Case Study -1 Reducing Commercial Aviation Fatalities

1.0 Introduction

Booz Allen Hamilton is a consulting firm with expertise in analytics, digital, engineering, and cyber, businesses, government, and military organizations has conducted this Kaggle competition. They have many complex problems real world related. Aviation fatality means the death of one or more persons inside or outside of an aircraft, spacecraft, or any other aerospace vehicle that occurs during a flight operation or any other operation involving that vehicle. These fatalities are directly associated with the operation of the aircraft and are called aviation accidents. Reducing fatalities in commercial aviation - This can be achieved, If we could detect the troubling events that alerts the pilots and aviation staff. So then we will find the solution for the problem before it gets worse. This is our aim for this problem Cause - Most flight-related fatalities stem from a loss of “airplane state awareness.” That is, ineffective attention management on the part of pilots who may be distracted, sleepy or in other dangerous cognitive states.

2.0 Business problem

People from different countries travel across or within countries through commercial aviation. So for the passengers to travel safe depends on some following conditions

1. Pilots should be healthy to operate
2. Plane outer body should be checked before take off
3. Engine should be tested So, our problem revolves around pilots due to more flights and odd timings pilots suffers from fatigue, jetlag and respiratory issues. Due to this when pilot operates flight, they go into airplane state awareness (ASA). Airplane state awareness (ASA) is a pilot performance attribute where in the pilot should be able to realize and respond quickly to any change of state of the airplane. Loss of airplane state awareness may lead to many dangerous situations and may result in loss of airplane control wherein an extreme deviation from the intended flight path may occur or in other words ineffective attention management on the part of pilots who may be distracted, sleepy or in other dangerous cognitive states. In this dataset, you are provided with real physiological data from pilots who were subjected to various distracting events. The pilots experienced distractions intended to induce one of the following three cognitive states:

   • Channelized Attention (CA) is, roughly speaking, the state of being focused on one task to the exclusion of all others. This is induced in benchmarking by having the subjects play an engaging puzzle-based video game.

• Diverted Attention (DA) is the state of having one's attention diverted by actions or thought processes associated with a decision. This is induced by having the subjects perform a display monitoring task. Periodically, a math problem showed up which had to be solved before returning to the monitoring task.

• Startle/Surprise (SS) is induced by having the subjects watch movie clips with jump scares. The aim is to build a model that can estimate the state of mind of the pilot in real-time using the physiological data given. When the pilot enters into any one of the above mentioned dangerous cognitive states, he/she should be alerted, thereby preventing any possible accident.

3.0 Dataset analysis

Three CSV files are provided for this competition. The first one is train.csv in which all the data which is to be used for training. Test.csv is provided to test the model. Sample_submission.csv is provided to submit the final output in the CSV format. Now, let's analyze each attribute in the dataset. The training data consist of three experiments: CA, DA, and SS. The output is one of the four labels: Baseline(no event), CA, DA, or SS. For example, if the experiment is CA, the output is either CA or Baseline(no event). The test data is taken from a full flight simulator. Here the experiment is called LOFT or Line Oriented Flight Training where the training of the pilot is carried out in a flight simulator, which artificially creates the environment of a real flight. In the test data, the experiment is given as LOFT and the output can be one of the four states at a given time. To predict the state of a pilot, physiological data are required. We have data from four sensors — EEG, ECG, Respiration, Galvanic skin response. Let's analyse each attribute of the dataset.

• Id: Unique identifier for crew + time combination. A pilot with a particular time into the experiment is represented using an id. So for each id, we need to predict the state

• Crew: Unique id for a pair or pilot

• Experiment: For training, it will be either CA or DA or SS. For testing, it will be LOFT

• Time: Seconds into the experiment

• Seat: Seat of the pilot- 0 means left, 1 means right

EEG (Electroencephalogram) — EEG's role has been greatly overstated over the years, and it's definitely not a panacea of brain activity. Clinically, you can usefully tell if someone is awake, asleep, brain dead, having a seizure, and a handful of other things. EEG is a summation of all the electrical activity on the surface of the brain. This is the summation of all activities on the surface of the brain.

Data from 20 electrodes are given to us. Each electrode lead is placed near a particular part of the brain (prefrontal(fp), temporal(t), frontal(f), parietal(p), occipital(o), central(c)). The odd

numbers in the representation indicate that the electrode is placed on the left side of the brain, even numbers indicate the right side, and

z indicate the middle region.

The below figure gives an idea about the position of each electrode.

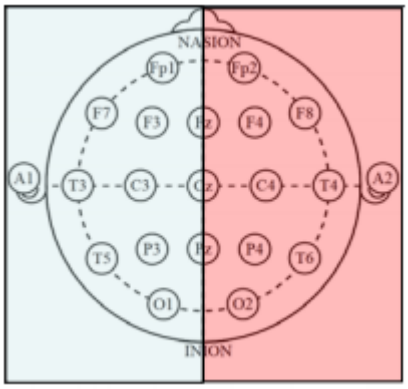


Figure 1: Position of electrodes in the scalp

• Eeg_f7: Data from the electrode near the prefrontal portion — left side

• Eeg_f8: Data from the electrode near the frontal area — right side

• Eeg_t4: Data from the electrode near the temporal area — right side

• Eeg_t6: Data from the electrode near the temporal area — right side

• Eeg_t5: Data from the electrode near the temporal area — left side

• Eeg_t3: Data from the electrode near the temporal area — left side

• Eeg_fp2: Data from the electrode near the prefrontal area — right side

• Eeg_o1: Data from the electrode near the occipital area — left side

• Eeg_p3: Data from the electrode near the parietal area — left side

• Eeg_pz: Data from the electrode near the parietal area — middle region

• Eeg_f3: Data from the electrode near the frontal area — left side

• Eeg_fz: Data from the electrode near the frontal area — middle region

• Eeg_f4: Data from the electrode near the frontal area — right side

• Eeg_c4: Data from the electrode near the central area — right side

• Eeg_p4: Data from the electrode near the parietal area — right side

• Eeg_poz: Data from the electrode near the parietal-occipital junction — Middle region

• Eeg_c3: Data from the electrode near the central area — left side

• Eeg_cz: Data from the electrode near the central area — middle region

• Eeg_o2: Data from the electrode near the occipital area — right side

• Ecg: Three-point electrocardiogram (ECG) signal — It measures the electrical activity of the heart (sensor output is in microvolts).

• R: Respiration sensor — It measures the rise and fall of the chest (Sensor output is in microvolts)

• Gsr: Galvanic skin response — The measure of electrodermal activity (Sensor output is in microvolts)

• Event: The output which is to be predicted — The state of the pilot at a given time. It will be

4.0 ML Formulation

4.1 Problem Type: - The model needs to predict one of the following cognitive states A (Baseline), B (SS), C (CA), and D (DA) of the pilot over a time during an experiment.

4.2 Types of data: - Data set here is of a time series nature and consists of medical science parameters like ECG, EEG, Respiration, and GSR which are electrical signals recorded over time.

4.3 Challenges to the data: - Dataset consists of physiological data collected from different machines being operated on 18 different pilots and wires of these machines tend to pick up unnecessary signals, hence there is some noise in the data

4.4 Business constraints: - It should be able to predict the pilot's cognitive state in real-time so that if a pilot is in a troubling state they could be alerted right away, preventing accidents and saving lives.

4.5 Performance metrics: - Multi-class log loss is used for performance evaluation of the model.

Multi-class log loss

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification


from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import plot_model
from matplotlib import pyplot


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import load_model
```

```
from tensorflow.keras.models import load_model
from sklearn.dummy import DummyClassifier
from sklearn.metrics import log_loss
```

Libraries important to visualize the data

```
!unzip '/content/drive/MyDrive/train.csv.zip'
```

```
Archive:  /content/drive/MyDrive/train.csv.zip
  inflating: train.csv
```

Unziping data to use csv file it has.

```
df= pd.read_csv('train.csv')
```

Loading the data

```
df
```

	crew	experiment	time	seat	eeg_fp1	eeg_f7	eeg_f8	eeg_
0	1	CA	0.011719	1	-5.285450	26.775801	-9.527310	-12.7932
1	1	CA	0.015625	1	-2.428420	28.430901	-9.323510	-3.7572
2	1	CA	0.019531	1	10.671500	30.420200	15.350700	24.7240
3	1	CA	0.023438	1	11.452500	25.609800	2.433080	12.4125
4	1	CA	0.027344	1	7.283210	25.942600	0.113564	5.7480
...	...	...	...	...	...	...	...	
4867416	13	SS	99.991005	1	-47.758202	-71.050400	-37.980801	-33.8529
4867417	13	SS	99.993004	0	-24.536699	79.787399	-45.435699	-6.3292
4867418	13	SS	99.994003	1	-42.078499	-66.937401	-27.298201	-25.9319
4867419	13	SS	99.997002	0	20.536301	10.485500	63.723400	44.4226
4867420	13	SS	99.998001	1	-12.230700	-39.962601	-2.675310	5.6991

4867421 rows × 28 columns

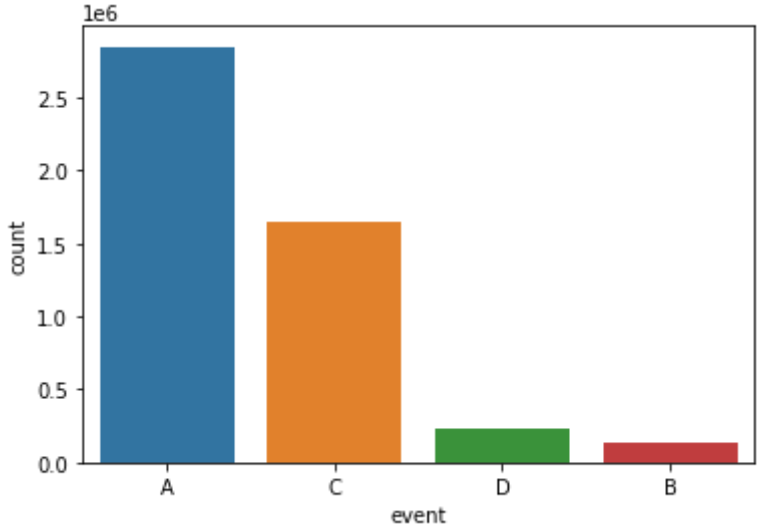
```
print(df['event'].value_counts())
```

```
A    2848809
C    1652686
D     235329
B     130597
Name: event, dtype: int64
```

Here A,B,C,D are the cognitive states and with value_counts we can observe that out data is imbalanced

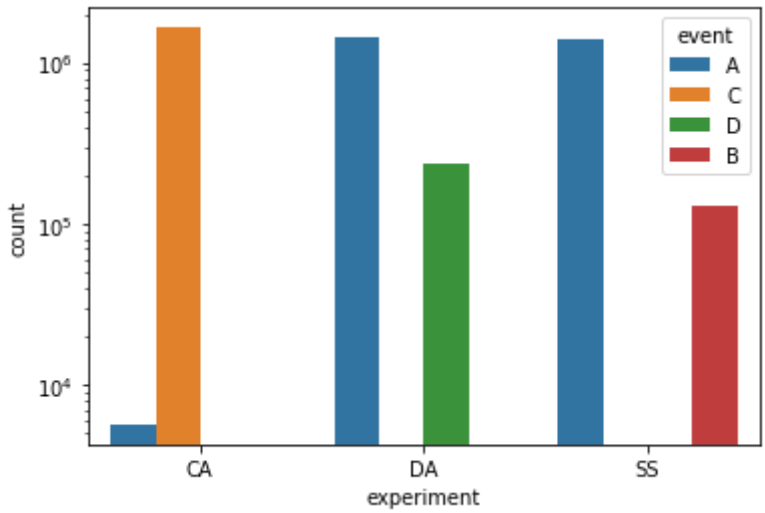
```
sns.countplot(df.event)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b1ab72d10>
```



We can see that our data is imblanced but seeing the relation between experiment and event we don't require to balance the data

```
sns.countplot(x='experiment',hue='event',data=df)
plt.yscale('log')
plt.show()
```

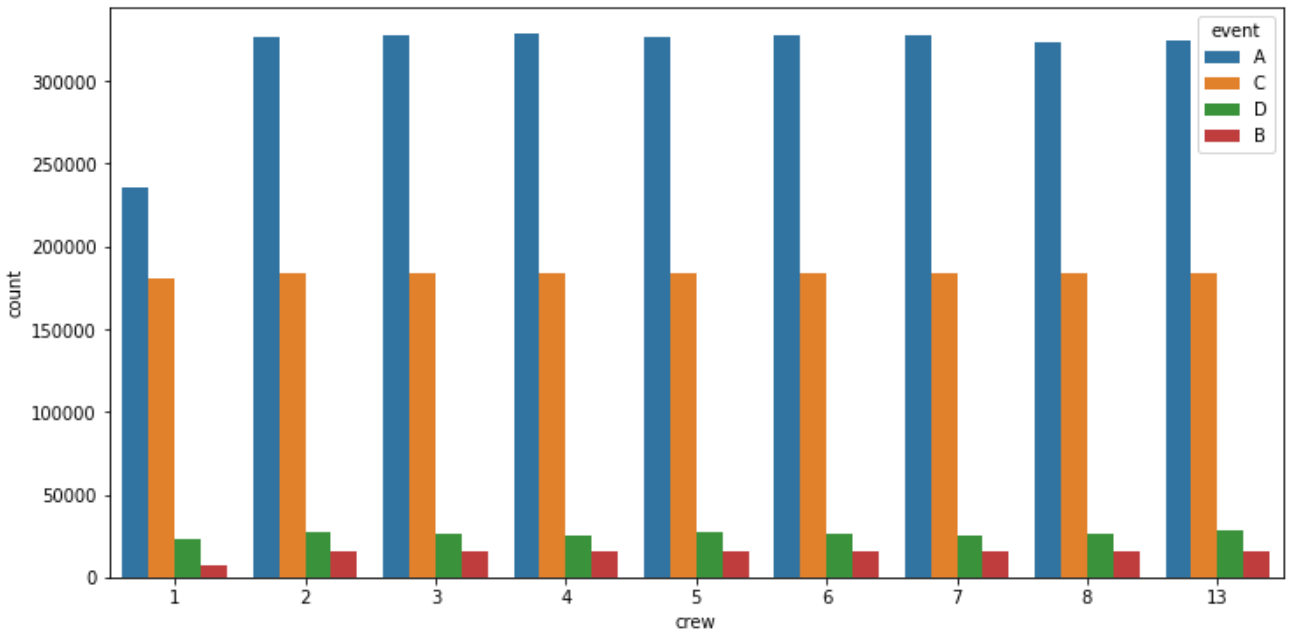


Observation of the above graph

1. In comparison with all the states CA is more to occur
2. In DA and SS, the no of experiments are more than the event
3. In CA the no of experiment is much less than the event CA.

```
plt.figure(figsize=(12,6))
dict_crew = dict(df.crew.value_counts())
for (key,value) in sorted(dict_crew.items()):
  print(key,value)

sns.countplot(x='crew',hue='event',data=df)
```

```
1 447652
2 552868
3 552795
4 552881
5 552815
6 552958
7 552769
8 549959
13 552724
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b1aaa14d0>
```



Observation

1. Here crew is devided into 9 parts
2. With this countplot we can visualize the distribution between all the crew is almost same

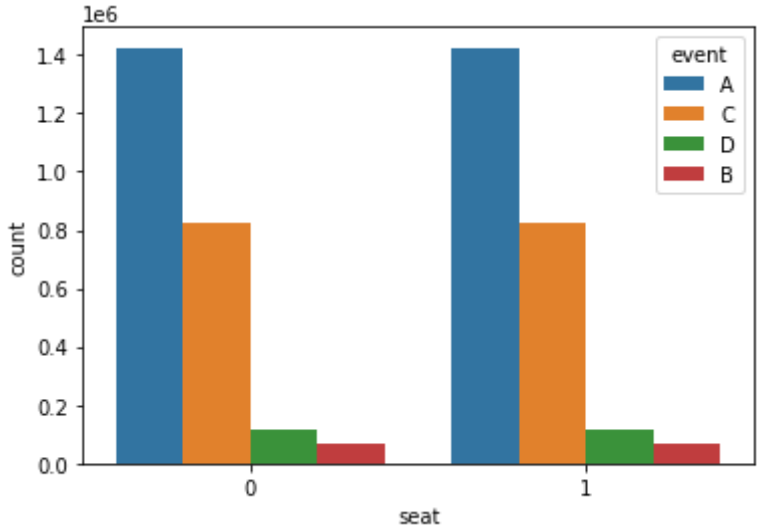
lets see a graph between all the events and see which cognitive state is more common to occur

```
dict_exp = dict(df.experiment.value_counts())
for (key,value) in sorted(dict_exp.items()):
  print(key,value)
```

```
CA 1658376
DA 1658393
SS 1550652
```

```
sns.countplot(x='seat',hue='event',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b1a3532d0>
```



Observation

1. One thing we could say before hand that seat would not help in prediction as mental state is independent for where the pilot sit.

```
plt.figure(figsize=(12,6))
sns.violinplot(x='event',y='time',data=df)
```

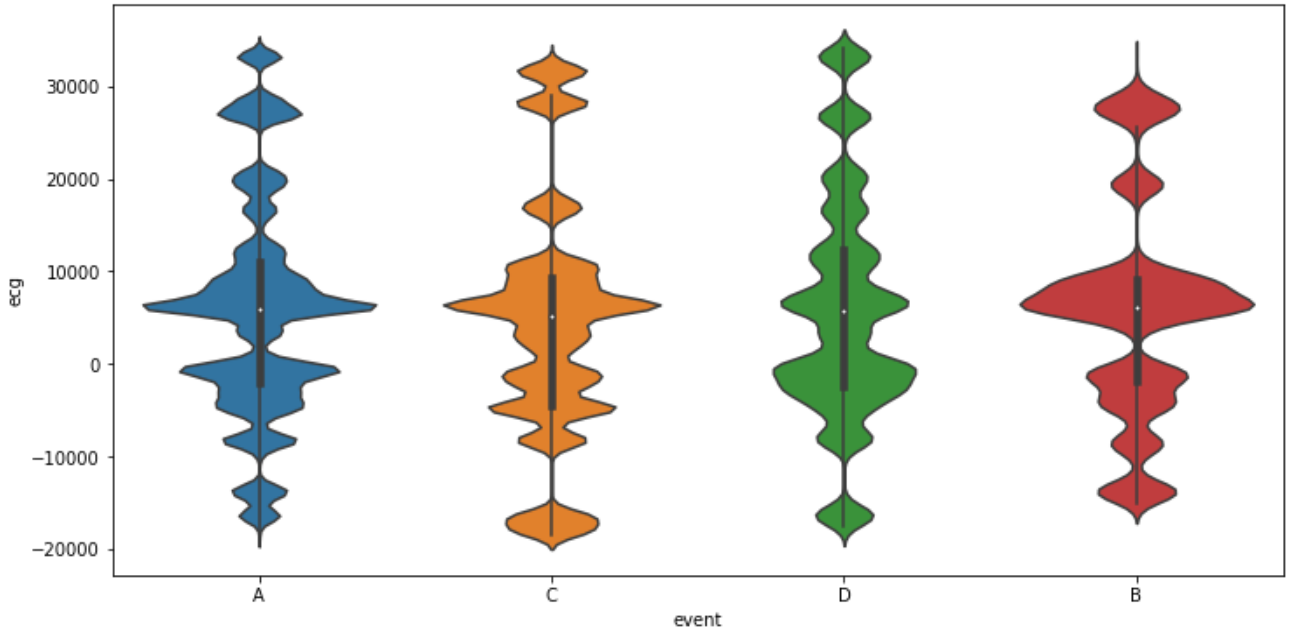
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b1a3ffc10>
```

Observation

1. Violin plots visualizes boxplot in it and shows denser , sparser regions in it.
2. Here the plot shows various patterns for all the cognitive states

```
plt.figure(figsize=(12,6))
sns.violinplot(x='event',y='ecg',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b1a39ae90>
```

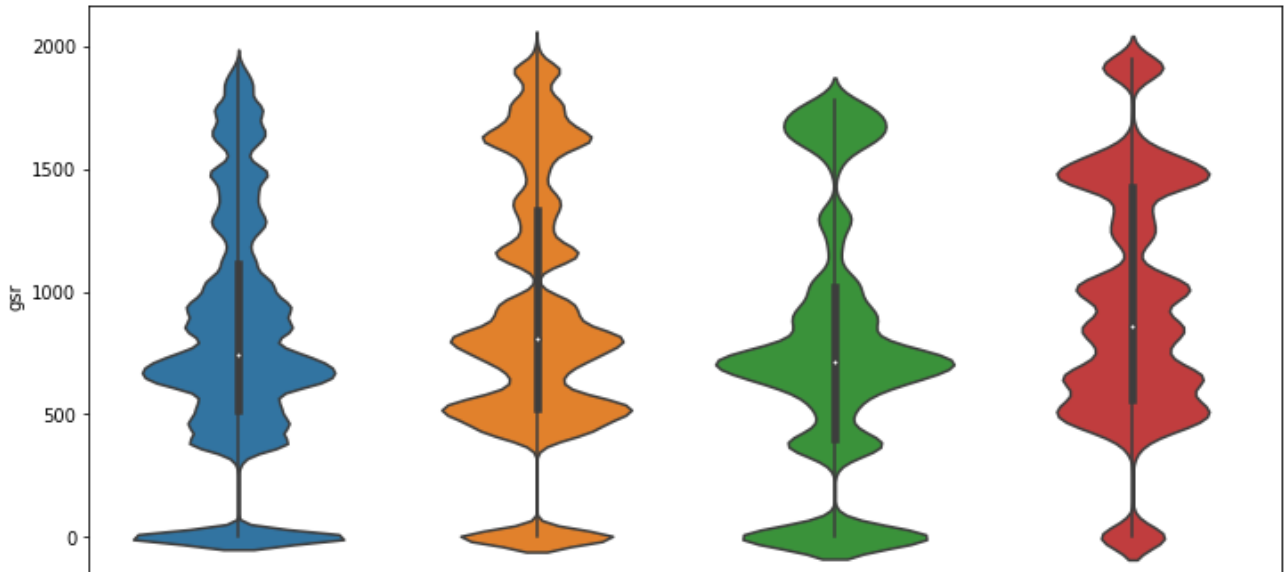


Observation between event and ecg

1. After visualizing the plot we could say that for some area value is missing and for some area the values are overlaping
2. For A between -10k to 20k it is denser region and for 20k to 30k it is sparser region
3. For C between -10k to 10k it is denser region and for 25k to 30k it is sparser region
4. For D -15k to 22k it is denser region and for -20k to -15k it is sparser region
5. for B 0 to 10k it is denser region and for 25k to 30k it is sparser region

```
plt.figure(figsize=(12,6))
sns.violinplot(x='event',y='gsr',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b4f5d7a90>
```

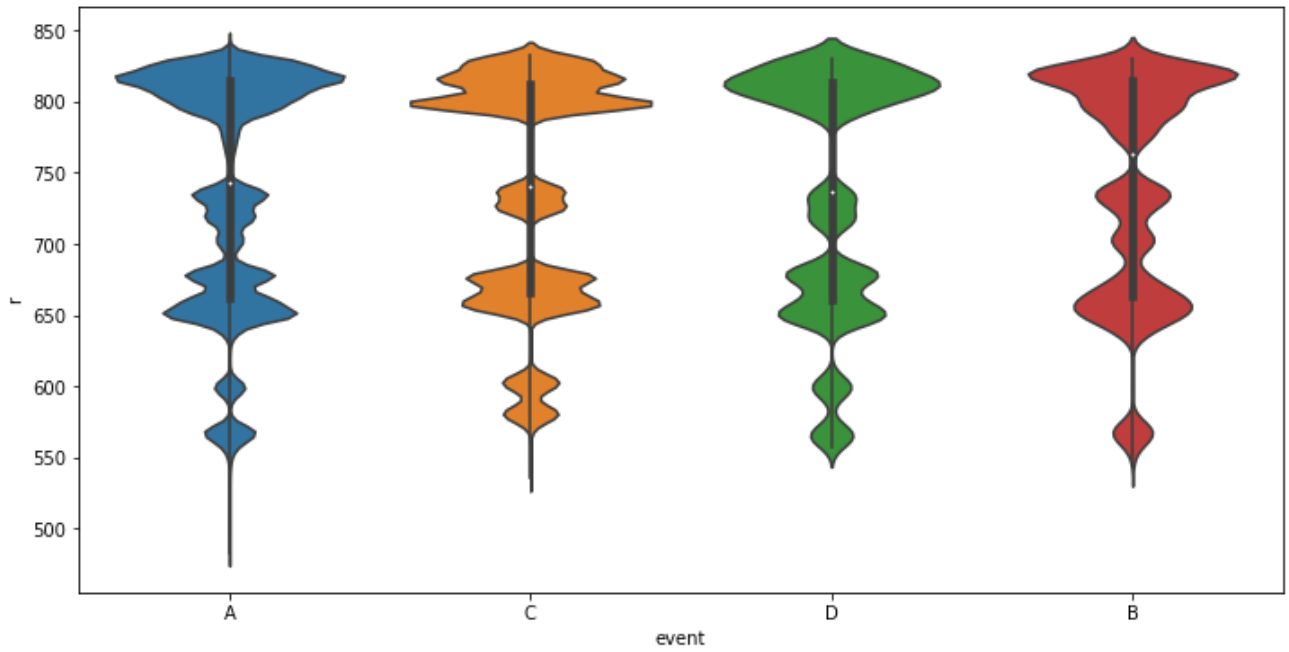


Observation

1. Here the regions in the bottom part show missing values for all the three cognitive state
2. Here the negative values are not there in visualization
3. Graph is similar to ecg

```
plt.figure(figsize=(12,6))
sns.violinplot(x='event',y='r',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b10ef9150>
```



Observation

1. here the we can say the range is defined betweem 500 to 850
2. shows respiration for all the 4 cognitive states

3. graphs looks simillar for all the 4 cognitive states
4. More the different the features are more better the features is .

Bivariate analysis

```
# sns.pairplot(df, hue="event")
```

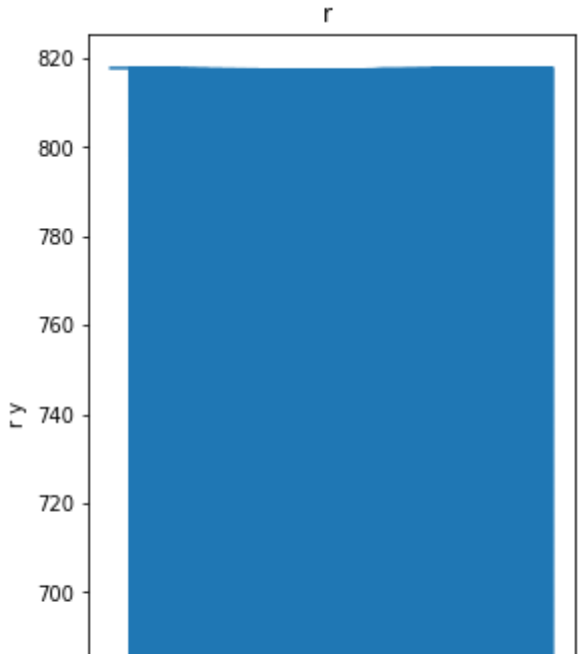
```
import scipy.signal as signal
# https://stackoverflow.com/questions/35588782/how-to-average-a-signal-to-remove-noise-wit
def noise_removal(noisy_data,Wn):
    N = 3
    B, A = signal.butter(N, Wn)
    return signal.filtfilt(B,A, noisy_data)
```

Noise removal in respiration data

```
df['r']
```

```
0          817.705994
1          817.705994
2          817.705994
3          817.705994
4          817.705994
              ...
4867416    758.981018
4867417    666.289978
4867418    758.981018
4867419    666.289978
4867420    758.981018
Name: r, Length: 4867421, dtype: float64
```

```
plt.figure(figsize=(20,40))
features = ['r']
for i,j in enumerate(features):
    plt.subplot(5, 4, i+1)
    plt.plot(df[j][:1000])
    plt.title("r ")
    plt.xlabel("indices")
    plt.ylabel("r y ")
```

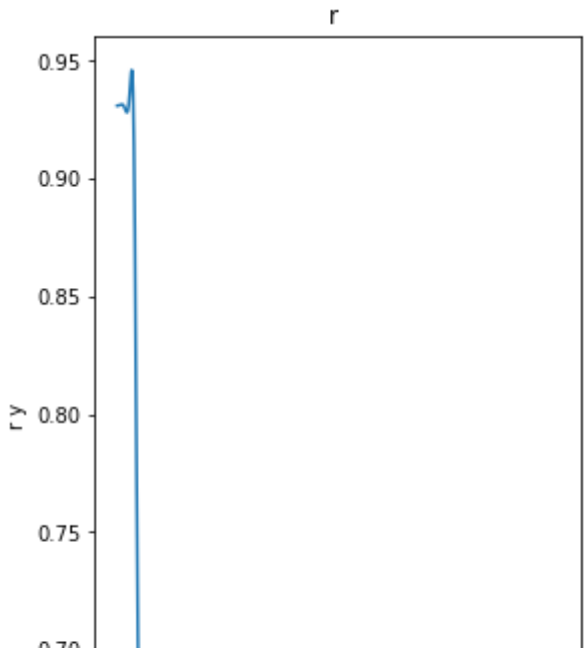


With graph we can say that out data has noise so to remove this noise I used butterfilter

```
for i in features:
  df[i]=noise_removal(df[i],0.1)
  df[i]=scaler.fit_transform(df[[i]])
  print(df['r'])
```

```
0          0.931052
1          0.931066
2          0.931093
3          0.931133
4          0.931187
             ...
4867416    0.616414
4867417    0.639896
4867418    0.667354
4867419    0.697692
4867420    0.729460
Name: r, Length: 4867421, dtype: float64
```

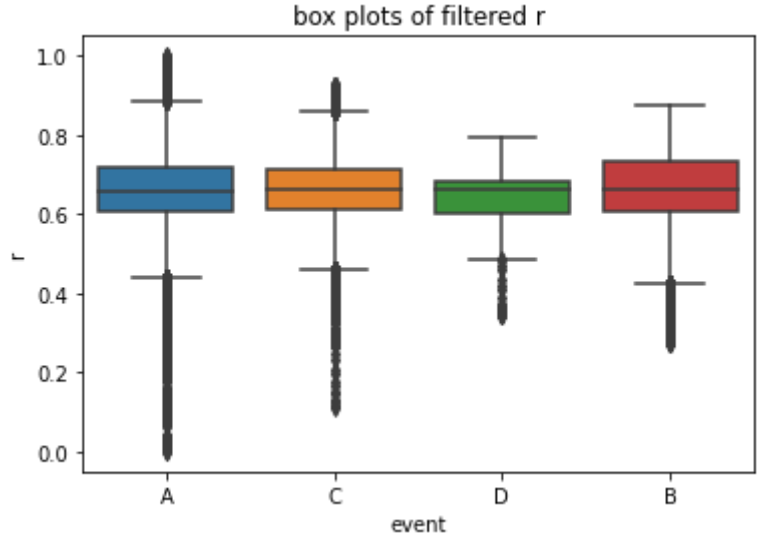
```
plt.figure(figsize=(20,40))
features = ['r']
for i,j in enumerate(features):
    plt.subplot(5, 4, i+1)
    plt.plot(df[j][:1000])
    plt.title("r ")
    plt.xlabel("indices")
    plt.ylabel("r y ")
```



After removing noise our data looks clean and we can clearly visualize the data

```
sns.boxplot(df["event"],df["r"])
plt.title("box plots of filtered r")
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```



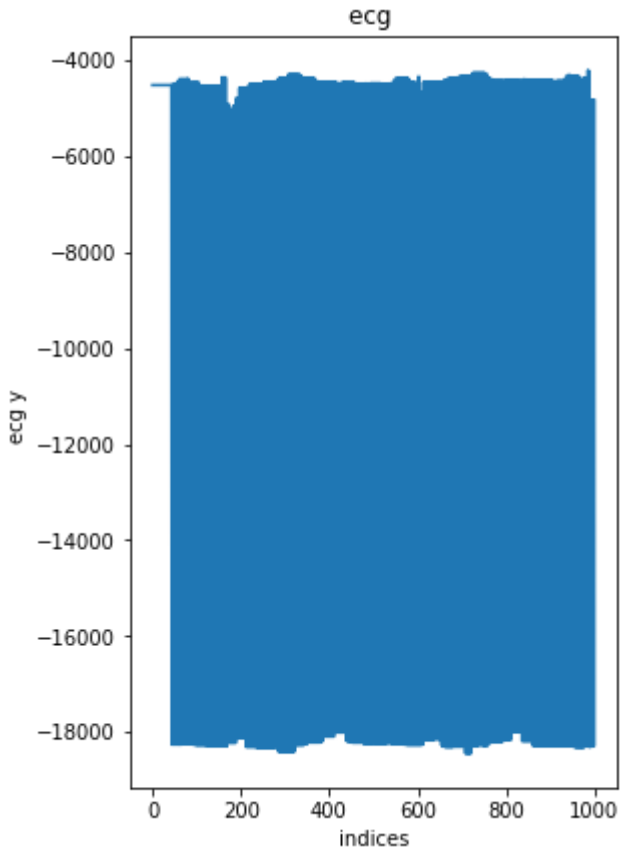
Noise removal in ECG data

```
df['ecg']
```

```
0          -4520.000000
1          -4520.000000
2          -4520.000000
```

```
3          -4520.000000
4          -4520.000000
                ...
4867416    -8395.570313
4867417   -13660.400391
4867418    -8395.570313
4867419   -13660.400391
4867420    -8395.570313
Name: ecg, Length: 4867421, dtype: float64
```

```
plt.figure(figsize=(20,40))
features = ['ecg']
for i,j in enumerate(features):
    plt.subplot(5, 4, i+1)
    plt.plot(df[j][:1000])
    plt.title("ecg ")
    plt.xlabel("indices")
    plt.ylabel("ecg y ")
```

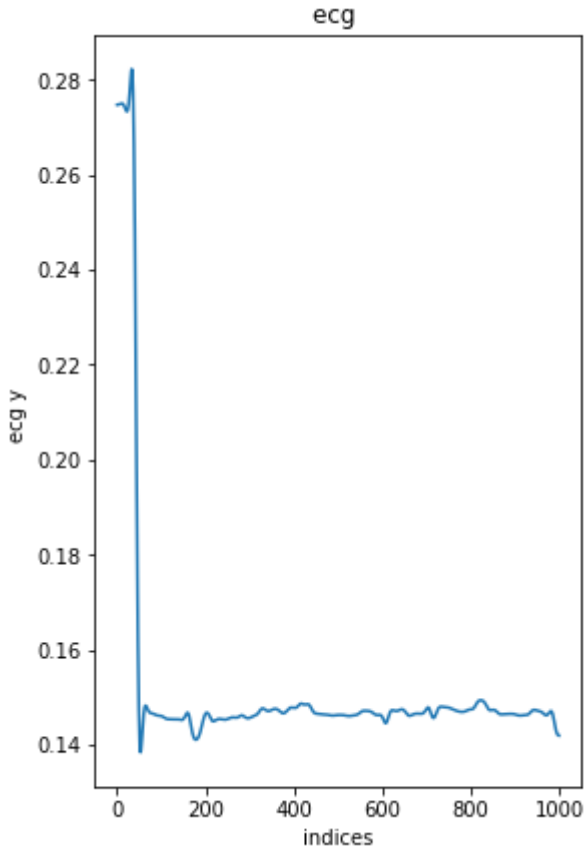



```
for i in features:
  df[i]=noise_removal(df[i],0.1)
  df[i]=scaler.fit_transform(df[[i]])
  print(df['ecg'])
```

```
0          0.274755
1          0.274762
2          0.274775
3          0.274795
4          0.274822
              ...
4867416    0.165976
```

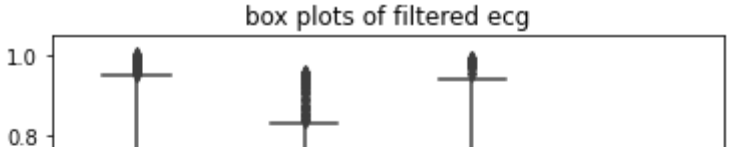
```
4867417    0.173328
4867418    0.181932
4867419    0.191441
4867420    0.201402
Name: ecg, Length: 4867421, dtype: float64
```

```
plt.figure(figsize=(20,40))
features = ['ecg']
for i,j in enumerate(features):
    plt.subplot(5, 4, i+1)
    plt.plot(df[j][:1000])
    plt.title("ecg ")
    plt.xlabel("indices")
    plt.ylabel("ecg y ")
```




```
sns.boxplot(df["event"],df["ecg"])
plt.title("box plots of filtered ecg")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
Text(0.5, 1.0, 'box plots of filtered ecg')
```

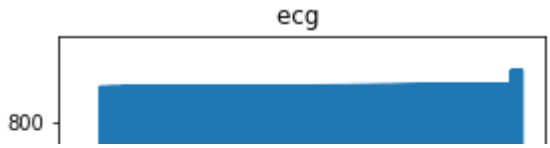


Noise removal in GSR data


```
df['gsr']
```

```
0          388.829987
1          388.829987
2          388.829987
3          388.829987
4          388.829987
              ...
4867416    658.107971
4867417    547.533997
4867418    658.107971
4867419    547.533997
4867420    658.107971
Name: gsr, Length: 4867421, dtype: float64
```

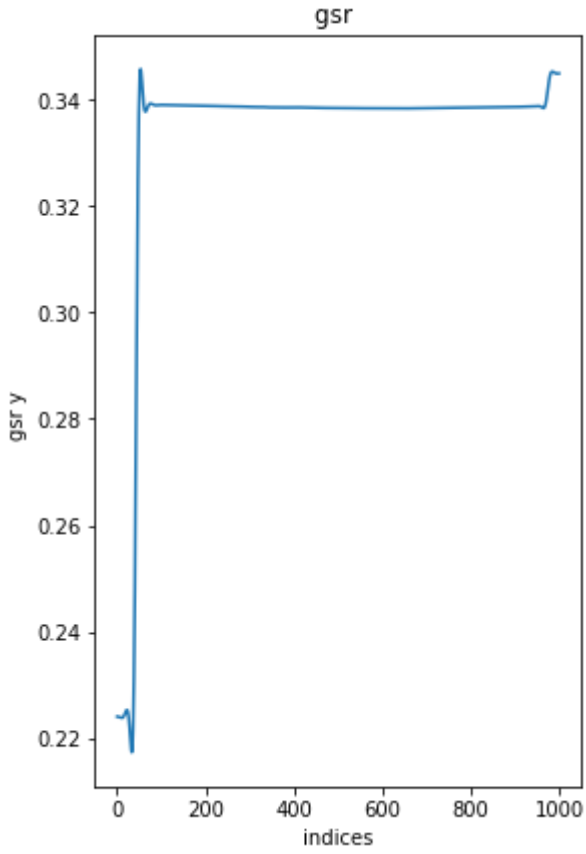
```
plt.figure(figsize=(20,40))
features = ['gsr']
for i,j in enumerate(features):
    plt.subplot(5, 4, i+1)
    plt.plot(df[j][:1000])
    plt.title("ecg ")
    plt.xlabel("indices")
    plt.ylabel("ecg y ")
```




```
for i in features:
  df[i]=noise_removal(df[i],0.1)
  df[i]=scaler.fit_transform(df[[i]])
  print(df['gsr'])
```

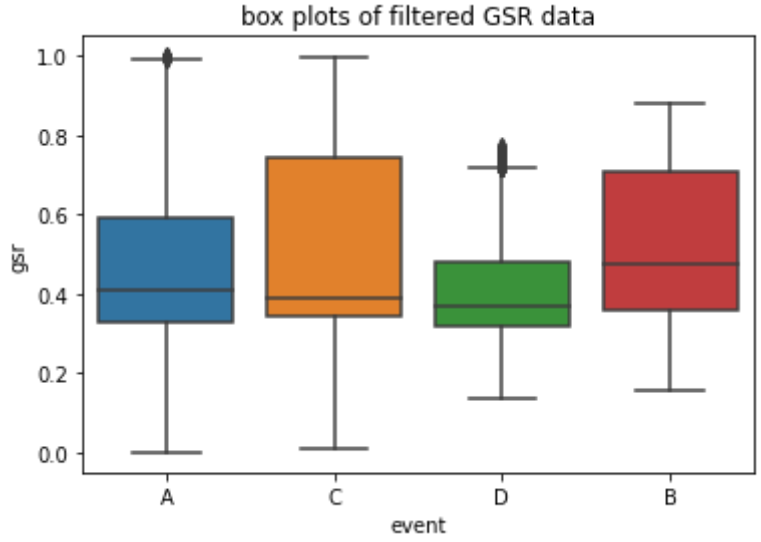
```
0          0.223995
1          0.223989
2          0.223977
3          0.223959
4          0.223935
             ...
4867416    0.345512
4867417    0.349876
4867418    0.354979
4867419    0.360618
4867420    0.366522
Name: gsr, Length: 4867421, dtype: float64
```

```
plt.figure(figsize=(20,40))
features = ['gsr']
for i,j in enumerate(features):
    plt.subplot(5, 4, i+1)
    plt.plot(df[j][:1000])
    plt.title("gsr ")
    plt.xlabel("indices")
    plt.ylabel("gsr y ")
```



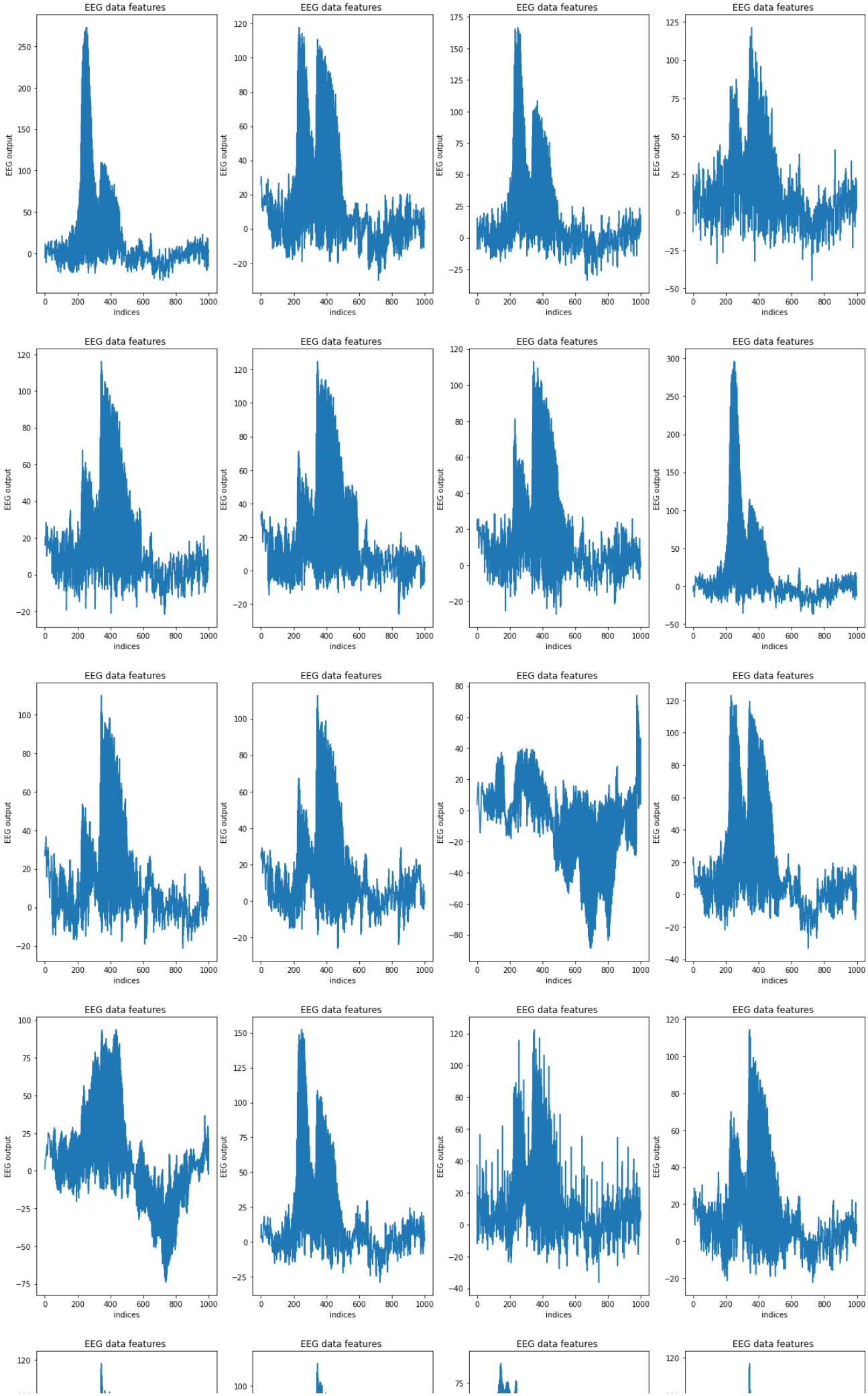
```
sns.boxplot(df["event"],df["gsr"])
plt.title("box plots of filtered GSR data")
plt.show()
```

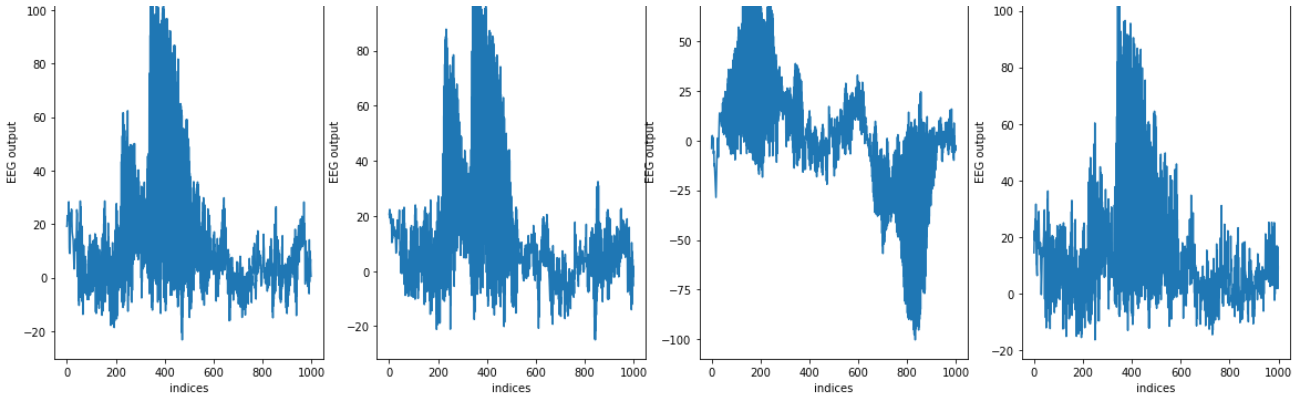
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```



Noise Removal in EEG data

```
plt.figure(figsize=(20,40))
features = ["eeg_fp1","eeg_f7","eeg_f8","eeg_t4","eeg_t6","eeg_t5","eeg_t3","eeg_fp2","eeg
for i,j in enumerate(features):
    plt.subplot(5, 4, i+1)
    plt.plot(df[j][:1000])
    plt.title("EEG data features ")
    plt.xlabel("indices")
    plt.ylabel("EEG output")

plt.show()
```


EEG data features
EEG output
indices

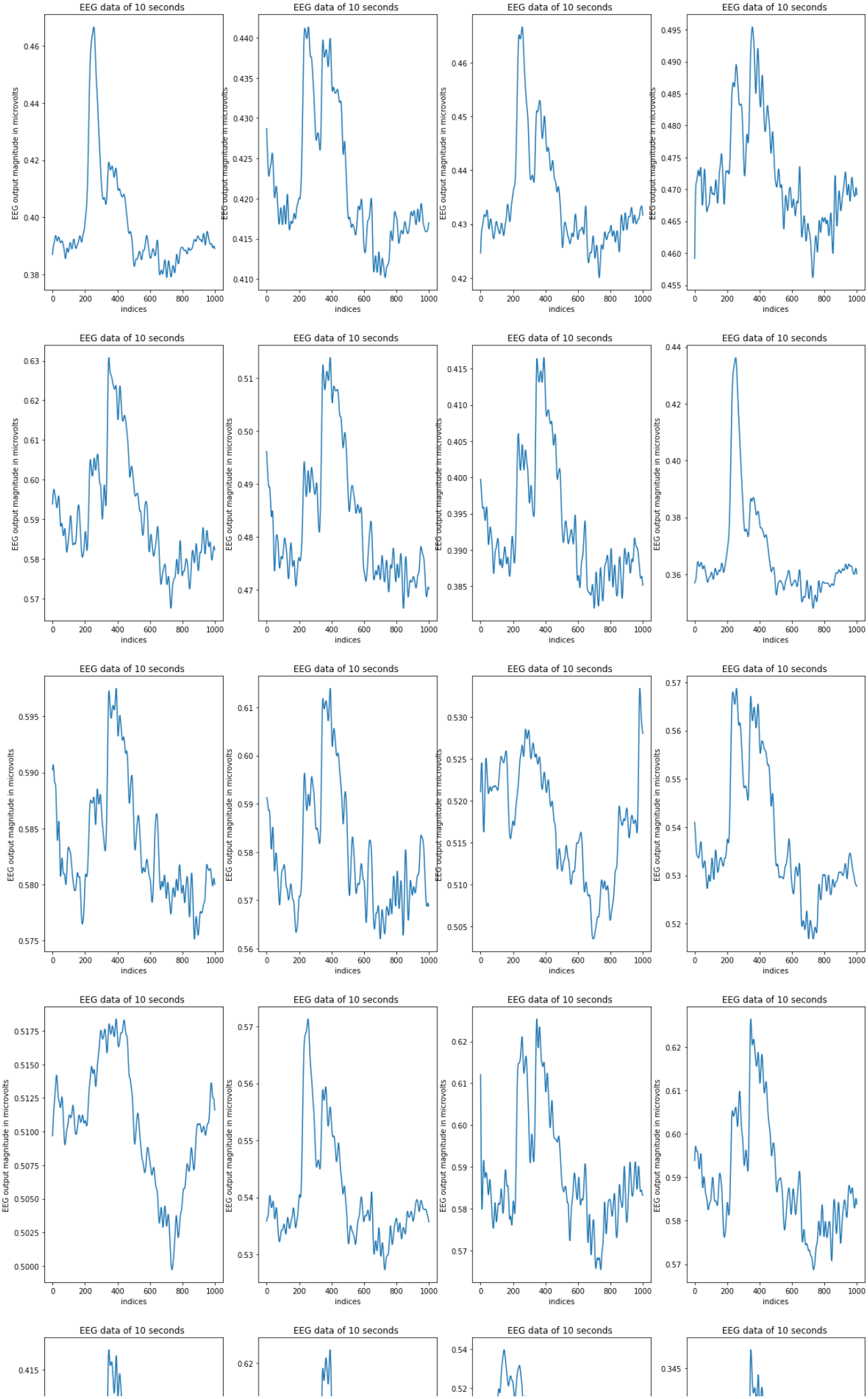


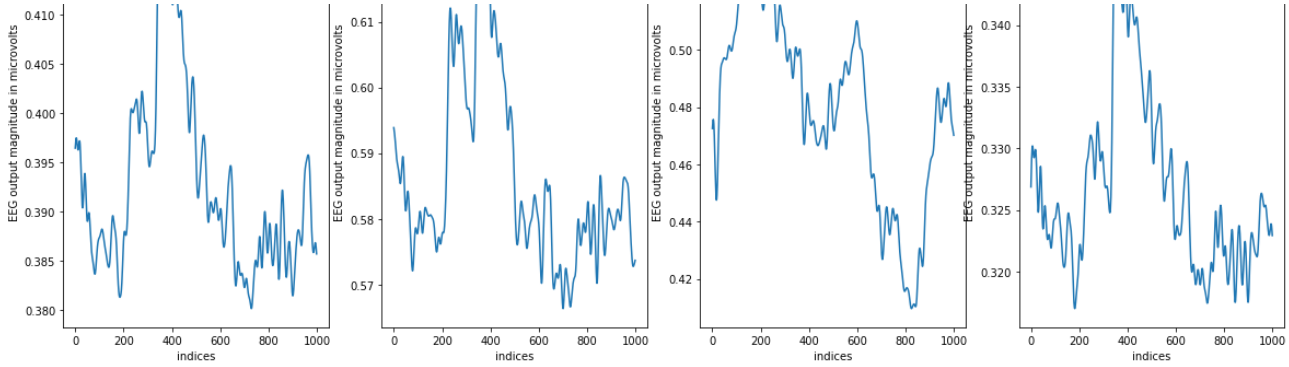

```
features = ["eeg_fp1","eeg_f7","eeg_f8","eeg_t4","eeg_t6","eeg_t5","eeg_t3","eeg_fp2","eeg
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
```

```
for i in features:
  df[i]=noise_removal(df[i],0.1)
  df[i]=scaler.fit_transform(df[[i]])
print(df[["eeg_fp1","eeg_f7","eeg_f8","eeg_t4","eeg_t6","eeg_t5","eeg_t3","eeg_fp2","eeg_o
```

```
            eeg_fp1    eeg_f7    eeg_f8  ...    eeg_c3    eeg_cz    eeg_o2
0          0.387046  0.428686  0.424623  ...  0.593875  0.472502  0.326867
1          0.387816  0.428217  0.425544  ...  0.593614  0.473759  0.327722
2          0.388530  0.427731  0.426408  ...  0.593335  0.474812  0.328513
3          0.389144  0.427218  0.427171  ...  0.593024  0.475491  0.329185
4          0.389635  0.426676  0.427806  ...  0.592672  0.475655  0.329700
...             ...       ...       ...  ...       ...       ...       ...
4867416    0.368885  0.393428  0.430548  ...  0.576525  0.460367  0.318230
4867417    0.371975  0.394283  0.429593  ...  0.577562  0.465460  0.319325
4867418    0.375441  0.395206  0.428769  ...  0.578592  0.470965  0.320423
4867419    0.379141  0.396137  0.428079  ...  0.579513  0.476647  0.321491
4867420    0.382919  0.397052  0.427499  ...  0.580336  0.482300  0.322502

[4867421 rows x 20 columns]
```

```
plt.figure(figsize=(20,40))
feats = ["eeg_fp1","eeg_f7","eeg_f8","eeg_t4","eeg_t6","eeg_t5","eeg_t3","eeg_fp2","eeg_o1
for i,j in enumerate(feats):
    plt.subplot(5, 4, i+1)
    plt.plot(df[j][:1000])
    plt.title("EEG data of 10 seconds ")
    plt.xlabel("indices")
    plt.ylabel("EEG output magnitude in microvolts")

plt.show()
```


EEG data of 10 seconds
EEG output magnitude in microvolts
indices



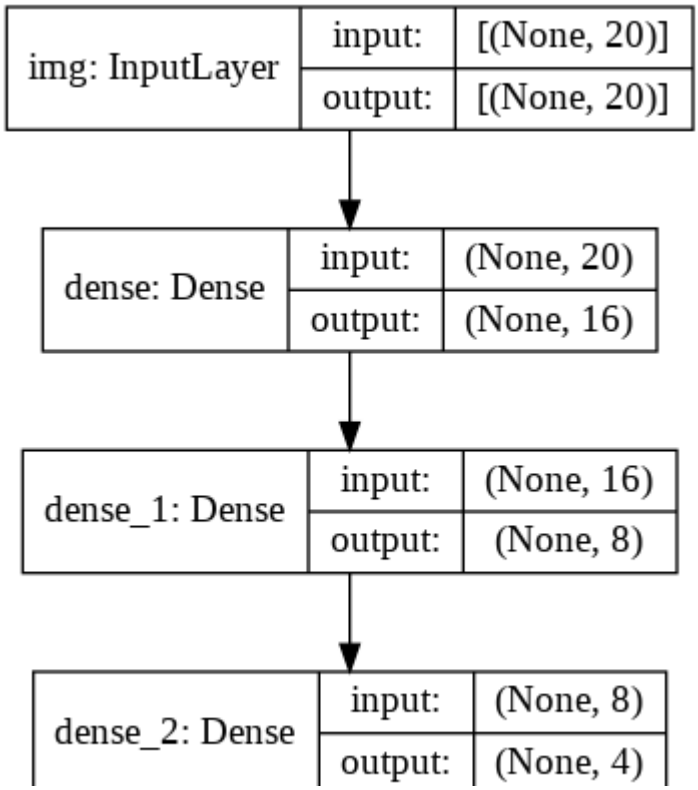
Feature Engineering on EEG DATA

```
features = ["eeg_fp1","eeg_f7","eeg_f8","eeg_t4","eeg_t6","eeg_t5","eeg_t3","eeg_fp2","eeg
X=df[features]
```

```
y=df['event']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
```

```
# visible = Input(shape=(n_inputs,))

encoder_input = keras.Input(shape=(20,), name="img")
x = Dense(16)(encoder_input)
x = Dense(8)(x)
# encoder_output = Dense(4)(x)
n_inputs=X_train.shape[1]
n_bottleneck = 4
bottleneck = Dense(n_bottleneck)(x)



# encoder = keras.Model(encoder_input, encoder_output, name="encoder")
# encoder.summary()


# decoder_input = keras.Input(shape=(4,), name="encoded_img")
# x = layers.Reshape((4, 1, 1))(decoder_input)
d=Dense(4)(bottleneck)
x = layers.Dense(8)(d)
x = layers.Dense(16)(x)
# decoder_output = Dense(20)(x)
output = Dense(n_inputs, activation='linear')(x)
# decoder = keras.Model(decoder_input, decoder_output, name="decoder")
# decoder.summary()
model = Model(inputs=encoder_input, outputs=output)


model.compile(optimizer='adam', loss='mse')
plot_model(model, 'autoencoder_no_compress.png', show_shapes=True)
```




```
history = model.fit(X_train, X_train, epochs=20, batch_size=16, verbose=2, validation_data
# plot loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

```
# define an encoder model (without the decoder)
encoder = Model(inputs=encoder_input, outputs=bottleneck)
plot_model(encoder, 'encoder_no_compress.png', show_shapes=True)
# save the encoder to file
encoder.save('encoder.h5')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-034a7d834309> in <module>()
----> 1 history = model.fit(X_train, X_train, epochs=20, batch_size=16, verbose=2,
validation_data=(X_test,X_test))
      2 # plot loss
      3 pyplot.plot(history.history['loss'], label='train')
      4 pyplot.plot(history.history['val_loss'], label='test')
      5 pyplot.legend()

NameError: name 'model' is not defined
```

SEARCH STACK OVERFLOW

```
encoder = load_model('/content/encoder.h5')
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was
```

```
# encode the train data
X_train_encode = encoder.predict(X_train)
# encode the test data
X_test_encode = encoder.predict(X_test)
```

```
X_train_encode.shape
```

```
(3261172, 4)
```

```
X_tr = np.hstack((X_train,X_train_encode))

X_te = np.hstack((X_test,X_test_encode ))

print(X_tr.shape,y_train.shape)
print(X_te.shape,y_test.shape)
```

```
(3261172, 24) (3261172,)
(1606249, 24) (1606249,)
```

Key take aways from the FE on eeg

Feature engineering is performed on eeg features Here I tried to use auto encoders to compress to 4 features out of 20 eeg features and using hstack combined 20+4 features then used random model got the log loss performance .

```
df.head()
```

	crew	experiment	time	seat	eeg_fp1	eeg_f7	eeg_f8	eeg_t4	eeg_t6
0	1	CA	0.011719	1	0.387046	0.428686	0.424623	0.459177	0.593822
1	1	CA	0.015625	1	0.387816	0.428217	0.425544	0.461508	0.594570
2	1	CA	0.019531	1	0.388530	0.427731	0.426408	0.463713	0.595288
3	1	CA	0.023438	1	0.389144	0.427218	0.427171	0.465683	0.595944
4	1	CA	0.027344	1	0.389635	0.426676	0.427806	0.467343	0.596511

```
x_final = np.vstack((X_train_encode ,X_test_encode))
```

```
df['eeg_fe_1'] = x_final[:,0]
df['eeg_fe_2'] = x_final[:,1]
df['eeg_fe_3'] = x_final[:,2]
df['eeg_fe_4'] = x_final[:,3]
```

```
feats= ['eeg_fe_1','eeg_fe_2','eeg_fe_3','eeg_fe_4']
for i in feats:
```

```
for i in feats:
    df[i]=scaler.fit_transform(df[[i]])

df
```

	crew	experiment	time	seat	eeg_fp1	eeg_f7	eeg_f8	eeg_t4	e
0	1	CA	0.011719	1	0.387046	0.428686	0.424623	0.459177	0.5
1	1	CA	0.015625	1	0.387816	0.428217	0.425544	0.461508	0.5
2	1	CA	0.019531	1	0.388530	0.427731	0.426408	0.463713	0.5
3	1	CA	0.023438	1	0.389144	0.427218	0.427171	0.465683	0.5
4	1	CA	0.027344	1	0.389635	0.426676	0.427806	0.467343	0.5
...	...	...	...	...	...	...	...	...	
4867416	13	SS	99.991005	1	0.368885	0.393428	0.430548	0.459547	0.5
4867417	13	SS	99.993004	0	0.371975	0.394283	0.429593	0.461101	0.5
4867418	13	SS	99.994003	1	0.375441	0.395206	0.428769	0.463158	0.5
4867419	13	SS	99.997002	0	0.379141	0.396137	0.428079	0.465587	0.5
4867420	13	SS	99.998001	1	0.382919	0.397052	0.427499	0.468233	0.5

4867421 rows × 32 columns

```
Y=df['event']

x=df.drop(['experiment','event'],axis=1)


X_trains, X_tests, y_trains, y_tests = train_test_split(x, Y, test_size=0.33, random_state
```

Base_line Solution

```
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(X_trains, y_trains)

y_pred=dummy_clf.predict_proba(X_tests)

Log=log_loss(y_tests, y_pred)
print(Log)
dummy_clf.get_params()
```

```
14.3046422941805
{'constant': None, 'random_state': None, 'strategy': 'most_frequent'}
```

```
df['event'].unique()
```

```
array(['A', 'C', 'D', 'B'], dtype=object)
```

logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import RandomizedSearchCV
```

```
cfl=SGDClassifier(loss='log',max_iter=1000)
prams={
    'alpha':[0.001,0.005,0.01],
    'penalty':['l1','l2']
}
lr_cfl=GridSearchCV(x_cfl,prams,verbose=10)
lr_cfl.fit(X_trains,y_trains)
print(lr_cfl.best_estimator_)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] alpha=0.001, penalty=l1 .....................................
[CV] ............. alpha=0.001, penalty=l1, score=0.626, total= 1.2min
[CV] alpha=0.001, penalty=l1 .....................................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.2min remaining:    0.0s
[CV] ............. alpha=0.001, penalty=l1, score=0.623, total= 1.2min
[CV] alpha=0.001, penalty=l1 .....................................
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  2.4min remaining:    0.0s
[CV] ............. alpha=0.001, penalty=l1, score=0.627, total= 1.3min
[CV] alpha=0.001, penalty=l1 .....................................
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  3.6min remaining:    0.0s
[CV] ............. alpha=0.001, penalty=l1, score=0.626, total= 1.2min
[CV] alpha=0.001, penalty=l1 .....................................
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:  4.9min remaining:    0.0s
[CV] ............. alpha=0.001, penalty=l1, score=0.625, total= 1.2min
[CV] alpha=0.001, penalty=l2 .....................................
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  6.1min remaining:    0.0s
[CV] ............. alpha=0.001, penalty=l2, score=0.479, total= 1.9min
[CV] alpha=0.001, penalty=l2 .....................................
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:  8.0min remaining:    0.0s
[CV] ............. alpha=0.001, penalty=l2, score=0.625, total= 2.0min
[CV] alpha=0.001, penalty=l2 .....................................
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:  9.9min remaining:    0.0s
[CV] ............. alpha=0.001, penalty=l2, score=0.623, total= 2.0min
[CV] alpha=0.001, penalty=l2 .....................................
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed: 11.9min remaining:    0.0s
[CV] ............. alpha=0.001, penalty=l2, score=0.628, total= 2.0min
[CV] alpha=0.001, penalty=l2 .....................................
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed: 13.9min remaining:    0.0s
```

```
[CV] ..... alpha=0.001, penalty=l2, score=0.626, total= 1.9min
[CV] alpha=0.005, penalty=l1 .....................................
[CV] ..... alpha=0.005, penalty=l1, score=0.596, total=  46.1s
[CV] alpha=0.005, penalty=l1 .....................................
[CV] ..... alpha=0.005, penalty=l1, score=0.585, total=  48.8s
[CV] alpha=0.005, penalty=l1 .....................................
[CV] ..... alpha=0.005, penalty=l1, score=0.604, total=  47.3s
[CV] alpha=0.005, penalty=l1 .....................................
[CV] ..... alpha=0.005, penalty=l1, score=0.563, total=  46.4s
[CV] alpha=0.005, penalty=l1 .....................................
[CV] ..... alpha=0.005, penalty=l1, score=0.585, total=  47.5s
[CV] alpha=0.005, penalty=l2 .....................................
[CV] ..... alpha=0.005, penalty=l2, score=0.608, total= 1.1min
[CV] alpha=0.005, penalty=l2 .....................................
[CV] ..... alpha=0.005, penalty=l2, score=0.585, total= 1.0min
[CV] alpha=0.005, penalty=l2 .....................................
[CV] ..... alpha=0.005, penalty=l2, score=0.585, total= 1.0min
[CV] alpha=0.005, penalty=l2 .....................................
[CV] ..... alpha=0.005, penalty=l2, score=0.600, total= 1.0min
[CV] alpha=0.005, penalty=l2 .....................................
[CV] ..... alpha=0.005, penalty=l2, score=0.589, total= 1.0min
[CV] alpha=0.01, penalty=l1 ......................................
[CV] ..... alpha=0.01, penalty=l1, score=0.585, total=  42.4s
[CV] alpha=0.01, penalty=l1 ......................................
[CV] ..... alpha=0.01, penalty=l1, score=0.585, total=  43.8s
[CV] alpha=0.01, penalty=l1 ......................................
[CV] ..... alpha=0.01, penalty=l1, score=0.585, total=  44.9s
[CV] alpha=0.01, penalty=l1 ......................................
[CV] ..... alpha=0.01, penalty=l1, score=0.585, total=  42.7s
```

```
logistic_model=SGDClassifier(alpha=0.001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=1000,
              n_iter_no_change=5, n_jobs=None, penalty='l1', power_t=0.5,
              random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
logistic_model.fit(X_trains,y_trains)
```

```
SGDClassifier(alpha=0.001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=1000,
              n_iter_no_change=5, n_jobs=None, penalty='l1', power_t=0.5,
              random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

```
y_pred=logistic_model.predict_proba(X_tests)

Log=log_loss(y_tests, y_pred)
print(Log)
# https://github.com/WillKoehrsen/Machine-Learning-Projects/blob/master/random_forest_expl
```

```
0.9109333704940372
```

```
# from sklearn.model_selection import RandomizedSearchCV
# import seaborn as sea
# from xgboost import XGBClassifier
```

```
# xgb_clf = XGBClassifier()

# parameters = {'learning_rate' : [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3] , 'n_estimators':[5

# classifier = RandomizedSearchCV(xgb_clf, parameters, cv=2, scoring='neg_log_loss',return
# classifier.fit(X_trains, y_trains)


from sklearn.ensemble import RandomForestClassifier
# https://github.com/WillKoehrsen/Machine-Learning-Projects/blob/master/random_forest_expl
random_forest_cfl=RandomForestClassifier()
prams={
     "n_estimators":[50],
     "max_depth":[40,50,60],
     "min_samples_split":[2,3,4,6,],
     "min_samples_leaf": [1,2,3]
}
random_cfl=RandomizedSearchCV(random_forest_cfl,param_distributions=prams,cv=3,verbose=10,
random_cfl.fit(X_train,y_train)
print(random_cfl.best_estimator_)a
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed: 14.7min
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed: 14.7min
[Parallel(n_jobs=-1)]: Done   3 tasks      | elapsed: 14.7min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 14.7min
---------------------------------------------------------------------------
TerminatedWorkerError                     Traceback (most recent call last)
<ipython-input-45-065ee4e04440> in <module>()
      9 }
     10
random_cfl=RandomizedSearchCV(random_forest_cfl,param_distributions=prams,cv=3,verbos

---> 11 random_cfl.fit(X_train,y_train)
     12 print(random_cfl.best_estimator_)

                                  7 frames
/usr/lib/python3.7/concurrent/futures/_base.py in __get_result(self)
    382     def __get_result(self):
    383         if self._exception:
--> 384             raise self._exception
    385         else:
    386             return self._result

TerminatedWorkerError: A worker process managed by the executor was unexpectedly
terminated. This could be caused by a segmentation fault while calling the function
or by an excessive memory usage causing the Operating System to kill the worker.

The exit codes of the workers are {SIGKILL(-9)}
```

```
from sklearn.ensemble import RandomForestClassifier
# https://github.com/WillKoehrsen/Machine-Learning-Projects/blob/master/random_forest_expl
```

```
random_forest_cfl=RandomForestClassifier()
prams={
     "n_estimators":[30],
     "max_depth":[5,10,15],
     "min_samples_split":[2,3,4,6],
     "min_samples_leaf": [1,2,3]
}
random_cfl=RandomizedSearchCV(random_forest_cfl,param_distributions=prams,cv=3,verbose=10,
random_cfl.fit(X_trains,y_trains)
print(random_cfl.best_estimator_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] n_estimators=30, min_samples_split=6, min_samples_leaf=3, max_depth=15
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-48-d5350ab82160> in <module>()
      9 }
     10
random_cfl=RandomizedSearchCV(random_forest_cfl,param_distributions=prams,cv=3,verbos

---> 11 random_cfl.fit(X_trains,y_trains)
     12 print(random_cfl.best_estimator_)

 21 frames
/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py in fit(self, X, y,
sample_weight, check_input, X_idx_sorted)
    365                                            min_impurity_split)
    366
--> 367         builder.build(self.tree_, X, y, sample_weight, X_idx_sorted)
    368
    369         if self.n_outputs_ == 1 and is_classifier(self):

KeyboardInterrupt:
```

```
random_forest_cfl=RandomForestClassifier()
prams={
     "n_estimators":[30],
     "max_depth":[15],
     "min_samples_split":[4],
     "min_samples_leaf": [1]
}
random_cfl=RandomizedSearchCV(random_forest_cfl,param_distributions=prams,cv=3,verbose=10,
random_cfl.fit(X_trains,y_trains)
print(random_cfl.best_estimator_)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py:281: UserWa
  % (grid_size, self.n_iter, grid_size), UserWarning)
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] n_estimators=30, min_samples_split=4, min_samples_leaf=1, max_depth=15
[CV]  n_estimators=30, min_samples_split=4, min_samples_leaf=1, max_depth=15, score=(
[CV] n_estimators=30, min_samples_split=4, min_samples_leaf=1, max_depth=15
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed: 10.1min remaining:    0.0s
[CV]  n_estimators=30, min_samples_split=4, min_samples_leaf=1, max_depth=15, score=(
[CV] n_estimators=30, min_samples_split=4, min_samples_leaf=1, max_depth=15
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed: 19.8min remaining:    0.0s
```

```
[CV]  n_estimators=30, min_samples_split=4, min_samples_leaf=1, max_depth=15, score=(
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed: 29.4min remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed: 29.4min finished
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=15, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=4,
                       min_weight_fraction_leaf=0.0, n_estimators=30,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
import pickle
pickle.dump(random_cfl,open('Random_Forest_load','wb'))


y_train_pred=random_cfl.predict_proba(X_trains)
y_test_pred=random_cfl.predict_proba(X_tests)
Log_train=log_loss(y_trains,y_train_pred)
Log_test=log_loss(y_tests, y_test_pred)
print(Log_train)
print(Log_test)
```

LightGBM

```
import lightgbm
from lightgbm import LGBMClassifier


params = {"n_estimators":[10,20,30,50,100],
 "num_leaves" : range(1,50),
 "learning_rate" : [1e-4,0.0001,0.001,0.01,0.1],
 "bagging_fraction" : list(np.arange(0.0,1.0,0.1)),
 "colsample_bytree" : list(np.arange(0.0,1.0,0.1)),
 'min_data_in_leaf':[1,10,20,50,80,100]
 }



model=LGBMClassifier()
model_LGBM=RandomizedSearchCV(model,param_distributions=params,verbose=10,return_train_sco



model_LGBM.fit(X_trains,y_trains)
print(model_LGBM.best_estimator_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] num_leaves=47, n_estimators=50, min_data_in_leaf=100, learning_rate=0.001, co
[CV]  num_leaves=47, n_estimators=50, min_data_in_leaf=100, learning_rate=0.001, c
[CV] num_leaves=47, n_estimators=50, min_data_in_leaf=100, learning_rate=0.001, co
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   45.7s remaining:    0.0s
[CV]  num_leaves=47, n_estimators=50, min_data_in_leaf=100, learning_rate=0.001, c
[CV] num_leaves=47, n_estimators=50, min_data_in_leaf=100, learning_rate=0.001, cc
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  1.5min remaining:    0.0s
[CV]  num_leaves=47, n_estimators=50, min_data_in_leaf=100, learning_rate=0.001, c
[CV] num_leaves=47, n_estimators=50, min_data_in_leaf=100, learning_rate=0.001, cc
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  2.2min remaining:    0.0s
[CV]  num_leaves=47, n_estimators=50, min_data_in_leaf=100, learning_rate=0.001, c
[CV] num_leaves=47, n_estimators=50, min_data_in_leaf=100, learning_rate=0.001, cc
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:  2.9min remaining:    0.0s
[CV]  num_leaves=47, n_estimators=50, min_data_in_leaf=100, learning_rate=0.001, c
[CV] num_leaves=3, n_estimators=30, min_data_in_leaf=50, learning_rate=0.0001, col
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  3.6min remaining:    0.0s
[CV]  num_leaves=3, n_estimators=30, min_data_in_leaf=50, learning_rate=0.0001, cc
[CV] num_leaves=3, n_estimators=30, min_data_in_leaf=50, learning_rate=0.0001, col
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:  3.7min remaining:    0.0s
[CV]  num_leaves=3, n_estimators=30, min_data_in_leaf=50, learning_rate=0.0001, cc
[CV] num_leaves=3, n_estimators=30, min_data_in_leaf=50, learning_rate=0.0001, col
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:  3.9min remaining:    0.0s
[CV]  num_leaves=3, n_estimators=30, min_data_in_leaf=50, learning_rate=0.0001, cc
[CV] num_leaves=3, n_estimators=30, min_data_in_leaf=50, learning_rate=0.0001, col
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:  4.0min remaining:    0.0s
[CV]  num_leaves=3, n_estimators=30, min_data_in_leaf=50, learning_rate=0.0001, cc
[CV] num_leaves=3, n_estimators=30, min_data_in_leaf=50, learning_rate=0.0001, col
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:  4.1min remaining:    0.0s
[CV]  num_leaves=3, n_estimators=30, min_data_in_leaf=50, learning_rate=0.0001, cc
[CV] num_leaves=25, n_estimators=30, min_data_in_leaf=80, learning_rate=0.1, colsa
[CV]  num_leaves=25, n_estimators=30, min_data_in_leaf=80, learning_rate=0.1, cols
[CV] num_leaves=25, n_estimators=30, min_data_in_leaf=80, learning_rate=0.1, colsa
[CV]  num_leaves=25, n_estimators=30, min_data_in_leaf=80, learning_rate=0.1, cols
[CV] num_leaves=25, n_estimators=30, min_data_in_leaf=80, learning_rate=0.1, colsa
[CV]  num_leaves=25, n_estimators=30, min_data_in_leaf=80, learning_rate=0.1, cols
[CV] num_leaves=25, n_estimators=30, min_data_in_leaf=80, learning_rate=0.1, colsa
[CV]  num_leaves=25, n_estimators=30, min_data_in_leaf=80, learning_rate=0.1, cols
[CV] num_leaves=25, n_estimators=30, min_data_in_leaf=80, learning_rate=0.1, colsa
[CV]  num_leaves=25, n_estimators=30, min_data_in_leaf=80, learning_rate=0.1, cols
[CV] num_leaves=5, n_estimators=20, min_data_in_leaf=100, learning_rate=0.01, cols
[CV]  num_leaves=5, n_estimators=20, min_data_in_leaf=100, learning_rate=0.01, col
[CV] num_leaves=5, n_estimators=20, min_data_in_leaf=100, learning_rate=0.01, cols
[CV]  num_leaves=5, n_estimators=20, min_data_in_leaf=100, learning_rate=0.01, col
[CV] num_leaves=5, n_estimators=20, min_data_in_leaf=100, learning_rate=0.01, cols
[CV]  num_leaves=5, n_estimators=20, min_data_in_leaf=100, learning_rate=0.01, col
[CV] num_leaves=5, n_estimators=20, min_data_in_leaf=100, learning_rate=0.01, cols
[CV]  num_leaves=5, n_estimators=20, min_data_in_leaf=100, learning_rate=0.01, col
[CV] num_leaves=5, n_estimators=20, min_data_in_leaf=100, learning_rate=0.01, cols
[CV]  num_leaves=5, n_estimators=20, min_data_in_leaf=100, learning_rate=0.01, col
[CV] num_leaves=18, n_estimators=100, min_data_in_leaf=10, learning_rate=0.0001, c
[CV]  num_leaves=18, n_estimators=100, min_data_in_leaf=10, learning_rate=0.0001,
[CV] num_leaves=18, n_estimators=100, min_data_in_leaf=10, learning_rate=0.0001, c
[CV]  num_leaves=18, n_estimators=100, min_data_in_leaf=10, learning_rate=0.0001,
[CV] num_leaves=18, n_estimators=100, min_data_in_leaf=10, learning_rate=0.0001, c
[CV]  num_leaves=18, n_estimators=100, min_data_in_leaf=10, learning_rate=0.0001,
```

```
model_LGBM=LGBMClassifier(bagging_fraction=0.1, boosting_type='gbdt', class_weight=None,
               colsample_bytree=0.7000000000000001, importance_type='split',
               learning_rate=0.1, max_depth=-1, min_child_samples=20,
               min_child_weight=0.001, min_data_in_leaf=80, min_split_gain=0.0,
               n_estimators=30, n_jobs=-1, num_leaves=25, objective=None,
               random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
```

```
                subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
model_LGBM.fit(X_trains,y_trains)
```

```
LGBMClassifier(bagging_fraction=0.1, boosting_type='gbdt', class_weight=None,
               colsample_bytree=0.7000000000000001, importance_type='split',
               learning_rate=0.1, max_depth=-1, min_child_samples=20,
               min_child_weight=0.001, min_data_in_leaf=80, min_split_gain=0.0,
               n_estimators=30, n_jobs=-1, num_leaves=25, objective=None,
               random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
               subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

```
y_train_pred=model_LGBM.predict_proba(X_trains)
y_test_pred=model_LGBM.predict_proba(X_tests)
Log_train_LGBM=log_loss(y_trains,y_train_pred)
Log_test_LGBM=log_loss(y_tests, y_test_pred)
print(Log_train_LGBM)
print(Log_test_LGBM)
```

```
0.3442437083104418
0.34412745341105566
```

```
import joblib
joblib.dump(model_LGBM,'lgbm_better.pkl')
```

```
['lgbm_better.pkl']
```

AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier

params = {"n_estimators":[50],
        "learning_rate" : [0.01,0.1]
          }
```

```
AdaBoost=AdaBoostClassifier()
Adaboost = GridSearchCV(AdaBoost,params,verbose=10)
Adaboost.fit(X_trains.iloc[2174115:],y_trains.iloc[2174115:])
```