Scope of Project

The goal is to predict whether a person has any of the three medical conditions or none of these along with its respective probability. Class 1 will be a person with prescence of any conditions and Class 0 will be person with no medical conditions.

```
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
/kaggle/input/tabpfn/tabpfn-0.1.9-py3-none-any.whl
/kaggle/input/tabpfn/prior_diff_real_checkpoint_n_0_epoch_100.cpkt
/kaggle/input/icr-identify-age-related-conditions/sample_submission.cs
v
/kaggle/input/icr-identify-age-related-conditions/greeks.csv
/kaggle/input/icr-identify-age-related-conditions/train.csv
/kaggle/input/icr-identify-age-related-conditions/test.csv
```

Import Libraries

```
# install Tabpfn
!pip install tabpfn --no-index --find-
links=file:///kaggle/input/tabpfn
!mkdir -p /opt/conda/lib/python3.10/site-packages/tabpfn/models_diff
```

```
!cp /kaggle/input/tabpfn/prior_diff_real_checkpoint_n_0_epoch_100.cpkt
/opt/conda/lib/python3.10/site-packages/tabpfn/models_diff/
```

```
Looking in links: file:///kaggle/input/tabpfn
Processing /kaggle/input/tabpfn/tabpfn-0.1.9-py3-none-any.whl
Requirement already satisfied: numpy>=1.21.2 in
/opt/conda/lib/python3.10/site-packages (from tabpfn) (1.23.5)
Requirement already satisfied: pyyaml>=5.4.1 in
/opt/conda/lib/python3.10/site-packages (from tabpfn) (6.0)
Requirement already satisfied: requests>=2.23.0 in
/opt/conda/lib/python3.10/site-packages (from tabpfn) (2.31.0)
Requirement already satisfied: scikit-learn>=0.24.2 in
/opt/conda/lib/python3.10/site-packages (from tabpfn) (1.2.2)
Requirement already satisfied: torch>=1.9.0 in
/opt/conda/lib/python3.10/site-packages (from tabpfn) (2.0.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/opt/conda/lib/python3.10/site-packages (from requests>=2.23.0-
>tabpfn) (3.1.0)
Requirement already satisfied: idna<4,>=2.5 in
/opt/conda/lib/python3.10/site-packages (from requests>=2.23.0-
>tabpfn) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.10/site-packages (from requests>=2.23.0-
>tabpfn) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.10/site-packages (from requests>=2.23.0-
>tabpfn) (2023.5.7)
Requirement already satisfied: scipy>=1.3.2 in
/opt/conda/lib/python3.10/site-packages (from scikit-learn>=0.24.2-
>tabpfn) (1.11.1)
Requirement already satisfied: joblib>=1.1.1 in
/opt/conda/lib/python3.10/site-packages (from scikit-learn>=0.24.2-
>tabpfn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/conda/lib/python3.10/site-packages (from scikit-learn>=0.24.2-
>tabpfn) (3.1.0)
Requirement already satisfied: filelock in
/opt/conda/lib/python3.10/site-packages (from torch>=1.9.0->tabpfn)
(3.12.2)
Requirement already satisfied: typing-extensions in
/opt/conda/lib/python3.10/site-packages (from torch>=1.9.0->tabpfn)
(4.6.3)
Requirement already satisfied: sympy in
/opt/conda/lib/python3.10/site-packages (from torch>=1.9.0->tabpfn)
(1.12)
Requirement already satisfied: networkx in
/opt/conda/lib/python3.10/site-packages (from torch>=1.9.0->tabpfn)
(3.1)
Requirement already satisfied: jinja2 in
/opt/conda/lib/python3.10/site-packages (from torch>=1.9.0->tabpfn)
```

```
(3.1.2)
Requirement already satisfied: MarkupSafe>=2.0 in
/opt/conda/lib/python3.10/site-packages (from jinja2->torch>=1.9.0-
>tabpfn) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in
/opt/conda/lib/python3.10/site-packages (from sympy->torch>=1.9.0-
>tabpfn) (1.3.0)
Installing collected packages: tabpfn
Successfully installed tabpfn-0.1.9
```

```
import random
import sklearn
import warnings
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn import metrics
from matplotlib import style
from tqdm.notebook import tqdm
from collections import Counter
from datetime import datetime
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from tabpfn import TabPFNClassifier
from sklearn.impute import KNNImputer
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import RandomOverSampler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.utils.class_weight import compute_sample_weight
from sklearn.feature_selection import SequentialFeatureSelector as sfs
from sklearn.metrics import log_loss, balanced_accuracy_score,
f1_score
from sklearn.metrics import accuracy_score, precision_score,
recall_score, roc_auc_score

plt.style.use('ggplot')

#  suppress warning
warnings.filterwarnings('ignore')
```

Import Data

File Descriptions

1. **train.csv** - The training set.
- Id - Unique identifier for each observation.
- AB-GL - Fifty-six anonymized health characteristics. *All are numeric except for EJ, which is categorical.*
- Class A binary target: 1 indicates the subject has been diagnosed with one of the three conditions, 0 indicates they have not.
1. **test.csv** - The test set.
- Goal is to predict the probability that a subject in this set belongs to each of the two classes.
1. **greeks.csv** - Supplemental metadata, only available for the training set.
- Alpha - Identifies the type of age-related condition, if present.
- A - No age-related condition. Corresponds to class 0.
- B, D, G - The three age-related conditions. Correspond to class 1.
- Beta, Gamma, Delta - Three experimental characteristics.
- Epsilon - The date that the data was collected from subject. All of the data in the training set was collected before test set.
1. **sample_submission.csv** - Demonstration of correct submission format.

```
# for suppressing some sections when debugging
flag_debug = True
flag_EDA = True
flag_backElimination = True
flag_smote = True

# load data
greeks_ori = pd.read_csv('/kaggle/input/icr-identify-age-related-
conditions/greeks.csv')
train_ori = pd.read_csv('/kaggle/input/icr-identify-age-related-
conditions/train.csv')
test_ori = pd.read_csv('/kaggle/input/icr-identify-age-related-
conditions/test.csv')
sample_submission = pd.read_csv('/kaggle/input/icr-identify-age-
related-conditions/sample_submission.csv')

if False:
    n = 40
    greeks_ori = greeks_ori[:n]
    train_ori = train_ori[:n]
    test_ori = test_ori[:n]


# make a copy from original train data
greeks = greeks_ori.copy(deep = True)
```

```
train = train_ori.copy(deep = True)
test = test_ori.copy(deep = True)
```

Helper Function

```
# function for getting some basic information about dataset
def data_info(dataset):
    print(f'Data has {dataset.shape[0]} rows and {dataset.shape[1]} columns \n')
    display(dataset.head())
    print('\n Info. of data: \n')
    print(dataset.info())
    print('-----------------------------------------------------')

def balanced_log_loss(y_true, y_pred):
    if flag_debug:
        print('bll y_true: ', y_true)
        print('bll y_pred: ', y_pred)
    N_0 = np.sum(1 - y_true) # count number of class 0
    N_1 = np.sum(y_true) # count number of class 1

    w_0 = 1 / N_0
    w_1 = 1 / N_1

    p_1 = np.clip(y_pred, 1e-15, 1 - 1e-15) # probability of observation belongs to class1
    p_0 = 1 - p_1 # probability of observation belongs to class 0

    log_loss_0 = -np.sum((1 - y_true) * np.log(p_0))
    log_loss_1 = -np.sum(y_true * np.log(p_1))

    balanced_log_loss = (w_0 * log_loss_0 + w_1 * log_loss_1) / 2

    return balanced_log_loss

# function for model evaluation
def eval_metrics(model, y, y_pred, y_pred_proba):
    y_df_pred = pd.DataFrame(y_pred)
    sampleWeight = compute_sample_weight(class_weight = 'balanced', y = y)
    logloss = log_loss(y, y_pred_proba, eps = 1e-15, sample_weight = sampleWeight)
    print('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
    print(f'Scoring Metrics for {model}')
    print('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
    print(f'\n Log loss: {logloss}\n')
```

```
    print('-----------------------------------')
    clf_report = classification_report(y, y_pred)
    print('\n Classification report: \n', clf_report)
    print('-----------------------------------')
    print('Balanced Accuracy Score:
{:.2f}'.format(metrics.balanced_accuracy_score(y, y_df_pred)))
    print('Accuracy Score: {:.2f}'.format(metrics.accuracy_score(y,
y_df_pred)))
    print('Precision Score: {:.2f}'.format(metrics.precision_score(y,
y_df_pred)))
    print('Recall Score: {:.2f}'.format(metrics.recall_score(y,
y_df_pred)))
    print('F1 Score: {:.2f}'.format(metrics.f1_score(y, y_df_pred)))
    print('ROC AUC Score: {:.2f}\n\n'.format(metrics.roc_auc_score(y,
y_df_pred)))
```

Exploratory Data Analysis (EDA)

```
# get some basic info. of file greeks
data_info(greeks)
```

```
Data has 617 rows and 6 columns


             Id Alpha Beta Gamma Delta    Epsilon
0  000ff2bfdfe9     B    C     G     D  3/19/2019
1  007255e47698     A    C     M     B    Unknown
2  013f2bd269f5     A    C     M     B    Unknown
3  043ac50845d5     A    C     M     B    Unknown
4  044fb8a146ec     D    B     F     B  3/25/2020


 Info. of data:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 617 entries, 0 to 616
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Id       617 non-null    object
 1   Alpha    617 non-null    object
 2   Beta     617 non-null    object
 3   Gamma    617 non-null    object
 4   Delta    617 non-null    object
 5   Epsilon  617 non-null    object
dtypes: object(6)
memory usage: 29.0+ KB
```

```
None
------------------------------------------------------
```

```python
# get some basic info. of training data
data_info(train)
```

```
Data has 617 rows and 58 columns

             Id        AB          AF          AH         AM        AR  \
0  000ff2bfdfe9  0.209377  3109.03329   85.200147  22.394407  8.138688

1  007255e47698  0.145282   978.76416   85.200147  36.968889  8.138688

2  013f2bd269f5  0.470030  2635.10654   85.200147  32.360553  8.138688

3  043ac50845d5  0.252107  3819.65177  120.201618  77.112203  8.138688

4  044fb8a146ec  0.380297  3733.04844   85.200147  14.103738  8.138688


         AX        AY         AZ          BC  ...        FL        FR  \
0  0.699861  0.025578   9.812214    5.555634  ...  7.298162   1.73855

1  3.632190  0.025578  13.517790    1.229900  ...  0.173229   0.49706

2  6.732840  0.025578  12.824570    1.229900  ...  7.709560   0.97556

3  3.685344  0.025578  11.053708    1.229900  ...  6.122162   0.49706

4  3.942255  0.054810   3.396778  102.151980  ...  8.153058  48.50134


         FS         GB          GE            GF         GH         GI  \
0  0.094822  11.339138   72.611063   2003.810319  22.136229  69.834944

1  0.568932   9.292698   72.611063  27981.562750  29.135430  32.131996

2  1.198821  37.077772   88.609437  13676.957810  28.022851  35.192676

3  0.284466  18.529584   82.416803   2094.262452  39.948656  90.493248

4  0.121914  16.408728  146.109943   8524.370502  45.381316  36.262628


          GL  Class
0   0.120343      1
1  21.978000      0
2   0.196941      0
```

```
3   0.155829      0
4   0.096614      1

[5 rows x 58 columns]


 Info. of data:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 617 entries, 0 to 616
Data columns (total 58 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Id      617 non-null    object
 1   AB      617 non-null    float64
 2   AF      617 non-null    float64
 3   AH      617 non-null    float64
 4   AM      617 non-null    float64
 5   AR      617 non-null    float64
 6   AX      617 non-null    float64
 7   AY      617 non-null    float64
 8   AZ      617 non-null    float64
 9   BC      617 non-null    float64
 10  BD      617 non-null    float64
 11  BN      617 non-null    float64
 12  BP      617 non-null    float64
 13  BQ      557 non-null    float64
 14  BR      617 non-null    float64
 15  BZ      617 non-null    float64
 16  CB      615 non-null    float64
 17  CC      614 non-null    float64
 18  CD      617 non-null    float64
 19  CF      617 non-null    float64
 20  CH      617 non-null    float64
 21  CL      617 non-null    float64
 22  CR      617 non-null    float64
 23  CS      617 non-null    float64
 24  CU      617 non-null    float64
 25  CW      617 non-null    float64
 26  DA      617 non-null    float64
 27  DE      617 non-null    float64
 28  DF      617 non-null    float64
 29  DH      617 non-null    float64
 30  DI      617 non-null    float64
 31  DL      617 non-null    float64
 32  DN      617 non-null    float64
 33  DU      616 non-null    float64
 34  DV      617 non-null    float64
 35  DY      617 non-null    float64
 36  EB      617 non-null    float64
```

```
 37  EE      617 non-null    float64
 38  EG      617 non-null    float64
 39  EH      617 non-null    float64
 40  EJ      617 non-null    object
 41  EL      557 non-null    float64
 42  EP      617 non-null    float64
 43  EU      617 non-null    float64
 44  FC      616 non-null    float64
 45  FD      617 non-null    float64
 46  FE      617 non-null    float64
 47  FI      617 non-null    float64
 48  FL      616 non-null    float64
 49  FR      617 non-null    float64
 50  FS      615 non-null    float64
 51  GB      617 non-null    float64
 52  GE      617 non-null    float64
 53  GF      617 non-null    float64
 54  GH      617 non-null    float64
 55  GI      617 non-null    float64
 56  GL      616 non-null    float64
 57  Class   617 non-null    int64
dtypes: float64(55), int64(1), object(2)
memory usage: 279.7+ KB
None
------------------------------------------------------
```

Data type of column **Id** and **EJ** are object instead of numerical, so label encoding may need to apply to column **EJ** for downstream consumption.

```
# get some info. of test data
data_info(test)

Data has 5 rows and 57 columns


             Id   AB   AF   AH   AM   AR   AX   AY   AZ   BC  ...   FI
FL  \
0  00eed32682bb  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
1  010ebe33f668  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
2  02fa521e1838  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
3  040e15f562a2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
4  046e85c7cc7f  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0

    FR   FS   GB   GE   GF   GH   GI   GL
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 57 columns]


 Info. of data:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 57 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Id      5 non-null      object
 1   AB      5 non-null      float64
 2   AF      5 non-null      float64
 3   AH      5 non-null      float64
 4   AM      5 non-null      float64
 5   AR      5 non-null      float64
 6   AX      5 non-null      float64
 7   AY      5 non-null      float64
 8   AZ      5 non-null      float64
 9   BC      5 non-null      float64
 10  BD      5 non-null      float64
 11  BN      5 non-null      float64
 12  BP      5 non-null      float64
 13  BQ      5 non-null      float64
 14  BR      5 non-null      float64
 15  BZ      5 non-null      float64
 16  CB      5 non-null      float64
 17  CC      5 non-null      float64
 18  CD      5 non-null      float64
 19  CF      5 non-null      float64
 20  CH      5 non-null      float64
 21  CL      5 non-null      float64
 22  CR      5 non-null      float64
 23  CS      5 non-null      float64
 24  CU      5 non-null      float64
 25  CW      5 non-null      float64
 26  DA      5 non-null      float64
 27  DE      5 non-null      float64
 28  DF      5 non-null      float64
 29  DH      5 non-null      float64
 30  DI      5 non-null      float64
 31  DL      5 non-null      float64
 32  DN      5 non-null      float64
 33  DU      5 non-null      float64
 34  DV      5 non-null      float64
```

```
 35  DY       5 non-null      float64
 36  EB       5 non-null      float64
 37  EE       5 non-null      float64
 38  EG       5 non-null      float64
 39  EH       5 non-null      float64
 40  EJ       5 non-null      object
 41  EL       5 non-null      float64
 42  EP       5 non-null      float64
 43  EU       5 non-null      float64
 44  FC       5 non-null      float64
 45  FD       5 non-null      float64
 46  FE       5 non-null      float64
 47  FI       5 non-null      float64
 48  FL       5 non-null      float64
 49  FR       5 non-null      float64
 50  FS       5 non-null      float64
 51  GB       5 non-null      float64
 52  GE       5 non-null      float64
 53  GF       5 non-null      float64
 54  GH       5 non-null      float64
 55  GI       5 non-null      float64
 56  GL       5 non-null      float64
dtypes: float64(55), object(2)
memory usage: 2.4+ KB
None
------------------------------------------------------
```

As what are shown in training data set, data type of column **Id** and **EJ** are object instead numerical.

```
# get some info. of sample submission and its format
data_info(sample_submission)
```

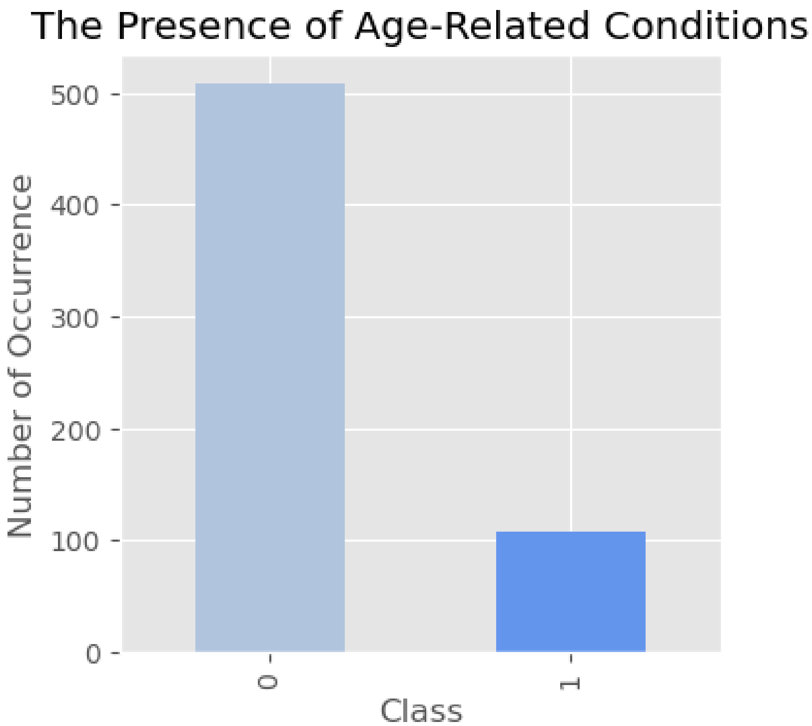
```
Data has 5 rows and 3 columns


             Id  class_0  class_1
0  00eed32682bb      0.5      0.5
1  010ebe33f668      0.5      0.5
2  02fa521e1838      0.5      0.5
3  040e15f562a2      0.5      0.5
4  046e85c7cc7f      0.5      0.5


 Info. of data:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
```

```
---  ------   --------------  -----
 0   Id       5 non-null      object
 1   class_0  5 non-null      float64
 2   class_1  5 non-null      float64
dtypes: float64(2), object(1)
memory usage: 248.0+ bytes
None
------------------------------------------------------
```

```
# find number of occurrences of Class 0 and Class 1
print('Value counts of class 0 and class 1: ',
train['Class'].value_counts(), sep = '\n')

# plot bar chart for visualization
fig, ax = plt.subplots()
train['Class'].value_counts().plot(kind = 'bar', figsize = [4,4],
                                   color = ['lightsteelblue',
'cornflowerblue'],
                                   title = 'The Presence of Age-
Related Conditions')
ax.set_xlabel('Class')
ax.set_ylabel('Number of Occurrence')
```

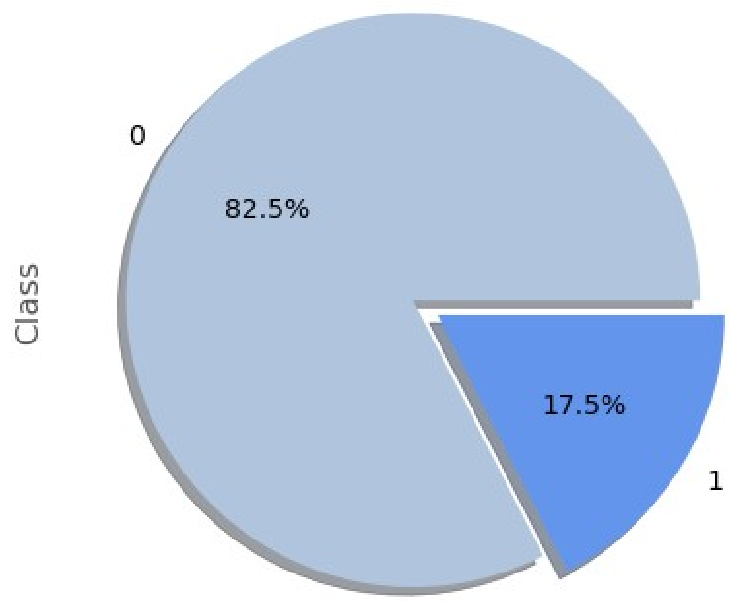
```
Value counts of class 0 and class 1:
0    509
1    108
Name: Class, dtype: int64
```

```
Text(0, 0.5, 'Number of Occurrence')
```




```
# calculate rate of occurrences of Class 0 and Class 1
print('Percentage of class 0 and class 1: ',
train['Class'].value_counts(normalize = True)*100, sep = '\n')

# plot pie chart for visualization
train['Class'].value_counts().plot(kind = 'pie', autopct = '%1.1f%%',
                                   colors = ['lightsteelblue',
'cornflowerblue'],
                                   shadow = True,
                                   explode = ([0,0.1]))
```

```
Percentage of class 0 and class 1:
0    82.495948
1    17.504052
Name: Class, dtype: float64

<Axes: ylabel='Class'>
```



From the rate shown above, Class 0 makes up 82.5% of the training dataset and Class 1 only makes up about 17.5%. Thus, the data is imbalanced and requires to be processed with oversampling or undersampling later before modelling.

- **Class 0** means a subject **has not** diagnosed with any of the age-related medical conditions.
- **Class 1** means a subject **has** diagnosed with any of the three age-related medical conditions.

```
# check for duplication
print('Check for duplication: ',train.duplicated().all())

Check for duplication:  False
```

From the above result, there is no duplication in the training data.

```
# check for missing/null values in train file
print('Check for missing/null values in ' + '\033[1m' + 'train' + '\
033[0m' + ' file: \n')
train.isnull().sum()

Check for missing/null values in train file:


Id        0
AB        0
AF        0
AH        0
```

AM	0
AR	0
AX	0
AY	0
AZ	0
BC	0
BD	0
BN	0
BP	0
BQ	60
BR	0
BZ	0
CB	2
CC	3
CD	0
CF	0
CH	0
CL	0
CR	0
CS	0
CU	0
CW	0
DA	0
DE	0
DF	0
DH	0
DI	0
DL	0
DN	0
DU	1
DV	0
DY	0
EB	0
EE	0
EG	0
EH	0
EJ	0
EL	60
EP	0
EU	0
FC	1
FD	0
FE	0
FI	0
FL	1
FR	0
FS	2
GB	0
GE	0

```
GF        0
GH        0
GI        0
GL        1
Class     0
dtype: int64
```

```python
# check for missing/null values in test file
print('Check for missing/null values in ' + '\033[1m' + 'test' + '\033[0m' + ' file: \n')
test.isnull().sum()
```

```
Check for missing/null values in test file:


Id     0
AB     0
AF     0
AH     0
AM     0
AR     0
AX     0
AY     0
AZ     0
BC     0
BD     0
BN     0
BP     0
BQ     0
BR     0
BZ     0
CB     0
CC     0
CD     0
CF     0
CH     0
CL     0
CR     0
CS     0
CU     0
CW     0
DA     0
DE     0
DF     0
DH     0
DI     0
DL     0
DN     0
DU     0
DV     0
```

```
DY      0
EB      0
EE      0
EG      0
EH      0
EJ      0
EL      0
EP      0
EU      0
FC      0
FD      0
FE      0
FI      0
FL      0
FR      0
FS      0
GB      0
GE      0
GF      0
GH      0
GI      0
GL      0
dtype: int64
```

```
# check for missing/null values in greeks file
print('Check for missing/null values in ' + '\033[1m' + 'greeks' + '\033[0m' + ' file: \n')
greeks.isnull().sum()
```

```
Check for missing/null values in greeks file:


Id          0
Alpha       0
Beta        0
Gamma       0
Delta       0
Epsilon     0
dtype: int64
```

From the results shown, there are some columns with missing/null values in train file. Thus, the missing values may need to be dealt with before modelling.

There is no missing/null values in both test and greeks file repectively.

```
# get some ideas about descriptive statistics
train.describe().T
```

```
       count          mean           std          min          25%  \
AB     617.0      0.477149      0.468388     0.081187     0.252107
```

AF	617.0	3502.013221	2300.322717	192.593280	2197.345480
AH	617.0	118.624513	127.838950	85.200147	85.200147
AM	617.0	38.968552	69.728226	3.177522	12.270314
AR	617.0	10.128242	10.518877	8.138688	8.138688
AX	617.0	5.545576	2.551696	0.699861	4.128294
AY	617.0	0.060320	0.416817	0.025578	0.025578
AZ	617.0	10.566447	4.350645	3.396778	8.129580
BC	617.0	8.053012	65.166943	1.229900	1.229900
BD	617.0	5350.388655	3021.326641	1693.624320	4155.702870
BN	617.0	21.419492	3.478278	9.886800	19.420500
BP	617.0	231.322223	183.992505	72.948951	156.847239
BQ	557.0	98.328737	96.479371	1.331155	27.834425
BR	617.0	1218.133238	7575.293707	51.216883	424.990642
BZ	617.0	550.632525	2076.371275	257.432377	257.432377
CB	615.0	77.104151	159.049302	12.499760	23.317567
CC	614.0	0.688801	0.263994	0.176874	0.563688
CD	617.0	90.251735	51.585130	23.387600	64.724192
CF	617.0	11.241064	13.571133	0.510888	5.066306
CH	617.0	0.030615	0.014808	0.003184	0.023482
CL	617.0	1.403761	1.922210	1.050225	1.050225
CR	617.0	0.742262	0.281195	0.069225	0.589575
CS	617.0	36.917590	17.266347	13.784111	29.782467
CU	617.0	1.383792	0.538717	0.137925	1.070298
CW	617.0	27.165653	14.645993	7.030640	7.030640
DA	617.0	51.128326	21.210888	6.906400	37.942520
DE	617.0	401.901299	317.745623	35.998895	188.815690
DF	617.0	0.633884	1.912384	0.238680	0.238680
DH	617.0	0.367002	0.112989	0.040995	0.295164
DI	617.0	146.972099	86.084419	60.232470	102.703553
DL	617.0	94.795377	28.243187	10.345600	78.232240
DN	617.0	26.370568	8.038825	6.339496	20.888264
DU	616.0	1.802900	9.034721	0.005518	0.005518
DV	617.0	1.924830	1.484555	1.743070	1.743070
DY	617.0	26.388989	18.116679	0.804068	14.715792
EB	617.0	9.072700	6.200281	4.926396	5.965392
EE	617.0	3.064778	2.058344	0.286201	1.648679
EG	617.0	1731.248215	1790.227476	185.594100	1111.160625
EH	617.0	0.305107	1.847499	0.003042	0.003042
EL	557.0	69.582596	38.555707	5.394675	30.927468
EP	617.0	105.060712	68.445620	78.526968	78.526968
EU	617.0	69.117005	390.187057	3.828384	4.324656
FC	616.0	71.341526	165.551545	7.534128	25.815384
FD	617.0	6.930086	64.754262	0.296850	0.296850
FE	617.0	10306.810737	11331.294051	1563.136688	5164.666260
FI	617.0	10.111079	2.934025	3.583450	8.523098
FL	616.0	5.433199	11.496257	0.173229	0.173229
FR	617.0	3.533905	50.181948	0.497060	0.497060
FS	615.0	0.421501	1.305365	0.067730	0.067730
GB	617.0	20.724856	9.991907	4.102182	14.036718

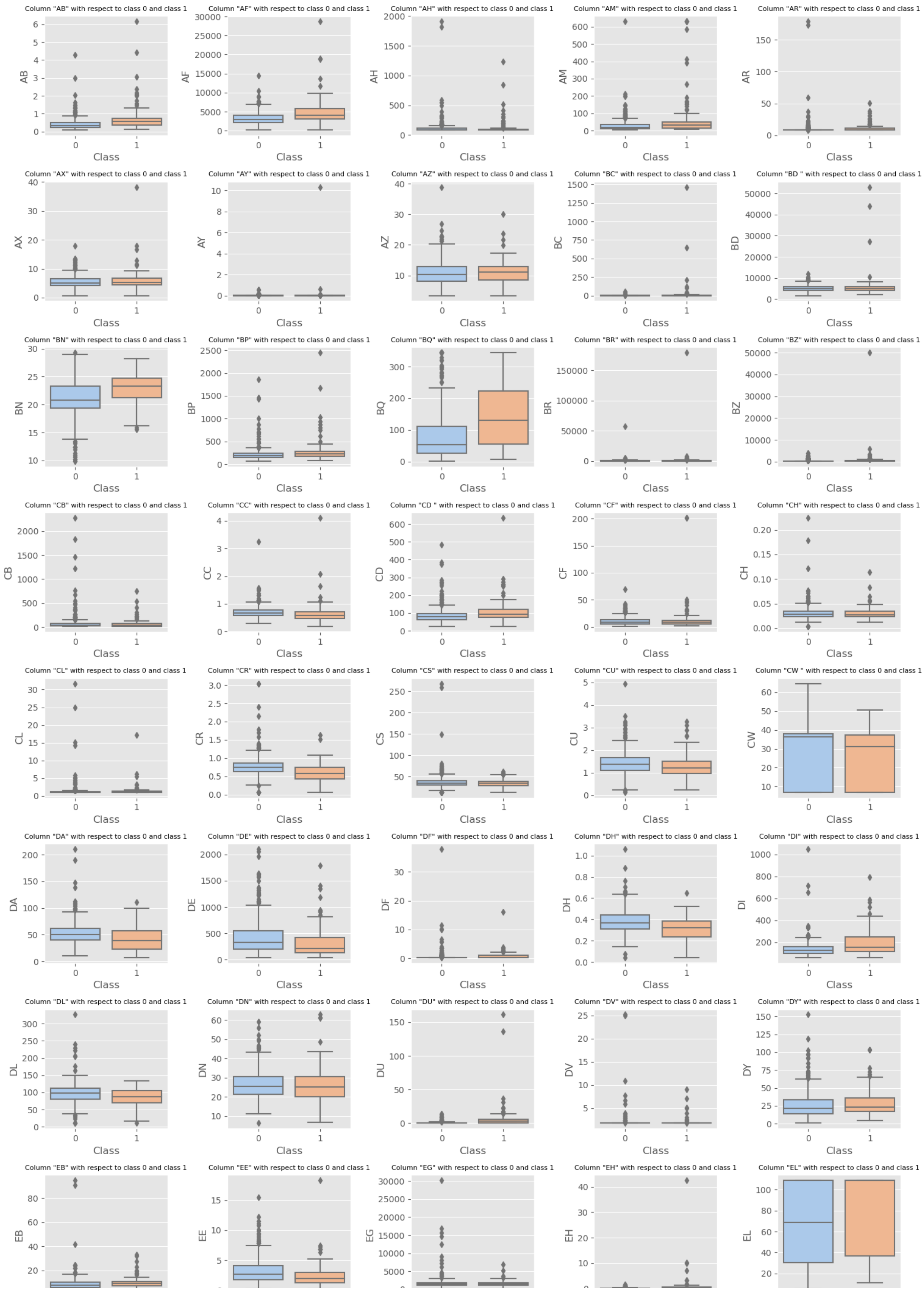
GE	617.0	131.714987	144.181524	72.611063	72.611063
GF	617.0	14679.595398	19352.959387	13.038894	2798.992584
GH	617.0	31.489716	9.864239	9.432735	25.034888
GI	617.0	50.584437	36.266251	0.897628	23.011684
GL	616.0	8.530961	10.327010	0.001129	0.124392
Class	617.0	0.175041	0.380310	0.000000	0.000000

	50%	75%	max
AB	0.354659	0.559763	6.161666
AF	3120.318960	4361.637390	28688.187660
AH	85.200147	113.739540	1910.123198
AM	20.533110	39.139886	630.518230
AR	8.138688	8.138688	178.943634
AX	5.031912	6.431634	38.270880
AY	0.025578	0.036845	10.315851
AZ	10.461320	12.969516	38.971568
BC	1.229900	5.081244	1463.693448
BD	4997.960730	6035.885700	53060.599240
BN	21.186000	23.657700	29.307300
BP	193.908816	247.803462	2447.810550
BQ	61.642115	134.009015	344.644105
BR	627.417402	975.649259	179250.252900
BZ	257.432377	257.432377	50092.459300
CB	42.554330	77.310097	2271.436167
CC	0.658715	0.772206	4.103032
CD	79.819104	99.813520	633.534408
CF	9.123000	13.565901	200.967526
CH	0.027860	0.034427	0.224074
CL	1.050225	1.228445	31.688153
CR	0.730800	0.859350	3.039675
CS	34.835130	40.529401	267.942823
CU	1.351665	1.660617	4.951507
CW	36.019104	37.935832	64.521624
DA	49.180940	61.408760	210.330920
DE	307.509595	507.896200	2103.405190
DF	0.238680	0.238680	37.895013
DH	0.358023	0.426348	1.060404
DI	130.050630	165.836955	1049.168078
DL	96.264960	110.640680	326.236200
DN	25.248800	30.544224	62.808096
DU	0.251741	1.058690	161.355315
DV	1.743070	1.743070	25.192930
DY	21.642456	34.058344	152.355164
EB	8.149404	10.503048	94.958580
EE	2.616119	3.910070	18.324926
EG	1493.817413	1905.701475	30243.758780
EH	0.085176	0.237276	42.569748
EL	71.949306	109.125159	109.125159
EP	78.526968	112.766654	1063.594578

```
EU       22.641144     49.085352    6501.264480
FC       36.394008     56.714448    3030.655824
FD        1.870155      4.880214    1578.654237
FE     7345.143424  10647.951650  143224.682300
FI        9.945452     11.516657      35.851039
FL        3.028141      6.238814     137.932739
FR        1.131000      1.512060    1244.227020
FS        0.250601      0.535067      31.365763
GB       18.771436     25.608406     135.781294
GE       72.611063    127.591671    1497.351958
GF     7838.273610  19035.709240  143790.071200
GH       30.608946     36.863947      81.210825
GI       41.007968     67.931664     191.194764
GL        0.337827     21.978000      21.978000
Class     0.000000      0.000000       1.000000
```

The means of the characteristics presented range widely from the descriptive statistics, so these characteristics may need to be standardized before modelling.

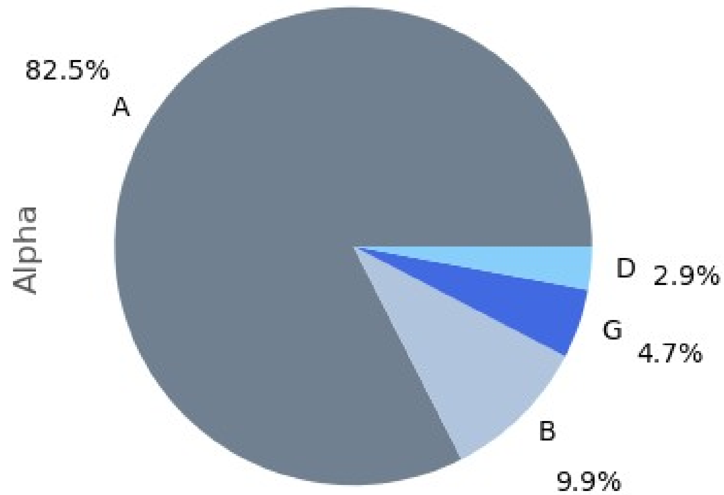
```python
if flag_EDA:
    # boxplot for visually looking at descriptive statistics
    count = 1
    fig = plt.figure(figsize = (15,30))

    for col in train.loc[:, ~train.columns.isin(['Id', 'EJ',
'Class'])]:
        plt.subplot(11,5, count)
        plt.title(f'Column "{col}" with respect to class 0 and class
1', fontsize = 8)
        sns.boxplot(x = train['Class'], y = train[col], palette =
'pastel')
        count = count + 1

    plt.tight_layout()
    plt.show()
```



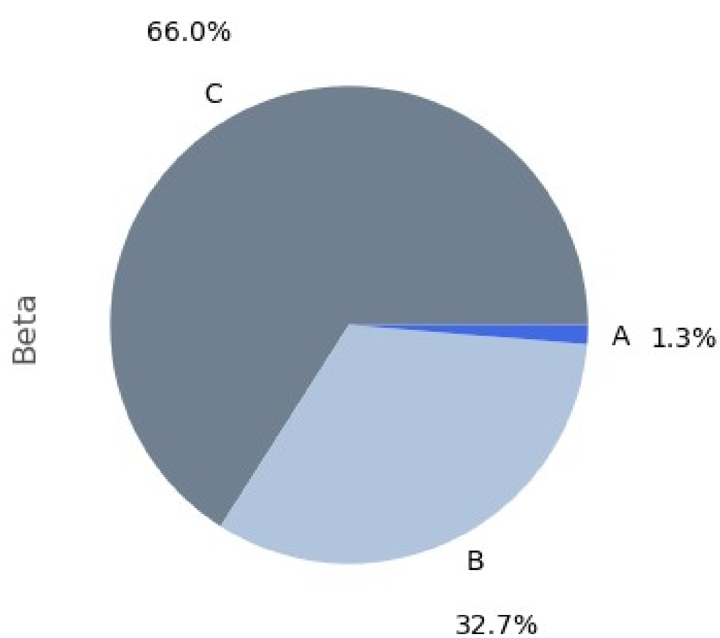
Column "AB" with respect to class 0 and class 1
Column "AF" with respect to class 0 and class 1
Column "AH" with respect to class 0 and class 1
Column "AM" with respect to class 0 and class 1
Column "AR" with respect to class 0 and class 1
Column "AX" with respect to class 0 and class 1
Column "AY" with respect to class 0 and class 1
Column "AZ" with respect to class 0 and class 1
Column "BC" with respect to class 0 and class 1
Column "BD " with respect to class 0 and class 1
Column "BN" with respect to class 0 and class 1
Column "BP" with respect to class 0 and class 1
Column "BQ" with respect to class 0 and class 1
Column "BR" with respect to class 0 and class 1
Column "BZ" with respect to class 0 and class 1
Column "CB" with respect to class 0 and class 1
Column "CC" with respect to class 0 and class 1
Column "CD " with respect to class 0 and class 1
Column "CF" with respect to class 0 and class 1
Column "CH" with respect to class 0 and class 1
Column "CL" with respect to class 0 and class 1
Column "CR" with respect to class 0 and class 1
Column "CS" with respect to class 0 and class 1
Column "CU" with respect to class 0 and class 1
Column "CW " with respect to class 0 and class 1
Column "DA" with respect to class 0 and class 1
Column "DE" with respect to class 0 and class 1
Column "DF" with respect to class 0 and class 1
Column "DH" with respect to class 0 and class 1
Column "DI" with respect to class 0 and class 1
Column "DL" with respect to class 0 and class 1
Column "DN" with respect to class 0 and class 1
Column "DU" with respect to class 0 and class 1
Column "DV" with respect to class 0 and class 1
Column "DY" with respect to class 0 and class 1
Column "EB" with respect to class 0 and class 1
Column "EE" with respect to class 0 and class 1
Column "EG" with respect to class 0 and class 1
Column "EH" with respect to class 0 and class 1
Column "EL" with respect to class 0 and class 1
Class

```
if flag_EDA:
    # check the percentage of presence of age-related conditions and
the 3 experimental characteristics
    colors = ['slategrey', 'lightsteelblue', 'royalblue',
'lightskyblue', 'steelblue', 'lavender', 'cadetblue']

    # loop via greeks columns to get number of occurrence of each
category and create pie plot
    for i in greeks.columns[1:-1]:
        print('\033[1m' + '**' + i + '**' + '\033[0m')
        value_count = greeks[i].value_counts()
        print('\nNumber of occurrence: \n', value_count, sep = '\n')

        # plot pie chart
        value_count.plot(kind = 'pie', figsize = (4,4), autopct =
'%1.1f%%', pctdistance = 1.4, colors = colors)
        plt.show()

print('---------------------------------------------------------------
---')

print('---------------------------------------------------------------
---')
```

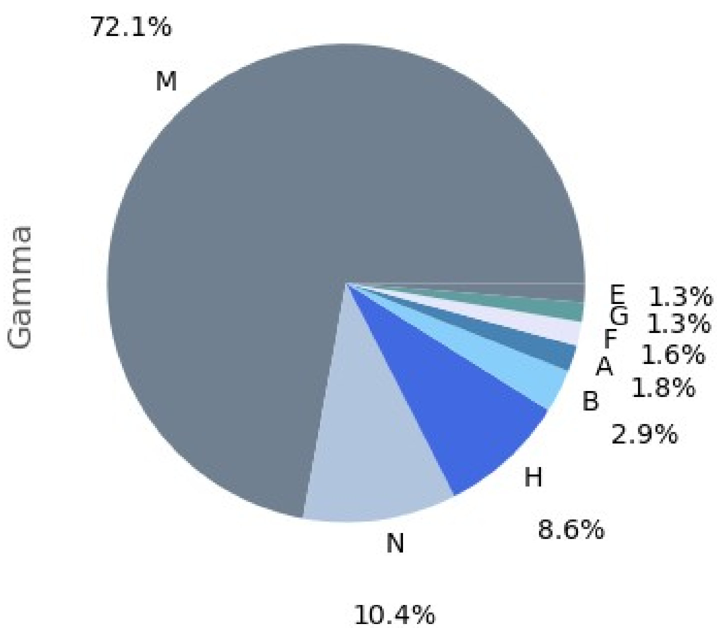
```
**Alpha**

Number of occurrence:

A    509
B     61
G     29
D     18
Name: Alpha, dtype: int64
```



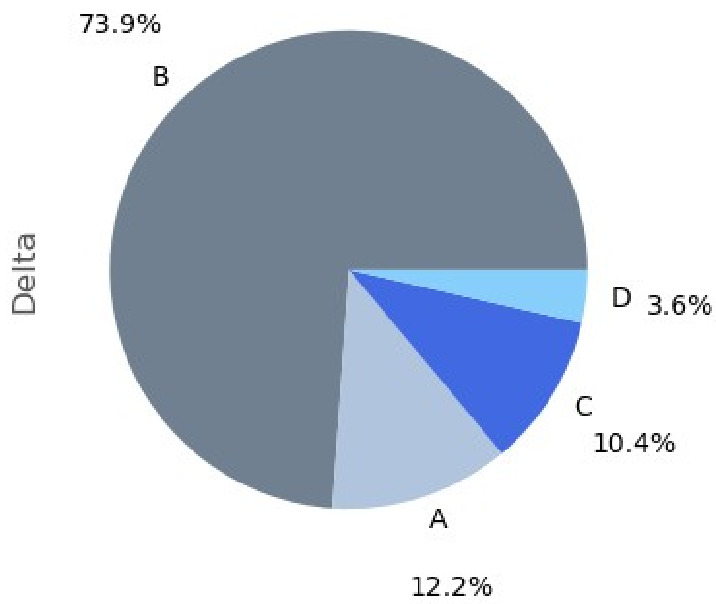

```
-------------------------------------------------------------------
-------------------------------------------------------------------
**Beta**

Number of occurrence:

C    407
B    202
A      8
Name: Beta, dtype: int64
```



```
----------------------------------------------------------------------
----------------------------------------------------------------------
**Gamma**

Number of occurrence:

M    445
N     64
H     53
B     18
A     11
F     10
G      8
E      8
Name: Gamma, dtype: int64
```

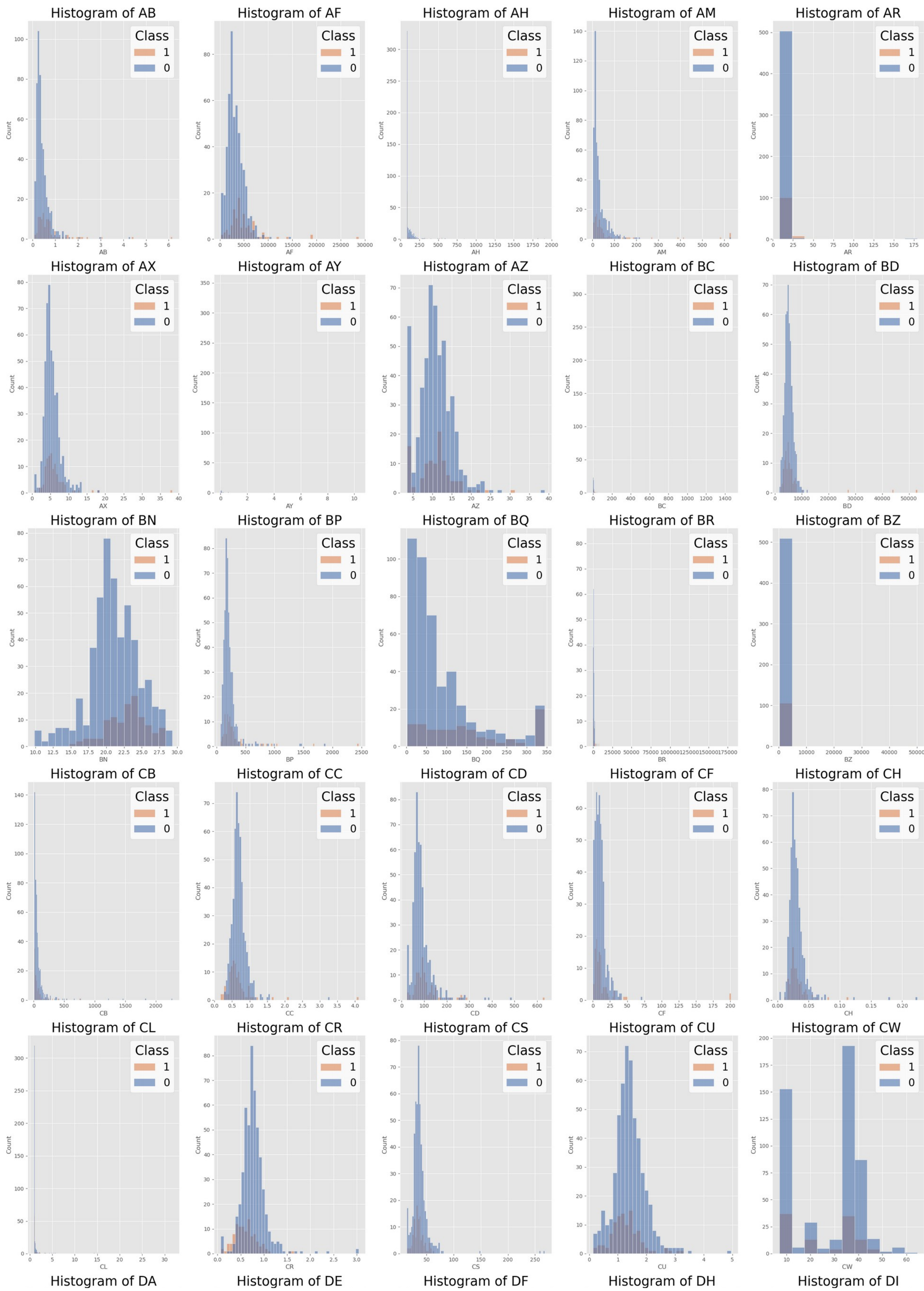



```
----------------------------------------------------------------------
----------------------------------------------------------------------
**Delta**

Number of occurrence:

B    456
A     75
C     64
D     22
Name: Delta, dtype: int64
```




```
--------------------------------------------------------------------------
--------------------------------------------------------------------------
```

```
if flag_EDA:
    # plot histogram to check out the distribution of each numerical
column from training data
    fig = plt.figure(figsize = (25,75))
    count = 1

    for col in train.loc[:,~train.columns.isin(['Id', 'EJ',
'Class'])]:
        plt.subplot(11, 5, count)
        plt.title(f'Histogram of {col}', fontsize = 25)
        sns.histplot(x = train[col], hue = train['Class'], palette =
'deep', alpha = 0.6)
        plt.legend(train['Class'], title = 'Class', title_fontsize =
25, fontsize = 20, facecolor = 'white')
        count = count + 1

    plt.tight_layout()
    plt.show()
```

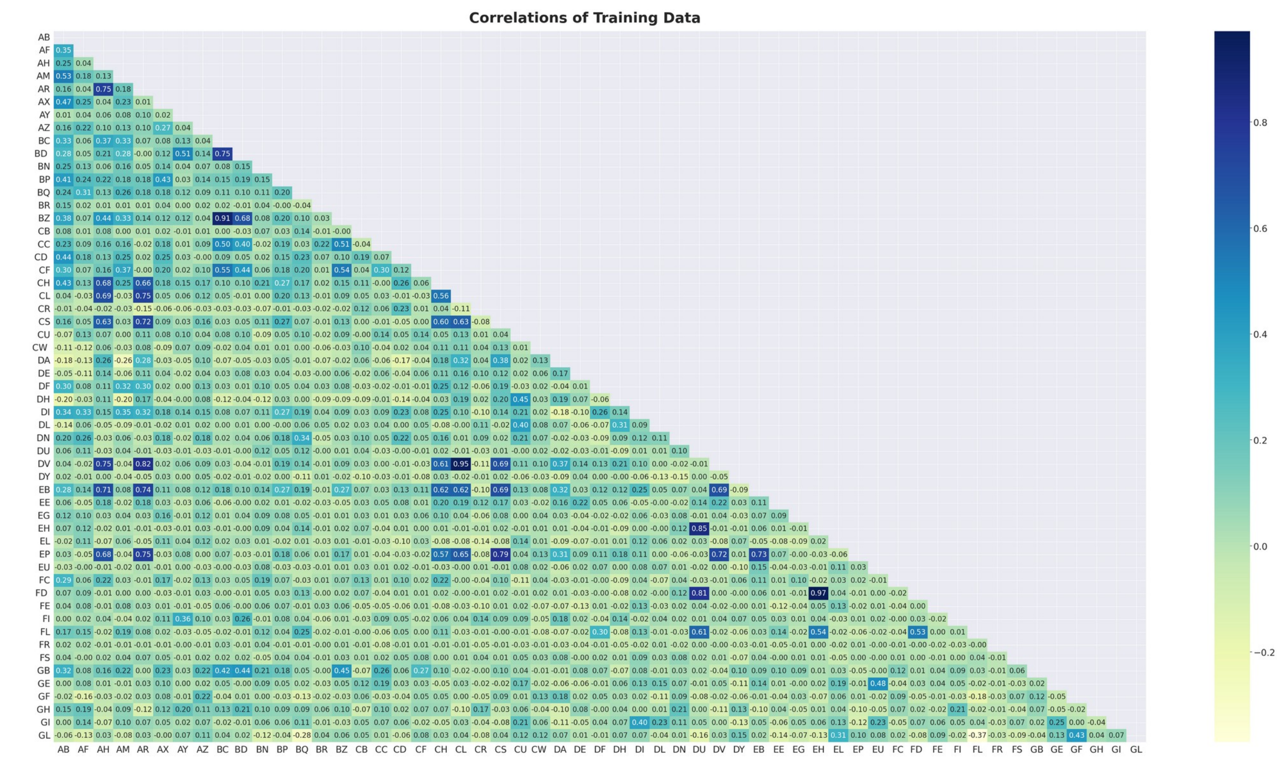

Histogram of AB
Histogram of AF
Histogram of AH
Histogram of AM
Histogram of AR
Histogram of AX
Histogram of AY
Histogram of AZ
Histogram of BC
Histogram of BD
Histogram of BN
Histogram of BP
Histogram of BQ
Histogram of BR
Histogram of BZ
Histogram of CB
Histogram of CC
Histogram of CD
Histogram of CF
Histogram of CH
Histogram of CL
Histogram of CR
Histogram of CS
Histogram of CU
Histogram of CW
Histogram of DA
Histogram of DE
Histogram of DF
Histogram of DH
Histogram of DI
Class
1
0
Count

The distribution from the resulting histogram for some columns are unclear due to the bins and outliers. However, some plots already clearly show the sign of long tails. To have a better look at the distribution, kernel density plot of each column may help further.

```
if flag_EDA:
    # plot kernel density plot(KDE) to examine tthe distribution of
each numerical column from training data
    fig = plt.figure(figsize = (15,15))
    count = 1

    for col in train.loc[:, ~train.columns.isin(['Id','EJ','Class'])]:
        plt.subplot(11, 5, count)
        plt.title(f'KDE plot of {col}', fontsize = 12)
        sns.kdeplot(data = train, x = train[col], hue =
train['Class'], multiple = 'stack', palette = 'deep')
        plt.legend(train['Class'], title = 'Class', title_fontsize =
8, fontsize = 5, facecolor = 'white')
        count = count + 1

    plt.tight_layout()
    plt.show()
```


```
if flag_EDA:
    # create a triangle correlation heatmap (remove the repeated
columns and rows)
    fig = plt.figure(figsize = (55,30))
    sns.set(font_scale = 2.3)

    # create a mask
    mask_train = np.triu(np.ones_like(train.loc[:,
~train.columns.isin(['Id', 'EJ', 'Class'])].corr()))

    # plot correlation heatmap
    train_corr_heatmap = sns.heatmap(train.loc[:,
```

```
~train.columns.isin(['Id', 'EJ', 'Class'])].corr(),
                                                     cmap = 'YlGnBu',
                                                     annot = True,
                                                     fmt = '.2f',
                                                     linewidth = 0,
                                                     mask = mask_train,
                                                     annot_kws = {'size':20})

    plt.suptitle('Correlations of Training Data', x = 0.4, fontsize =
40, fontweight = 'bold')
    plt.tight_layout()
    plt.show()
```



From the heatmap, it shows some features are highly correlated. Thus, some models required these highly correlated features to be removed first before training for better prediction results.

Imputation of Missing Values and Standardization

```
# extract column names from train/test data set
train_col_name = train.columns.drop(['Id', 'EJ', 'Class'])
test_col_name = test.columns.drop(['Id', 'EJ'])

# instantiate KNN imputer for fill out missing values
```

```
imputer = KNNImputer(n_neighbors = 10)

# instantiate scaler to scale features
scaler = RobustScaler()

train_fillna = imputer.fit_transform(train.iloc[:,
~train.columns.isin(['Id', 'EJ', 'Class'])])
train_scaled = pd.DataFrame(scaler.fit_transform(train_fillna))

# no missing/null value in test set, so test set only needs to be
scaled
test_scaled = pd.DataFrame(scaler.fit_transform(test.iloc[:,
~test.columns.isin(['Id', 'EJ'])]))

# add column names back to processed train data/test data
train_scaled.columns = train_col_name
test_scaled.columns = test_col_name

# drop all columns that is in train_scaled/test_scaled from train/test
train = train.drop(train_scaled, axis = 1)
test = test.drop(test_scaled, axis = 1)

# concatenate train/test and train_scaled/test_scaled together to make
train/test set complete
train = pd.concat([train, train_scaled], axis = 1)
test = pd.concat([test, test_scaled], axis = 1)

print('Check whether missing values are filled in' + '\033[1m' + '
train ' + '\033[0m' + 'data: \n', train.isnull().sum(), sep = '\n')
print('\n Check the shape of' + '\033[1m' + ' train ' + '\033[0m' +
'data: ', train.shape, sep = '\n')
display(train.head())
print('-------------------------------------------------------------------
---')
print('-------------------------------------------------------------------
---')
print('\n Check the shape of' + '\033[1m' + ' test ' + '\033[0m' +
'data: ', test.shape, sep = '\n')
display(test.head())
```

```
Check whether missing values are filled in train data:

Id        0
EJ        0
Class     0
AB        0
AF        0
AH        0
AM        0
AR        0
```

AX	0
AY	0
AZ	0
BC	0
BD	0
BN	0
BP	0
BQ	0
BR	0
BZ	0
CB	0
CC	0
CD	0
CF	0
CH	0
CL	0
CR	0
CS	0
CU	0
CW	0
DA	0
DE	0
DF	0
DH	0
DI	0
DL	0
DN	0
DU	0
DV	0
DY	0
EB	0
EE	0
EG	0
EH	0
EL	0
EP	0
EU	0
FC	0
FD	0
FE	0
FI	0
FL	0
FR	0
FS	0
GB	0
GE	0
GF	0
GH	0
GI	0

```
GL       0
dtype: int64

 Check the shape of train data:
(617, 58)

             Id EJ  Class        AB        AF        AH        AM   AR
\
0  000ff2bfdfe9  B      1 -0.472222 -0.005214  0.000000  0.069272  0.0

1  007255e47698  A      0 -0.680556 -0.989494  0.000000  0.611687  0.0

2  013f2bd269f5  B      0  0.375000 -0.224190  0.000000  0.440180  0.0

3  043ac50845d5  B      0 -0.333333  0.323123  1.226427  2.105694  0.0

4  044fb8a146ec  B      1  0.083333  0.283109  0.000000 -0.239281  0.0


         AX        AY  ...        FI        FL         FR        FS
GB  \
0 -1.880769  0.000000  ... -2.125230  0.702705   0.598571 -0.347826 -
0.642283
1 -0.607692  0.000000  ...  0.138122 -0.472232  -0.624571  0.666667 -
0.819132
2  0.738462  0.000000  ...  0.561694  0.770546  -0.153143  2.014493
1.581994
3 -0.584615  0.000000  ...  1.639042  0.508776  -0.624571  0.057971 -
0.020900
4 -0.473077  2.594595  ...  1.243094  0.843681  46.670286 -0.289855 -
0.204180

         GE        GF        GH        GI        GL
0  0.000000 -0.359338 -0.716263  0.641741 -0.010025
1  0.000000  1.240601 -0.124567 -0.197595  0.990161
2  0.290982  0.359598 -0.218622 -0.129459 -0.006520
3  0.178349 -0.353767  0.789556  1.101632 -0.008401
4  1.336815  0.042256  1.248820 -0.105640 -0.011111

[5 rows x 58 columns]

------------------------------------------------------------------
------------------------------------------------------------------

 Check the shape of test data:
(5, 57)

             Id EJ   AB   AF   AH   AM   AR   AX   AY   AZ  ...   FI
FL  \
0  00eed32682bb  A  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
```

```
1  010ebe33f668  A  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
2  02fa521e1838  A  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
3  040e15f562a2  A  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
4  046e85c7cc7f  A  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0

    FR   FS   GB   GE   GF   GH   GI   GL
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 57 columns]
```

All missing values are filled and all numerical columns are standardized.

Label Encoding

```
# instantiate label encoding function
le = sklearn.preprocessing.LabelEncoder()
train['EJ'] = le.fit_transform(train['EJ'])
test['EJ'] = le.fit_transform(test['EJ'])
print('\n Check the difference unique values in column "EJ" from' + '\
033[1m' + ' train ' + '\033[0m' + ':', train['EJ'].unique())
print('\n After label encoding,' + '\033[1m' + ' train ' + '\033[0m' +
'looks like:')
display(train)
print('\n Check the difference unique values in column "EJ" from' + '\
033[1m' + ' test ' + '\033[0m' + ':', test['EJ'].unique())
print('\n After label encoding,' + '\033[1m' + ' test ' + '\033[0m' +
'looks like:')
display(test)
```

```
 Check the difference unique values in column "EJ" from train : [1 0]

 After label encoding, train looks like:

               Id  EJ  Class        AB        AF         AH        AM
\
0    000ff2bfdfe9   1      1 -0.472222 -0.005214   0.000000  0.069272

1    007255e47698   0      0 -0.680556 -0.989494   0.000000  0.611687

2    013f2bd269f5   1      0  0.375000 -0.224190   0.000000  0.440180
```

```
3    043ac50845d5   1      0 -0.333333  0.323123   1.226427  2.105694
4    044fb8a146ec   1      1  0.083333  0.283109   0.000000 -0.239281
..            ...  ..    ...       ...       ...        ...       ...
612  fd3dafe738fd   0      0 -0.666667  0.004501   1.351236 -0.410097
613  fd895603f071   1      0  0.263889  1.081978   0.000000  0.968303
614  fd8ef6377f76   0      0  0.236111 -0.305510   1.574611  1.295989
615  fe1942975e40   1      0  0.027778 -0.857917   0.000000  0.117335
616  ffcca4ded3bb   0      0  0.416667 -0.206897  16.169362  3.404334

           AR        AX        AY  ...        FI        FL         FR
\
0    0.000000 -1.880769  0.000000  ... -2.125230  0.702705   0.598571
1    0.000000 -0.607692  0.000000  ...  0.138122 -0.472232  -0.624571
2    0.000000  0.738462  0.000000  ...  0.561694  0.770546  -0.153143
3    0.000000 -0.584615  0.000000  ...  1.639042  0.508776  -0.624571
4    0.000000 -0.473077  2.594595  ...  1.243094  0.843681  46.670286
..        ...       ...       ...  ...       ...       ...        ...
612  4.882164 -0.665385  4.594595  ... -0.022099 -0.472232   0.128000
613  7.834536  0.411538  0.027027  ...  0.322284  1.185050   0.109714
614  1.866864  1.319231  0.000000  ...  0.696133 -0.472232  -0.624571
615  0.000000  1.280769  0.000000  ... -0.640884  1.025726  -0.338286
616  0.000000 -0.796154  8.108108  ... -0.734807 -0.472232   0.013714

           FS        GB        GE        GF        GH        GI
GL
0   -0.347826 -0.642283  0.000000 -0.359338 -0.716263  0.641741 -
0.010025
1    0.666667 -0.819132  0.000000  1.240601 -0.124567 -0.197595
0.990161
2    2.014493  1.581994  0.290982  0.359598 -0.218622 -0.129459 -
0.006520
```

```
3    0.057971 -0.020900  0.178349 -0.353767  0.789556  1.101632 -
0.008401
4   -0.289855 -0.204180  1.336815  0.042256  1.248820 -0.105640 -
0.011111
..        ...       ...       ...       ...       ...       ...
...
612 -0.405797 -0.847267  2.628881  0.015869 -0.504561  0.627426
0.990161
613  0.362319  1.479904  7.718779 -0.292729 -0.081158  1.865560 -
0.008881
614 -0.405797  0.102894  1.023740 -0.083984 -0.375590  1.748697
0.990161
615  0.884058  0.503215  0.000000 -0.361707 -0.464297 -0.085772 -
0.007084
616 -0.231884 -0.440514  0.000000 -0.060837  1.279648  1.643687
0.990161

[617 rows x 58 columns]


 Check the difference unique values in column "EJ" from test : [0]

 After label encoding, test looks like:

             Id  EJ   AB   AF   AH   AM   AR   AX   AY   AZ  ...   FI
FL  \
0  00eed32682bb   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
1  010ebe33f668   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
2  02fa521e1838   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
3  040e15f562a2   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
4  046e85c7cc7f   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0

    FR   FS   GB   GE   GF   GH   GI   GL
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 57 columns]
```

Logistic Regression

```
# make a copy of train set
train_lr = train.copy(deep = True)

# set features and target
X_lr = train_lr.drop(['Id', 'Class'], axis = 1) # features
y_lr = train_lr['Class'] # target

# split training set to training (80%) and testing (20%)
X_lr_train, X_lr_test, y_lr_train, y_lr_test = train_test_split(X_lr,
y_lr, test_size = 0.2, random_state = 42)
```

Create a baseline model with balanced weight and all features

```
# instantiate Logistic Regression
lr = LogisticRegression(class_weight = 'balanced', solver = 'lbfgs',
random_state = 42, max_iter = 300)

# fit the Logistic Regression model
lr_baseline = lr.fit(X_lr_train, y_lr_train)

'''Test Part of the Train Set'''

# making predictions on test part of the data from train set
pred_lr_baseline = lr_baseline.predict(X_lr_test)
# calculating probability on test part of the data form train set
pred_proba_lr_baseline = lr_baseline.predict_proba(X_lr_test)

'''Train Part of the Train Set'''

# making predictions on train part of the data from train set
train_pred_lr_baseline = lr_baseline.predict(X_lr_train)
# calculating probability on train part of the data from train set
train_pred_proba_lr_baseline = lr_baseline.predict_proba(X_lr_train)

# checking how prediction results look
print('Actual results vs. Prediction results (class 0 or class 1) of
baseline logistic regression model: ')
compare_result_lr = pd.DataFrame({'actual_result': y_lr_test,
'prediction_result': pred_lr_baseline})
display(compare_result_lr.head())

print('\nProbability of each subject is in to each class respectively
for baseline logistic regression model: ')
proba_lr_baseline = pd.DataFrame(pred_proba_lr_baseline)
proba_lr_baseline = proba_lr_baseline.set_axis(['class_0', 'class_1'],
axis = 1)
display(proba_lr_baseline.head())
```

```
Actual results vs. Prediction results (class 0 or class 1) of baseline
logistic regression model:

     actual_result  prediction_result
49               1                  1
581              0                  0
82               0                  0
304              1                  1
109              1                  1


Probability of each subject is in to each class respectively for
baseline logistic regression model:

    class_0   class_1
0  0.010608  0.989392
1  0.607021  0.392979
2  0.992295  0.007705
3  0.228700  0.771300
4  0.000054  0.999946
```

Evaluation on Logistic Regression

```
eval_metrics('Logistic Regression on Test Set', y_lr_test,
pred_lr_baseline, pred_proba_lr_baseline)
print(f'Balanced log loss of logistic regression with backward
elimination on test set: {balanced_log_loss(y_lr_test,
pred_lr_baseline):.2f}')

eval_metrics('Logistic Regression on Train Set', y_lr_train,
train_pred_lr_baseline, train_pred_proba_lr_baseline)
print(f'Balanced log loss of logistic regression with backward
elimination on test set: {balanced_log_loss(y_lr_train,
train_pred_lr_baseline):.2f}')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Scoring Metrics for Logistic Regression on Test Set
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 Log loss: 1.1247529187320637

------------------------------------

 Classification report:
               precision    recall  f1-score   support

           0       0.97      0.91      0.94       101
           1       0.69      0.87      0.77        23

    accuracy                           0.90       124
   macro avg       0.83      0.89      0.85       124
```

```
weighted avg       0.92      0.90      0.91       124

------------------------------------
Balanced Accuracy Score: 0.89
Accuracy Score: 0.90
Precision Score: 0.69
Recall Score: 0.87
F1 Score: 0.77
ROC AUC Score: 0.89


bll y_true:  49     1
581    0
82     0
304    1
109    1
      ..
6      0
104    0
114    0
158    0
181    1
Name: Class, Length: 124, dtype: int64
bll y_pred:  [1 0 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0 0
 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1
0 1
 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0
0 0
 0 0 0 1 0 0 0 0 0 0 0 0 1]
Balanced log loss of logistic regression with backward elimination on
test set: 3.79
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Scoring Metrics for Logistic Regression on Train Set
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 Log loss: 0.21703182066900373

------------------------------------

 Classification report:
               precision    recall  f1-score   support

           0       0.99      0.92      0.95       408
           1       0.70      0.95      0.81        85

    accuracy                           0.92       493
   macro avg       0.85      0.93      0.88       493
weighted avg       0.94      0.92      0.93       493
```

```
------------------------------------
Balanced Accuracy Score: 0.93
Accuracy Score: 0.92
Precision Score: 0.70
Recall Score: 0.95
F1 Score: 0.81
ROC AUC Score: 0.93


bll y_true:  530    1
363    0
177    0
312    0
199    0
      ..
71     0
106    0
270    0
435    0
102    0
Name: Class, Length: 493, dtype: int64
bll y_pred:  [1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0
1 0 0 1 0 0 0 1 0
 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 1 0 1 0 0 1 0 0 0
0 0
 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0
1 1
 0 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0
0 1
 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 1 0 0 1 1 1 0 0
0 1
 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0
 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0
0 0
 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0
 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 0 1 0 0 1 0 1 0 0 0 0 1 1 0 0
1 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1
 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0
0 1
 0 0 1 0 0 1 0 0 0 0 0 1]
```

```
Balanced log loss of logistic regression with backward elimination on
test set: 2.25
```

Logistic Regression with Backward Elimination Method

```
if flag_backElimination:
    # backward feature selection
    sbs = sfs(lr, n_features_to_select = 'auto', direction =
'backward', cv = 5)

    # fit model
    sbs = sbs.fit(X_lr_train, y_lr_train)

    # check the selected features
    features_selected = sbs.get_support()
    features_selected = X_lr.columns[features_selected].tolist()
    print(f'Number of features selected: {len(features_selected)}\n')
    print(f'Selected features: {features_selected}\n')

    X_sbs = pd.DataFrame(X_lr[features_selected])

    display(X_sbs.head())
```

```
Number of features selected: 28

Selected features: ['EJ', 'AB', 'AH', 'AX', 'BQ', 'BZ', 'CB', 'CC',
'CD ', 'CR', 'DE', 'DH', 'DI', 'DU', 'DV', 'EG', 'EP', 'EU', 'FC', 'FD
', 'FE', 'FI', 'FR', 'FS', 'GB', 'GE', 'GF', 'GH']
```

```
   EJ        AB        AH        AX        BQ   BZ        CB        CC
\
0   1 -0.472222  0.000000 -1.880769  0.821764  0.0  0.086549 -0.463220

1   0 -0.680556  0.000000 -0.607692 -0.528767  0.0 -0.227447 -0.841935

2   1  0.375000  0.000000  0.738462  1.473886  0.0 -0.185195 -0.788369

3   1 -0.333333  1.226427 -0.584615 -0.565031  0.0 -0.507028  0.279115

4   1  0.083333  0.000000 -0.473077  0.792487  0.0  0.735156 -0.593097


        CD         CR  ...        FC        FD         FE        FI
FR  \
0 -1.608224 -2.452321  ... -0.746073  1.831606  0.306960 -2.125230
0.598571
1 -0.831902  1.434529  ... -0.623729 -0.343264 -0.102154  0.138122  -
0.624571
2  0.174876 -0.112872  ...  6.074663  1.500000  0.181235  0.561694  -
0.153143
```

```
3  0.237686 -0.351126  ...  0.739235  1.312176  0.660302  1.639042  -
0.624571
4 -0.204474 -0.139561  ... -0.237849  0.524611  1.614526  1.243094
46.670286

         FS        GB        GE        GF        GH
0 -0.347826 -0.642283  0.000000 -0.359338 -0.716263
1  0.666667 -0.819132  0.000000  1.240601 -0.124567
2  2.014493  1.581994  0.290982  0.359598 -0.218622
3  0.057971 -0.020900  0.178349 -0.353767  0.789556
4 -0.289855 -0.204180  1.336815  0.042256  1.248820

[5 rows x 28 columns]
```

```
if flag_backElimination:
    # split the training set to training (80%) and testing (20%) from
feature selected data set
    X_sbs_train, X_sbs_test, y_sbs_train, y_sbs_test =
train_test_split(X_sbs, y_lr, test_size = 0.2, random_state = 42)

    # create Logistic Regression model with selected features
    lr_sbs = lr.fit(X_sbs_train, y_sbs_train)

    '''Test Part of the Train Set'''

    # make predictions on test set of training data
    pred_lr_sbs = lr_sbs.predict(X_sbs_test)
    # calculating probability on test set of training data
    pred_proba_lr_sbs = lr_sbs.predict_proba(X_sbs_test)

    '''Train Part of the Train Set'''

    # making predictions on train part of the data from train set
    train_pred_lr_sbs = lr_sbs.predict(X_sbs_train)
    # calculating probability on train part of the data from train set
    train_pred_proba_lr_sbs = lr_sbs.predict_proba(X_sbs_train)
```

Evaluation on Logistic Regression With Backward Elimination Method

```
if flag_backElimination:
    eval_metrics('Logistic Regression with Backward Elimination on
Test Set', y_sbs_test, pred_lr_sbs, pred_proba_lr_sbs)
    print(f'Balanced log loss of logistic regression with backward
elimination on test set: {balanced_log_loss(y_sbs_test,
pred_lr_sbs):.2f}')

    eval_metrics('Logistic Regression with Backward Elimination on
Train Set', y_sbs_train, train_pred_lr_sbs, train_pred_proba_lr_sbs)
    print(f'Balanced log loss of logistic regression with backward
```

```
elimination on test set: {balanced_log_loss(y_sbs_train,
train_pred_lr_sbs):.2f}')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Scoring Metrics for Logistic Regression with Backward Elimination on
Test Set
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 Log loss: 0.35410450629046336

------------------------------------

 Classification report:
               precision    recall  f1-score   support

           0       0.94      0.83      0.88       101
           1       0.51      0.78      0.62        23

    accuracy                           0.82       124
   macro avg       0.73      0.81      0.75       124
weighted avg       0.86      0.82      0.84       124

------------------------------------
Balanced Accuracy Score: 0.81
Accuracy Score: 0.82
Precision Score: 0.51
Recall Score: 0.78
F1 Score: 0.62
ROC AUC Score: 0.81


bll y_true:  49     1
581    0
82     0
304    1
109    1
      ..
6      0
104    0
114    0
158    0
181    1
Name: Class, Length: 124, dtype: int64
bll y_pred:  [1 1 0 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0 0
 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1
0 0
 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0
0 0
 0 0 0 1 0 0 1 0 0 0 0 0 1]
```

```
Balanced log loss of logistic regression with backward elimination on
test set: 6.66
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Scoring Metrics for Logistic Regression with Backward Elimination on
Train Set
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 Log loss: 0.29136773407318306

------------------------------------

 Classification report:
               precision    recall  f1-score   support

           0       0.98      0.92      0.95       408
           1       0.70      0.92      0.79        85

    accuracy                           0.92       493
   macro avg       0.84      0.92      0.87       493
weighted avg       0.93      0.92      0.92       493

------------------------------------
Balanced Accuracy Score: 0.92
Accuracy Score: 0.92
Precision Score: 0.70
Recall Score: 0.92
F1 Score: 0.79
ROC AUC Score: 0.92


bll y_true:  530    1
363    0
177    0
312    0
199    0
      ..
71     0
106    0
270    0
435    0
102    0
Name: Class, Length: 493, dtype: int64
bll y_pred:  [1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0
1 0 0 1 0 0 0 1 0
 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 1 0 1 1 0 1 0 0 0
0 0
 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 1
 0 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
```

```
 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0
0 0
 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 1 1 1 0 1
0 1
 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0
 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0
0 0
 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0
 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 0 1 0 0 1 0 1 0 0 0 0 1 1 0 1
1 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 1
 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0
0 1
 0 0 1 0 0 1 1 0 0 0 0 1]
Balanced log loss of logistic regression with backward elimination on
test set: 2.86
```

Oversampling - SMOTE

```
if flag_smote:
    # make a copy of train data
    train_os = train.copy(deep = True)

    # apply feature selection to train data
    train_os_X = train_os[features_selected]

    # create target
    train_os_y = train['Class']

    X_train, X_test, y_train, y_test = train_test_split(train_os_X,
train_os_y, test_size = 0.2, random_state = 42)

    print('Shape of train: ', X_train.shape)
    print('Shape of test: ', X_test.shape)

    # set a counter to count the number of data in each class
    counter = Counter(y_train)
    print('\033[1m' + 'Before ' + '\033[0m' + 'applying SMOTE to
train: ', counter)

    # perform oversampling
    oversample = SMOTE(sampling_strategy = {0:600, 1:600},
random_state = 42)
```

```
    X_smote, y_smote = oversample.fit_resample(X_train, y_train)
    counter = Counter(y_smote)
    print('\033[1m' + 'After ' + '\033[0m' + 'applying SMOTE to train:
', counter)
```

```
Shape of train:  (493, 28)
Shape of test:  (124, 28)
Before applying SMOTE to train:  Counter({0: 408, 1: 85})
After applying SMOTE to train:  Counter({1: 600, 0: 600})
```

Now, the data of each class is balance weighted and can proceed with modeling.

Random Forest Classifier

```
# create parameter grid
param_rf = {'max_depth' : [3,6,9],
            'n_estimators' : [100,300,500],
            'min_samples_leaf' : [2,4,6],
            'max_features' : [None]}

skf = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 42)

if flag_smote:
    # instantiate random forest classifier
    rf = RandomForestClassifier(class_weight = 'balanced',
random_state = 42)

    # randomized search on hyper paramters
    rf_clf = RandomizedSearchCV(rf,
                                param_distributions = param_rf,
                                n_iter = 100,
                                cv = skf.split(X_smote, y_smote),
                                scoring = 'roc_auc',
                                verbose = -1,
                                random_state = 42)

    # fit a random forest model
    rf_clf.fit(X_smote, y_smote)

if flag_smote:
    # find best forest estimation and parameters
    print('Best random forest estimation - ', rf_clf.best_estimator_,
sep = '\n')
    print('Best parameters - ', rf_clf.best_params_)
```

```
Best random forest estimation -
RandomForestClassifier(class_weight='balanced', max_depth=9,
max_features=None,
                       min_samples_leaf=2, n_estimators=500,
```

```
random_state=42)
Best parameters -  {'n_estimators': 500, 'min_samples_leaf': 2,
'max_features': None, 'max_depth': 9}
```

```
if flag_smote:
    # use the best estimator and optimize the model
    rf_best =
RandomForestClassifier(**rf_clf.best_estimator_.get_params())

    rf_best.fit(X_smote, y_smote)
    rf_pred = rf_best.predict(X_test)
    accuracy = rf_best.score(X_test, y_test)
    print('Accuracy of Random Forest: {}'.format(accuracy))
    rf_log_loss = balanced_log_loss(y_test, rf_pred)
    print('Balanced log loss of random forest: ', rf_log_loss)
```

```
Accuracy of Random Forest: 0.8870967741935484
bll y_true:  49     1
581    0
82     0
304    1
109    1
      ..
6      0
104    0
114    0
158    0
181    1
Name: Class, Length: 124, dtype: int64
bll y_pred:  [1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0
0 0 0 1 0 0 1 0 0
 0 0 0 0 1 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 1
 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1
0 0
 0 0 0 1 0 0 0 0 1 0 0 1 1]
Balanced log loss of random forest:  3.553541929758511
```

XGBoost

```
param_xgb = {'gamma' : [0, 0.1, 0.2, 0.3],
             'max_depth' : [6, 9, 12, 15],
             'subsample' : [0.7, 0.8, 0.9, 1.0],
             'colsample_bytree' : [0.7, 0.8, 0.9, 1.0],
             'min_child_weight' : [1, 2, 3, 4],
             'scale_pos_weight' : [2, 3, 4, 5],
             'reg_lambda' : [0, 0.25, 0.5, 0.75, 1],
```

```
              'reg_alpha' : [0, 0.25, 0.5, 0.75, 1],
              }

if flag_smote:
    # instantiate XGBoost Classifier
    xgb = XGBClassifier(n_estimators = 1000,
                        learning_rate = 0.01,
                        objective = 'binary:logistic')

    # randomize search on hyperparameter search
    xgb_clf = RandomizedSearchCV(xgb,
                                 param_distributions = param_xgb,
                                 cv = skf.split(X_smote, y_smote),
                                 scoring = 'roc_auc',
                                 verbose = -1,
                                 random_state = 42)

    # fit the model using data processed by SMOTE
    xgb_clf.fit(X_smote, y_smote)

if flag_smote:
    # find best estimators and parameters
    print('Best estimator - ', xgb_clf.best_estimator_)
    print('Best parameters - ', xgb_clf.best_params_)
```

```
Best estimator -  XGBClassifier(base_score=None, booster=None,
callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=1.0, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None,
feature_types=None,
              gamma=0, gpu_id=None, grow_policy=None,
importance_type=None,
              interaction_constraints=None, learning_rate=0.01,
max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=6, max_leaves=None,
              min_child_weight=1, missing=nan,
monotone_constraints=None,
              n_estimators=1000, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
Best parameters -  {'subsample': 0.7, 'scale_pos_weight': 3,
'reg_lambda': 0.25, 'reg_alpha': 0, 'min_child_weight': 1,
'max_depth': 6, 'gamma': 0, 'colsample_bytree': 1.0}
```

```
if flag_smote:
    # use best estimator selected and optimize model
    xgb_best = XGBClassifier(**xgb_clf.best_estimator_.get_params())

    xgb_best.fit(X_smote, y_smote)
```

```
    xgb_pred = xgb_best.predict(X_test)
    accuracy = xgb_best.score(X_test, y_test)
    print('Accuracy of XGBoost: {}'.format(accuracy))
    xgb_log_loss = balanced_log_loss(y_test, xgb_pred)
    print('Balanced log loss of XGBoost: ', xgb_log_loss)
```

```
Accuracy of XGBoost: 0.9112903225806451
bll y_true:  49     1
581    0
82     0
304    1
109    1
      ..
6      0
104    0
114    0
158    0
181    1
Name: Class, Length: 124, dtype: int64
bll y_pred:  [1 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0
1 0 0 1 0 0 1 0 0
 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
0 1
 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1
0 0
 0 0 0 1 0 0 0 0 0 0 0 1 1]
Balanced log loss of XGBoost:  3.040577929872261
```

LGBM

```
param_lgbm = {'learning_rate': [0.01, 0.05, 0.1],
              'n_estimators' : [50, 100, 500, 1000],
              'colsample_bytree': [0.25, 0.5, 0.75, 0.9],
              'max_depth' : [2, 4, 6, 8]}

# instantiate LGBM model
lgbm = LGBMClassifier(n_estimators = 1000,
                      learning_rate = 0.01,
                      random_state = 42)

if flag_smote:
    # randomize search on hyperparameters
    lgbm_clf = RandomizedSearchCV(lgbm,
                                  param_distributions = param_lgbm,
                                  n_iter = 1000,
                                  scoring = 'roc_auc',
                                  n_jobs = -1,
                                  cv = skf.split(X_smote, y_smote),
```

```
                                    verbose = -1,
                                    random_state = 42)
    lgbm_clf.fit(X_smote, y_smote)
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
```

```
if flag_smote:
    # find best estimators and parameters
    print('Best estimator - ', lgbm_clf.best_estimator_)
    print('Best parameters - ', lgbm_clf.best_params_)
```

```
Best estimator -  LGBMClassifier(colsample_bytree=0.25,
learning_rate=0.05, max_depth=8,
               n_estimators=500, random_state=42)
Best parameters -  {'n_estimators': 500, 'max_depth': 8,
'learning_rate': 0.05, 'colsample_bytree': 0.25}
```

```
if flag_smote:
    # use best estimator selected and optimize model
    lgbm_best =
LGBMClassifier(**lgbm_clf.best_estimator_.get_params())

    lgbm_best.fit(X_smote, y_smote)
    lgbm_pred = lgbm_best.predict(X_test)
    accuracy = lgbm_best.score(X_test, y_test)
    print('Accuracy of LGBM: {}'.format(accuracy))
    lgbm_log_loss = balanced_log_loss(y_test, lgbm_pred)
    print('Balanced log loss of LGBM: ', lgbm_log_loss)
```

```
Accuracy of LGBM: 0.9435483870967742
bll y_true:  49     1
```

```
581    0
82     0
304    1
109    1
      ..
6      0
104    0
114    0
158    0
181    1
Name: Class, Length: 124, dtype: int64
bll y_pred:  [1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0
 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
0 1
 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0
0 0
 0 0 0 1 0 0 0 0 0 0 0 1 1]
Balanced log loss of LGBM:  3.516335860313267
```

LGBM with TabPFN

```python
class Ensemble():
    def __init__(self):
        self.imputer = KNNImputer(n_neighbors = 10) # fill
missing/null value with K-Nearest Neighbor
        # create model
        self.classifiers =
[LGBMClassifier(n_estimators=100,max_depth=3,learning_rate=0.001,subsa
mple=0.9,colsample_bytree=0.85),
                            LGBMClassifier(),

TabPFNClassifier(N_ensemble_configurations=24),

TabPFNClassifier(N_ensemble_configurations=64)]

    def fit(self, X, y):
        X = self.imputer.fit_transform(X) # fill missing/null value
with K-Nearest Neighbor

        for classifier in self.classifiers:
            # for tabpfn
            if classifier == self.classifiers[2] or classifier ==
self.classifiers[3]:
                classifier.fit(X, y, overwrite_warning = True)
            else:
                # for other ML models other than TabPFN
                classifier.fit(X, y)
```

```
    def predict_proba(self, x):
        x = self.imputer.transform(x)

        # put all probabilities from all classifiers into an array
        probabilities = np.stack([classifier.predict_proba(x) for
classifier in self.classifiers])
        # take average of probabilites
        averaged_probabilities = np.mean(probabilities, axis = 0)
        return averaged_probabilities

def training(model, x, y, y_meta):
    if flag_debug:
        print('x: ', x.shape)
        print('y: ', y.shape)
        print('y_meta: ', y_meta.shape)
    results = list()
    best_loss = np.inf

    split = 0
    splits = 5

    # K-Fold cross-validation is used instead of train-test-split
    cv = KFold(n_splits = splits, shuffle = True, random_state = 42)

    for train_idx, val_idx in tqdm(cv.split(x), total = splits):
        split += 1

        # use greeks for training and training data for validation
        # greeks has more info. detail on class 0 and 1
        x_train, x_val = x.iloc[train_idx], x.iloc[val_idx]
        y_train, y_val = y_meta[train_idx], y.iloc[val_idx]

        model.fit(x_train, y_train)

        y_pred = model.predict_proba(x_val)

        p0 = y_pred[:,0] # obtain probability of class 0
        p0 = np.where(p0 >= 0.5, 0, 1) # if p0 is great equal than
0.5, then class 0; otherwise, class 1
        p0 = p0.reshape(len(p0))

        loss = balanced_log_loss(y_val, p0)

        if flag_debug:
            print('split: ', split)
            print('train_idx: ', train_idx, 'val_idx: ', val_idx)
            print('x_train: ', x_train, 'y_train: ', y_train)
            print('x_val: ', x_val, 'y_val: ', y_val)
            print('y_pred: ', y_pred)
```

```python
        if loss < best_loss:
            best_model = model
            best_loss = loss
            print('best_model_saved')
        results.append(loss)
        print('> val_loss = %.5f, split = %.1f' %(loss, split))
    print('LOSS: %.5f' % (np.mean(results)))
    return best_model

ros = RandomOverSampler(random_state = 42) # random oversampling
train_ros, y_ros = ros.fit_resample(train, greeks['Alpha'])
_, y_ros = np.unique(y_ros, return_inverse = True)

if flag_debug:
    print('train: ', train.shape)
    print('train: ', train)
    print('greeks: ', greeks.shape)
    print('greeks: ', greeks)
    print('train_ros: ', train_ros.shape, 'y_ros: ', y_ros.shape)
    print('train_ros: ', train_ros, 'y_ros: ', y_ros)
    print('y_ros: ', y_ros.shape)
    print('y_ros: ', y_ros)
```

```
train:  (617, 58)
train:                  Id  EJ  Class        AB        AF         AH
AM  \
0    000ff2bfdfe9   1       1 -0.472222 -0.005214   0.000000  0.069272

1    007255e47698   0       0 -0.680556 -0.989494   0.000000  0.611687

2    013f2bd269f5   1       0  0.375000 -0.224190   0.000000  0.440180

3    043ac50845d5   1       0 -0.333333  0.323123   1.226427  2.105694

4    044fb8a146ec   1       1  0.083333  0.283109   0.000000 -0.239281

..            ...  ..     ...       ...       ...        ...       ...

612  fd3dafe738fd   0       0 -0.666667  0.004501   1.351236 -0.410097

613  fd895603f071   1       0  0.263889  1.081978   0.000000  0.968303

614  fd8ef6377f76   0       0  0.236111 -0.305510   1.574611  1.295989

615  fe1942975e40   1       0  0.027778 -0.857917   0.000000  0.117335

616  ffcca4ded3bb   0       0  0.416667 -0.206897  16.169362  3.404334


           AR        AY        AY  ...        FI        FL         FR
```

```
\
0    0.000000 -1.880769  0.000000  ... -2.125230  0.702705   0.598571

1    0.000000 -0.607692  0.000000  ...  0.138122 -0.472232  -0.624571

2    0.000000  0.738462  0.000000  ...  0.561694  0.770546  -0.153143

3    0.000000 -0.584615  0.000000  ...  1.639042  0.508776  -0.624571

4    0.000000 -0.473077  2.594595  ...  1.243094  0.843681  46.670286

..        ...       ...       ...  ...       ...       ...        ...

612  4.882164 -0.665385  4.594595  ... -0.022099 -0.472232   0.128000

613  7.834536  0.411538  0.027027  ...  0.322284  1.185050   0.109714

614  1.866864  1.319231  0.000000  ...  0.696133 -0.472232  -0.624571

615  0.000000  1.280769  0.000000  ... -0.640884  1.025726  -0.338286

616  0.000000 -0.796154  8.108108  ... -0.734807 -0.472232   0.013714


           FS        GB        GE        GF        GH        GI
GL
0   -0.347826 -0.642283  0.000000 -0.359338 -0.716263  0.641741 -
0.010025
1    0.666667 -0.819132  0.000000  1.240601 -0.124567 -0.197595
0.990161
2    2.014493  1.581994  0.290982  0.359598 -0.218622 -0.129459 -
0.006520
3    0.057971 -0.020900  0.178349 -0.353767  0.789556  1.101632 -
0.008401
4   -0.289855 -0.204180  1.336815  0.042256  1.248820 -0.105640 -
0.011111
..        ...       ...       ...       ...       ...       ...
...
612 -0.405797 -0.847267  2.628881  0.015869 -0.504561  0.627426
0.990161
613  0.362319  1.479904  7.718779 -0.292729 -0.081158  1.865560 -
0.008881
614 -0.405797  0.102894  1.023740 -0.083984 -0.375590  1.748697
0.990161
615  0.884058  0.503215  0.000000 -0.361707 -0.464297 -0.085772 -
0.007084
616 -0.231884 -0.440514  0.000000 -0.060837  1.279648  1.643687
0.990161

[617 rows x 58 columns]
```

```
greeks:  (617, 6)
greeks:                  Id Alpha Beta Gamma Delta    Epsilon
0    000ff2bfdfe9     B    C     G     D  3/19/2019
1    007255e47698     A    C     M     B    Unknown
2    013f2bd269f5     A    C     M     B    Unknown
3    043ac50845d5     A    C     M     B    Unknown
4    044fb8a146ec     D    B     F     B  3/25/2020
..            ...   ...  ...   ...   ...        ...
612  fd3dafe738fd     A    B     M     B  9/13/2020
613  fd895603f071     A    B     M     B   9/8/2020
614  fd8ef6377f76     A    C     M     B  7/24/2019
615  fe1942975e40     A    C     M     B  1/31/2019
616  ffcca4ded3bb     A    C     M     B    Unknown

[617 rows x 6 columns]
train_ros:  (2036, 58) y_ros:  (2036,)
train_ros:                  Id  EJ  Class        AB        AF
AH        AM  \
0     000ff2bfdfe9   1      1 -0.472222 -0.005214   0.000000  0.069272

1     007255e47698   0      0 -0.680556 -0.989494   0.000000  0.611687

2     013f2bd269f5   1      0  0.375000 -0.224190   0.000000  0.440180

3     043ac50845d5   1      0 -0.333333  0.323123   1.226427  2.105694

4     044fb8a146ec   1      1  0.083333  0.283109   0.000000 -0.239281

...            ...  ..    ...       ...       ...        ...       ...

2031  f16b095a433f   0      1  6.555556  1.811240  26.662191  0.084140

2032  a92580fda1c3   0      1  0.763889  0.283647   0.000000  1.522245

2033  679465bb38ba   1      1  1.347222  1.698131   0.000000  0.954818

2034  582ef2696f72   1      1  1.263889  1.816902   0.000000 -0.001037

2035  679465bb38ba   1      1  1.347222  1.698131   0.000000  0.954818


            AR        AX        AY  ...        FI        FL        FR
\
0     0.000000 -1.880769  0.000000  ... -2.125230  0.702705  0.598571

1     0.000000 -0.607692  0.000000  ...  0.138122 -0.472232 -0.624571

2     0.000000  0.738462  0.000000  ...  0.561694  0.770546 -0.153143

3     0.000000 -0.584615  0.000000  ...  1.639042  0.508776 -0.624571
```

```
4     0.000000 -0.473077  2.594595  ...  1.243094  0.843681  46.670286

...        ...       ...       ...  ...       ...       ...        ...

2031  0.000000  0.253846  0.000000  ... -0.854512 -0.472232   1.924571

2032  0.000000  0.094231  0.000000  ... -1.055249 -0.472232  -0.624571

2033  9.350052  0.280769  0.000000  ... -1.543278 -0.018171  -0.231429

2034  0.000000 -0.323077  1.567568  ... -0.895028  1.047681  -0.624571

2035  9.350052  0.280769  0.000000  ... -1.543278 -0.018171  -0.231429


            FS        GB        GE        GF        GH        GI
GL
0    -0.347826 -0.642283  0.000000 -0.359338 -0.716263  0.641741 -
0.010025
1     0.666667 -0.819132  0.000000  1.240601 -0.124567 -0.197595
0.990161
2     2.014493  1.581994  0.290982  0.359598 -0.218622 -0.129459 -
0.006520
3     0.057971 -0.020900  0.178349 -0.353767  0.789556  1.101632 -
0.008401
4    -0.289855 -0.204180  1.336815  0.042256  1.248820 -0.105640 -
0.011111
...        ...       ...       ...       ...       ...       ...
...
2031 -0.405797  0.643891  0.000000  1.328281  1.296005 -0.303750
0.990161
2032 -0.333333 -0.111736  0.000000  0.070104  0.267694 -0.329402
0.990161
2033  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011
2034  0.985507 -0.854502  0.000000  0.192884  1.035546  0.062439 -
0.011296
2035  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011

[2036 rows x 58 columns] y_ros:  [1 0 0 ... 3 3 3]
y_ros:  (2036,)
y_ros:  [1 0 0 ... 3 3 3]
```

```
yt = Ensemble()
```

```
Loading model that can be used for inference only
Using a Transformer with 25.82 M parameters
Loading model that can be used for inference only
Using a Transformer with 25.82 M parameters
```

```
x_ros = train_ros.drop(['Class', 'Id'], axis = 1)
print('x_ros: ', x_ros)
y_ = train_ros.Class
print('y_: ', y_)
```

```
x_ros:          EJ        AB         AF          AH         AM         AR
AX  \
0       1 -0.472222 -0.005214   0.000000  0.069272  0.000000 -1.880769

1       0 -0.680556 -0.989494   0.000000  0.611687  0.000000 -0.607692

2       1  0.375000 -0.224190   0.000000  0.440180  0.000000  0.738462

3       1 -0.333333  0.323123   1.226427  2.105694  0.000000 -0.584615

4       1  0.083333  0.283109   0.000000 -0.239281  0.000000 -0.473077

...    ..       ...       ...        ...       ...       ...       ...

2031    0  6.555556  1.811240  26.662191  0.084140  0.000000  0.253846

2032    0  0.763889  0.283647   0.000000  1.522245  0.000000  0.094231

2033    1  1.347222  1.698131   0.000000  0.954818  9.350052  0.280769

2034    1  1.263889  1.816902   0.000000 -0.001037  0.000000 -0.323077

2035    1  1.347222  1.698131   0.000000  0.954818  9.350052  0.280769


            AY        AZ         BC  ...        FI        FL
FR  \
0     0.000000 -0.134115   1.123175  ... -2.125230  0.702705
0.598571
1     0.000000  0.631510   0.000000  ...  0.138122 -0.472232  -
0.624571
2     0.000000  0.488281   0.000000  ...  0.561694  0.770546  -
0.153143
3     0.000000  0.122396   0.000000  ...  1.639042  0.508776  -
0.624571
4     2.594595 -1.459635  26.204380  ...  1.243094  0.843681
46.670286
...        ...       ...        ...  ...       ...       ...        ..
.
2031  0.000000  1.420573   0.779197  ... -0.854512 -0.472232
1.924571
2032  0.000000  0.433594   0.000000  ... -1.055249 -0.472232  -
0.624571
2033  0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429
2034  1.567568 -0.386719   1.597628  ... -0.895028  1.047681  -
```

```
0.624571
2035  0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429

            FS        GB        GE        GF        GH        GI
GL
0    -0.347826 -0.642283  0.000000 -0.359338 -0.716263  0.641741 -
0.010025
1     0.666667 -0.819132  0.000000  1.240601 -0.124567 -0.197595
0.990161
2     2.014493  1.581994  0.290982  0.359598 -0.218622 -0.129459 -
0.006520
3     0.057971 -0.020900  0.178349 -0.353767  0.789556  1.101632 -
0.008401
4    -0.289855 -0.204180  1.336815  0.042256  1.248820 -0.105640 -
0.011111
...        ...       ...       ...       ...       ...       ...
...
2031 -0.405797  0.643891  0.000000  1.328281  1.296005 -0.303750
0.990161
2032 -0.333333 -0.111736  0.000000  0.070104  0.267694 -0.329402
0.990161
2033  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011
2034  0.985507 -0.854502  0.000000  0.192884  1.035546  0.062439 -
0.011296
2035  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011

[2036 rows x 56 columns]
y_:  0       1
1       0
2       0
3       0
4       1
       ..
2031    1
2032    1
2033    1
2034    1
2035    1
Name: Class, Length: 2036, dtype: int64
```

```
print('x_ros', x_ros)
print(x_ros['EJ'])
print('y_', y_)
print('y_ros', y_ros)
```

```
x_ros        EJ        AB        AF         AH        AM        AR
AX  \
```

```
0       1 -0.472222 -0.005214   0.000000  0.069272  0.000000 -1.880769

1       0 -0.680556 -0.989494   0.000000  0.611687  0.000000 -0.607692

2       1  0.375000 -0.224190   0.000000  0.440180  0.000000  0.738462

3       1 -0.333333  0.323123   1.226427  2.105694  0.000000 -0.584615

4       1  0.083333  0.283109   0.000000 -0.239281  0.000000 -0.473077

...    ..       ...       ...        ...       ...       ...       ...

2031    0  6.555556  1.811240  26.662191  0.084140  0.000000  0.253846

2032    0  0.763889  0.283647   0.000000  1.522245  0.000000  0.094231

2033    1  1.347222  1.698131   0.000000  0.954818  9.350052  0.280769

2034    1  1.263889  1.816902   0.000000 -0.001037  0.000000 -0.323077

2035    1  1.347222  1.698131   0.000000  0.954818  9.350052  0.280769


            AY        AZ         BC  ...        FI        FL
FR  \
0     0.000000 -0.134115   1.123175  ... -2.125230  0.702705
0.598571
1     0.000000  0.631510   0.000000  ...  0.138122 -0.472232  -
0.624571
2     0.000000  0.488281   0.000000  ...  0.561694  0.770546  -
0.153143
3     0.000000  0.122396   0.000000  ...  1.639042  0.508776  -
0.624571
4     2.594595 -1.459635  26.204380  ...  1.243094  0.843681
46.670286
...        ...       ...        ...  ...       ...       ...         ..
.
2031  0.000000  1.420573   0.779197  ... -0.854512 -0.472232
1.924571
2032  0.000000  0.433594   0.000000  ... -1.055249 -0.472232  -
0.624571
2033  0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429
2034  1.567568 -0.386719   1.597628  ... -0.895028  1.047681  -
0.624571
2035  0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429

            FS        GB        GE        GF        GH        GI
GL
```

```
0    -0.347826 -0.642283  0.000000 -0.359338 -0.716263  0.641741 -
0.010025
1     0.666667 -0.819132  0.000000  1.240601 -0.124567 -0.197595
0.990161
2     2.014493  1.581994  0.290982  0.359598 -0.218622 -0.129459 -
0.006520
3     0.057971 -0.020900  0.178349 -0.353767  0.789556  1.101632 -
0.008401
4    -0.289855 -0.204180  1.336815  0.042256  1.248820 -0.105640 -
0.011111
...        ...       ...       ...       ...       ...       ...
...
2031 -0.405797  0.643891  0.000000  1.328281  1.296005 -0.303750
0.990161
2032 -0.333333 -0.111736  0.000000  0.070104  0.267694 -0.329402
0.990161
2033  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011
2034  0.985507 -0.854502  0.000000  0.192884  1.035546  0.062439 -
0.011296
2035  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011

[2036 rows x 56 columns]
0       1
1       0
2       1
3       1
4       1
       ..
2031    0
2032    0
2033    1
2034    1
2035    1
Name: EJ, Length: 2036, dtype: int64
y_ 0       1
1       0
2       0
3       0
4       1
       ..
2031    1
2032    1
2033    1
2034    1
2035    1
Name: Class, Length: 2036, dtype: int64
y_ros [1 0 0 ... 3 3 3]
```

```
model = training(yt, x_ros, y_, y_ros)
```

```
x:  (2036, 56)
y:  (2036,)
y_meta:  (2036,)
```

{"model_id":"4f4f6448e3ab479799eb608e57eebf47","version_major":2,"version_minor":0}

```
bll y_true:  23      0
29      0
30      0
32      1
39      0
       ..
2002    1
2023    1
2026    1
2028    1
2031    1
Name: Class, Length: 408, dtype: int64
bll y_pred:  [0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 1 0
 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0
0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 1
0 0
 1 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1]
split:  1
train_idx:  [   0    1    2 ... 2033 2034 2035] val_idx:  [  23   29
30   32   39   44   45   49   56   59   63   65   67   69
   70   73   76   78   99  100  109  111  115  120  123  124  128  135
  162  163  168  173  175  185  188  194  196  203  210  211  212  218
  220  231  233  237  239  247  251  254  256  266  270  275  281  289
```

```
  297  298  300  303  305  306  307  316  321  322  324  331  342  344
  350  351  352  353  354  361  366  367  368  374  382  383  393  394
  411  414  416  420  422  427  429  432  433  450  453  462  471  478
  479  480  482  483  485  486  494  495  514  519  526  527  529  530
  532  534  535  538  543  544  548  552  554  555  561  567  570  582
  584  590  591  593  596  599  611  613  614  617  619  628  637  650
  654  670  674  678  679  680  692  694  701  706  712  715  727  733
  736  741  759  771  772  774  780  782  806  807  809  812  818  845
  849  855  859  862  869  873  881  886  905  907  909  916  923  930
  931  932  938  939  940  942  944  949  966  973  974  982  984  985
  987  990  997  998  999 1004 1006 1018 1022 1029 1034 1043 1054 1063
 1068 1073 1084 1102 1103 1106 1110 1112 1113 1121 1124 1128 1137 1160
 1185 1189 1202 1206 1216 1220 1233 1239 1240 1242 1244 1245 1255 1258
 1268 1273 1278 1280 1281 1287 1293 1304 1313 1315 1317 1318 1334 1336
 1343 1345 1355 1356 1358 1360 1368 1370 1376 1380 1383 1385 1387 1393
 1402 1405 1412 1419 1421 1423 1431 1435 1440 1441 1447 1449 1450 1454
 1456 1457 1464 1467 1476 1486 1490 1491 1510 1511 1514 1524 1526 1531
 1538 1540 1541 1549 1554 1557 1559 1560 1563 1564 1571 1575 1578 1580
 1581 1582 1583 1592 1593 1594 1599 1600 1608 1619 1626 1627 1635 1637
 1639 1642 1644 1649 1655 1660 1662 1667 1673 1676 1680 1684 1692 1696
 1699 1703 1706 1708 1717 1721 1727 1729 1740 1743 1746 1749 1752 1755
 1756 1760 1761 1763 1769 1771 1788 1793 1794 1802 1805 1808 1810 1817
 1827 1831 1832 1833 1843 1844 1850 1852 1853 1867 1879 1882 1885 1892
 1901 1909 1919 1920 1939 1945 1950 1952 1955 1957 1959 1960 1963 1966
 1967 1968 1970 1971 1973 1974 1986 1990 1992 1998 2001 2002 2023 2026
 2028 2031]
x_train:         EJ        AB        AF        AH        AM        AR
AX  \
0      1 -0.472222 -0.005214  0.000000  0.069272  0.000000 -1.880769

1      0 -0.680556 -0.989494  0.000000  0.611687  0.000000 -0.607692

2      1  0.375000 -0.224190  0.000000  0.440180  0.000000  0.738462

3      1 -0.333333  0.323123  1.226427  2.105694  0.000000 -0.584615

4      1  0.083333  0.283109  0.000000 -0.239281  0.000000 -0.473077

...   ..       ...       ...       ...       ...       ...       ...

2030   0  1.986111  7.208136  0.000000  0.028239  1.489296  0.500000

2032   0  0.763889  0.283647  0.000000  1.522245  0.000000  0.094231

2033   1  1.347222  1.698131  0.000000  0.954818  9.350052  0.280769

2034   1  1.263889  1.816902  0.000000 -0.001037  0.000000 -0.323077

2035   1  1.347222  1.698131  0.000000  0.954818  9.350052  0.280769
```

```
            AY        AZ         BC  ...        FI        FL  
FR  \
0     0.000000 -0.134115   1.123175  ... -2.125230  0.702705   
0.598571
1     0.000000  0.631510   0.000000  ...  0.138122 -0.472232  -
0.624571
2     0.000000  0.488281   0.000000  ...  0.561694  0.770546  -
0.153143
3     0.000000  0.122396   0.000000  ...  1.639042  0.508776  -
0.624571
4     2.594595 -1.459635  26.204380  ...  1.243094  0.843681  
46.670286
...        ...       ...        ...  ...       ...       ...        ..
.
2030  0.000000  0.345052  10.346715  ... -0.978821 -0.472232  -
0.624571
2032  0.000000  0.433594   0.000000  ... -1.055249 -0.472232  -
0.624571
2033  0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429
2034  1.567568 -0.386719   1.597628  ... -0.895028  1.047681  -
0.624571
2035  0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429

            FS        GB        GE        GF        GH        GI  
GL
0    -0.347826 -0.642283  0.000000 -0.359338 -0.716263  0.641741 -
0.010025
1     0.666667 -0.819132  0.000000  1.240601 -0.124567 -0.197595  
0.990161
2     2.014493  1.581994  0.290982  0.359598 -0.218622 -0.129459 -
0.006520
3     0.057971 -0.020900  0.178349 -0.353767  0.789556  1.101632 -
0.008401
4    -0.289855 -0.204180  1.336815  0.042256  1.248820 -0.105640 -
0.011111
...        ...       ...       ...       ...       ...       ...  
...
2030 -0.405797 -0.728296  0.000000 -0.469769  2.633533 -0.105268  
0.990161
2032 -0.333333 -0.111736  0.000000  0.070104  0.267694 -0.329402  
0.990161
2033  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279  
0.066011
2034  0.985507 -0.854502  0.000000  0.192884  1.035546  0.062439 -
0.011296
2035  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
```

```
0.066011

[1628 rows x 56 columns] y_train:  [1 0 0 ... 3 3 3]
x_val:        EJ         AB        AF         AH         AM         AR
AX   AY  \
23     1   0.055556 -0.353824   0.694843  -0.572499   0.000000
2.334615  0.0
29     1   3.152778 -0.758551   0.000000   1.521208   9.565056
0.603846  0.0
30     0   0.458333 -0.330432   0.000000   1.475911   0.000000
0.384615  0.0
32     1   0.347222  0.523806   0.000000   1.864338   0.000000
0.115385  0.0
39     1  -0.763889  0.120439   0.000000  -0.605233   0.000000 -
0.921154  0.0
...   ..        ...       ...        ...        ...        ...        .
..  ...
2002   0   0.750000  1.124108   0.000000   0.082065   0.000000
0.873077  0.0
2023   0   0.861111  0.478620   0.301800   0.362955   8.327472
1.296154  0.0
2026   0  13.263889  0.796241  40.391211  22.701706  11.736072
2.700000  0.0
2028   0   1.138889  0.488936   0.000000   0.275588   0.000000 -
0.053846  0.0
2031   0   6.555556  1.811240  26.662191   0.084140   0.000000
0.253846  0.0

            AZ          BC  ...        FI         FL        FR
FS  \
23    0.220052    0.395985  ... -0.488950   0.265698 -0.624571
1.304348
29   -0.286458    2.802920  ...  0.902394   0.496557  0.384000
0.869565
30    0.194010    0.000000  ... -0.666667  -0.472232 -0.624571 -
0.260870
32   -1.459635    0.961679  ... -0.443831  12.064035  1.739143
1.463768
39   -0.587240    0.000000  ... -0.887661   1.511568  0.065143 -
0.405797
...        ...         ...  ...       ...        ...       ...        .
..
2002 -0.200521    0.923358  ... -0.233886  -0.472232 -0.624571
0.115942
2023 -1.180990    2.583942  ... -0.114180  -0.472232  0.289143 -
0.405797
2026  0.514323  379.728102  ... -1.072744  -0.472232 -0.624571
2.260870
2028  0.326823    0.000000  ...  0.287293  -0.472232  0.430857 -
```

```
0.057971
2031  1.420573    0.779197  ... -0.854512  -0.472232  1.924571 -
0.405797

             GB        GE        GF        GH        GI        GL
23     0.250000  2.289134 -0.113974  0.169236 -0.578557 -0.009283
29    -0.809486  0.000000 -0.099416 -1.540107  0.681534 -0.007501
30    -0.243569  0.000000  1.017274  0.025165 -0.315087  0.990161
32     0.409164  0.000000  0.140967  0.240642 -0.556313 -0.013903
39    -1.037781  5.714641 -0.327545  0.293174  0.143201 -0.007730
...         ...       ...       ...       ...       ...       ...
2002  -0.817524  0.000000  0.404923 -0.448883  0.955626  0.990161
2023  -0.841640  0.000000 -0.045577  0.933627  0.173547  0.990161
2026  10.111736  5.284113 -0.447118  1.069204 -0.059347  0.990161
2028   1.219453  0.032215  0.956860 -0.474992  0.444833  0.990161
2031   0.643891  0.000000  1.328281  1.296005 -0.303750  0.990161

[408 rows x 56 columns] y_val:  23      0
29      0
30      0
32      1
39      0
       ..
2002    1
2023    1
2026    1
2028    1
2031    1
Name: Class, Length: 408, dtype: int64
y_pred:  [[0.82691516 0.05797252 0.06117717 0.05393512]
 [0.75730318 0.09112631 0.06172227 0.08984826]
 [0.82163729 0.0606567  0.06127315 0.05643286]
 ...
 [0.05678569 0.05898774 0.05997464 0.82425194]
 [0.0578373  0.05939324 0.05994175 0.82282771]
 [0.06052839 0.06296229 0.06367355 0.81283577]]
best_model_saved
> val_loss = 0.61678, split = 1.0
bll y_true:  2       0
6       0
15      0
18      0
31      1
       ..
2016    1
2017    1
2021    1
2025    1
2034    1
```

```
Name: Class, Length: 407, dtype: int64
bll y_pred:  [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 1 0 1 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0
0 1
 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1]
split:  2
train_idx:  [   0    1    3 ... 2032 2033 2035] val_idx:  [   2    6
15   18   31   43   48   51   54   58   71   72   81   83
   84   86  101  107  113  118  141  147  148  155  170  174  177  179
  181  182  184  192  198  199  208  214  221  222  226  236  240  243
  244  250  259  261  265  271  272  273  274  277  285  286  287  292
  296  308  309  310  311  312  326  327  329  332  334  339  341  346
  358  360  363  365  370  371  376  380  381  398  405  408  413  415
  423  425  426  428  430  435  436  438  439  442  445  451  457  461
  464  465  468  481  490  493  497  500  505  506  507  513  518  522
  528  551  557  560  566  572  575  576  579  581  583  585  588  589
  598  601  602  607  609  610  618  620  621  629  630  631  643  651
  660  669  677  693  700  704  705  707  710  720  721  722  730  743
  744  745  746  752  755  756  757  764  765  767  777  781  785  787
  788  792  802  808  813  817  819  824  829  834  836  838  857  861
  874  879  889  900  903  904  922  925  926  936  937  943  948  950
  962  963  964  965  976  978  979  986  988  993  994 1005 1007 1009
 1013 1023 1033 1037 1040 1041 1047 1050 1052 1055 1058 1067 1087 1090
 1094 1100 1107 1114 1117 1120 1125 1131 1132 1133 1164 1176 1177 1179
 1182 1187 1192 1210 1211 1221 1223 1225 1229 1235 1237 1249 1251 1259
 1261 1262 1265 1270 1271 1272 1290 1298 1301 1305 1316 1319 1322 1323
 1329 1333 1335 1338 1340 1341 1342 1350 1351 1352 1357 1359 1378 1379
 1389 1391 1392 1401 1406 1407 1414 1417 1418 1424 1432 1446 1463 1465
 1466 1469 1473 1481 1487 1501 1502 1503 1507 1512 1519 1536 1545 1555
 1561 1565 1566 1568 1573 1576 1586 1587 1591 1604 1606 1610 1611 1613
 1614 1616 1620 1621 1624 1625 1630 1632 1641 1645 1646 1650 1651 1657
```

```
 1664 1672 1674 1679 1689 1704 1709 1710 1713 1716 1718 1726 1728 1731
 1735 1744 1753 1773 1777 1779 1781 1784 1787 1797 1798 1799 1800 1814
 1818 1821 1822 1834 1839 1848 1849 1859 1874 1876 1881 1888 1894 1896
 1897 1900 1902 1903 1905 1910 1912 1929 1933 1934 1935 1942 1947 1948
 1949 1975 1977 1979 1981 1996 2008 2010 2012 2013 2016 2017 2021 2025
 2034]
x_train:         EJ        AB        AF         AH        AM        AR
AX  \
0      1 -0.472222 -0.005214   0.000000  0.069272  0.000000 -1.880769

1      0 -0.680556 -0.989494   0.000000  0.611687  0.000000 -0.607692

3      1 -0.333333  0.323123   1.226427  2.105694  0.000000 -0.584615

4      1  0.083333  0.283109   0.000000 -0.239281  0.000000 -0.473077

5      0 -0.472222 -0.233104   0.000000 -0.446289  0.000000 -0.442308

...   ..       ...       ...        ...       ...       ...       ...

2030   0  1.986111  7.208136   0.000000  0.028239  1.489296  0.500000

2031   0  6.555556  1.811240  26.662191  0.084140  0.000000  0.253846

2032   0  0.763889  0.283647   0.000000  1.522245  0.000000  0.094231

2033   1  1.347222  1.698131   0.000000  0.954818  9.350052  0.280769

2035   1  1.347222  1.698131   0.000000  0.954818  9.350052  0.280769


            AY        AZ         BC  ...        FI        FL
FR  \
0     0.000000 -0.134115   1.123175  ... -2.125230  0.702705
0.598571
1     0.000000  0.631510   0.000000  ...  0.138122 -0.472232  -
0.624571
3     0.000000  0.122396   0.000000  ...  1.639042  0.508776  -
0.624571
4     2.594595 -1.459635  26.204380  ...  1.243094  0.843681
46.670286
5     0.000000  0.430990   0.000000  ...  0.346225 -0.472232  -
0.624571
...        ...       ...        ...  ...       ...       ...        ..
.
2030  0.000000  0.345052  10.346715  ... -0.978821 -0.472232  -
0.624571
2031  0.000000  1.420573   0.779197  ... -0.854512 -0.472232
1.924571
2032  0.000000  0.433594   0.000000  ... -1.055249 -0.472232  -
```

```
0.624571
2033  0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429
2035  0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429

            FS        GB        GE        GF        GH        GI
GL
0    -0.347826 -0.642283  0.000000 -0.359338 -0.716263  0.641741 -
0.010025
1     0.666667 -0.819132  0.000000  1.240601 -0.124567 -0.197595
0.990161
3     0.057971 -0.020900  0.178349 -0.353767  0.789556  1.101632 -
0.008401
4    -0.289855 -0.204180  1.336815  0.042256  1.248820 -0.105640 -
0.011111
5     1.942029  0.271704  0.000000  1.006319 -0.176156  0.924306
0.990161
...        ...       ...       ...       ...       ...       ...
...
2030 -0.405797 -0.728296  0.000000 -0.469769  2.633533 -0.105268
0.990161
2031 -0.405797  0.643891  0.000000  1.328281  1.296005 -0.303750
0.990161
2032 -0.333333 -0.111736  0.000000  0.070104  0.267694 -0.329402
0.990161
2033  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011
2035  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011

[1629 rows x 56 columns] y_train:  [1 0 0 ... 3 3 3]
x_val:        EJ        AB        AF        AH        AM        AR
AX  \
2      1  0.375000 -0.224190  0.000000  0.440180  0.000000  0.738462

6      1 -0.020833 -0.640701  0.000000 -0.452397  7.173792 -1.353846

15     1  1.013889  0.248332  7.676533  2.042646  0.000000  3.380769

18     1 -0.277778 -0.708611  0.000000 -0.226026  0.000000 -0.053846

31     1  1.263889  0.151361  0.000000  2.192024  0.000000  0.100000

...   ..       ...       ...       ...       ...       ...       ...

2016   0  1.986111  7.208136  0.000000  0.028239  1.489296  0.500000

2017   0  0.763889  0.283647  0.000000  1.522245  0.000000  0.094231
```

```
2021   1  0.402778  0.957328  3.832469 -0.271784  0.000000  0.980769

2025   1  1.208333  2.125606  0.000000  0.448940  0.000000  0.873077

2034   1  1.263889  1.816902  0.000000 -0.001037  0.000000 -0.323077


            AY        AZ         BC  ...        FI        FL
FR  \
2     0.000000  0.488281   0.000000  ...  0.561694   0.770546 -
0.153143
6     0.000000 -0.808594   0.000000  ...  0.278085   0.226183 -
0.265714
15    0.000000 -0.473958   0.000000  ... -0.742173  10.691755 -
0.348571
18    0.648649 -0.713542   0.594891  ...  1.263352  15.545042
1.582286
31    0.000000  0.393229   2.543796  ...  1.388582   1.040633
0.576000
...        ...       ...        ...  ...       ...        ...        ..
.
2016  0.000000  0.345052  10.346715  ... -0.978821  -0.472232 -
0.624571
2017  0.000000  0.433594   0.000000  ... -1.055249  -0.472232 -
0.624571
2021  0.000000 -0.156250   1.910584  ... -1.530387   1.262091
0.716857
2025  0.000000  0.174479   0.676095  ... -1.171271   0.375757 -
0.624571
2034  1.567568 -0.386719   1.597628  ... -0.895028   1.047681 -
0.624571

            FS        GB        GE        GF        GH        GI
GL
2     2.014493  1.581994  0.290982  0.359598 -0.218622 -0.129459 -
0.006520
6     0.449275 -0.077170  2.179715 -0.277507  0.300723  1.315202 -
0.012553
15    0.695652 -1.267685  0.000000 -0.346118 -0.575024 -0.114973 -
0.014835
18    0.971014 -0.208199  0.000000 -0.261161  0.061025 -0.346464 -
0.015029
31    1.492754  0.515273  0.000000 -0.347892  0.587292  0.303808 -
0.011722
...        ...       ...       ...       ...       ...       ...
...
2016 -0.405797 -0.728296  0.000000 -0.469769  2.633533 -0.105268
0.990161
2017 -0.333333 -0.111736  0.000000  0.070104  0.267694 -0.329402
0.990161
```

```
2021 -0.405797 -0.549035  0.000000 -0.455998  0.321485 -0.094360 -
0.013242
2025  5.043478 -0.725080  0.000000 -0.285817 -0.060082 -0.686058
0.011649
2034  0.985507 -0.854502  0.000000  0.192884  1.035546  0.062439 -
0.011296

[407 rows x 56 columns] y_val:  2       0
6       0
15      0
18      0
31      1
       ..
2016    1
2017    1
2021    1
2025    1
2034    1
Name: Class, Length: 407, dtype: int64
y_pred:  [[0.76272425 0.12175662 0.05876834 0.05675076]
 [0.82582411 0.05825581 0.05818326 0.05773679]
 [0.62565357 0.26195964 0.05557778 0.05680901]
 ...
 [0.05893005 0.05928525 0.06291899 0.81886574]
 [0.05556568 0.05614654 0.06051495 0.82777283]
 [0.05905958 0.05965726 0.0626911  0.81859209]]
best_model_saved
> val_loss = 0.31400, split = 2.0
bll y_true:  0       1
3       0
5       0
7       0
10      1
       ..
2011    1
2022    1
2027    1
2029    1
2030    1
Name: Class, Length: 407, dtype: int64
bll y_pred:  [1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
0 1 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0
1 0
 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
0 1
 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1]
split:  3
train_idx:  [   1    2    4 ... 2033 2034 2035] val_idx:  [   0    3
5    7   10   12   24   25   27   33   41   42   47   52
   55   60   62   66   68   74   77   80   82   85   88   92   94   96
   97  102  104  105  106  110  117  125  126  129  131  132  136  137
  138  139  140  142  144  156  158  164  165  167  171  178  183  193
  195  204  209  213  215  224  227  228  232  235  238  242  248  249
  258  260  267  280  282  290  291  294  299  302  314  318  319  323
  325  328  333  336  348  349  355  359  362  364  375  377  388  390
  404  409  410  419  421  424  434  440  446  447  448  458  467  477
  503  516  523  525  531  533  536  541  542  545  547  549  553  558
  571  573  578  594  597  603  605  615  622  624  626  634  636  638
  644  649  661  664  665  666  668  672  673  676  682  688  691  695
  711  714  716  718  723  724  737  754  762  770  778  786  793  796
  798  799  803  810  811  816  820  826  832  839  841  842  843  844
  846  847  858  864  865  867  887  888  892  893  898  899  901  906
  910  914  915  917  918  921  941  946  952  968  983  989 1000 1001
 1010 1026 1027 1030 1036 1046 1049 1053 1057 1061 1074 1075 1078 1079
 1080 1083 1085 1089 1091 1096 1101 1105 1108 1111 1116 1118 1134 1138
 1144 1146 1151 1157 1159 1161 1163 1165 1169 1170 1173 1175 1178 1181
 1190 1193 1196 1198 1200 1204 1208 1222 1226 1228 1230 1231 1232 1234
 1247 1263 1269 1274 1283 1284 1285 1286 1288 1289 1299 1302 1303 1309
 1310 1314 1320 1324 1326 1328 1330 1331 1339 1344 1353 1361 1362 1364
 1366 1381 1384 1395 1427 1433 1436 1442 1452 1453 1455 1460 1461 1472
 1474 1475 1477 1480 1488 1489 1492 1494 1506 1509 1525 1530 1532 1543
 1544 1547 1548 1553 1556 1562 1567 1574 1601 1605 1607 1609 1615 1617
 1618 1622 1640 1647 1668 1670 1671 1677 1681 1688 1694 1697 1701 1719
 1736 1737 1741 1747 1758 1764 1766 1775 1786 1789 1791 1804 1813 1816
 1824 1825 1835 1837 1838 1851 1855 1858 1861 1865 1866 1868 1871 1878
 1889 1890 1891 1893 1904 1906 1907 1908 1922 1932 1938 1956 1958 1962
 1965 1969 1980 1982 1983 1993 1995 1999 2005 2009 2011 2022 2027 2029
 2030]
x_train:         EJ        AB        AF         AH        AM        AR
AX  \
1      0 -0.680556 -0.989494   0.000000  0.611687  0.000000 -0.607692

2      1  0.375000 -0.224190   0.000000  0.440180  0.000000  0.738462
```

```
4       1  0.083333  0.283109   0.000000 -0.239281  0.000000 -0.473077
6       1 -0.020833 -0.640701   0.000000 -0.452397  7.173792 -1.353846
8       1 -0.027778  0.054576   0.000000  0.310973  0.000000 -0.438462
...    ..       ...       ...        ...       ...       ...       ...
2031    0  6.555556  1.811240  26.662191  0.084140  0.000000  0.253846
2032    0  0.763889  0.283647   0.000000  1.522245  0.000000  0.094231
2033    1  1.347222  1.698131   0.000000  0.954818  9.350052  0.280769
2034    1  1.263889  1.816902   0.000000 -0.001037  0.000000 -0.323077
2035    1  1.347222  1.698131   0.000000  0.954818  9.350052  0.280769

            AY        AZ         BC  ...        FI        FL
FR  \
1     0.000000  0.631510   0.000000  ...  0.138122 -0.472232  -
0.624571
2     0.000000  0.488281   0.000000  ...  0.561694  0.770546  -
0.153143
4     2.594595 -1.459635  26.204380  ...  1.243094  0.843681
46.670286
6     0.000000 -0.808594   0.000000  ...  0.278085  0.226183  -
0.265714
8     0.000000 -0.458333   0.622263  ...  0.502762  0.284527
0.050857
...        ...       ...        ...  ...       ...       ...         ..
.
2031  0.000000  1.420573   0.779197  ... -0.854512 -0.472232
1.924571
2032  0.000000  0.433594   0.000000  ... -1.055249 -0.472232  -
0.624571
2033  0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429
2034  1.567568 -0.386719   1.597628  ... -0.895028  1.047681  -
0.624571
2035  0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429

            FS        GB        GE        GF        GH        GI
GL
1     0.666667 -0.819132  0.000000  1.240601 -0.124567 -0.197595
0.990161
2     2.014493  1.581994  0.290982  0.359598 -0.218622 -0.129459 -
```

```
0.006520
4    -0.289855 -0.204180  1.336815  0.042256  1.248820 -0.105640 -
0.011111
6     0.449275 -0.077170  2.179715 -0.277507  0.300723  1.315202 -
0.012553
8    -0.405797 -0.131833  1.356979 -0.200081 -0.070777  0.304266 -
0.012173
...        ...       ...       ...       ...       ...       ...
...
2031 -0.405797  0.643891  0.000000  1.328281  1.296005 -0.303750
0.990161
2032 -0.333333 -0.111736  0.000000  0.070104  0.267694 -0.329402
0.990161
2033  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011
2034  0.985507 -0.854502  0.000000  0.192884  1.035546  0.062439 -
0.011296
2035  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011

[1629 rows x 56 columns] y_train:  [0 0 2 ... 3 3 3]
x_val:        EJ         AB        AF          AH          AM          AR
AX  \
0      1  -0.472222 -0.005214   0.000000   0.069272   0.000000 -
1.880769
3      1  -0.333333  0.323123   1.226427   2.105694   0.000000 -
0.584615
5      0  -0.472222 -0.233104   0.000000  -0.446289   0.000000 -
0.442308
7      1  -0.277778 -0.995182   0.000000   0.023859   0.000000 -
0.019231
10     0   0.486111  1.371810   0.000000   6.298870   8.888580
0.507692
...   ..        ...       ...         ...         ...         ...        .
..
2011   0   0.763889  0.283647   0.000000   1.522245   0.000000
0.094231
2022   0   0.527778  0.290919   0.418370  -0.206662   0.000000 -
0.292308
2027   0   0.763889  0.283647   0.000000   1.522245   0.000000
0.094231
2029   0  13.263889  0.796241  40.391211  22.701706  11.736072
2.700000
2030   0   1.986111  7.208136   0.000000   0.028239   1.489296
0.500000

              AY        AZ          BC  ...        FI        FL
FR  \
0       0.000000 -0.134115    1.123175  ... -2.125230  0.702705
```

```
0.598571
3       0.000000  0.122396    0.000000  ...  1.639042  0.508776 -
0.624571
5       0.000000  0.430990    0.000000  ...  0.346225 -0.472232 -
0.624571
7       0.000000 -0.217448    0.000000  ...  0.132597  0.586238 -
0.624571
10    913.351351  0.833333   54.889599  ...  8.653775 -0.472232
0.846000
...          ...       ...         ...  ...       ...       ...
...
2011    0.000000  0.433594    0.000000  ... -1.055249 -0.472232 -
0.624571
2022    1.081081  1.122396    0.897810  ... -0.038674 -0.472232
0.318857
2027    0.000000  0.433594    0.000000  ... -1.055249 -0.472232 -
0.624571
2029    0.000000  0.514323  379.728102  ... -1.072744 -0.472232 -
0.624571
2030    0.000000  0.345052   10.346715  ... -0.978821 -0.472232 -
0.624571

            FS         GB        GE        GF        GH        GI
GL
0    -0.347826  -0.642283  0.000000 -0.359338 -0.716263  0.641741 -
0.010025
3     0.057971  -0.020900  0.178349 -0.353767  0.789556  1.101632 -
0.008401
5     1.942029   0.271704  0.000000  1.006319 -0.176156  0.924306
0.990161
7     0.043478  -0.028135  0.674473  0.833511  0.388802 -0.213341 -
0.011282
10   -0.405797   0.816720  0.979233  0.263454  4.277760  0.711995
0.990161
...        ...        ...       ...       ...       ...       ...
...
2011 -0.333333  -0.111736  0.000000  0.070104  0.267694 -0.329402
0.990161
2022 -0.405797   1.067122  0.000000 -0.440104 -0.502674 -0.070255
0.990161
2027 -0.333333  -0.111736  0.000000  0.070104  0.267694 -0.329402
0.990161
2029  2.260870  10.111736  5.284113 -0.447118  1.069204 -0.059347
0.990161
2030 -0.405797  -0.728296  0.000000 -0.469769  2.633533 -0.105268
0.990161

[407 rows x 56 columns] y_val:  0       1
3       0
```

```
5       0
7       0
10      1
       ..
2011    1
2022    1
2027    1
2029    1
2030    1
Name: Class, Length: 407, dtype: int64
y_pred:  [[0.05944087 0.82599708 0.05614796 0.0584141 ]
 [0.80754806 0.08334094 0.05304981 0.05606116]
 [0.82818471 0.05686048 0.05666786 0.05828693]
 ...
 [0.05626109 0.05684354 0.0589075  0.82798786]
 [0.05650557 0.05680727 0.05859734 0.82808985]
 [0.05564395 0.05627082 0.05856815 0.82951708]]
> val_loss = 0.62233, split = 3.0
bll y_true:  4       1
9       0
11      0
16      0
17      0
       ..
2019    1
2020    1
2024    1
2033    1
2035    1
Name: Class, Length: 407, dtype: int64
bll y_pred:  [1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 1 0
0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0
0 0
 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1]
split:  4
train_idx:  [   0    1    2 ... 2031 2032 2034] val_idx:  [   4    9
11   16   17   19   22   28   35   36   38   46   50   57
   61   75   79   89   90   93  108  114  116  119  127  133  145  149
  153  154  157  159  169  172  176  180  190  191  217  223  234  245
  255  257  263  264  268  278  283  284  301  304  313  320  335  338
  340  347  356  357  369  372  373  386  389  395  396  399  407  412
  417  431  443  444  449  454  456  460  470  473  475  476  487  489
  491  496  498  499  501  504  511  512  515  517  521  537  539  546
  559  568  569  574  580  587  595  604  606  616  625  633  635  652
  653  655  656  657  658  662  667  671  675  684  685  689  690  696
  697  703  708  713  717  726  728  731  732  734  735  738  739  740
  750  753  758  760  761  768  773  783  784  789  790  797  801  814
  822  823  825  827  828  830  833  837  848  850  851  852  853  866
  868  872  875  876  882  884  885  890  894  895  902  908  911  912
  919  920  924  927  933  934  935  945  947  953  958  959  961  967
  969  970  971  977  980  991  992  996 1002 1003 1008 1011 1015 1019
 1024 1031 1032 1035 1039 1042 1048 1062 1065 1066 1069 1070 1072 1077
 1081 1088 1092 1093 1097 1098 1099 1115 1119 1127 1139 1140 1141 1142
 1145 1148 1149 1150 1155 1156 1166 1167 1168 1172 1174 1188 1191 1195
 1197 1199 1201 1203 1205 1209 1212 1213 1214 1217 1219 1227 1236 1243
 1246 1252 1253 1260 1276 1277 1279 1292 1295 1296 1307 1308 1311 1312
 1325 1346 1347 1348 1365 1373 1374 1375 1377 1386 1394 1399 1403 1404
 1416 1420 1422 1425 1428 1429 1444 1448 1458 1468 1470 1471 1483 1493
 1497 1498 1504 1505 1516 1517 1518 1521 1523 1535 1537 1539 1542 1546
 1550 1551 1552 1558 1569 1572 1588 1596 1602 1612 1623 1628 1629 1652
 1654 1656 1658 1659 1665 1666 1669 1686 1690 1691 1702 1712 1720 1723
 1730 1738 1739 1745 1754 1767 1774 1776 1778 1783 1807 1811 1815 1820
 1823 1826 1830 1840 1841 1846 1847 1854 1860 1864 1872 1884 1887 1911
 1914 1915 1917 1921 1923 1925 1926 1928 1940 1943 1944 1946 1953 1954
 1972 1985 1991 1994 1997 2000 2004 2007 2014 2015 2019 2020 2024 2033
 2035]
x_train:        EJ         AB         AF          AH          AM
AR         AX  \
0      1  -0.472222 -0.005214   0.000000   0.069272   0.000000 -
1.880769
1      0  -0.680556 -0.989494   0.000000   0.611687   0.000000 -
0.607692
2      1   0.375000 -0.224190   0.000000   0.440180   0.000000
0.738462
3      1  -0.333333  0.323123   1.226427   2.105694   0.000000 -
0.584615
5      0  -0.472222 -0.233104   0.000000  -0.446289   0.000000 -
0.442308
...   ..        ...        ...         ...         ...         ...        .
..
```

```
2029   0  13.263889  0.796241  40.391211  22.701706  11.736072
2.700000
2030   0   1.986111  7.208136   0.000000   0.028239   1.489296
0.500000
2031   0   6.555556  1.811240  26.662191   0.084140   0.000000
0.253846
2032   0   0.763889  0.283647   0.000000   1.522245   0.000000
0.094231
2034   1   1.263889  1.816902   0.000000  -0.001037   0.000000 -
0.323077

            AY        AZ          BC  ...        FI        FL
FR  \
0     0.000000 -0.134115    1.123175  ... -2.125230  0.702705
0.598571
1     0.000000  0.631510    0.000000  ...  0.138122 -0.472232 -
0.624571
2     0.000000  0.488281    0.000000  ...  0.561694  0.770546 -
0.153143
3     0.000000  0.122396    0.000000  ...  1.639042  0.508776 -
0.624571
5     0.000000  0.430990    0.000000  ...  0.346225 -0.472232 -
0.624571
...        ...       ...         ...  ...       ...       ...       ..
.
2029  0.000000  0.514323  379.728102  ... -1.072744 -0.472232 -
0.624571
2030  0.000000  0.345052   10.346715  ... -0.978821 -0.472232 -
0.624571
2031  0.000000  1.420573    0.779197  ... -0.854512 -0.472232
1.924571
2032  0.000000  0.433594    0.000000  ... -1.055249 -0.472232 -
0.624571
2034  1.567568 -0.386719    1.597628  ... -0.895028  1.047681 -
0.624571

            FS         GB        GE        GF        GH        GI
GL
0    -0.347826  -0.642283  0.000000 -0.359338 -0.716263  0.641741 -
0.010025
1     0.666667  -0.819132  0.000000  1.240601 -0.124567 -0.197595
0.990161
2     2.014493   1.581994  0.290982  0.359598 -0.218622 -0.129459 -
0.006520
3     0.057971  -0.020900  0.178349 -0.353767  0.789556  1.101632 -
0.008401
5     1.942029   0.271704  0.000000  1.006319 -0.176156  0.924306
0.990161
...        ...        ...       ...       ...       ...       ...
```

```
...
2029  2.260870  10.111736  5.284113 -0.447118  1.069204 -0.059347
0.990161
2030 -0.405797  -0.728296  0.000000 -0.469769  2.633533 -0.105268
0.990161
2031 -0.405797   0.643891  0.000000  1.328281  1.296005 -0.303750
0.990161
2032 -0.333333  -0.111736  0.000000  0.070104  0.267694 -0.329402
0.990161
2034  0.985507  -0.854502  0.000000  0.192884  1.035546  0.062439 -
0.011296

[1629 rows x 56 columns] y_train:  [1 0 0 ... 3 3 3]
x_val:        EJ        AB        AF        AH        AM        AR
AX  \
4      1  0.083333  0.283109  0.000000 -0.239281  0.000000 -0.473077

9      0 -0.097222  0.955677  0.000000 -0.281524  0.000000 -0.190385

11     1  1.319444  1.773067  4.025633  0.109036  0.000000  1.126923

16     0 -0.333333  0.951745  0.000000 -0.226833  0.000000 -0.092308

17     1  0.305556  0.183106  0.849863  1.343937  0.000000  0.361538

...   ..       ...       ...       ...       ...       ...       ...

2019   0  1.986111  7.208136  0.000000  0.028239  1.489296  0.500000

2020   0  1.305556  1.773192  5.064998  5.257377  0.000000  0.761538

2024   1  1.347222  1.698131  0.000000  0.954818  9.350052  0.280769

2033   1  1.347222  1.698131  0.000000  0.954818  9.350052  0.280769

2035   1  1.347222  1.698131  0.000000  0.954818  9.350052  0.280769


             AY        AZ         BC  ...        FI        FL
FR  \
4      2.594595 -1.459635  26.204380  ...  1.243094  0.843681
46.670286
9      0.000000  0.046224   0.000000  ...  0.796501 -0.472232
0.434000
11     0.000000  1.308594   0.000000  ...  1.335175  0.635411  -
0.040286
16    14.108108  0.037760   1.941606  ... -0.279926 -0.472232  -
0.403429
17     0.270270  0.148437   0.000000  ... -0.092081  0.235439
0.400571
```

```
...          ...       ...         ...  ...       ...       ...         .
..
2019   0.000000  0.345052  10.346715  ... -0.978821 -0.472232  -
0.624571
2020   0.000000 -0.302083   0.000000  ... -0.870166 -0.472232  -
0.624571
2024   0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429
2033   0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429
2035   0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429

            FS        GB        GE        GF        GH        GI
GL
4    -0.289855 -0.204180  1.336815  0.042256  1.248820 -0.105640 -
0.011111
9     0.130435  0.496383  0.478691 -0.140123 -0.728216 -0.171572
0.990161
11    2.159420  2.087621  0.000000  0.021824  0.173325  0.285084
0.004503
16   -0.405797 -0.438103  0.000000  1.813010  1.161057  0.234183
0.990161
17    0.000000 -0.742765  0.000000  2.550560  0.653979 -0.750072 -
0.004833
...        ...       ...       ...       ...       ...       ...
...
2019 -0.405797 -0.728296  0.000000 -0.469769  2.633533 -0.105268
0.990161
2020  0.565217 -0.116559  0.000000  0.278114 -1.370557 -0.113828
0.990161
2024  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011
2033  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011
2035  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011

[407 rows x 56 columns] y_val:  4       1
9       0
11      0
16      0
17      0
       ..
2019    1
2020    1
2024    1
2033    1
2035    1
```

```
Name: Class, Length: 407, dtype: int64
y_pred:  [[0.05876995 0.05333654 0.82896183 0.05893166]
 [0.83120387 0.05487917 0.05532177 0.0585952 ]
 [0.82103222 0.06227561 0.05568336 0.06100885]
 ...
 [0.05906967 0.05365887 0.05594927 0.83132219]
 [0.05906967 0.05365887 0.05594927 0.83132219]
 [0.05906967 0.05365887 0.05594927 0.83132219]]
best_model_saved
> val_loss = 0.19189, split = 4.0
bll y_true:  1       0
8       0
13      1
14      0
20      0
       ..
1989    1
2003    1
2006    1
2018    1
2032    1
Name: Class, Length: 407, dtype: int64
bll y_pred:  [0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 0 1
0 0 1 0 1 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0
0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 1 1 0 1 0 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1]
split:  5
train_idx:  [   0    2    3 ... 2033 2034 2035] val_idx:  [   1    8
13   14   20   21   26   34   37   40   53   64   87   91
   95   98  103  112  121  122  130  134  143  146  150  151  152  160
  161  166  186  187  189  197  200  201  202  205  206  207  216  219
```

```
  225  229  230  241  246  252  253  262  269  276  279  288  293  295
  315  317  330  337  343  345  378  379  384  385  387  391  392  397
  400  401  402  403  406  418  437  441  452  455  459  463  466  469
  472  474  484  488  492  502  508  509  510  520  524  540  550  556
  562  563  564  565  577  586  592  600  608  612  623  627  632  639
  640  641  642  645  646  647  648  659  663  681  683  686  687  698
  699  702  709  719  725  729  742  747  748  749  751  763  766  769
  775  776  779  791  794  795  800  804  805  815  821  831  835  840
  854  856  860  863  870  871  877  878  880  883  891  896  897  913
  928  929  951  954  955  956  957  960  972  975  981  995 1012 1014
 1016 1017 1020 1021 1025 1028 1038 1044 1045 1051 1056 1059 1060 1064
 1071 1076 1082 1086 1095 1104 1109 1122 1123 1126 1129 1130 1135 1136
 1143 1147 1152 1153 1154 1158 1162 1171 1180 1183 1184 1186 1194 1207
 1215 1218 1224 1238 1241 1248 1250 1254 1256 1257 1264 1266 1267 1275
 1282 1291 1294 1297 1300 1306 1321 1327 1332 1337 1349 1354 1363 1367
 1369 1371 1372 1382 1388 1390 1396 1397 1398 1400 1408 1409 1410 1411
 1413 1415 1426 1430 1434 1437 1438 1439 1443 1445 1451 1459 1462 1478
 1479 1482 1484 1485 1495 1496 1499 1500 1508 1513 1515 1520 1522 1527
 1528 1529 1533 1534 1570 1577 1579 1584 1585 1589 1590 1595 1597 1598
 1603 1631 1633 1634 1636 1638 1643 1648 1653 1661 1663 1675 1678 1682
 1683 1685 1687 1693 1695 1698 1700 1705 1707 1711 1714 1715 1722 1724
 1725 1732 1733 1734 1742 1748 1750 1751 1757 1759 1762 1765 1768 1770
 1772 1780 1782 1785 1790 1792 1795 1796 1801 1803 1806 1809 1812 1819
 1828 1829 1836 1842 1845 1856 1857 1862 1863 1869 1870 1873 1875 1877
 1880 1883 1886 1895 1898 1899 1913 1916 1918 1924 1927 1930 1931 1936
 1937 1941 1951 1961 1964 1976 1978 1984 1987 1988 1989 2003 2006 2018
 2032]
x_train:         EJ        AB        AF         AH        AM        AR
AX  \
0      1 -0.472222 -0.005214   0.000000  0.069272  0.000000 -1.880769

2      1  0.375000 -0.224190   0.000000  0.440180  0.000000  0.738462

3      1 -0.333333  0.323123   1.226427  2.105694  0.000000 -0.584615

4      1  0.083333  0.283109   0.000000 -0.239281  0.000000 -0.473077

5      0 -0.472222 -0.233104   0.000000 -0.446289  0.000000 -0.442308

...   ..       ...       ...        ...       ...       ...       ...

2030   0  1.986111  7.208136   0.000000  0.028239  1.489296  0.500000

2031   0  6.555556  1.811240  26.662191  0.084140  0.000000  0.253846

2033   1  1.347222  1.698131   0.000000  0.954818  9.350052  0.280769

2034   1  1.263889  1.816902   0.000000 -0.001037  0.000000 -0.323077

2035   1  1.347222  1.698131   0.000000  0.954818  9.350052  0.280769
```

```
              AY        AZ         BC  ...        FI        FL
FR  \
0     0.000000 -0.134115   1.123175  ... -2.125230  0.702705
0.598571
2     0.000000  0.488281   0.000000  ...  0.561694  0.770546  -
0.153143
3     0.000000  0.122396   0.000000  ...  1.639042  0.508776  -
0.624571
4     2.594595 -1.459635  26.204380  ...  1.243094  0.843681
46.670286
5     0.000000  0.430990   0.000000  ...  0.346225 -0.472232  -
0.624571
...        ...       ...        ...  ...       ...       ...        ..
.
2030  0.000000  0.345052  10.346715  ... -0.978821 -0.472232  -
0.624571
2031  0.000000  1.420573   0.779197  ... -0.854512 -0.472232
1.924571
2033  0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429
2034  1.567568 -0.386719   1.597628  ... -0.895028  1.047681  -
0.624571
2035  0.000000 -0.290365   0.815693  ... -1.543278 -0.018171  -
0.231429

              FS        GB        GE        GF        GH        GI
GL
0    -0.347826 -0.642283  0.000000 -0.359338 -0.716263  0.641741 -
0.010025
2     2.014493  1.581994  0.290982  0.359598 -0.218622 -0.129459 -
0.006520
3     0.057971 -0.020900  0.178349 -0.353767  0.789556  1.101632 -
0.008401
4    -0.289855 -0.204180  1.336815  0.042256  1.248820 -0.105640 -
0.011111
5     1.942029  0.271704  0.000000  1.006319 -0.176156  0.924306
0.990161
...        ...       ...       ...       ...       ...       ...
...
2030 -0.405797 -0.728296  0.000000 -0.469769  2.633533 -0.105268
0.990161
2031 -0.405797  0.643891  0.000000  1.328281  1.296005 -0.303750
0.990161
2033  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
0.066011
2034  0.985507 -0.854502  0.000000  0.192884  1.035546  0.062439 -
0.011296
2035  1.478261 -0.165595  0.000000 -0.470590 -1.117332  0.482279
```

```
0.066011

[1629 rows x 56 columns] y_train:  [1 0 0 ... 3 3 3]
x_val:         EJ        AB        AF        AH         AM        AR
AX  \
1      0 -0.680556 -0.989494  0.000000  0.611687   0.000000 -0.607692

8      1 -0.027778  0.054576  0.000000  0.310973   0.000000 -0.438462

13     0  0.583333  0.769066  0.192859 -0.438912   0.000000  0.855769

14     0  0.500000 -0.687611  0.000000 -0.322614   9.478530  0.900000

20     0  0.652778  1.438079  0.000000  0.193753   0.000000 -0.015385

...   ..       ...       ...       ...       ...        ...       ...

1989   0  2.125000  2.978081  0.000000  1.203550   6.371460  1.659615

2003   0  0.361111  0.169840  0.000000 -0.011065  18.574248  1.034615

2006   1  1.263889  1.816902  0.000000 -0.001037   0.000000 -0.323077

2018   0  0.527778  0.290919  0.418370 -0.206662   0.000000 -0.292308

2032   0  0.763889  0.283647  0.000000  1.522245   0.000000  0.094231


             AY        AZ        BC  ...        FI        FL
FR  \
1      0.000000  0.631510  0.000000  ...  0.138122 -0.472232  -
0.624571
8      0.000000 -0.458333  0.622263  ...  0.502762  0.284527
0.050857
13     0.000000 -0.063802  3.532847  ... -0.325046 -0.472232
43.833714
14    18.513514  0.511719  2.650547  ... -2.125230 -0.472232
0.446571
20     0.000000 -0.957031  0.000000  ...  1.388582 -0.472232
0.146857
...         ...       ...       ...  ...       ...       ...        ..
.
1989   0.000000  1.311198  0.000000  ... -0.171271 -0.472232  -
0.624571
2003   2.756757 -1.459635  0.000000  ... -0.751381 -0.472232  -
0.624571
2006   1.567568 -0.386719  1.597628  ... -0.895028  1.047681  -
0.624571
2018   1.081081  1.122396  0.897810  ... -0.038674 -0.472232
0.318857
```

```
2032   0.000000  0.433594  0.000000  ... -1.055249 -0.472232  -
0.624571

            FS        GB        GE        GF        GH        GI
GL
1     0.666667 -0.819132  0.000000  1.240601 -0.124567 -0.197595
0.990161
8    -0.405797 -0.131833  1.356979 -0.200081 -0.070777  0.304266 -
0.012173
13   -0.405797  1.470257  0.000000 -0.427596  1.307015  0.468308
0.990161
14   -0.405797 -0.650322  0.000000  0.242590  0.327461 -0.722244
0.990161
20   -0.188406  1.066318  0.000000 -0.452747  0.173010  1.295219
0.990161
...        ...       ...       ...       ...       ...       ...
...
1989 -0.405797 -0.253215  0.932557 -0.382526 -0.522491  1.024764
0.990161
2003  0.000000 -0.585209  0.000000  0.424901 -0.674740  0.531806
0.990161
2006  0.985507 -0.854502  0.000000  0.192884  1.035546  0.062439 -
0.011296
2018 -0.405797  1.067122  0.000000 -0.440104 -0.502674 -0.070255
0.990161
2032 -0.333333 -0.111736  0.000000  0.070104  0.267694 -0.329402
0.990161

[407 rows x 56 columns] y_val:  1       0
8       0
13      1
14      0
20      0
       ..
1989    1
2003    1
2006    1
2018    1
2032    1
Name: Class, Length: 407, dtype: int64
y_pred:  [[0.83008055 0.06063098 0.05524565 0.05404281]
 [0.82865808 0.06204619 0.05561187 0.05368387]
 [0.05927665 0.05713828 0.82776712 0.05581794]
 ...
 [0.05955224 0.05727438 0.05647316 0.82670023]
 [0.06638457 0.06016351 0.05943937 0.81401254]
 [0.06175992 0.05944434 0.05927922 0.81951653]]
> val_loss = 0.40162, split = 5.0
LOSS: 0.42932
```

Prediction and Submission

XGBoost with LGBM

```
display(test.head())

# store column 'Id' from test set for submission use
test_id = test['Id']
```

```
              Id  EJ   AB   AF   AH   AM   AR   AX   AY   AZ  ...   FI
FL  \
0  00eed32682bb   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
1  010ebe33f668   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
2  02fa521e1838   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
3  040e15f562a2   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0
4  046e85c7cc7f   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0
0.0

    FR   FS   GB   GE   GF   GH   GI   GL
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 57 columns]
```

```
# xgb_b_acu_s = metrics.balanced_accuracy_score(y_test, xgb_pred)
# lgbm_b_acu_s = metrics.balanced_accuracy_score(y_test, lgbm_pred)

# xgb_lgbm_weight = xgb_b_acu_s / (xgb_b_acu_s + lgbm_b_acu_s)
# lgbm_xgb_weight = lgbm_b_acu_s / (xgb_b_acu_s + lgbm_b_acu_s)

# print('xgb_lgbm_weight: ', xgb_lgbm_weight)
# print('lgbm_xgb_weight: ', lgbm_xgb_weight)

# xgb_pred_t = xgb_best.predict(test[features_selected])
# xgb_pred_proba_t =
pd.DataFrame(xgb_best.predict_proba(test[features_selected]))
# lgbm_pred_t = lgbm_best.predict(test[features_selected])
# lgbm_pred_proba_t =
pd.DataFrame(lgbm_best.predict_proba(test[features_selected]))

# submission = pd.DataFrame(test_id, columns = ['Id'])
# submission['Id'] = test.reset_index()['Id']
```

```
# submission['class_0'] = (xgb_pred_proba_t[0]*xgb_lgbm_weight) +
(lgbm_pred_proba_t[0]*lgbm_xgb_weight)
# submission['class_1'] = (xgb_pred_proba_t[1]*xgb_lgbm_weight) +
(lgbm_pred_proba_t[1]*lgbm_xgb_weight)

# submission.set_index('Id').to_csv('submission.csv', index = False)
# submission

test=test.drop(['Id'], axis = 1)
display(test.head())
y_pred = model.predict_proba(test)
p0 = y_pred[:,0]
```

```
   EJ   AB   AF   AH   AM   AR   AX   AY   AZ   BC  ...   FI   FL   FR
FS  \
0   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
0.0
1   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
0.0
2   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
0.0
3   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
0.0
4   0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
0.0

    GB   GE   GF   GH   GI   GL
0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 56 columns]
```

```
submission = pd.DataFrame(test_id, columns = ['Id'])

submission['class_0'] = p0
submission['class_1'] = 1 - p0

submission.to_csv('submission.csv', index = False)
submission
```

```
             Id   class_0   class_1
0  00eed32682bb  0.830898  0.169102
1  010ebe33f668  0.830898  0.169102
2  02fa521e1838  0.830898  0.169102
3  040e15f562a2  0.830898  0.169102
4  046e85c7cc7f  0.830898  0.169102
```