P8"])
```

```
occipital = np.array(["O1", "Oz", "O2", "PO3", "PO4"])
central = np.array(["CP5", "CP1", "Cz", "C4", "C3", "CP6", "CP2"])
```

Dataframe for Theta power values

```
# Transform 40 x 32 x 7680 => 40 x 32
eeg_theta = []
for i in range (len(filter_data)):
  for j in range (len(filter_data[0])):
    eeg_theta.append(get_band_power(i,j,"theta"))
eeg_theta = np.reshape(eeg_theta, (40, 32))

df_theta = pd.DataFrame(data = eeg_theta, columns=eeg_channels)
theta_feature = (df_theta.describe())
print(theta_feature)
print(df_theta)
#data frame to CSV conversion
#theta_feature.to_csv('theta_feature.csv')
df_theta.to_csv('df_theta.csv')
#print(df_theta.describe())
#print(df_theta)
```

```
             Fp1        AF3        F3  ...         P8        PO4         O2
count  40.000000  40.000000  40.000000  ...  40.000000  40.000000  40.000000
mean    4.305472   5.823744   6.593334  ...   3.244052   7.898259   6.239054
std     0.926782   1.612571   1.867168  ...   0.910896   2.170938   1.457809
min     2.597929   2.947654   3.081066  ...   2.023093   3.641643   3.649540
25%     3.670649   4.590691   5.270173  ...   2.665397   6.342811   5.211397
50%     4.084414   5.605281   6.499234  ...   2.998740   7.419454   6.185236
75%     4.912856   6.887547   7.684318  ...   3.430199   9.407444   7.142219
max     6.453942  10.406575  11.543171  ...   6.419693  13.655592  10.356962

[8 rows x 32 columns]
         Fp1        AF3         F3  ...        P8        PO4         O2
0   4.045076   5.203156   5.961250  ...  2.902942   6.889217   5.521877
1   5.143294   7.048838   7.716178  ...  3.292742  10.013705   7.827210
2   4.673507   6.338758   7.163341  ...  3.125049   9.175377   6.630233
3   4.307117   6.193349   6.949383  ...  2.851142   7.944904   6.348572
4   2.597929   2.947654   3.081066  ...  2.187631   3.641643   3.649540
5   3.795476   4.571316   5.078570  ...  2.927377   7.204717   6.615830
6   4.068964   4.977074   5.429359  ...  2.661178   7.080222   5.239325
7   6.019001   8.142001   8.430279  ...  4.259537  10.174216   8.252348
8   6.422583   9.553065  10.722361  ...  4.740372  13.655592   9.491739
9   4.187269   5.207327   5.525896  ...  2.342736   7.251529   5.127611
10  4.927137   7.786849   9.133717  ...  3.472281  10.448939   6.136668
11  4.645741   5.784158   6.460921  ...  4.862409   6.451171   6.166275
12  4.774451   7.093040   8.445423  ...  2.987840   9.735694   6.264766
13  6.453942  10.406575  11.543171  ...  6.419693  11.830851  10.356962
14  4.048969   5.489338   5.767944  ...  3.416172   7.155874   6.657485
15  3.714454   4.491621   4.929171  ...  2.666674   6.136519   4.960623
16  3.383708   3.859078   4.309600  ...  2.692890   5.152050   4.869711
17  3.519180   4.597149   5.313904  ...  2.295503   6.454736   4.648324
18  3.948217   5.776391   6.813692  ...  3.044591   9.995619   6.226555
19  4.265771   5.969937   7.273101  ...  2.927889  10.051783   7.187147
20  4.909462   7.454451   9.228544  ...  2.947139  11.807826   7.515163
```

```
21  4.099864   5.483966   6.001196  ...  2.550975   7.546222   5.796858
22  3.931726   5.050962   5.940689  ...  3.266565   5.967485   5.475998
23  3.539233   4.730610   5.541762  ...  2.910874   6.411574   5.486542
24  5.058462   6.880121   7.360376  ...  4.592746   9.360324   7.276205
25  5.088548   6.758397   8.714246  ...  4.611145   8.518127   8.124890
26  4.923038   6.632065   7.673698  ...  4.175681   8.782656   7.127243
27  3.980936   5.490097   6.666788  ...  2.820279   8.868138   5.999217
28  3.069260   3.893784   3.996605  ...  3.035789   5.204941   5.873683
29  5.652110   6.909825   7.330997  ...  3.268765   8.586946   7.231704
30  2.804424   3.692797   4.160087  ...  2.023825   5.484806   3.856568
31  4.623225   5.597538   5.978023  ...  3.009641   7.292686   6.036049
32  4.893092   7.632705   8.497258  ...  4.417263   8.510571   6.830109
33  3.825363   5.613024   6.881612  ...  3.333494   7.268530   6.204198
34  3.269116   4.401850   5.138978  ...  2.023093   6.121506   4.651324
35  4.041719   5.909749   6.537548  ...  3.169158   8.640427   6.667455
36  5.532622   7.459127   8.484552  ...  3.924468   9.548806   8.029789
37  3.473881   4.158621   4.534252  ...  2.661566   5.446494   4.737481
38  3.459412   3.955837   4.715690  ...  2.585044   5.229283   4.149099
39  3.101600   3.807568   4.302113  ...  2.357936   4.888646   4.313804

[40 rows x 32 columns]
```

Dataframe for Alpha power values

```
# Transform 40 x 32 x 7680 => 40 x 32
eeg_alpha = []
for i in range (len(filter_data)):
  for j in range (len(filter_data[0])):
    eeg_alpha.append(get_band_power(i,j,"alpha"))
eeg_alpha = np.reshape(eeg_alpha, (40, 32))

df_alpha = pd.DataFrame(data = eeg_alpha, columns=eeg_channels)
alpha_feature = (df_alpha.describe())
print(alpha_feature)
print(df_alpha)
#data frame to CSV conversion
#alpha_feature.to_csv('alpha_feature.csv')
df_alpha.to_csv('df_alpha.csv')
#print(df_theta.describe())
#print(df_theta)
```

```
             Fp1        AF3         F3  ...         P8        PO4         O2
count  40.000000  40.000000  40.000000  ...  40.000000  40.000000  40.000000
mean    3.762202   4.441650   4.903746  ...   3.809602   7.166256   6.477558
std     0.662314   0.907576   1.136222  ...   0.783262   1.272645   1.068724
min     2.666660   2.841876   3.021739  ...   2.372774   4.847106   4.357820
25%     3.306380   3.799863   4.092241  ...   3.269975   6.332106   5.784090
50%     3.766980   4.417777   4.884817  ...   3.750908   7.226185   6.482779
75%     4.214168   5.119865   5.577940  ...   4.104603   7.786522   7.175366
max     5.053208   6.312305   8.509200  ...   5.742830  10.265297   9.635522

[8 rows x 32 columns]
         Fp1       AF3        F3  ...        P8       PO4        O2
0   3.398845  3.989256  4.515779  ...  3.435137  6.255098  5.370451
1   3.883059  4.615910  4.954047  ...  3.426914  7.245613  6.436137
2   4.172697  4.867400  5.181410  ...  4.122377  8.400493  7.678708
```

```
3   3.377279  4.043221  4.364608  ...  2.809046   6.433562  5.908209
4   2.683563  2.864437  3.021739  ...  3.120202   4.943390  5.038999
5   3.351979  3.678299  4.021338  ...  3.186407   6.666349  5.797813
6   3.748695  4.250157  4.827904  ...  3.421963   7.072426  6.420735
7   4.596282  5.823375  6.387123  ...  5.742830  10.265297  9.635522
8   5.035217  6.149034  6.723983  ...  4.643232   9.730360  7.987899
9   3.651818  4.156247  4.338791  ...  3.108650   6.487638  5.452295
10  3.760204  4.884986  5.540340  ...  3.774696   6.848356  5.742921
11  4.647695  5.241081  5.476047  ...  5.042135   7.785917  7.172417
12  4.436727  5.452762  6.395493  ...  4.097321   9.055071  6.958429
13  4.179838  5.356268  6.094366  ...  5.181689   8.006611  7.413013
14  3.776158  4.625633  4.801940  ...  3.668633   7.309992  6.801717
15  3.255305  3.431056  3.565772  ...  3.655263   5.872276  5.919061
16  3.058422  3.295024  3.462511  ...  3.577137   6.328675  6.688554
17  2.807646  3.481768  3.790626  ...  2.744121   5.488031  4.626314
18  3.773755  4.566537  4.949067  ...  3.766575   8.147395  6.676416
19  3.575482  4.414482  5.062547  ...  3.276533   7.489067  6.384782
20  3.981305  5.098477  6.080846  ...  3.735241   9.456323  7.426966
21  3.780244  4.421073  4.665369  ...  3.336556   7.206758  6.401150
22  3.941902  4.344795  4.902596  ...  4.948155   7.065337  6.727856
23  3.318851  3.840384  4.197498  ...  3.168432   6.333250  5.728430
24  4.576886  5.448664  5.855972  ...  4.911485   7.727790  6.846808
25  4.680666  6.312305  8.509200  ...  5.020437   7.525770  7.702020
26  5.053208  5.746763  6.192725  ...  4.988686   8.936431  7.719087
27  3.268968  3.960680  4.579225  ...  3.332736   7.423089  6.252132
28  3.004775  3.345307  3.533071  ...  3.842904   6.211225  6.279723
29  4.582046  5.289498  5.609883  ...  3.995261   7.838070  7.134467
30  2.666660  2.841876  3.107444  ...  2.545685   4.847106  4.357820
31  4.256118  4.720651  4.893838  ...  3.778856   7.515770  7.335543
32  3.886357  5.066913  5.567293  ...  4.098678   6.870524  6.223245
33  3.525435  4.241796  4.875796  ...  3.963688   7.565765  7.430019
34  2.757852  3.174377  3.496461  ...  2.372774   5.081408  4.686759
35  4.200185  4.919485  5.330874  ...  4.093205   7.788337  6.814952
36  4.533248  5.184028  5.778097  ...  4.441921   7.530409  7.184214
37  3.554770  3.946679  4.115876  ...  3.823172   6.836087  6.529420
38  2.872213  3.268699  3.716342  ...  3.250299   5.109423  4.545731
39  2.875738  3.306634  3.665990  ...  2.935037   5.949740  5.665576

[40 rows x 32 columns]
```

Dataframe for Beta power values

```
# Transform 40 x 32 x 7680 => 40 x 32
eeg_beta = []
for i in range (len(filter_data)):
  for j in range (len(filter_data[0])):
    eeg_beta.append(get_band_power(i,j,"beta"))
eeg_beta = np.reshape(eeg_beta, (40, 32))

df_beta = pd.DataFrame(data = eeg_beta, columns=eeg_channels)
beta_feature = (df_beta.describe())
print(beta_feature)
print(df_beta)
#data frame to CSV conversion
#alpha_feature.to_csv('alpha_feature.csv')
df_beta.to_csv('df_beta.csv')
#print(df_theta.describe())
#print(df_theta)
```

```
             Fp1        AF3        F3  ...        P8       PO4        O2
count  40.000000  40.000000  40.000000  ...  40.000000  40.000000  40.000000
mean    5.999201   7.082044   7.976174  ...   5.067042   7.648056   7.048770
std     0.840605   1.349351   1.835525  ...   1.139815   0.968834   0.901107
min     4.745370   5.340509   5.823536  ...   3.465957   6.008508   5.277376
25%     5.307373   5.977582   6.812266  ...   4.422545   6.834445   6.437713
50%     5.952609   7.010898   7.854018  ...   4.738749   7.662486   6.976903
75%     6.453396   7.591288   8.816876  ...   5.576926   8.315726   7.579501
max     9.015656  13.175819  16.944850  ...   9.327020   9.482370   9.164525

[8 rows x 32 columns]
         Fp1        AF3         F3  ...        P8       PO4        O2
0   5.921890   6.724025   7.305003  ...  4.480089  7.091182  6.202064
1   6.175955   7.033744   7.636457  ...  4.651400  7.475481  6.555805
2   6.706055   7.924134   8.812951  ...  5.375542  8.809592  8.165761
3   5.292459   5.987910   6.819542  ...  3.504195  6.634186  6.295400
4   4.869721   5.340509   5.856276  ...  4.390148  6.113248  6.115323
5   5.247273   5.918595   6.702882  ...  4.428226  6.838434  6.016293
6   5.533536   6.957050   9.045172  ...  4.623270  7.641467  7.156553
7   6.683513   8.252988   9.194394  ...  6.520311  9.482370  9.012947
8   7.282756   8.719999   9.747554  ...  6.563056  9.410428  8.321691
9   5.741234   6.849063   7.703313  ...  4.083776  7.664456  6.979811
10  5.688381   6.764195   7.904921  ...  4.728124  6.436139  5.733217
11  6.739414   8.100882   8.828652  ...  5.852210  8.516916  7.659322
12  6.365159   7.619152   9.164879  ...  5.551777  8.335119  6.973994
13  6.421224   7.280822   8.110566  ...  6.072746  7.660515  6.869096
14  5.654419   6.781343   7.234876  ...  4.260204  7.949313  7.552895
15  5.312344   5.710374   6.122933  ...  5.652376  6.487477  6.449856
16  4.950712   5.598912   6.061675  ...  3.991805  6.822478  6.847441
17  4.745370   5.599632   5.978489  ...  3.677759  6.008508  5.277376
18  6.248502   7.413434   7.992494  ...  4.749373  8.349315  7.511891
19  6.012444   7.199948   7.862293  ...  4.716582  8.102493  7.025902
20  6.328630   7.447485   8.503898  ...  4.666118  9.097350  7.542206
21  6.334704   7.473329   7.845744  ...  4.405502  8.206536  7.548812
22  6.518300   7.359175   8.238735  ...  7.012076  7.935638  7.551224
23  5.473310   6.350462   7.025162  ...  4.718832  6.975457  6.401285
24  6.877722   8.262923   9.103705  ...  6.823458  8.149121  7.781071
25  9.015656  13.175819  16.944850  ...  9.327020  9.104660  9.164525
26  7.380066   8.255746   9.344142  ...  6.789700  8.829282  7.858610
27  5.634627   6.778748   7.763985  ...  4.783150  7.625435  6.923299
28  5.086452   5.718074   6.284306  ...  4.531052  6.702635  6.587971
29  6.546035   8.156367   9.425978  ...  4.985239  7.962075  7.401752
30  5.108968   5.713330   6.319108  ...  3.817774  6.455506  5.753501
31  6.431762   7.582000   8.263652  ...  4.766203  8.128217  7.764926
32  5.848443   7.174567   8.572264  ...  5.932437  7.056209  6.567589
33  5.983328   6.988052   7.786428  ...  4.759392  8.762442  8.170801
34  4.871070   5.386515   5.823536  ...  3.465957  6.351413  6.189863
35  6.408845   7.448809   8.273009  ...  4.891153  8.154323  7.499183
36  6.531870   7.807501   9.020001  ...  5.217102  8.309261  7.891851
37  5.623852   6.560948   6.951924  ...  4.933388  7.075421  6.458633
38  5.200477   5.946596   6.790440  ...  4.694611  6.170843  5.542299
39  5.171544   5.918616   6.680757  ...  4.288540  7.041315  6.628770

[40 rows x 32 columns]
```

Dataframe for Gamma power values

```
# Transform 40 x 32 x 7680 => 40 x 32
eeg_gamma = []
for i in range (len(filter_data)):
  for j in range (len(filter_data[0])):
    eeg_gamma.append(get_band_power(i,j,"gamma"))
eeg_gamma = np.reshape(eeg_gamma, (40, 32))

df_gamma = pd.DataFrame(data = eeg_gamma, columns=eeg_channels)
gamma_feature = (df_gamma.describe())
print(gamma_feature)
print(df_gamma)
#data frame to CSV conversion
#alpha_feature.to_csv('alpha_feature.csv')
df_gamma.to_csv('df_gamma.csv')
#print(df_theta.describe())
#print(df_theta)
```

```
             Fp1        AF3        F3  ...         P8        PO4         O2
count  40.000000  40.000000  40.000000  ...  40.000000  40.000000  40.000000
mean    1.486692   1.581345   1.872482  ...   1.607013   1.437616   1.311753
std     0.392132   0.653346   1.119080  ...   0.868679   0.205085   0.270454
min     1.159532   1.231137   1.328071  ...   0.801404   1.120329   0.950432
25%     1.313642   1.361393   1.532119  ...   1.088946   1.323672   1.143811
50%     1.403192   1.427084   1.639636  ...   1.339545   1.409186   1.246689
75%     1.513599   1.608228   1.780154  ...   1.853287   1.523086   1.460911
max     3.600448   5.475807   8.550272  ...   5.592532   2.250519   2.482995

[8 rows x 32 columns]
         Fp1       AF3        F3  ...        P8       PO4        O2
0   1.308929  1.403657  1.522686  ...  1.148110  1.291682  1.095876
1   1.448926  1.424581  1.565883  ...  1.311693  1.331894  1.180496
2   1.500272  1.558377  1.710590  ...  1.370533  1.513840  1.406613
3   1.326880  1.327685  1.543712  ...  0.801404  1.207338  1.087293
4   1.159532  1.231137  1.352958  ...  1.345094  1.120329  1.046113
5   1.316018  1.270956  1.456950  ...  1.333996  1.322582  1.145186
6   1.299125  1.699172  2.592254  ...  1.116570  1.365478  1.239544
7   1.593003  1.612116  1.717490  ...  1.890685  1.624892  1.710846
8   1.761178  1.708140  1.898597  ...  2.638149  1.722127  1.563641
9   1.370363  1.442856  1.643215  ...  1.067343  1.559798  1.604263
10  1.290278  1.429587  1.733043  ...  1.435003  1.207184  1.057079
11  1.432906  1.540871  1.675891  ...  1.681494  1.389500  1.205549
12  1.730252  1.862685  2.458710  ...  1.872457  1.623495  1.308106
13  2.011043  1.746269  1.818714  ...  2.390470  1.416687  1.154951
14  1.289817  1.365002  1.477718  ...  0.953039  1.493702  1.519858
15  1.336296  1.338812  1.482534  ...  2.160238  1.372679  1.253834
16  1.194058  1.259112  1.392754  ...  0.819898  1.337777  1.130735
17  1.196156  1.245796  1.328071  ...  0.953366  1.136308  0.950432
18  1.344013  1.417210  1.561063  ...  1.139734  1.409223  1.339051
19  1.591328  1.633406  1.861766  ...  1.700788  1.711402  1.270537
20  1.497394  1.552498  1.751058  ...  1.393494  1.607777  1.369818
21  1.554941  1.606932  1.634827  ...  1.095909  1.496589  1.612317
22  1.474474  1.517304  1.767301  ...  3.099296  1.591846  1.455929
23  1.344474  1.302652  1.440753  ...  1.846897  1.251310  1.158160
24  1.553583  1.669101  1.717053  ...  2.521991  1.423491  1.526657
25  3.600448  5.475807  8.550272  ...  5.592532  2.250519  2.482995
26  1.983172  1.839902  2.148646  ...  2.533989  1.746219  1.475856
27  1.346440  1.422115  1.664083  ...  1.670110  1.413322  1.505437
28  1.163427  1.263190  1.534022  ...  1.244601  1.174941  1.139689
29  1.452514  1.752757  2.285832  ...  1.418334  1.352341  1.264317
```

```
30  1.410933  1.391191  1.526410  ...  1.084893  1.409148  1.120873
31  1.293626  1.401728  1.550546  ...  1.090297  1.275131  1.186545
32  1.420073  1.499641  2.130436  ...  2.699154  1.324035  1.221039
33  1.349080  1.424300  1.636057  ...  1.009279  1.550822  1.255657
34  1.395451  1.329044  1.433855  ...  0.906726  1.465345  1.631698
35  1.596693  1.565312  1.705337  ...  1.068196  1.502740  1.196738
36  1.441995  1.600276  1.889534  ...  1.116853  1.510915  1.333718
37  1.315213  1.369012  1.598855  ...  1.356364  1.282915  1.192252
38  1.488077  1.403060  1.595416  ...  1.332139  1.337097  0.983661
39  1.285305  1.350568  1.544395  ...  1.069396  1.380214  1.086775

[40 rows x 32 columns]
```

Fisher's Score for Theta Band

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split

from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest, SelectPercentile


# load dataset
data = pd.read_csv('1df_theta.csv')
data.shape
```

```
(40, 33)
```

```
data.head()
```

	videos	Fp1	AF3	F3	F7	FC5	FC1	C3	
0	0	4.045076	5.203156	5.961250	5.864659	2.973758	2.841117	3.273879	6.434
1	1	5.143294	7.048838	7.716178	8.193027	3.714211	4.247095	4.606335	10.285
2	2	4.673507	6.338758	7.163341	7.306396	3.961363	3.359052	4.220336	9.115
3	3	4.307117	6.193349	6.949383	6.434644	3.598289	2.657834	3.454831	6.613
4	4	2.597929	2.947654	3.081066	3.541055	1.826381	2.056701	1.806602	3.713

```
X = data[['Fp1',    'AF3',  'F3',   'F7',   'FC5',  'FC1',  'C3',   'T7',   'CP5',  'CP1',
X.head()
```

	Fp1	AF3	F3	F7	FC5	FC1	C3	T7	
0	4.045076	5.203156	5.961250	5.864659	2.973758	2.841117	3.273879	6.434752	2.74
1	5.143294	7.048838	7.716178	8.193027	3.714211	4.247095	4.606335	10.285405	3.46
2	4.673507	6.338758	7.163341	7.306396	3.961363	3.359052	4.220336	9.115732	3.38
3	4.307117	6.193349	6.949383	6.434644	3.598289	2.657834	3.454831	6.613919	3.28
4	2.597929	2.947654	3.081066	3.541055	1.826381	2.056701	1.806602	3.713862	1.82

```
y = data['videos']
y.head()
```

```
0    0
1    1
2    2
3    3
4    4
Name: videos, dtype: int64
```

```
X.shape, y.shape
```

```
((40, 32), (40,))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state =
```

```
f_score = chi2(X_train, y_train)
```

```
# separate train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    data[['Fp1',    'AF3',  'F3',   'F7',   'FC5',  'FC1',  'C3',   'T7',   'CP5',  'CP1',
    data['videos'],
    test_size=0.3,
    random_state=0)

X_train.shape, X_test.shape
```

```
((28, 32), (12, 32))
```

```
f_score = chi2(X_train.fillna(0), y_train)
f_score
```

```
(array([  5.73313589,  13.30551522,  14.41257628,  12.1104467 ,
          8.5911387 ,   7.76273454,   5.10788197,  18.17581648,
          4.52269415,   0.91918237,  20.8012056 ,  14.59212764,
          5.43636672,   6.76179663,   8.53607593,   1.82127231,
         30.12052326,  27.89277362, 103.41024579,  53.06269386,
          4.16675661,   4.14301664,  46.34429029,  28.13293201,
          4.35716441,  13.51260228,   3.32159185,   0.445687  ,
         22.3147356 ,   8.05408439,  15.27925655,  10.74716599]),
 array([9.99995429e-01, 9.87158636e-01, 9.76890858e-01, 9.93854813e-01,
        9.99708313e-01, 9.99891838e-01, 9.99998719e-01, 8.98054150e-01,
        9.99999676e-01, 1.00000000e+00, 7.95510349e-01, 9.74772367e-01,
```

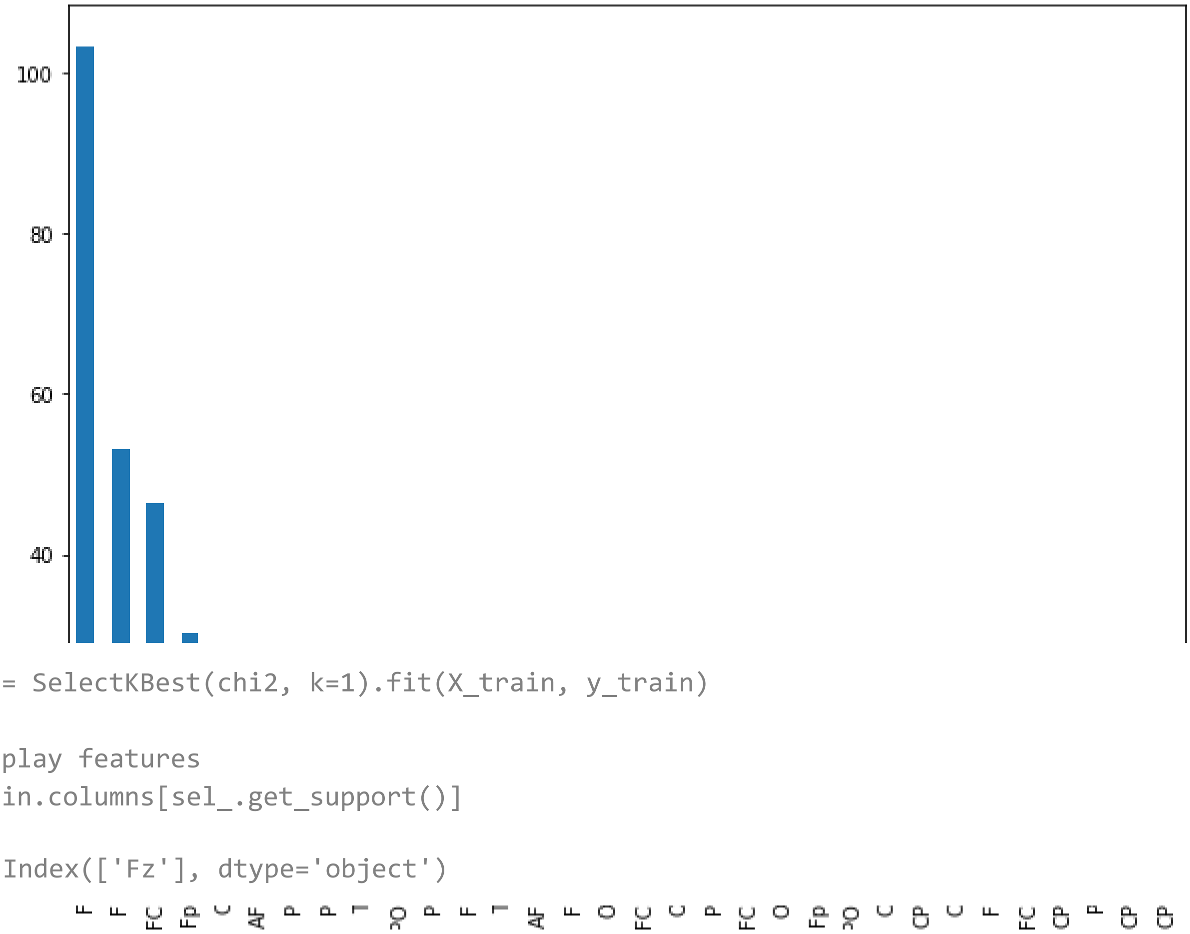
```
            9.99997444e-01, 9.99973511e-01, 9.99725792e-01, 1.00000000e+00,
            3.08765717e-01, 4.16517857e-01, 7.04160179e-11, 1.97780483e-03,
            9.99999874e-01, 9.99999882e-01, 1.16792025e-02, 4.04144188e-01,
            9.99999788e-01, 9.85573948e-01, 9.99999991e-01, 1.00000000e+00,
            7.21217453e-01, 9.99844230e-01, 9.65330399e-01, 9.97739573e-01]))
```

```
fvalues = pd.Series(f_score[0])
fvalues.index = X_train.columns
fvalues.sort_values(ascending=False)
fvalues.to_csv('fscore_theta.csv')
print(fvalues)
```

```
Fp1      5.733136
AF3     13.305515
F3      14.412576
F7      12.110447
FC5      8.591139
FC1      7.762735
C3       5.107882
T7      18.175816
CP5      4.522694
CP1      0.919182
P3      20.801206
P7      14.592128
PO3      5.436367
O1       6.761797
Oz       8.536076
Pz       1.821272
Fp2     30.120523
AF4     27.892774
Fz     103.410246
F4      53.062694
F8       4.166757
FC6      4.143017
FC2     46.344290
Cz      28.132932
C4       4.357164
T8      13.512602
CP6      3.321592
CP2      0.445687
P4      22.314736
P8       8.054084
PO4     15.279257
O2      10.747166
dtype: float64
```

```
#fvalues.plot.bar()
fvalues.sort_values(ascending=False).plot.bar(figsize=(10,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcfdd190dd0>
```




```
sel_ = SelectKBest(chi2, k=1).fit(X_train, y_train)

# display features
X_train.columns[sel_.get_support()]
```

```
Index(['Fz'], dtype='object')
```

Fisher Score for Alpha Band

```
# load dataset
data = pd.read_csv('2df_alpha.csv')
data.shape
```

```
(40, 33)
```

```
data.head()
```

	videos	Fp1	AF3	F3	F7	FC5	FC1	C3	
0	0	3.398845	3.989256	4.515779	4.596829	2.399088	2.220344	2.882789	4.9453
1	1	3.883059	4.615910	4.954047	5.601205	2.925839	2.640588	3.256620	6.4311
2	2	4.172697	4.867400	5.181410	5.726629	3.148036	2.388484	3.385071	6.3513
3	3	3.377279	4.043221	4.364608	4.661950	2.509896	2.082791	2.750999	5.1765
4	4	2.683563	2.864437	3.021739	3.196781	1.898775	1.659160	1.926744	3.8118

```
X.shape, y.shape
```

```
((40, 32), (40,))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state =
```

```
f_score = chi2(X_train, y_train)
```

```
# separate train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    data[['Fp1',   'AF3',  'F3',   'F7',   'FC5',  'FC1',  'C3',   'T7',   'CP5',  'CP1',
    data['videos'],
    test_size=0.3,
    random_state=0)

X_train.shape, X_test.shape
```

```
((28, 32), (12, 32))
```

```
f_score = chi2(X_train.fillna(0), y_train)
f_score
```

```
(array([ 3.28484908,  4.85784063,  5.60837883,  6.79310743,  5.35666852,
         2.53601759,  2.67267443,  7.00652862,  4.06404925,  1.89852151,
         8.62359414,  7.18493685,  9.65832647,  4.76576003,  4.27933727,
         3.30751588,  9.32582485,  6.82938375, 18.63829103, 14.24898683,
         3.24909092,  2.77709596,  9.06783204,  7.20526661,  2.73414112,
         6.71034638,  2.51713591,  1.16421122,  7.03378514,  4.99153607,
         6.71070305,  5.89705121]),
 array([0.99999999, 0.99999927, 0.9999964 , 0.99997221, 0.99999783,
        1.        , 1.        , 0.99996172, 0.99999991, 1.        ,
        0.99969757, 0.99995045, 0.99912636, 0.99999941, 0.99999983,
        0.99999999, 0.99936716, 0.99997063, 0.88280333, 0.97870319,
        0.99999999, 1.        , 0.99951294, 0.999949  , 1.        ,
        0.99997554, 1.        , 1.        , 0.99996016, 0.99999901,
        0.99997552, 0.9999938 ]))
```

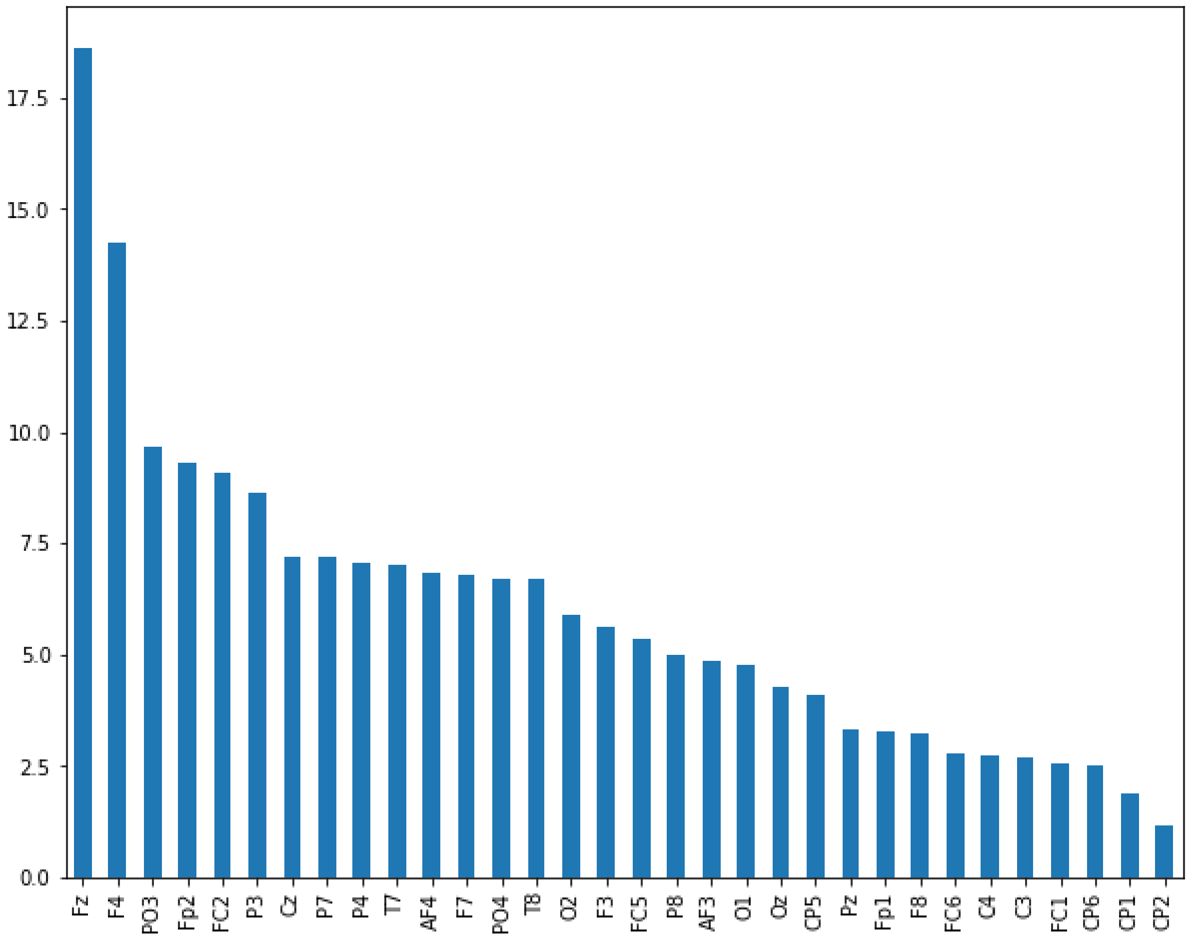
```
fvalues = pd.Series(f_score[0])
fvalues.index = X_train.columns
fvalues.sort_values(ascending=False)
fvalues.to_csv('fscore_alpha.csv')
print(fvalues)
```

```
Fp1     3.284849
AF3     4.857841
F3      5.608379
F7      6.793107
FC5     5.356669
FC1     2.536018
C3      2.672674
T7      7.006529
CP5     4.064049
CP1     1.898522
P3      8.623594
P7      7.184937
PO3     9.658326
O1      4.765760
Oz      4.279337
```

```
Pz       3.307516
Fp2      9.325825
AF4      6.829384
Fz      18.638291
F4      14.248987
F8       3.249091
FC6      2.777096
FC2      9.067832
Cz       7.205267
C4       2.734141
T8       6.710346
CP6      2.517136
CP2      1.164211
P4       7.033785
P8       4.991536
PO4      6.710703
O2       5.897051
dtype: float64
```

```
#fvalues.plot.bar()
fvalues.sort_values(ascending=False).plot.bar(figsize=(10,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcff2fe89d0>
```




```
sel_ = SelectKBest(chi2, k=1).fit(X_train, y_train)

# display features
X_train.columns[sel_.get_support()]
```

```
Index(['Fz'], dtype='object')
```

Fisher Score for Beta Band

```
# load dataset
data = pd.read_csv('3df_beta.csv')
data.shape
```

```
(40, 33)
```

```
X.shape, y.shape
```

```
((40, 32), (40,))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state =
```

```
f_score = chi2(X_train, y_train)
```

```
data.head()
```

	videos	Fp1	AF3	F3	F7	FC5	FC1	C3	
0	0	5.921890	6.724025	7.305003	6.297888	3.241240	3.698041	3.659401	7.2414
1	1	6.175955	7.033744	7.636457	8.110029	3.808173	4.136289	4.080968	8.4974
2	2	6.706055	7.924134	8.812951	7.684197	3.931223	4.169439	4.408886	8.3811
3	3	5.292459	5.987910	6.819542	6.080699	3.236244	3.763843	3.438532	8.2130
4	4	4.869721	5.340509	5.856276	4.812446	2.749350	3.043106	2.928019	8.1782

```
# separate train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    data[['Fp1',    'AF3',  'F3',   'F7',   'FC5',  'FC1',  'C3',   'T7',   'CP5',  'CP1',
    data['videos'],
    test_size=0.3,
    random_state=0)

X_train.shape, X_test.shape
```

```
((28, 32), (12, 32))
```

```
f_score = chi2(X_train.fillna(0), y_train)
f_score
```

```
(array([2.36539587, 3.40268531, 4.56706788, 3.65571759, 4.89374565,
        1.47617185, 1.53318004, 5.06184245, 2.46500835, 1.07711032,
        4.18750053, 5.91006534, 3.84020505, 2.24511146, 2.5116875 ,
        1.56772371, 5.04884054, 2.76325092, 6.29016204, 6.97684164,
```

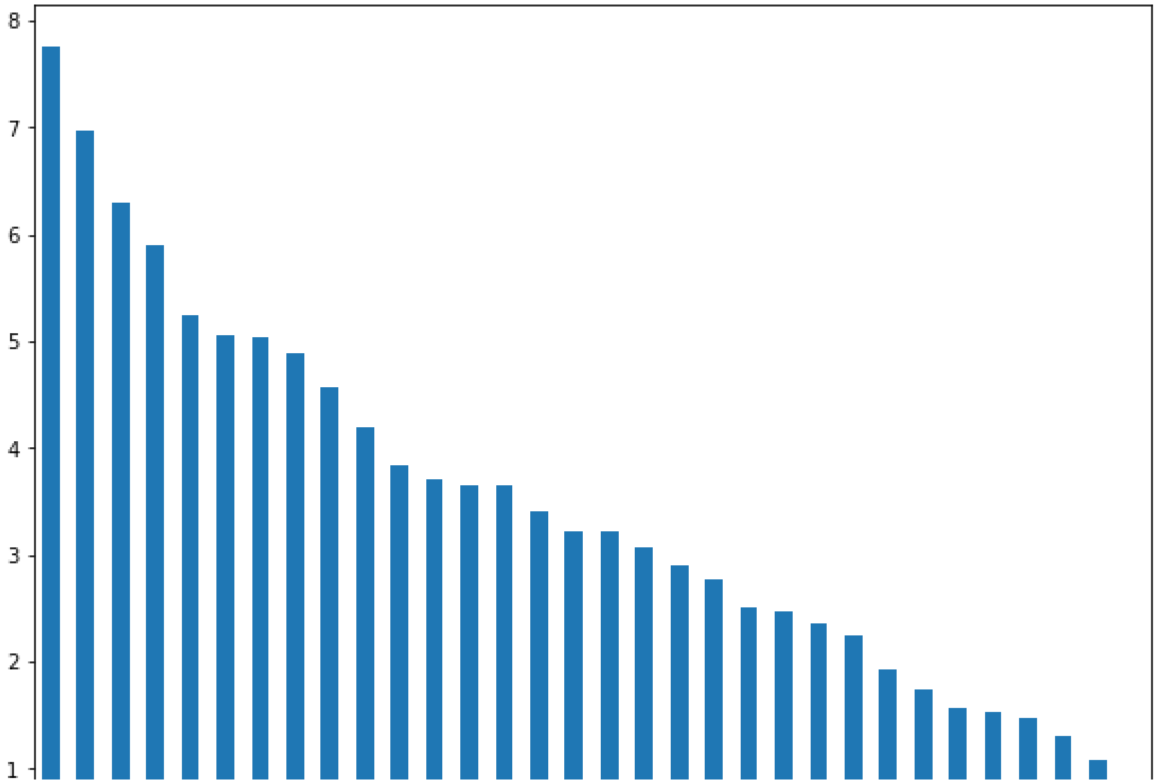
```
            3.64510035, 1.92658292, 3.70445304, 1.30071026, 1.7337007 ,
            7.76617927, 3.0762374 , 0.66349648, 2.89915551, 5.25385032,
            3.22939793, 3.21818291]),
     array([1.        , 0.99999999, 0.99999964, 0.99999997, 0.99999921,
            1.        , 1.        , 0.99999884, 1.        , 1.        ,
            0.99999987, 0.99999365, 0.99999995, 1.        , 1.        ,
            1.        , 0.99999887, 1.        , 0.99998762, 0.99996336,
            0.99999997, 1.        , 0.99999997, 1.        , 1.        ,
            0.99989136, 1.        , 1.        , 1.        , 0.99999825,
            0.99999999, 0.99999999]))
```

```
fvalues = pd.Series(f_score[0])
fvalues.index = X_train.columns
fvalues.sort_values(ascending=False)
fvalues.to_csv('fscore_beta.csv')
print(fvalues)
```

```
Fp1    2.365396
AF3    3.402685
F3     4.567068
F7     3.655718
FC5    4.893746
FC1    1.476172
C3     1.533180
T7     5.061842
CP5    2.465008
CP1    1.077110
P3     4.187501
P7     5.910065
PO3    3.840205
O1     2.245111
Oz     2.511687
Pz     1.567724
Fp2    5.048841
AF4    2.763251
Fz     6.290162
F4     6.976842
F8     3.645100
FC6    1.926583
FC2    3.704453
Cz     1.300710
C4     1.733701
T8     7.766179
CP6    3.076237
CP2    0.663496
P4     2.899156
P8     5.253850
PO4    3.229398
O2     3.218183
dtype: float64
```

```
#fvalues.plot.bar()
fvalues.sort_values(ascending=False).plot.bar(figsize=(10,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcfdcaf92d0>
```




```
sel_ = SelectKBest(chi2, k=1).fit(X_train, y_train)

# display features
X_train.columns[sel_.get_support()]
```

```
Index(['T8'], dtype='object')
```

Fisher Score for Gamma Band

```
# load dataset
data = pd.read_csv('4df_gamma.csv')
data.shape
```

```
(40, 33)
```

```
X.shape, y.shape
```

```
((40, 32), (40,))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state =
```

```
f_score = chi2(X_train, y_train)
```

```
data.head()
```

	videos	Fp1	AF3	F3	F7	FC5	FC1	C3	
0	0	1.308929	1.403657	1.522686	1.398537	0.811275	0.858045	0.901661	2.6539
1	1	1.448926	1.424581	1.565883	2.756226	0.932322	0.862945	0.935371	3.1520
2	2	1.500272	1.558377	1.710590	2.418392	1.017373	0.898782	1.074023	3.3774
3	3	1.326880	1.327685	1.543712	1.625128	0.895629	0.884229	0.884067	3.8040
4	4	1.159532	1.231137	1.352958	1.266736	0.760074	0.751901	0.766605	3.9900

```
# separate train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    data[['Fp1',    'AF3',  'F3',   'F7',   'FC5',  'FC1',  'C3',   'T7',   'CP5',  'CP1',
    data['videos'],
    test_size=0.3,
    random_state=0)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((28, 32), (12, 32), (28,), (12,))
```

```
f_score = chi2(X_train.fillna(0), y_train)
f_score
#f_score.to_csv('fscore.csv')
```

```
(array([ 0.80602972,  0.57535419,  1.4839313 ,  3.78515529,  7.45632062,
         0.2731191 ,  0.54685432,  4.8320275 ,  0.96161682,  0.08083882,
         0.63611926,  4.37397934,  0.31317709,  1.14663315,  0.41629454,
         0.09947068,  1.3489424 ,  0.80131724,  0.34831832,  0.85905867,
         2.63749167,  3.12044555,  0.21331475,  0.27860547,  1.52321518,
        18.48221735, 10.186954  ,  0.08602232,  0.92455459,  6.5660025 ,
         0.47736076,  0.93825316]),
 array([1.        , 1.        , 1.        , 0.99999996, 0.99992779,
        1.        , 1.        , 0.99999931, 1.        , 1.        ,
        1.        , 0.99999978, 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        0.88809119, 0.99858741, 1.        , 1.        , 0.99998052,
        1.        , 1.        ]))
```

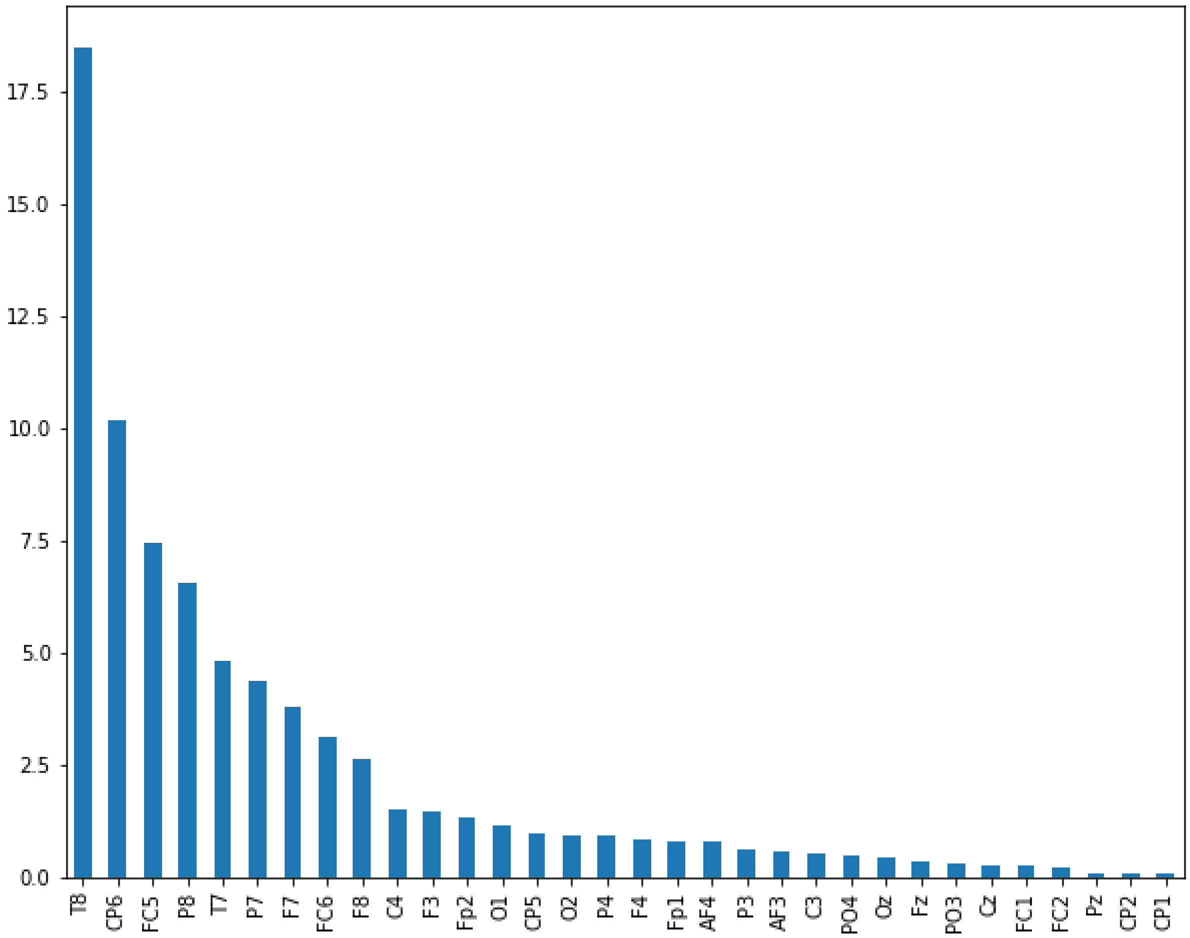
```
fvalues = pd.Series(f_score[0])
fvalues.index = X_train.columns
fvalues.sort_values(ascending=False)
fvalues.to_csv('fscore_gamma.csv')
print(fvalues)
```

```
Fp1     0.806030
AF3     0.575354
F3      1.483931
F7      3.785155
FC5     7.456321
FC1     0.273119
C3      0.546854
T7      4.832028
CP5     0.961617
CP1     0.080839
```

```
P3       0.636119
P7       4.373979
PO3      0.313177
O1       1.146633
Oz       0.416295
Pz       0.099471
Fp2      1.348942
AF4      0.801317
Fz       0.348318
F4       0.859059
F8       2.637492
FC6      3.120446
FC2      0.213315
Cz       0.278605
C4       1.523215
T8      18.482217
CP6     10.186954
CP2      0.086022
P4       0.924555
P8       6.566003
PO4      0.477361
O2       0.938253
dtype: float64
```

```
#fvalues.plot.bar()
fvalues.sort_values(ascending=False).plot.bar(figsize=(10,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcfdc954b10>
```




```
sel_ = SelectKBest(chi2, k=1).fit(X_train, y_train)
```

```
# display features
X_train.columns[sel_.get_support()]
```

```
Index(['T8'], dtype='object')
```

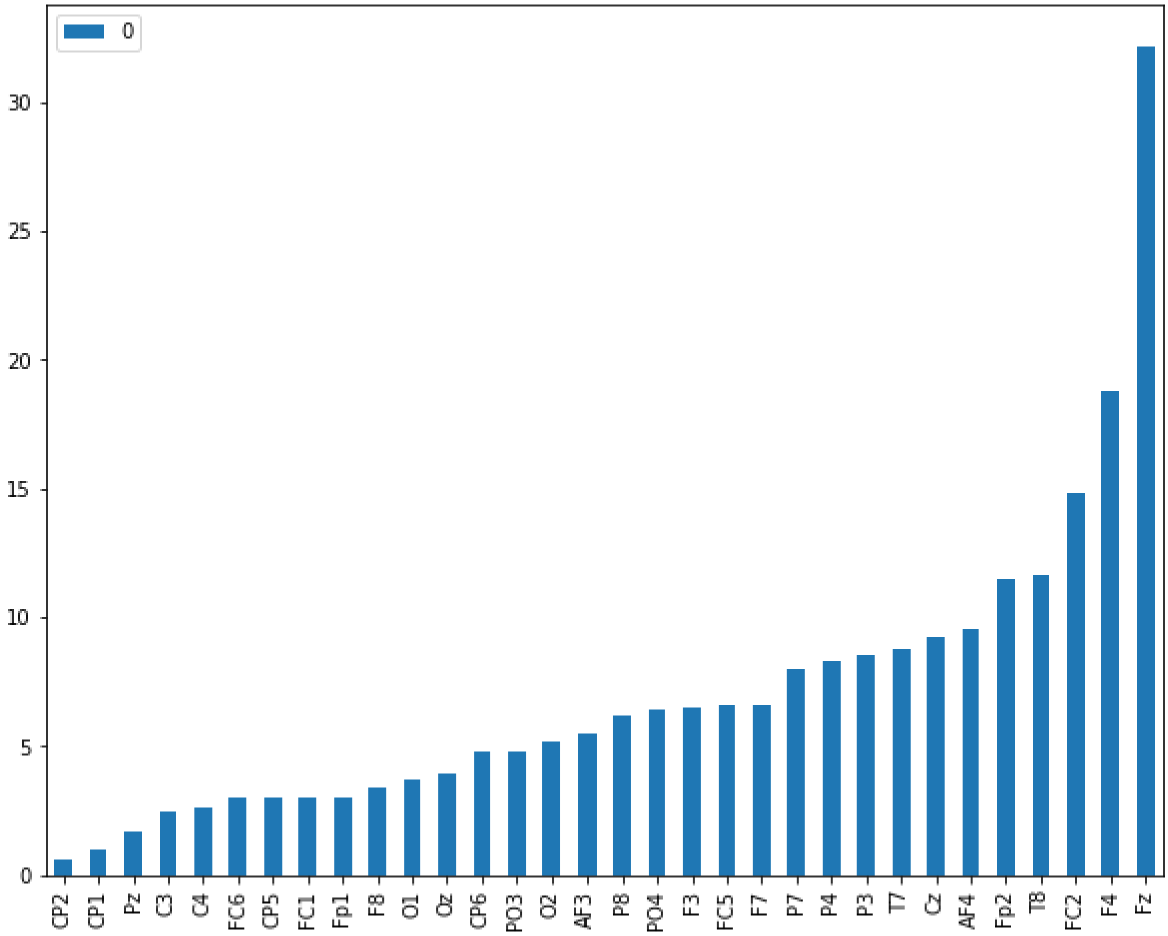
Total Avearge F-Score

```
total_fscore_data = pd.read_csv('fscore_total.csv',index_col=0).sort_values(by=['0'])
total_fscore_data.shape
total_fscore_data
```

	0
CP2	0.589854
CP1	0.993913
Pz	1.698996
C3	2.465148
C4	2.587055
FC6	2.991785
CP5	3.003342
FC1	3.012011

```
total_fscore_data.plot.bar(figsize=(10,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faeb212a050>
```



Cz 9.229379

Classification

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

F4 18.786805

```
import pandas as pd
data = pd.read_csv("Book2.csv")
```

```
data
```

	videos	FzTheta	FzAlpha	FzBeta	FzGamma
0	0	6.176246	4.322296	6.048135	1.191866
1	1	8.345299	4.812194	6.408611	1.168725
2	2	9.600425	5.405637	7.251806	1.331525
3	3	12.692512	5.619613	6.102789	1.151865
4	4	3.515012	2.915420	5.189118	1.083122
5	5	5.114116	3.692517	5.883927	1.165087
6	6	5.487069	4.349237	6.102311	1.244622
7	7	12.729863	7.938675	8.949391	1.453063

```
data = data.drop('videos', axis = 1)
x = np.array(data)
y = np.array(em_labels)
```

10	10	9.644406	5.286301	6.304967	1.223515

```
print(data)
```

```
     FzTheta    FzAlpha    FzBeta   FzGamma
0    6.176246   4.322296  6.048135  1.191866
1    8.345299   4.812194  6.408611  1.168725
2    9.600425   5.405637  7.251806  1.331525
3   12.692512   5.619613  6.102789  1.151865
4    3.515012   2.915420  5.189118  1.083122
5    5.114116   3.692517  5.883927  1.165087
6    5.487069   4.349237  6.102311  1.244622
7   12.729863   7.938675  8.949391  1.453063
8   20.178017   8.790646  9.052838  1.510332
9    6.775441   4.573829  7.073508  1.317985
10   9.644406   5.286301  6.304967  1.223515
11  11.147350   6.576466  8.287427  1.372574
12   7.229926   4.989285  6.646497  1.288661
13  30.218795  10.245903  8.304906  1.382939
14  10.510897   6.619012  7.722740  1.319610
15   5.659122   3.653861  5.280646  1.129614
16   6.206081   4.308195  5.498378  1.110447
17   4.954103   3.371939  4.874603  1.022129
18   5.056793   3.983282  6.664512  1.256192
19   5.833986   4.458821  6.386082  1.223614
20   7.244018   5.228023  7.309713  1.334074
21   6.263520   4.555388  7.108337  1.275401
22   8.167380   5.497220  7.408813  1.348390
23   6.102216   4.083782  6.536193  1.243604
24  11.013817   6.678075  8.874964  1.429372
25  11.507413   6.586624  8.438445  1.781435
26  12.186705   7.742324  8.764357  1.488059
27   6.180781   4.435207  6.579715  1.190804
28   8.519039   4.531809  5.908094  1.152743
29  12.683576   6.812811  8.177562  1.385015
30   3.940152   2.958678  5.627921  1.165622
31   7.872014   5.433421  7.532137  1.286289
32  20.781867   8.703683  8.471666  1.426315
33  12.944150   6.538457  7.840651  1.356258
34   5.269281   3.350110  5.300812  1.152070
```

```
35  11.418630   5.809676  7.357208  1.321633
36  15.605528   7.774339  8.888546  1.462974
37   5.263686   4.097542  6.039846  1.148033
38   4.706968   3.637400  5.811326  1.162409
39   4.769078   3.479449  5.795699  1.192860
```

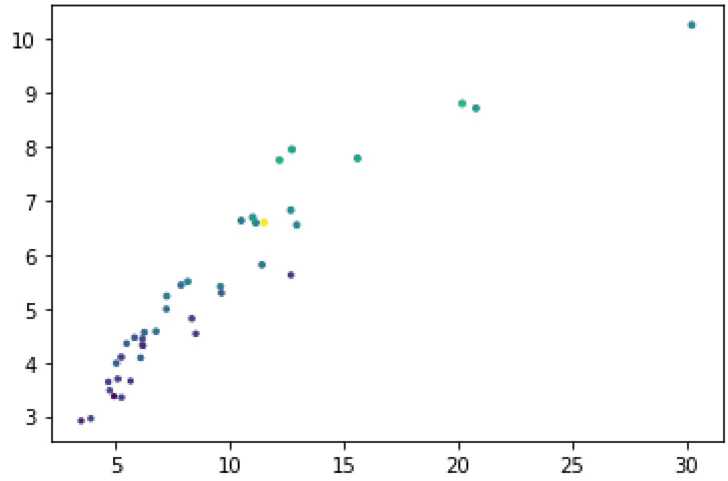
35 35 11.418630 5.809676 7.357208 1.321633

```
import matplotlib.pyplot as plt
%matplotlib inline
```

37 37 5.263686 4.097542 6.039846 1.148033

```
plt.scatter(data['FzTheta'],data['FzAlpha'],data['FzBeta'],data['FzGamma'])
```

```
<matplotlib.collections.PathCollection at 0x7efc4d2d7b10>
```




```
X = data[['FzTheta', 'FzAlpha', 'FzBeta', 'FzGamma']]
X.head()
```

	FzTheta	FzAlpha	FzBeta	FzGamma
0	6.176246	4.322296	6.048135	1.191866
1	8.345299	4.812194	6.408611	1.168725
2	9.600425	5.405637	7.251806	1.331525
3	12.692512	5.619613	6.102789	1.151865
4	3.515012	2.915420	5.189118	1.083122

```
y = data['videos']
y.head()
```

```
0    0
1    1
2    2
3    3
4    4
Name: videos, dtype: int64
```

```
X.shape, y.shape
```

```
((40, 4), (40,))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state =


# import SVC classifier
from sklearn.svm import SVC


# import metrics to compute accuracy
from sklearn.metrics import accuracy_score


# instantiate classifier with default hyperparameters
svc=SVC(kernel = 'linear')


# fit classifier to training set
svc.fit(X_train,y_train)


# make predictions on test set
y_pred=svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy_score
```

```
Model accuracy score with default hyperparameters: 0.3333
```

```
def run_randomForest(X_train, X_test, y_train, y_test):
    clf = RandomForestClassifier(n_estimators=100, random_state=0, n_jobs=-1)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print('Accuracy: ', accuracy_score(y_test, y_pred))


run_randomForest(X_train, X_test, y_train, y_test)
```

```
Accuracy:  0.25
```

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score


classifier = svm.SVC(kernel='linear')


classifier.fit(X_train,y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
```

```
        max_iter=-1, probability=False, random_state=None, shrinking=True,
        tol=0.001, verbose=False)
```

```
X_train_prediction = classifier.predict(X_train)
training_data_accuray = accuracy_score(X_train_prediction,y_train)
```

```
print('Accuracy on training data : ', training_data_accuray)
```

```
Accuracy on training data :  0.42857142857142855
```

```
X_test_prediction = classifier.predict(X_test)
test_data_accuray = accuracy_score(X_test_prediction,y_test)
```

```
print('Accuracy on test data : ', test_data_accuray)
```

```
Accuracy on test data :  0.3333333333333333
```

✓ 0s completed at 9:31 PM