



Brain electroencephalogram (EEG) activity analysis on big data platform

The final project of Data 603

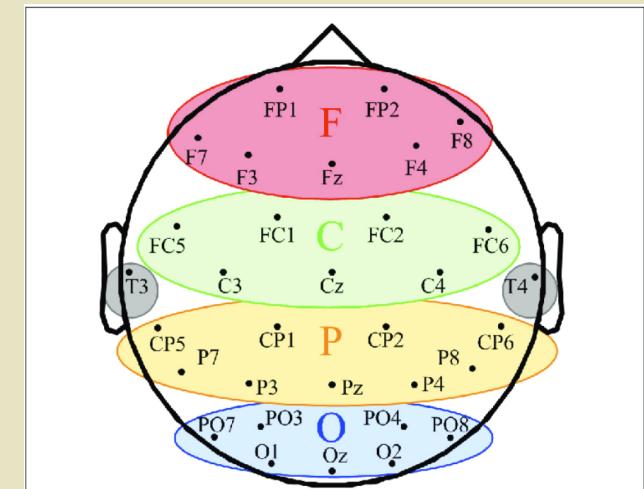
Ye Li (Tyler)

Introduction

- The development of cutting-edge recording technology in neuroscience research increase the data size and complexity.
- For example, the new electroencephalogram (EEG) experiment can simultaneously record several neural activities from multiple brain regions, which induce huge data size yield from one single experiment participants. (~500MB for 30min recording in one participants in compressed format)



Cited from: https://pressrelease.brainproducts.com/actichamp-plus_r-net/

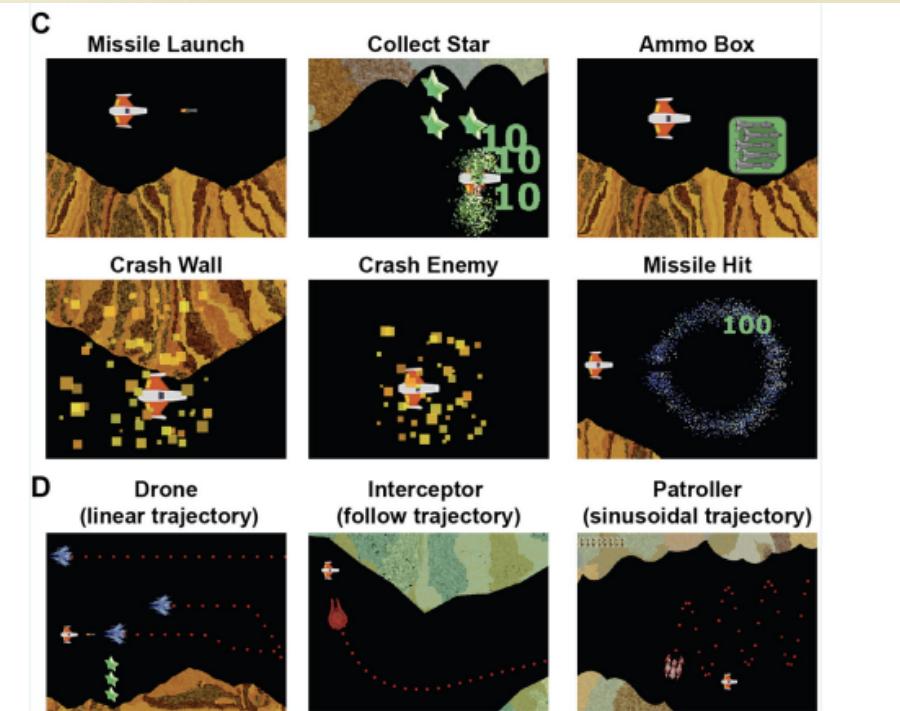


Cited from: Long-Range Temporal Correlations of Patients in Minimally Conscious State Modulated by Spinal Cord Stimulation

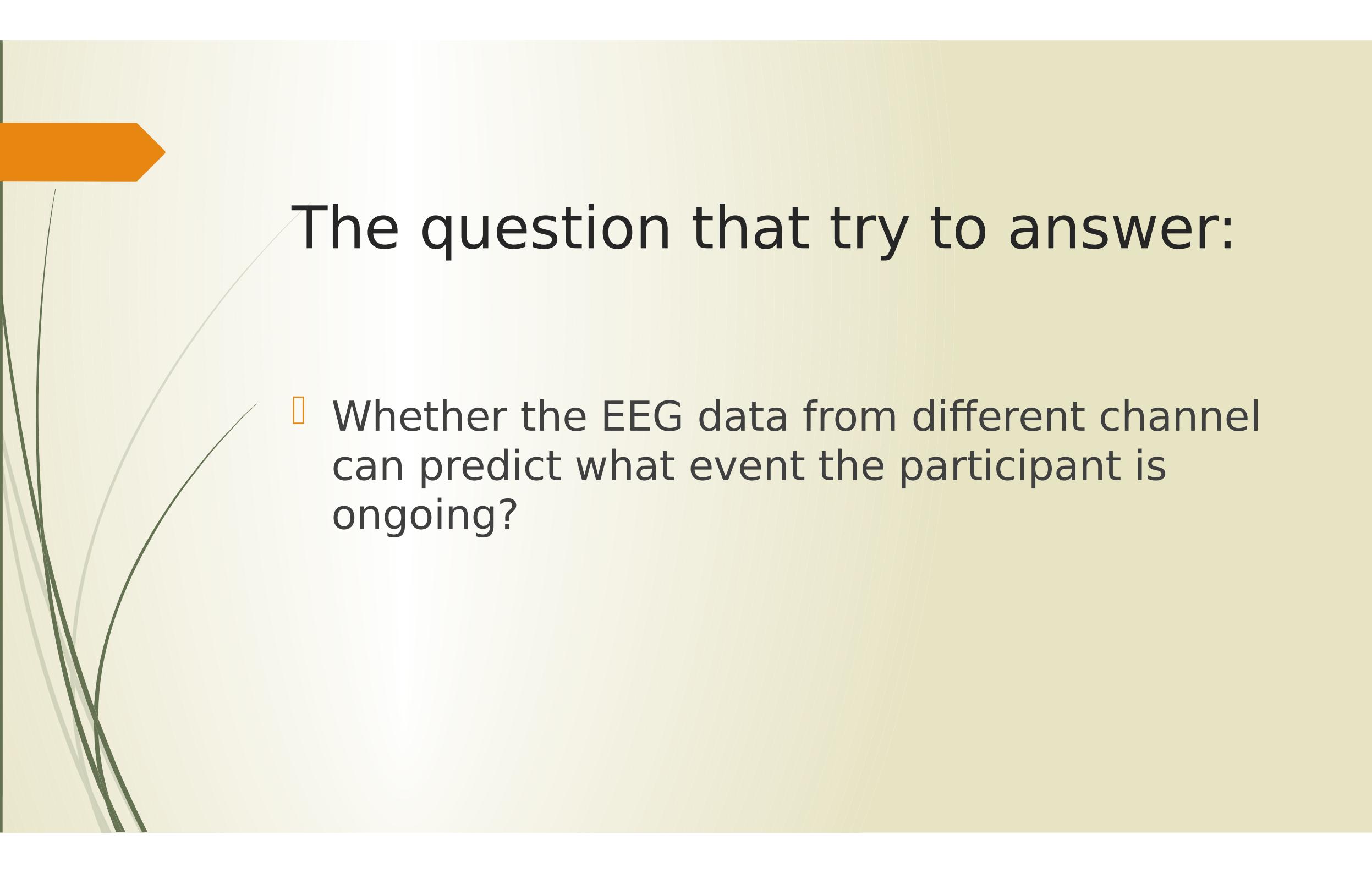
Dataset:

Link: <https://openneuro.org/datasets/ds003517/versions/1.1.0>

- Original data: EEG recording during continuous gameplay of an 8-bit style video game. Total participants number is 17. Several event in the game also recorded.



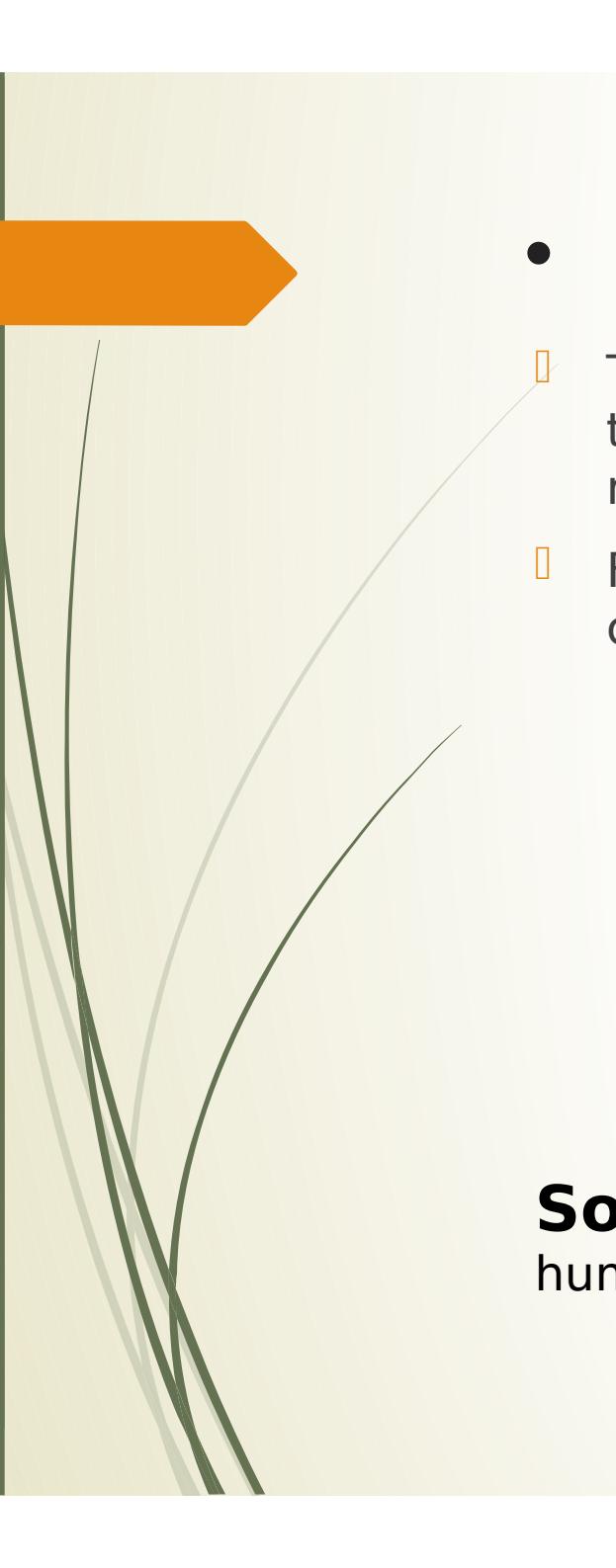
- Data used in this project: Since size of whole original dataset is about 2GB in compressed format, here, in this project I only use data from 1 participant's 1 session for demo purpose.



The question that try to answer:

- Whether the EEG data from different channel can predict what event the participant is ongoing?

- Technical hurdles to load raw EEG dataset
 - The dataset structure is presented by using multiple files, such as files to record channel info, files to record EEG voltage data, and files to record co-order system.
 - Files formats are different between different EEG machine. It could be a challenge if any vendor using a closed system.



```
sub-001_task-ContinuousVideoGamePlay_run-02_channels.tsv
sub-001_task-ContinuousVideoGamePlay_run-02_coordsystem.json
sub-001_task-ContinuousVideoGamePlay_run-02_eeg.fdt
sub-001_task-ContinuousVideoGamePlay_run-02_eeg.json
sub-001_task-ContinuousVideoGamePlay_run-02_eeg.set
sub-001_task-ContinuousVideoGamePlay_run-02_electrodes.tsv
sub-001_task-ContinuousVideoGamePlay_run-02_events.json
sub-001_task-ContinuousVideoGamePlay_run-02_events.tsv
```

Solution: I found Open-source Python package for loading, analysis human neurophysiological data, such as EEG dataset.

Workflow:

1. Pre-processing of raw data

- The package I used to load the raw EEG data is MNE.

```
sub-001_task-ContinuousVideoGamePlay_run-02_channels.tsv  
sub-001_task-ContinuousVideoGamePlay_run-02_coordsystem.json  
sub-001_task-ContinuousVideoGamePlay_run-02_eeg.fdt  
sub-001_task-ContinuousVideoGamePlay_run-02_eeg.json  
sub-001_task-ContinuousVideoGamePlay_run-02_eeg.set  
sub-001_task-ContinuousVideoGamePlay_run-02_electrodes.tsv  
sub-001_task-ContinuousVideoGamePlay_run-02_events.json  
sub-001_task-ContinuousVideoGamePlay_run-02_events.tsv
```

```
import mne
```

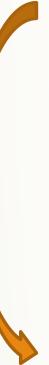
```
df=mne.io.read_raw_eeglab("sub-001_task-ContinuousVideoGamePlay_run-02_eeg.set")  
raw=df.to_data_frame()  
raw.head()
```

	time	Fp1	Fz	F3	F7	FT9	FC5	FC1
0	0	-21805.175781	8995.654297	-21166.406250	-24166.259766	-8705.029297	-17463.330078	-21718.554688
1	2	-21804.736328	8995.166016	-21168.115234	-24155.468750	-8712.548828	-17456.933594	-21721.826172
2	4	-21800.292969	8995.166016	-21167.529297	-24149.609375	-8687.158203	-17477.783203	-21724.316406
3	6	-21804.248047	8994.384766	-21164.404297	-24148.046875	-8684.326172	-17458.593750	-21721.826172
4	8	-21808.740234	8994.287109	-21161.816406	-24163.476562	-8707.568359	-17481.005859	-21715.576172

Workflow:

2. Load event data from EEG raw data

```
event=df.annotations.to_data_frame()  
event["onset"]=event["onset"].astype('datetime64[ns]').astype(np.int64) / int(1e6)  
event=event.rename(columns={'onset':'time', 'description':'event'})  
event
```



	time	duration	event				
0	0.0	0.000	boundary	4065	2794426.0	0.002 S 90	
1	90440.0	0.002	S100	4066	2795344.0	0.002 S 97	
2	91758.0	0.002	S 90	4067	2795390.0	0.002 S 90	
3	92110.0	0.002	S 97	4068	2795580.0	0.002 S 97	
4	94466.0	0.002	S 90	4069	2796230.0	0.002 S 98	
...				4070 rows × 3 columns			

Workflow:

3. Pre-processing data and loading the raw into pyspark session.

combine EEG voltage with event

```
total_raw=pd.merge(raw, event, on='time', how='outer')
```

add event label to raw data

```
#df=total_raw.copy()
for i in range(1, len(total_raw)):
    if pd.notna(total_raw.at[i, "event"]):
        total_raw.loc[i-1, "add_event"] =total_raw.at[i, 'event']
        total_raw.loc[i, "add_event"] =total_raw.at[i, 'event']
        total_raw.loc[i+1, "add_event"] =total_raw.at[i, 'event']
```

```
project_df=total_raw[total_raw["add_event"].notna()]
```

```
project_df.to_csv("project_data.csv", index=False)
```

```
project_df.shape
```

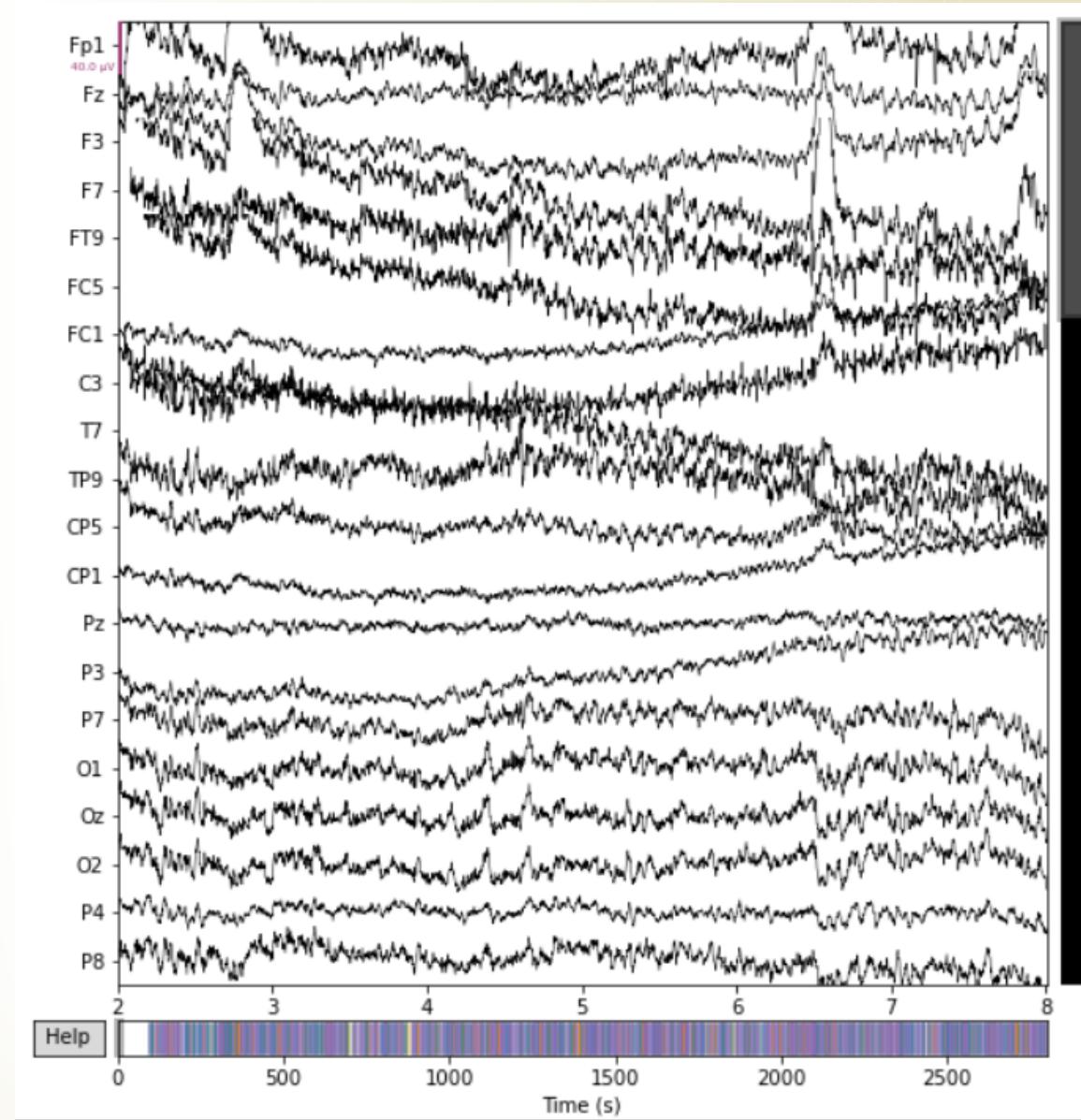
```
(12207, 69)
```

```
root
|-- time: integer (nullable = true)
|-- Fp1: double (nullable = true)
|-- Fz: double (nullable = true)
|-- F3: double (nullable = true)
|-- F7: double (nullable = true)
|-- FT9: double (nullable = true)
|-- FC5: double (nullable = true)
|-- FC1: double (nullable = true)
|-- C3: double (nullable = true)
|-- T7: double (nullable = true)
|-- TP9: double (nullable = true)
|-- CP5: double (nullable = true)
|-- CP1: double (nullable = true)
|-- Pz: double (nullable = true)
|-- P3: double (nullable = true)
|-- P7: double (nullable = true)
|-- O1: double (nullable = true)
|-- Oz: double (nullable = true)
|-- O2: double (nullable = true)
|-- P4: double (nullable = true)
|-- P8: double (nullable = true)
```

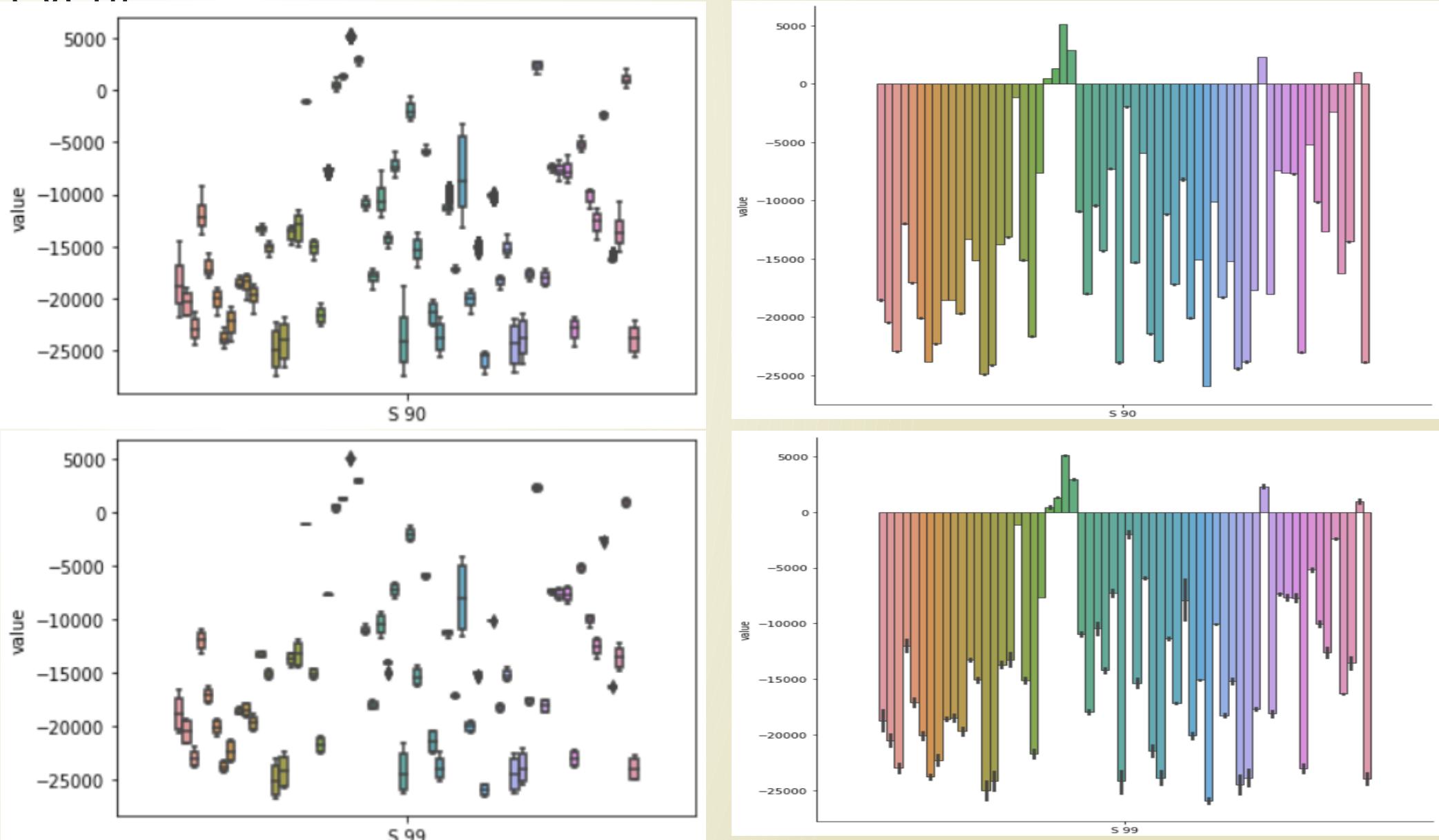
```
.....
```

Workflow:

4. Data visualization: EEG voltage in different channel



Workflow: 4. Data visualization: channel shape between different event



5.1 First, I try to predict three different event: ***game start*, *game over* and *in game***. In this way, I label all event in the game (such as SHOOT_BUTTON, PLAYER_CRASH_WALL, PLAYER_CRASH_ENEMY et al.) into same category.

```
formula = RFormula(  
    formula="add_event ~ .",  
    featuresCol="features",  
    labelCol="label")  
  
output = formula.fit(df1).transform(df1)  
output.select("features", "label").show()  
  
+-----+----+  
| features|label|  
+-----+----+  
|[ -21769.482421875,... | 1.0  
|[ -21771.093749999,... | 1.0  
|[ -21771.484375,92... | 1.0  
|[ -21738.427734375... | 0.0  
|[ -21739.697265625... | 0.0  
|[ -21739.306640624... | 0.0  
|[ -21748.4375,9236... | 0.0  
|[ -21748.53515625,... | 0.0  
|[ -21747.998046874... | 0.0  
|[ -21709.423828124... | 0.0  
|[ -21708.349609375... | 0.0  
|[ -21720.8984375,9... | 0.0  
|[ -21717.3828125,9... | 0.0  
|[ -21724.70703125,... | 0.0  
|[ -21724.21875,925... | 0.0  
|[ -21702.636718749... | 0.0  
|[ -21701.708984375... | 0.0  
|[ -21702.636718749... | 0.0  
|[ -21717.724609375... | 0.0  
|[ -21719.384765625... | 0.0|  
+-----+----+  
only showing top 20 rows
```

```
: from pyspark.ml import Pipeline  
from pyspark.ml.classification import DecisionTreeClassifier  
from pyspark.ml.feature import StringIndexer, VectorIndexer  
from pyspark.ml.evaluation import MulticlassClassificationEvaluator  
  
: labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(output)  
  
# Automatically identify categorical features, and index them.  
# We specify maxCategories so features with > 4 distinct values are treated as continuous.  
featureIndexer = (VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=3).fit(output))  
  
(trainingData, testData) = output.randomSplit([0.7, 0.3], seed=12345)  
  
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")  
  
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dt])  
  
# Train model. This also runs the indexers.  
model = pipeline.fit(trainingData)  
  
# Make predictions.  
predictions = model.transform(testData)  
  
# Select example rows to display.  
predictions.select("prediction", "indexedLabel", "features").show(5)  
  
# Select (prediction, true label) and compute test error  
evaluator = MulticlassClassificationEvaluator(  
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")  
accuracy = evaluator.evaluate(predictions)  
print("Test Error = %g" % (1.0 - accuracy))  
  
treeModel = model.stages[2]  
# summary only  
print(treeModel)  
  
+-----+-----+-----+  
|prediction|indexedLabel|      features|  
+-----+-----+-----+  
|     0.0|      0.0|[-21739.697265625...|  
|     0.0|      0.0|[-21724.70703125,...|  
|     0.0|      0.0|[-21717.724609375...|  
|     0.0|      0.0|[-21690.087890624...|  
|     0.0|      0.0|[-21690.087890624...|  
+-----+-----+-----+  
only showing top 5 rows  
  
Test Error = 0.00218221  
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_f529ff142492, depth=5, numNodes=25, numClasses=3, numFeatures=63
```

For predict ongoing event game start, in game and game over, the model had a great performance.

Workflow:

5. Predict ongoing event: decision tree model building

5.2 Here, I try to predict three different in game event: **crash wall, crash enemy and missile hit enemy**. In this way, I only label there three event in the dataframe. I also applied the paramgridbuilder method with hyperparameter turning.

```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

grid = ParamGridBuilder() \
    .addGrid(dt.impurity, ["gini", "entropy"]) \
    .addGrid(dt.maxBins, [5, 10, 15]) \
    .addGrid(dt.minInfoGain, [0.0, 0.2, 0.4]) \
    .addGrid(dt.maxDepth, [3, 5, 7]) \
    .build()

cv = CrossValidator(estimator=pipeline, evaluator=evaluator, estimatorParamMaps=grid, numFolds=3)

cvModel = cv.fit(trainingData)

#print out the hyperparameter of best model

best_Model = cvModel.bestModel

predictions=best_Model.transform(testData)
predictions.select("prediction", "indexedLabel", "features").show(5) #select("prediction", "indexedLabel", "features", "accuracy")

test_metric = evaluator.evaluate(best_Model.transform(testData))
print("Test Error = %g " % (1.0 - test_metric))

for x in range(len(best_Model.stages)):
    print(best_Model.stages[x])

java_model = best_Model.stages[-1]._java_obj
{param.name: java_model.getOrDefault(java_model.getParam(param.name))
 for param in grid[0]}
```

prediction	indexedLabel	features
0.0	0.0	[-21748.4375, 9236...]
0.0	0.0	[-21717.3828125, 9...]
0.0	0.0	[-21654.736328125...]
0.0	1.0	[-21593.701171875...]
2.0	0.0	[-21581.201171874...]

only showing top 5 rows

Test Error = 0.27176
StringIndexerModel: uid=StringIndexer_fb606add02bf, handleInvalid=error
VectorIndexerModel: uid=VectorIndexer_6fabd9376fb, numFeatures=63, handleInvalid=error
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_0ef63c4fe303, depth=7, numNodes=143, numClasses=3, numFeatures=63
{'impurity': 'gini', 'maxBins': 10, 'minInfoGain': 0.0, 'maxDepth': 7}



Next step:

1. Further raw data clean, such as clean the artificial signal in EEG, noise filtering, data re-sample, and epoch data transferring.
2. Analyzing all the data from participants.
3. Trying the other training model and ts-flint package.

THANKS:
Thanks!