Based

- https://www.kaggle.com/code/honganzhu/cmi-piu-competition?scriptVersionId=201912528 Version44 LB0.492

If you find this notebook useful, please upvote this and the based one.

Description of Imported Libraries

- **NumPy (`np`)**: Used for efficient numerical operations, including linear algebra and array manipulation.
- **Pandas (`pd`)**: Provides data structures like DataFrames for handling structured data, essential for data preprocessing.
- **Polars (`pl`)**: A faster alternative to pandas for DataFrame operations, particularly useful for large datasets.
- **Matplotlib & Seaborn (`plt`, `sns`)**: Visualization libraries. Matplotlib is used for basic plots, while Seaborn builds on it to create more advanced statistical visualizations.
- **LightGBM, XGBoost, CatBoost**: Machine learning libraries used for gradient boosting, which is efficient for both regression and classification tasks.
- **Colorama**: Enhances console output with colored text, making it easier to highlight important results or warnings.
- **SciPy (`minimize`)**: Provides optimization routines, such as adjusting thresholds to maximize performance metrics like kappa scores.
- **OS**: Used for file path manipulations and system-related functions.
- **Scikit-learn (`sklearn`)**: A powerful machine learning library, providing utilities for cross-validation, metrics, and model cloning.
- **YDF**: A specialized library for machine learning tasks, likely including decision forests.
- **ThreadPoolExecutor & TQDM**: Tools for parallelizing tasks and displaying progress bars for long-running loops, improving efficiency and usability.
- **Warnings**: Filters out unwanted warnings to keep the output clean, useful when dealing with noisy outputs from multiple libraries.
- **IPython display (`clear_output`)**: A utility for clearing the Jupyter notebook output, often used to avoid clutter in long-running scripts.

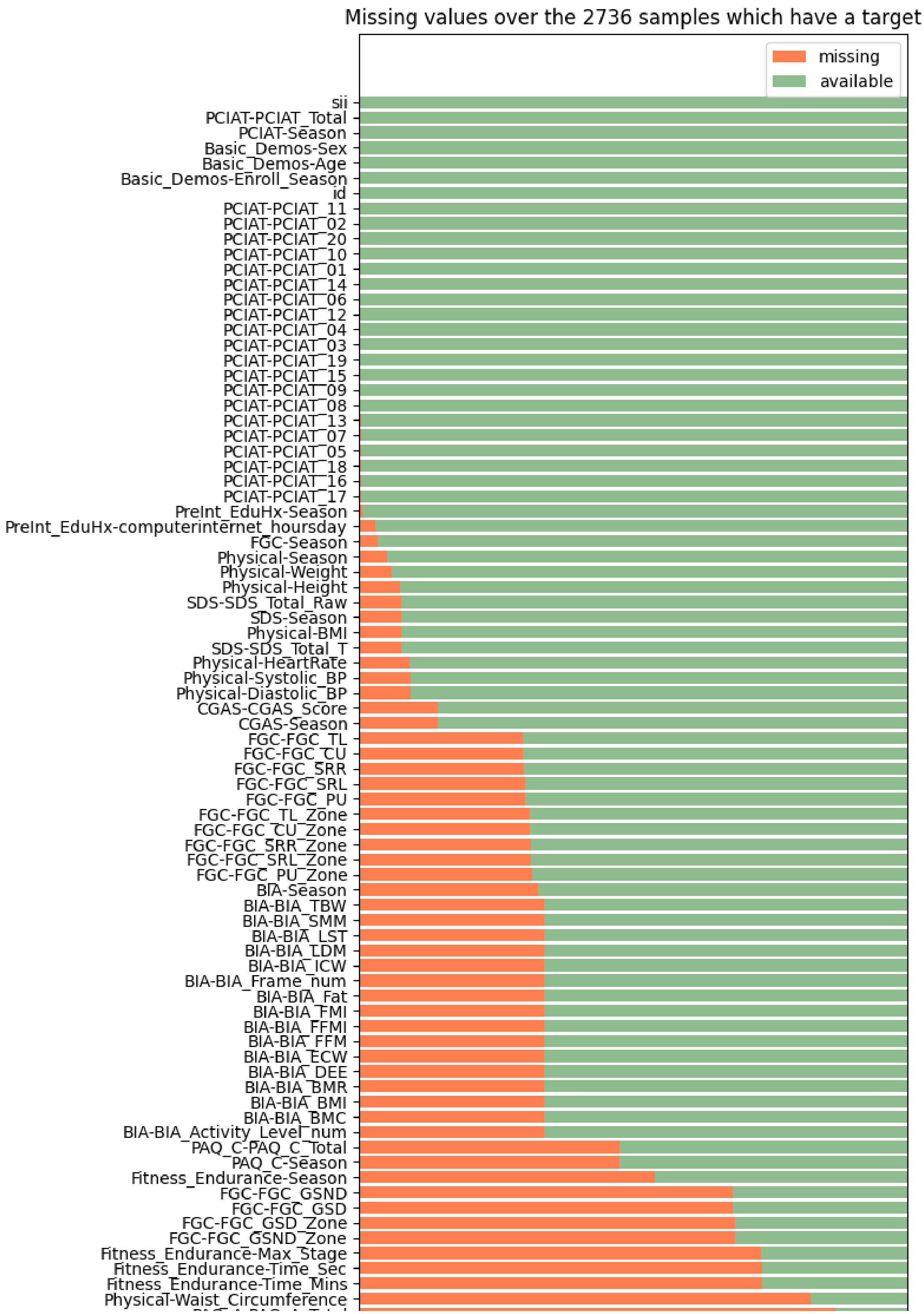
```
!pip -q install /kaggle/input/pytorchtabnet/pytorch_tabnet-4.1.0-py3-none-any.whl

from pytorch_tabnet.tab_model import TabNetRegressor
import torch

import numpy as np
import pandas as pd
import os
import re
from sklearn.base import clone
```

```python
from sklearn.metrics import cohen_kappa_score
from sklearn.model_selection import StratifiedKFold
from scipy.optimize import minimize
from concurrent.futures import ThreadPoolExecutor
from tqdm import tqdm
import polars as pl
import polars.selectors as cs
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator, FormatStrFormatter,
PercentFormatter
import seaborn as sns

from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from keras.models import Model
from keras.layers import Input, Dense
from keras.optimizers import Adam
import torch
import torch.nn as nn
import torch.optim as optim

from colorama import Fore, Style
from IPython.display import clear_output
import warnings
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from sklearn.ensemble import VotingRegressor, RandomForestRegressor,
GradientBoostingRegressor
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.pipeline import Pipeline
warnings.filterwarnings('ignore')
pd.options.display.max_columns = None

target_labels = ['None', 'Mild', 'Moderate', 'Severe']

season_dtype = pl.Enum(['Spring', 'Summer', 'Fall', 'Winter'])

train = (
    pl.read_csv('/kaggle/input/child-mind-institute-problematic-
internet-use/train.csv')
    .with_columns(pl.col('^.*Season$').cast(season_dtype))
)

test = (
    pl.read_csv('/kaggle/input/child-mind-institute-problematic-
internet-use/test.csv')
    .with_columns(pl.col('^.*Season$').cast(season_dtype))
)
```

```
train
test

shape: (20, 59)
┌──────────┬─────────────┬─────────────┬─────────────┬───┬─────────────┬────────
────┬─────────────┬─────────────┐
│ id       ┆ Basic_Dem   ┆ Basic_Dem   ┆ Basic_Dem   ┆ … ┆ SDS-SDS_T   ┆ SDS-
SDS_T ┆ PreInt_Ed ┆ PreInt_Ed │
│ ---      ┆ os-Enroll   ┆ os-Age      ┆ os-Sex      ┆   ┆ otal_Raw    ┆
otal_T    ┆ uHx-Seaso ┆ uHx-compu │
│ str      ┆ _Season     ┆ ---         ┆ ---         ┆   ┆ ---         ┆ ---
┆ n          ┆ terintern │
│          ┆ ---         ┆ i64         ┆ i64         ┆   ┆ i64         ┆ i64
┆ ---        ┆ et_…      │
│          ┆ enum        ┆             ┆             ┆   ┆             ┆
┆ enum       ┆ ---       │
│          ┆             ┆             ┆             ┆   ┆             ┆
┆            ┆ i64       │
╞══════════╪═════════════╪═════════════╪═════════════╪═══╪═════════════╪════════
════╪═════════════╪═════════════╡
│ 00008ff9 ┆ Fall        ┆ 5           ┆ 0           ┆ … ┆ null        ┆ null
┆ Fall       ┆ 3         │
│ 000fd460 ┆ Summer      ┆ 9           ┆ 0           ┆ … ┆ 46          ┆ 64
┆ Summer     ┆ 0         │
│ 00105258 ┆ Summer      ┆ 10          ┆ 1           ┆ … ┆ 38          ┆ 54
┆ Summer     ┆ 2         │
│ 00115b9f ┆ Winter      ┆ 9           ┆ 0           ┆ … ┆ 31          ┆ 45
┆ Winter     ┆ 0         │
│ 0016bb22 ┆ Spring      ┆ 18          ┆ 1           ┆ … ┆ null        ┆ null
┆ null       ┆ null      │
│ …        ┆ …           ┆ …           ┆ …           ┆ … ┆ …           ┆ …
┆ …          ┆ …         │
│ 00c0cd71 ┆ Winter      ┆ 7           ┆ 0           ┆ … ┆ 35          ┆ 50
┆ Winter     ┆ 2         │
│ 00d56d4b ┆ Spring      ┆ 5           ┆ 1           ┆ … ┆ 37          ┆ 53
┆ Spring     ┆ 0         │
│ 00d9913d ┆ Fall        ┆ 10          ┆ 1           ┆ … ┆ null        ┆ null
┆ Fall       ┆ 1         │
│ 00e6167c ┆ Winter      ┆ 6           ┆ 0           ┆ … ┆ 39          ┆ 55
┆ Winter     ┆ 3         │
│ 00ebc35d ┆ Winter      ┆ 10          ┆ 0           ┆ … ┆ null        ┆ null
┆ Winter     ┆ 2         │
└──────────┴─────────────┴─────────────┴─────────────┴───┴─────────────┴────────
────┴─────────────┴─────────────┘
```

For a supervised learning, we need the target value, but some (sii) are missing. So we only use the part with valid target value(sii).

```
supervised_usable = (
    train
    .filter(pl.col('sii').is_not_null())
)

missing_count = (
    supervised_usable
    .null_count()
    .transpose(include_header=True,
               header_name='feature',
               column_names=['null_count'])
    .sort('null_count', descending=True)
    .with_columns((pl.col('null_count') /
len(supervised_usable)).alias('null_ratio'))
)
plt.figure(figsize=(6, 15))
plt.title(f'Missing values over the {len(supervised_usable)} samples
which have a target')
plt.barh(np.arange(len(missing_count)),
missing_count.get_column('null_ratio'), color='coral',
label='missing')
plt.barh(np.arange(len(missing_count)),
         1 - missing_count.get_column('null_ratio'),
         left=missing_count.get_column('null_ratio'),
         color='darkseagreen', label='available')
plt.yticks(np.arange(len(missing_count)),
missing_count.get_column('feature'))
plt.gca().xaxis.set_major_formatter(PercentFormatter(xmax=1,
decimals=0))
plt.xlim(0, 1)
plt.legend()
plt.show()
```


Missing values over the 2736 samples which have a target
missing
available
sii
PCIAT-PCIAT_Total
PCIAT-Season
Basic_Demos-Sex
Basic_Demos-Age
Basic_Demos-Enroll_Season
id
PCIAT-PCIAT_11
PCIAT-PCIAT_02
PCIAT-PCIAT_20
PCIAT-PCIAT_10
PCIAT-PCIAT_01
PCIAT-PCIAT_14
PCIAT-PCIAT_06
PCIAT-PCIAT_12
PCIAT-PCIAT_04
PCIAT-PCIAT_03
PCIAT-PCIAT_19
PCIAT-PCIAT_15
PCIAT-PCIAT_09
PCIAT-PCIAT_08
PCIAT-PCIAT_13
PCIAT-PCIAT_07
PCIAT-PCIAT_05
PCIAT-PCIAT_18
PCIAT-PCIAT_16
PCIAT-PCIAT_17
PreInt_EduHx-Season
PreInt_EduHx-computerinternet_hoursday
FGC-Season
Physical-Season
Physical-Weight
Physical-Height
SDS-SDS_Total_Raw
SDS-Season
Physical-BMI
SDS-SDS_Total_T
Physical-HeartRate
Physical-Systolic_BP
Physical-Diastolic_BP
CGAS-CGAS_Score
CGAS-Season
FGC-FGC_TL
FGC-FGC_CU
FGC-FGC_SRR
FGC-FGC_SRL
FGC-FGC_PU
FGC-FGC_TL_Zone
FGC-FGC_CU_Zone
FGC-FGC_SRR_Zone
FGC-FGC_SRL_Zone
FGC-FGC_PU_Zone
BIA-Season
BIA-BIA_TBW
BIA-BIA_SMM
BIA-BIA_LST
BIA-BIA_LDM
BIA-BIA_ICW
BIA-BIA_Frame_num
BIA-BIA_Fat
BIA-BIA_FMI
BIA-BIA_FFMI
BIA-BIA_FFM
BIA-BIA_ECW
BIA-BIA_DEE
BIA-BIA_BMR
BIA-BIA_BMI
BIA-BIA_BMC
BIA-BIA_Activity_Level_num
PAQ_C-PAQ_C_Total
PAQ_C-Season
Fitness_Endurance-Season
FGC-FGC_GSND
FGC-FGC_GSD
FGC-FGC_GSD_Zone
FGC-FGC_GSND_Zone
Fitness_Endurance-Max_Stage
Fitness_Endurance-Time_Sec
Fitness_Endurance-Time_Mins
Physical-Waist_Circumference

```
print(train.select(pl.col('PCIAT-PCIAT_Total').is_null() ==
pl.col('sii').is_null()).to_series().mean())

(train
 .select(pl.col('PCIAT-PCIAT_Total'))
 .group_by(train.get_column('sii'))
 .agg(pl.col('PCIAT-PCIAT_Total').min().alias('PCIAT-PCIAT_Total
min'),
      pl.col('PCIAT-PCIAT_Total').max().alias('PCIAT-PCIAT_Total
max'),
      pl.col('PCIAT-PCIAT_Total').len().alias('count'))
 .sort('sii')
)
```

```
1.0
```

```
shape: (5, 4)
```

sii --- i64	PCIAT-PCIAT_Total min --- i64	PCIAT-PCIAT_Total max --- i64	count --- u32
null	null	null	1224
0	0	30	1594
1	31	49	730
2	50	79	378
3	80	93	34

Insight:

This dataset is imbalanced. Half of the samples are in class 0, while very few in class 3.

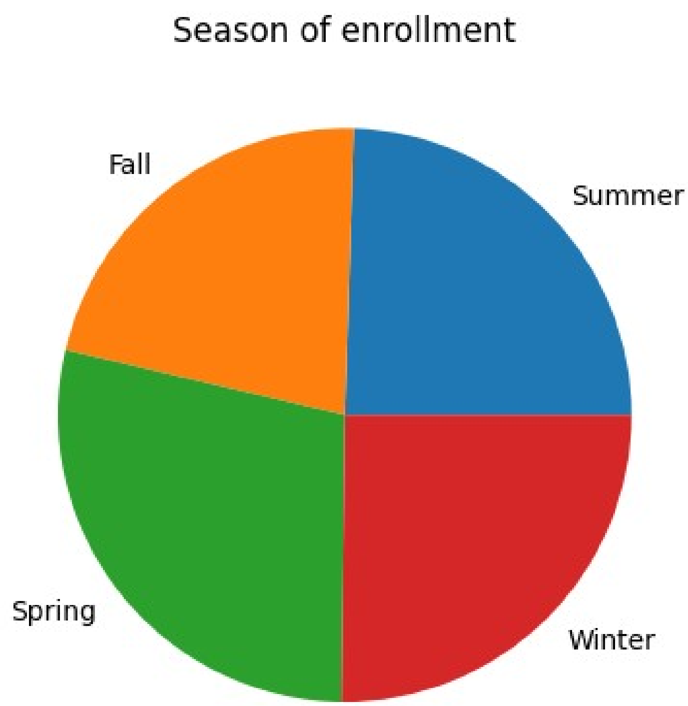
```
print('Columns missing in test:')
print([f for f in train.columns if f not in test.columns])
```

```
Columns missing in test:
['PCIAT-Season', 'PCIAT-PCIAT_01', 'PCIAT-PCIAT_02', 'PCIAT-PCIAT_03',
'PCIAT-PCIAT_04', 'PCIAT-PCIAT_05', 'PCIAT-PCIAT_06', 'PCIAT-
PCIAT_07', 'PCIAT-PCIAT_08', 'PCIAT-PCIAT_09', 'PCIAT-PCIAT_10',
'PCIAT-PCIAT_11', 'PCIAT-PCIAT_12', 'PCIAT-PCIAT_13', 'PCIAT-
PCIAT_14', 'PCIAT-PCIAT_15', 'PCIAT-PCIAT_16', 'PCIAT-PCIAT_17',
'PCIAT-PCIAT_18', 'PCIAT-PCIAT_19', 'PCIAT-PCIAT_20', 'PCIAT-
PCIAT_Total', 'sii']
```

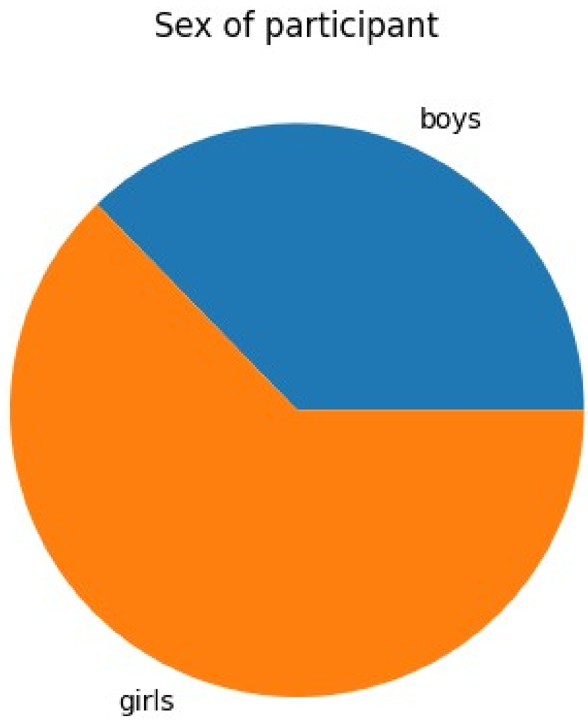
Demographics

Now we look at some basic demographics.

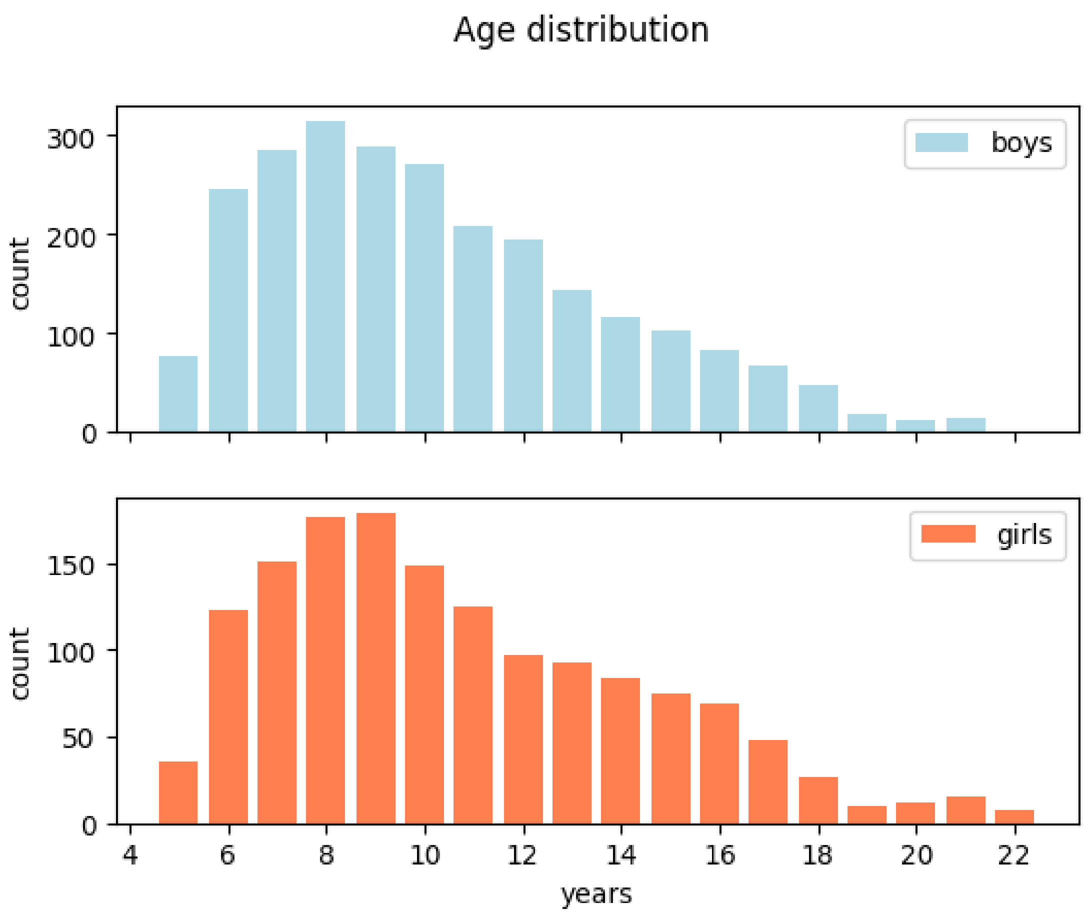
```
vc = train.get_column('Basic_Demos-Enroll_Season').value_counts()
plt.pie(vc.get_column('count'), labels=vc.get_column('Basic_Demos-
Enroll_Season'))
plt.title('Season of enrollment')
plt.show()
```



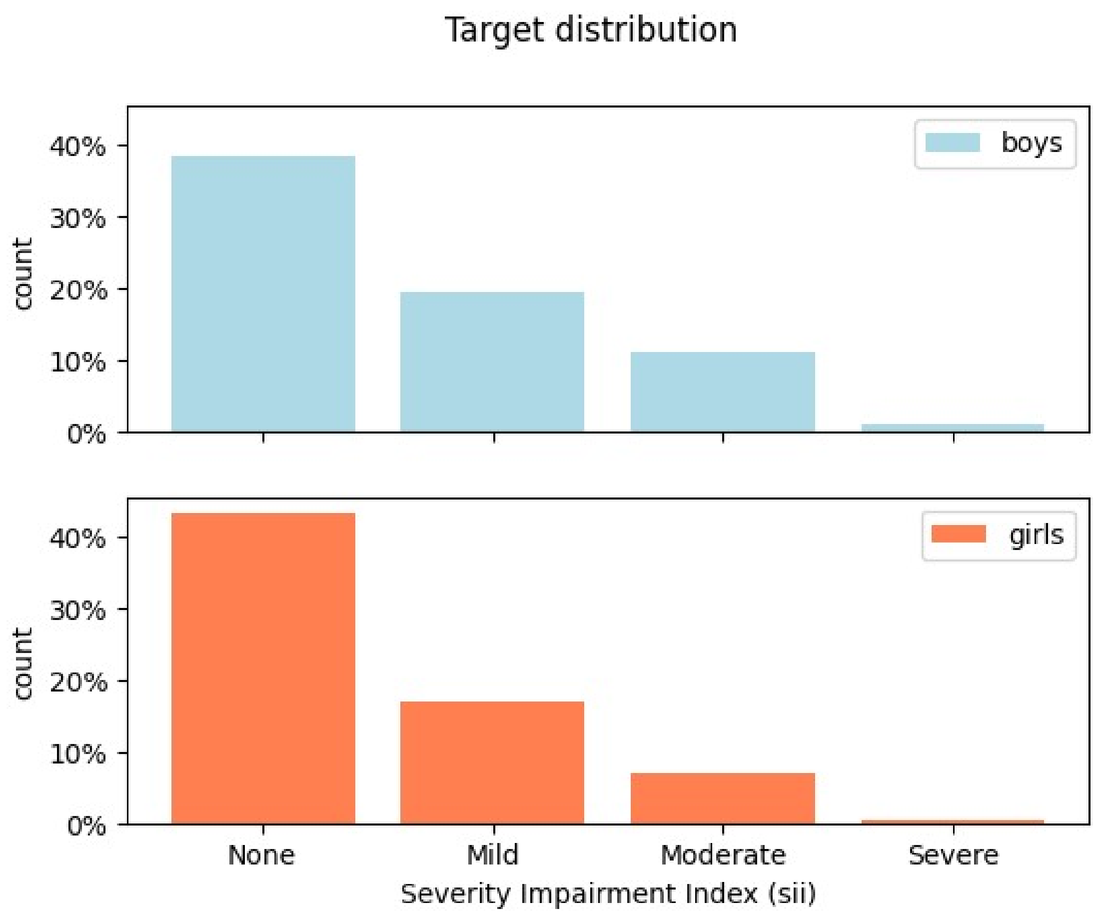

```
vc = train.get_column('Basic_Demos-Sex').value_counts()
plt.pie(vc.get_column('count'), labels=['boys', 'girls'])
plt.title('Sex of participant')
plt.show()
```




```
_, axs = plt.subplots(2, 1, sharex=True)
for sex in range(2):
    ax = axs.ravel()[sex]
    vc = train.filter(pl.col('Basic_Demos-Sex') ==
sex).get_column('Basic_Demos-Age').value_counts()
    ax.bar(vc.get_column('Basic_Demos-Age'),
           vc.get_column('count'),
           color=['lightblue', 'coral'][sex],
           label=['boys', 'girls'][sex])
    ax.xaxis.set_major_locator(MaxNLocator(integer=True))
    ax.set_ylabel('count')
    ax.legend()
plt.suptitle('Age distribution')
axs.ravel()[1].set_xlabel('years')
plt.show()
```




```
_, axs = plt.subplots(2, 1, sharex=True, sharey=True)
for sex in range(2):
    ax = axs.ravel()[sex]
    vc = train.filter(pl.col('Basic_Demos-Sex') ==
sex).get_column('sii').value_counts()
    ax.bar(vc.get_column('sii'),
           vc.get_column('count') / vc.get_column('count').sum(),
           color=['lightblue', 'coral'][sex],
           label=['boys', 'girls'][sex])
    ax.set_xticks(np.arange(4), target_labels)
    ax.yaxis.set_major_formatter(PercentFormatter(xmax=1, decimals=0))
    ax.set_ylabel('count')
    ax.legend()
plt.suptitle('Target distribution')
axs.ravel()[1].set_xlabel('Severity Impairment Index (sii)')
plt.show()
```



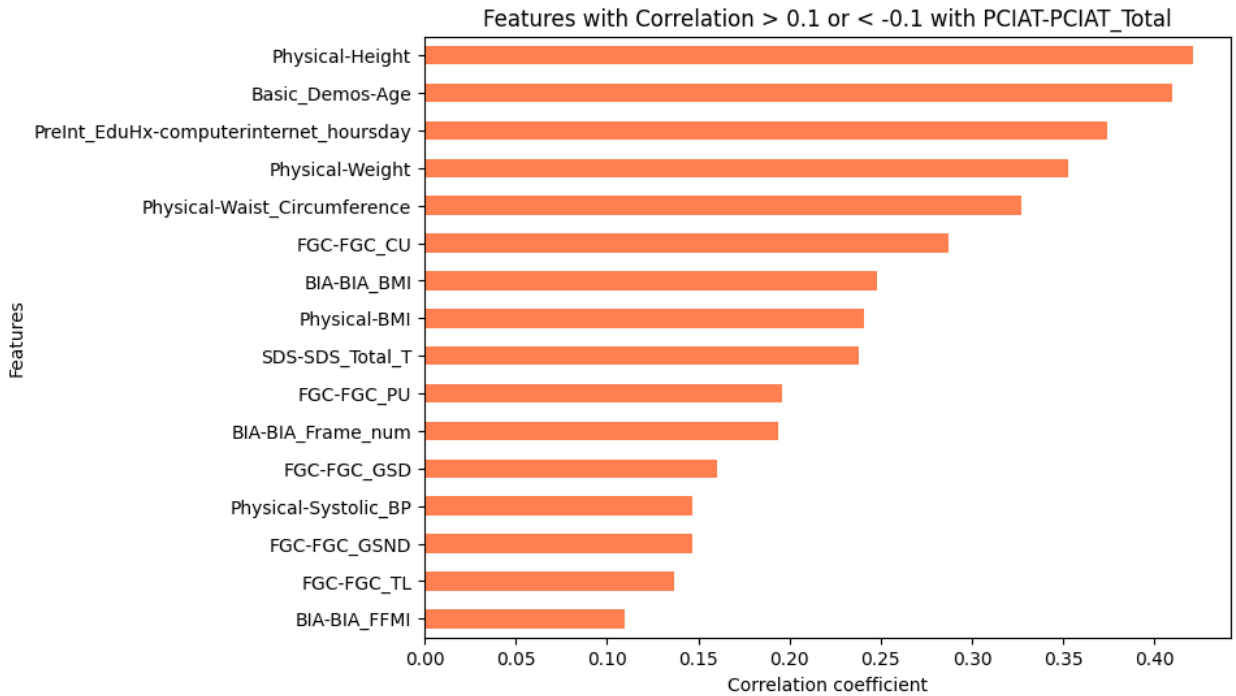
Now we look at correlations

```
plt.figure(figsize=(14, 12))
corr_matrix = supervised_usable.select([
    'PCIAT-PCIAT_Total', 'Basic_Demos-Age', 'Basic_Demos-Sex',
'Physical-BMI',
    'Physical-Height', 'Physical-Weight', 'Physical-
Waist_Circumference',
    'Physical-Diastolic_BP', 'Physical-Systolic_BP', 'Physical-
HeartRate',
    'PreInt_EduHx-computerinternet_hoursday', 'SDS-SDS_Total_T',
'PAQ_A-PAQ_A_Total',
    'PAQ_C-PAQ_C_Total', 'Fitness_Endurance-Max_Stage',
'Fitness_Endurance-Time_Mins','Fitness_Endurance-Time_Sec',
    'FGC-FGC_CU', 'FGC-FGC_GSND','FGC-FGC_GSD','FGC-FGC_PU','FGC-
FGC_SRL','FGC-FGC_SRR','FGC-FGC_TL','BIA-BIA_Activity_Level_num',
    'BIA-BIA_BMC', 'BIA-BIA_BMI', 'BIA-BIA_BMR', 'BIA-BIA_DEE', 'BIA-
BIA_ECW', 'BIA-BIA_FFM',
    'BIA-BIA_FFMI','BIA-BIA_FMI', 'BIA-BIA_Fat','BIA-
BIA_Frame_num','BIA-BIA_ICW','BIA-BIA_LDM','BIA-BIA_LST',
```

```
    'BIA-BIA_SMM','BIA-BIA_TBW'
    # Add other relevant columns
]).to_pandas().corr()

sii_corr = corr_matrix['PCIAT-PCIAT_Total'].drop('PCIAT-PCIAT_Total')
filtered_corr = sii_corr[(sii_corr > 0.1) | (sii_corr < -0.1)]

print(filtered_corr)

plt.figure(figsize=(8, 6))
filtered_corr.sort_values().plot(kind='barh', color='coral')
plt.title('Features with Correlation > 0.1 or < -0.1 with PCIAT-
PCIAT_Total')
plt.xlabel('Correlation coefficient')
plt.ylabel('Features')
plt.show()
```

```
Basic_Demos-Age                                    0.409559
Physical-BMI                                       0.240858
Physical-Height                                    0.420765
Physical-Weight                                    0.353048
Physical-Waist_Circumference                       0.327013
Physical-Systolic_BP                               0.147081
PreInt_EduHx-computerinternet_hoursday             0.374124
SDS-SDS_Total_T                                    0.237718
FGC-FGC_CU                                         0.287494
FGC-FGC_GSND                                       0.146813
FGC-FGC_GSD                                        0.160472
FGC-FGC_PU                                         0.196006
FGC-FGC_TL                                         0.136696
BIA-BIA_BMI                                        0.248060
BIA-BIA_FFMI                                       0.109694
BIA-BIA_Frame_num                                  0.193631
Name: PCIAT-PCIAT_Total, dtype: float64
```

```
<Figure size 1400x1200 with 0 Axes>
```



Actigraphy (time series)

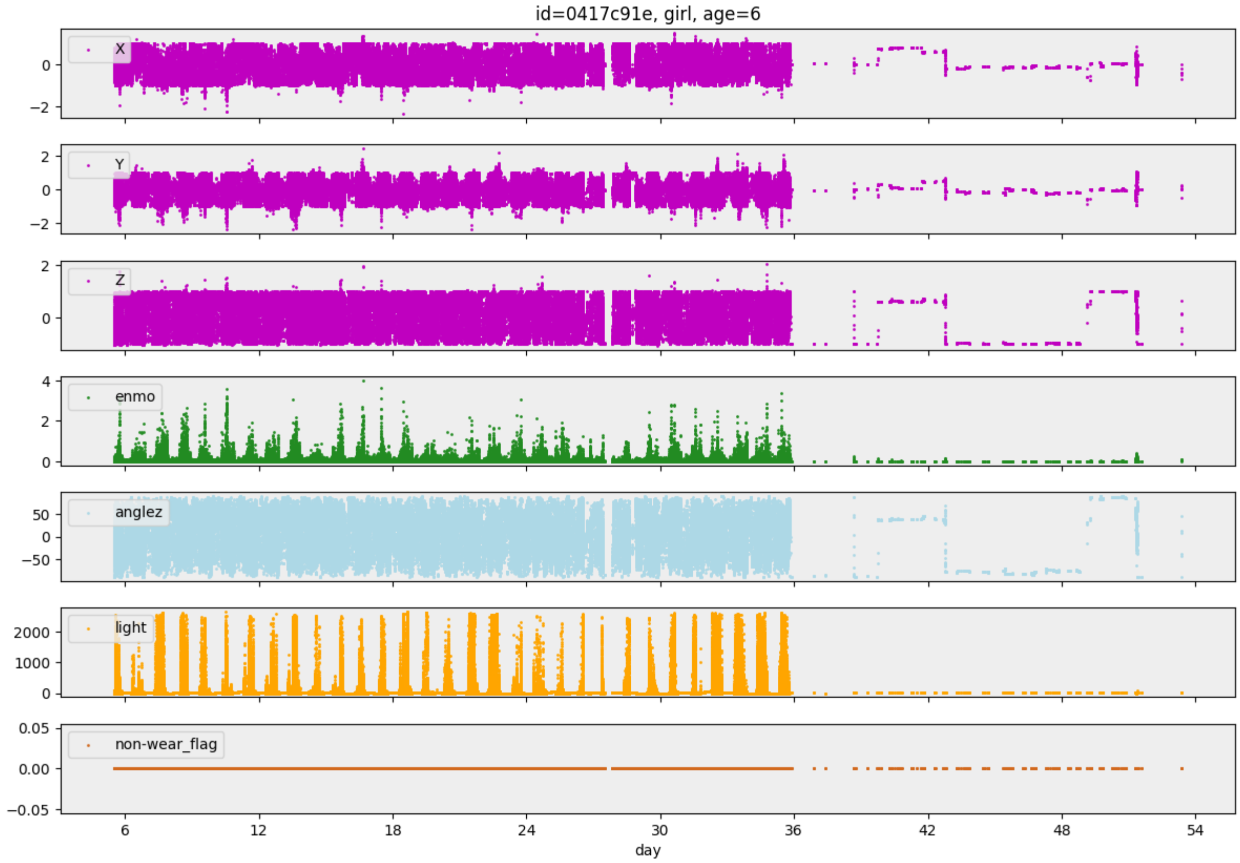
```
actigraphy = pl.read_parquet('/kaggle/input/child-mind-institute-
problematic-internet-use/series_train.parquet/id=0417c91e/part-
0.parquet')
actigraphy
```

```
shape: (287_179, 13)
┌─────────┬───────────┬───────────┬───────────┬───┬──────────────┬─────
────┬─────────┬─────────────┐
│ step    ┆ X         ┆ Y         ┆ Z         ┆ … ┆ time_of_day  ┆
weekday ┆ quarter ┆ relative_dat │
│ ---     ┆ ---       ┆ ---       ┆ ---       ┆   ┆ ---          ┆ ---
┆ ---     ┆ e_PCIAT     │
│ u32     ┆ f32       ┆ f32       ┆ f32       ┆   ┆ i64          ┆ i8
┆ i8      ┆ ---         │
│         ┆           ┆           ┆           ┆   ┆              ┆
┆         ┆ f32         │
╞═════════╪═══════════╪═══════════╪═══════════╪═══╪══════════════╪═════
════╪═════════╪═════════════╡
│ 0       ┆ 0.014375  ┆ -0.020112 ┆ -0.995358 ┆ … ┆ 441000000000 ┆ 2
┆ 2       ┆ 5.0         │
│         ┆           ┆           ┆           ┆   ┆ 00           ┆
┆         ┆             │
│ 1       ┆ 0.014167  ┆ -0.023278 ┆ -0.996164 ┆ … ┆ 441050000000 ┆ 2
┆ 2       ┆ 5.0         │
│         ┆           ┆           ┆           ┆   ┆ 00           ┆
```

```
¦              ¦                        ┃
│ 2       ¦ 0.014036  ¦ -0.022964 ¦ -0.99632  ¦ … ¦ 441100000000 ¦ 2
¦ 2        ¦ 5.0        ┃
│         ¦           ¦           ¦           ¦   ¦ 00           ¦
¦              ¦                        ┃
│ 3       ¦ 0.013593  ¦ -0.022048 ¦ -0.996762 ¦ … ¦ 441150000000 ¦ 2
¦ 2        ¦ 5.0        ┃
│         ¦           ¦           ¦           ¦   ¦ 00           ¦
¦              ¦                        ┃
│ 4       ¦ -0.061772 ¦ -0.065317 ¦ -0.973063 ¦ … ¦ 447800000000 ¦ 2
¦ 2        ¦ 5.0        ┃
│         ¦           ¦           ¦           ¦   ¦ 00           ¦
¦              ¦                        ┃
│ …       ¦ …         ¦ …         ¦ …         ¦ … ¦ …            ¦ …
¦ …        ¦ …          ┃
│ 287174 ¦ -0.407433 ¦ 0.091612  ¦ -0.377763 ¦ … ¦ 328750000000 ¦ 1
¦ 3        ¦ 53.0       ┃
│         ¦           ¦           ¦           ¦   ¦ 00           ¦
¦              ¦                        ┃
│ 287175 ¦ -0.703572 ¦ 0.016187  ¦ 0.15956   ¦ … ¦ 328800000000 ¦ 1
¦ 3        ¦ 53.0       ┃
│         ¦           ¦           ¦           ¦   ¦ 00           ¦
¦              ¦                        ┃
│ 287176 ¦ -0.209607 ¦ -0.4697   ¦ 0.636573  ¦ … ¦ 328850000000 ¦ 1
¦ 3        ¦ 53.0       ┃
│         ¦           ¦           ¦           ¦   ¦ 00           ¦
¦              ¦                        ┃
│ 287177 ¦ -0.390378 ¦ 0.284386  ¦ 0.147654  ¦ … ¦ 328900000000 ¦ 1
¦ 3        ¦ 53.0       ┃
│         ¦           ¦           ¦           ¦   ¦ 00           ¦
¦              ¦                        ┃
│ 287178 ¦ -0.48903  ¦ 0.179624  ¦ -0.509611 ¦ … ¦ 328950000000 ¦ 1
¦ 3        ¦ 53.0       ┃
│         ¦           ¦           ¦           ¦   ¦ 00           ¦
¦              ¦                        ┃
└─────────┴───────────┴───────────┴───────────┴───┴──────────────┴─────
────┴──────────┴────────────────┘
```

```python
def analyze_actigraphy(id, only_one_week=False, small=False):
    actigraphy = pl.read_parquet(f'/kaggle/input/child-mind-institute-problematic-internet-use/series_train.parquet/id={id}/part-0.parquet')
    day = actigraphy.get_column('relative_date_PCIAT') + actigraphy.get_column('time_of_day') / 86400e9
    sample = train.filter(pl.col('id') == id)
    age = sample.get_column('Basic_Demos-Age').item()
    sex = ['boy', 'girl'][sample.get_column('Basic_Demos-Sex').item()]
    actigraphy = (
        actigraphy
        .with_columns(
            (day.diff() * 86400).alias('diff_seconds'),
```

```
            (np.sqrt(np.square(pl.col('X')) + np.square(pl.col('Y')) +
np.square(pl.col('Z'))).alias('norm'))
        )
    )

    if only_one_week:
        start = np.ceil(day.min())
        mask = (start <= day.to_numpy()) & (day.to_numpy() <= start +
7*3)
        mask &= ~ actigraphy.get_column('non-
wear_flag').cast(bool).to_numpy()
    else:
        mask = np.full(len(day), True)

    if small:
        timelines = [
            ('enmo', 'forestgreen'),
            ('light', 'orange'),
        ]
    else:
        timelines = [
            ('X', 'm'),
            ('Y', 'm'),
            ('Z', 'm'),
#             ('norm', 'c'),
            ('enmo', 'forestgreen'),
            ('anglez', 'lightblue'),
            ('light', 'orange'),
            ('non-wear_flag', 'chocolate')
    #         ('diff_seconds', 'k'),
        ]

    _, axs = plt.subplots(len(timelines), 1, sharex=True, figsize=(12,
len(timelines) * 1.1 + 0.5))
    for ax, (feature, color) in zip(axs, timelines):
        ax.set_facecolor('#eeeeee')
        ax.scatter(day.to_numpy()[mask],
                   actigraphy.get_column(feature).to_numpy()[mask],
                   color=color, label=feature, s=1)
        ax.legend(loc='upper left', facecolor='#eeeeee')
        if feature == 'diff_seconds':
            ax.set_ylim(-0.5, 20.5)
    axs[-1].set_xlabel('day')
    axs[-1].xaxis.set_major_locator(MaxNLocator(integer=True))
    plt.tight_layout()
    axs[0].set_title(f'id={id}, {sex}, age={age}')
    plt.show()

analyze_actigraphy('0417c91e', only_one_week=False)
```




```
SEED = 42
n_splits = 5
```

Feature Engineering

- **Feature Selection**: The dataset contains features related to physical characteristics (e.g., BMI, Height, Weight), behavioral aspects (e.g., internet usage), and fitness data (e.g., endurance time).
- **Categorical Feature Encoding**: Categorical features are mapped to numerical values using custom mappings for each unique category within the dataset. This ensures compatibility with machine learning algorithms that require numerical input.
- **Time Series Aggregation**: Time series statistics (e.g., mean, standard deviation) from the actigraphy data are computed and merged into the main dataset to create additional features for model training.

```
def process_file(filename, dirname):
    df = pd.read_parquet(os.path.join(dirname, filename, 'part-
0.parquet'))
    df.drop('step', axis=1, inplace=True)
    return df.describe().values.reshape(-1), filename.split('=')[1]

def load_time_series(dirname) -> pd.DataFrame:
```

```
    ids = os.listdir(dirname)

    with ThreadPoolExecutor() as executor:
        results = list(tqdm(executor.map(lambda fname:
process_file(fname, dirname), ids), total=len(ids)))

    stats, indexes = zip(*results)

    df = pd.DataFrame(stats, columns=[f"stat_{i}" for i in
range(len(stats[0]))])
    df['id'] = indexes
    return df


class AutoEncoder(nn.Module):
    def __init__(self, input_dim, encoding_dim):
        super(AutoEncoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, encoding_dim*3),
            nn.ReLU(),
            nn.Linear(encoding_dim*3, encoding_dim*2),
            nn.ReLU(),
            nn.Linear(encoding_dim*2, encoding_dim),
            nn.ReLU()
        )
        self.decoder = nn.Sequential(
            nn.Linear(encoding_dim, input_dim*2),
            nn.ReLU(),
            nn.Linear(input_dim*2, input_dim*3),
            nn.ReLU(),
            nn.Linear(input_dim*3, input_dim),
            nn.Sigmoid()
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded


def perform_autoencoder(df, encoding_dim=50, epochs=50,
batch_size=32):
    scaler = StandardScaler()
    df_scaled = scaler.fit_transform(df)

    data_tensor = torch.FloatTensor(df_scaled)

    input_dim = data_tensor.shape[1]
    autoencoder = AutoEncoder(input_dim, encoding_dim)
```

```
    criterion = nn.MSELoss()
    optimizer = optim.Adam(autoencoder.parameters())

    for epoch in range(epochs):
        for i in range(0, len(data_tensor), batch_size):
            batch = data_tensor[i : i + batch_size]
            optimizer.zero_grad()
            reconstructed = autoencoder(batch)
            loss = criterion(reconstructed, batch)
            loss.backward()
            optimizer.step()

        if (epoch + 1) % 10 == 0:
            print(f'Epoch [{epoch + 1}/{epochs}], Loss:
{loss.item():.4f}]')

    with torch.no_grad():
        encoded_data = autoencoder.encoder(data_tensor).numpy()

    df_encoded = pd.DataFrame(encoded_data, columns=[f'Enc_{i + 1}'
for i in range(encoded_data.shape[1])])

    return df_encoded

def feature_engineering(df):
    season_cols = [col for col in df.columns if 'Season' in col]
    df = df.drop(season_cols, axis=1)
    df['BMI_Age'] = df['Physical-BMI'] * df['Basic_Demos-Age']
    df['Internet_Hours_Age'] = df['PreInt_EduHx-
computerinternet_hoursday'] * df['Basic_Demos-Age']
    df['BMI_Internet_Hours'] = df['Physical-BMI'] * df['PreInt_EduHx-
computerinternet_hoursday']
    df['BFP_BMI'] = df['BIA-BIA_Fat'] / df['BIA-BIA_BMI']
    df['FFMI_BFP'] = df['BIA-BIA_FFMI'] / df['BIA-BIA_Fat']
    df['FMI_BFP'] = df['BIA-BIA_FMI'] / df['BIA-BIA_Fat']
    df['LST_TBW'] = df['BIA-BIA_LST'] / df['BIA-BIA_TBW']
    df['BFP_BMR'] = df['BIA-BIA_Fat'] * df['BIA-BIA_BMR']
    df['BFP_DEE'] = df['BIA-BIA_Fat'] * df['BIA-BIA_DEE']
    df['BMR_Weight'] = df['BIA-BIA_BMR'] / df['Physical-Weight']
    df['DEE_Weight'] = df['BIA-BIA_DEE'] / df['Physical-Weight']
    df['SMM_Height'] = df['BIA-BIA_SMM'] / df['Physical-Height']
    df['Muscle_to_Fat'] = df['BIA-BIA_SMM'] / df['BIA-BIA_FMI']
    df['Hydration_Status'] = df['BIA-BIA_TBW'] / df['Physical-Weight']
    df['ICW_TBW'] = df['BIA-BIA_ICW'] / df['BIA-BIA_TBW']

    return df

train = pd.read_csv('/kaggle/input/child-mind-institute-problematic-
internet-use/train.csv')
test = pd.read_csv('/kaggle/input/child-mind-institute-problematic-
```

```
internet-use/test.csv')
sample = pd.read_csv('/kaggle/input/child-mind-institute-problematic-
internet-use/sample_submission.csv')

train_ts = load_time_series("/kaggle/input/child-mind-institute-
problematic-internet-use/series_train.parquet")
test_ts = load_time_series("/kaggle/input/child-mind-institute-
problematic-internet-use/series_test.parquet")

df_train = train_ts.drop('id', axis=1)
df_test = test_ts.drop('id', axis=1)

train_ts_encoded = perform_autoencoder(df_train, encoding_dim=60,
epochs=100, batch_size=32)
test_ts_encoded = perform_autoencoder(df_test, encoding_dim=60,
epochs=100, batch_size=32)

time_series_cols = train_ts_encoded.columns.tolist()
train_ts_encoded["id"]=train_ts["id"]
test_ts_encoded['id']=test_ts["id"]

train = pd.merge(train, train_ts_encoded, how="left", on='id')
test = pd.merge(test, test_ts_encoded, how="left", on='id')

imputer = KNNImputer(n_neighbors=5)
numeric_cols = train.select_dtypes(include=['float64',
'int64']).columns
imputed_data = imputer.fit_transform(train[numeric_cols])
train_imputed = pd.DataFrame(imputed_data, columns=numeric_cols)
train_imputed['sii'] = train_imputed['sii'].round().astype(int)
for col in train.columns:
    if col not in numeric_cols:
        train_imputed[col] = train[col]

train = train_imputed

train = feature_engineering(train)
train = train.dropna(thresh=10, axis=0)
test = feature_engineering(test)

train = train.drop('id', axis=1)
test  = test .drop('id', axis=1)


featuresCols = ['Basic_Demos-Age', 'Basic_Demos-Sex',
                'CGAS-CGAS_Score', 'Physical-BMI',
                'Physical-Height', 'Physical-Weight', 'Physical-
Waist_Circumference',
                'Physical-Diastolic_BP', 'Physical-HeartRate',
'Physical-Systolic_BP',
```

```
                'Fitness_Endurance-Max_Stage',
                'Fitness_Endurance-Time_Mins', 'Fitness_Endurance-
Time_Sec',
                'FGC-FGC_CU', 'FGC-FGC_CU_Zone', 'FGC-FGC_GSND',
                'FGC-FGC_GSND_Zone', 'FGC-FGC_GSD', 'FGC-
FGC_GSD_Zone', 'FGC-FGC_PU',
                'FGC-FGC_PU_Zone', 'FGC-FGC_SRL', 'FGC-FGC_SRL_Zone',
'FGC-FGC_SRR',
                'FGC-FGC_SRR_Zone', 'FGC-FGC_TL', 'FGC-FGC_TL_Zone',
                'BIA-BIA_Activity_Level_num', 'BIA-BIA_BMC', 'BIA-
BIA_BMI',
                'BIA-BIA_BMR', 'BIA-BIA_DEE', 'BIA-BIA_ECW', 'BIA-
BIA_FFM',
                'BIA-BIA_FFMI', 'BIA-BIA_FMI', 'BIA-BIA_Fat', 'BIA-
BIA_Frame_num',
                'BIA-BIA_ICW', 'BIA-BIA_LDM', 'BIA-BIA_LST', 'BIA-
BIA_SMM',
                'BIA-BIA_TBW', 'PAQ_A-PAQ_A_Total',
                'PAQ_C-PAQ_C_Total', 'SDS-SDS_Total_Raw',
                'SDS-SDS_Total_T',
                'PreInt_EduHx-computerinternet_hoursday', 'sii',
'BMI_Age','Internet_Hours_Age','BMI_Internet_Hours',
                'BFP_BMI', 'FFMI_BFP', 'FMI_BFP', 'LST_TBW',
'BFP_BMR', 'BFP_DEE', 'BMR_Weight', 'DEE_Weight',
                'SMM_Height', 'Muscle_to_Fat', 'Hydration_Status',
'ICW_TBW']

featuresCols += time_series_cols

train = train[featuresCols]
train = train.dropna(subset='sii')

featuresCols = ['Basic_Demos-Age', 'Basic_Demos-Sex',
                'CGAS-CGAS_Score', 'Physical-BMI',
                'Physical-Height', 'Physical-Weight', 'Physical-
Waist_Circumference',
                'Physical-Diastolic_BP', 'Physical-HeartRate',
'Physical-Systolic_BP',
                'Fitness_Endurance-Max_Stage',
                'Fitness_Endurance-Time_Mins', 'Fitness_Endurance-
Time_Sec',
                'FGC-FGC_CU', 'FGC-FGC_CU_Zone', 'FGC-FGC_GSND',
                'FGC-FGC_GSND_Zone', 'FGC-FGC_GSD', 'FGC-
FGC_GSD_Zone', 'FGC-FGC_PU',
                'FGC-FGC_PU_Zone', 'FGC-FGC_SRL', 'FGC-FGC_SRL_Zone',
'FGC-FGC_SRR',
                'FGC-FGC_SRR_Zone', 'FGC-FGC_TL', 'FGC-FGC_TL_Zone',
                'BIA-BIA_Activity_Level_num', 'BIA-BIA_BMC', 'BIA-
BIA_BMI',
                'BIA-BIA_BMR', 'BIA-BIA_DEE', 'BIA-BIA_ECW', 'BIA-
```

```
BIA_FFM',
                'BIA-BIA_FFMI', 'BIA-BIA_FMI', 'BIA-BIA_Fat', 'BIA-
BIA_Frame_num',
                'BIA-BIA_ICW', 'BIA-BIA_LDM', 'BIA-BIA_LST', 'BIA-
BIA_SMM',
                'BIA-BIA_TBW', 'PAQ_A-PAQ_A_Total',
                'PAQ_C-PAQ_C_Total', 'SDS-SDS_Total_Raw',
                'SDS-SDS_Total_T',
                'PreInt_EduHx-computerinternet_hoursday',
'BMI_Age','Internet_Hours_Age','BMI_Internet_Hours',
                'BFP_BMI', 'FFMI_BFP', 'FMI_BFP', 'LST_TBW',
'BFP_BMR', 'BFP_DEE', 'BMR_Weight', 'DEE_Weight',
                'SMM_Height', 'Muscle_to_Fat', 'Hydration_Status',
'ICW_TBW']

featuresCols += time_series_cols
test = test[featuresCols]
```

```
100%|██████████| 996/996 [01:23<00:00, 11.96it/s]
100%|██████████| 2/2 [00:00<00:00,  8.14it/s]
```

```
Epoch [10/100], Loss: 1.6593]
Epoch [20/100], Loss: 1.5514]
Epoch [30/100], Loss: 1.5332]
Epoch [40/100], Loss: 1.5184]
Epoch [50/100], Loss: 1.5153]
Epoch [60/100], Loss: 1.5180]
Epoch [70/100], Loss: 1.3878]
Epoch [80/100], Loss: 1.3803]
Epoch [90/100], Loss: 1.3777]
Epoch [100/100], Loss: 1.3771]
Epoch [10/100], Loss: 1.0089]
Epoch [20/100], Loss: 0.5724]
Epoch [30/100], Loss: 0.4271]
Epoch [40/100], Loss: 0.4271]
Epoch [50/100], Loss: 0.4271]
Epoch [60/100], Loss: 0.4271]
Epoch [70/100], Loss: 0.4271]
Epoch [80/100], Loss: 0.4271]
Epoch [90/100], Loss: 0.4271]
Epoch [100/100], Loss: 0.4271]
```

```
if np.any(np.isinf(train)):
    train = train.replace([np.inf, -np.inf], np.nan)

def quadratic_weighted_kappa(y_true, y_pred):
    return cohen_kappa_score(y_true, y_pred, weights='quadratic')

def threshold_Rounder(oof_non_rounded, thresholds):
    return np.where(oof_non_rounded < thresholds[0], 0,
```

```
                    np.where(oof_non_rounded < thresholds[1], 1,
                             np.where(oof_non_rounded < thresholds[2], 
2, 3)))

def evaluate_predictions(thresholds, y_true, oof_non_rounded):
    rounded_p = threshold_Rounder(oof_non_rounded, thresholds)
    return -quadratic_weighted_kappa(y_true, rounded_p)
```

Model Training and Evaluation

- **Model Types**: Various models are used, including:
  - **LightGBM**: A gradient-boosting framework known for its speed and efficiency with large datasets.
  - **XGBoost**: Another powerful gradient-boosting model used for structured data.
  - **CatBoost**: Optimized for categorical features without the need for extensive preprocessing.
  - **Voting Regressor**: An ensemble model that combines the predictions of LightGBM, XGBoost, and CatBoost for better accuracy.
- **Cross-Validation**: Stratified K-Folds cross-validation is employed to split the data into training and validation sets, ensuring balanced class distribution in each fold.
- **Quadratic Weighted Kappa (QWK)**: The performance of the models is evaluated using QWK, which measures the agreement between predicted and actual values, taking into account the ordinal nature of the target variable.
- **Threshold Optimization**: The `minimize` function from `scipy.optimize` is used to fine-tune decision thresholds that map continuous predictions to discrete categories (None, Mild, Moderate, Severe).

```
def TrainML(model_class, test_data):
    X = train.drop(['sii'], axis=1)
    y = train['sii']

    SKF = StratifiedKFold(n_splits=n_splits, shuffle=True, 
random_state=SEED)

    train_S = []
    test_S = []

    oof_non_rounded = np.zeros(len(y), dtype=float)
    oof_rounded = np.zeros(len(y), dtype=int)
    test_preds = np.zeros((len(test_data), n_splits))

    for fold, (train_idx, test_idx) in enumerate(tqdm(SKF.split(X, y), 
desc="Training Folds", total=n_splits)):
        X_train, X_val = X.iloc[train_idx], X.iloc[test_idx]
        y_train, y_val = y.iloc[train_idx], y.iloc[test_idx]

        model = clone(model_class)
```

```
        model.fit(X_train, y_train)

        y_train_pred = model.predict(X_train)
        y_val_pred = model.predict(X_val)

        oof_non_rounded[test_idx] = y_val_pred
        y_val_pred_rounded = y_val_pred.round(0).astype(int)
        oof_rounded[test_idx] = y_val_pred_rounded

        train_kappa = quadratic_weighted_kappa(y_train,
y_train_pred.round(0).astype(int))
        val_kappa = quadratic_weighted_kappa(y_val,
y_val_pred_rounded)

        train_S.append(train_kappa)
        test_S.append(val_kappa)

        test_preds[:, fold] = model.predict(test_data)

        print(f"Fold {fold+1} - Train QWK: {train_kappa:.4f},
Validation QWK: {val_kappa:.4f}")
        clear_output(wait=True)

    print(f"Mean Train QWK --> {np.mean(train_S):.4f}")
    print(f"Mean Validation QWK ---> {np.mean(test_S):.4f}")

    KappaOPtimizer = minimize(evaluate_predictions,
                              x0=[0.5, 1.5, 2.5], args=(y,
oof_non_rounded),
                              method='Nelder-Mead')
    assert KappaOPtimizer.success, "Optimization did not converge."

    oof_tuned = threshold_Rounder(oof_non_rounded, KappaOPtimizer.x)
    tKappa = quadratic_weighted_kappa(y, oof_tuned)

    print(f"----> || Optimized QWK SCORE :: {Fore.CYAN}{Style.BRIGHT}
{tKappa:.3f}{Style.RESET_ALL}")

    tpm = test_preds.mean(axis=1)
    tpTuned = threshold_Rounder(tpm, KappaOPtimizer.x)

    submission = pd.DataFrame({
        'id': sample['id'],
        'sii': tpTuned
    })

    return submission
```

Hyperparameter Tuning

- **LightGBM Parameters**: Hyperparameters such as `learning_rate`, `max_depth`, `num_leaves`, and `feature_fraction` are tuned to improve the performance of the LightGBM model. These parameters control the complexity of the model and its ability to generalize to new data.
- **XGBoost and CatBoost Parameters**: Similar tuning is applied for XGBoost and CatBoost, adjusting parameters such as `n_estimators`, `max_depth`, `learning_rate`, `subsample`, and `regularization` terms (`reg_alpha`, `reg_lambda`). These help in controlling overfitting and ensuring the model's robustness.

```
# Model parameters for LightGBM
Params = {
    'learning_rate': 0.046,
    'max_depth': 12,
    'num_leaves': 478,
    'min_data_in_leaf': 13,
    'feature_fraction': 0.893,
    'bagging_fraction': 0.784,
    'bagging_freq': 4,
    'lambda_l1': 10,  # Increased from 6.59
    'lambda_l2': 0.01,  # Increased from 2.68e-06
    'device': 'gpu'

}


# XGBoost parameters
XGB_Params = {
    'learning_rate': 0.05,
    'max_depth': 6,
    'n_estimators': 200,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'reg_alpha': 1,  # Increased from 0.1
    'reg_lambda': 5,  # Increased from 1
    'random_state': SEED,
    'tree_method': 'gpu_hist',

}


CatBoost_Params = {
    'learning_rate': 0.05,
    'depth': 6,
    'iterations': 200,
    'random_seed': SEED,
    'verbose': 0,
    'l2_leaf_reg': 10,  # Increase this value
    'task_type': 'GPU'
```

```
}

# New: TabNet

from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from pytorch_tabnet.callbacks import Callback
import os
import torch
from pytorch_tabnet.callbacks import Callback

class TabNetWrapper(BaseEstimator, RegressorMixin):
    def __init__(self, **kwargs):
        self.model = TabNetRegressor(**kwargs)
        self.kwargs = kwargs
        self.imputer = SimpleImputer(strategy='median')
        self.best_model_path = 'best_tabnet_model.pt'

    def fit(self, X, y):
        # Handle missing values
        X_imputed = self.imputer.fit_transform(X)

        if hasattr(y, 'values'):
            y = y.values

        # Create internal validation set
        X_train, X_valid, y_train, y_valid = train_test_split(
            X_imputed,
            y,
            test_size=0.2,
            random_state=42
        )

        # Train TabNet model
        history = self.model.fit(
            X_train=X_train,
            y_train=y_train.reshape(-1, 1),
            eval_set=[(X_valid, y_valid.reshape(-1, 1))],
            eval_name=['valid'],
            eval_metric=['mse'],
            max_epochs=500,
            patience=50,
            batch_size=1024,
            virtual_batch_size=128,
            num_workers=0,
            drop_last=False,
            callbacks=[
                TabNetPretrainedModelCheckpoint(
```

```
                    filepath=self.best_model_path,
                    monitor='valid_mse',
                    mode='min',
                    save_best_only=True,
                    verbose=True
                )
            ]
        )

        # Load the best model
        if os.path.exists(self.best_model_path):
            self.model.load_model(self.best_model_path)
            os.remove(self.best_model_path)  # Remove temporary file

        return self

    def predict(self, X):
        X_imputed = self.imputer.transform(X)
        return self.model.predict(X_imputed).flatten()

    def __deepcopy__(self, memo):
        # Add deepcopy support for scikit-learn
        cls = self.__class__
        result = cls.__new__(cls)
        memo[id(self)] = result
        for k, v in self.__dict__.items():
            setattr(result, k, deepcopy(v, memo))
        return result

# TabNet hyperparameters
TabNet_Params = {
    'n_d': 64,              # Width of the decision prediction layer
    'n_a': 64,              # Width of the attention embedding for
each step
    'n_steps': 5,           # Number of steps in the architecture
    'gamma': 1.5,           # Coefficient for feature selection
regularization
    'n_independent': 2,     # Number of independent GLU layer in each
GLU block
    'n_shared': 2,          # Number of shared GLU layer in each GLU
block
    'lambda_sparse': 1e-4,  # Sparsity regularization
    'optimizer_fn': torch.optim.Adam,
    'optimizer_params': dict(lr=2e-2, weight_decay=1e-5),
    'mask_type': 'entmax',
    'scheduler_params': dict(mode="min", patience=10, min_lr=1e-5,
factor=0.5),
    'scheduler_fn': torch.optim.lr_scheduler.ReduceLROnPlateau,
    'verbose': 1,
```

```
    'device_name': 'cuda' if torch.cuda.is_available() else 'cpu'
}

class TabNetPretrainedModelCheckpoint(Callback):
    def __init__(self, filepath, monitor='val_loss', mode='min',
                 save_best_only=True, verbose=1):
        super().__init__()  # Initialize parent class
        self.filepath = filepath
        self.monitor = monitor
        self.mode = mode
        self.save_best_only = save_best_only
        self.verbose = verbose
        self.best = float('inf') if mode == 'min' else -float('inf')

    def on_train_begin(self, logs=None):
        self.model = self.trainer  # Use trainer itself as model

    def on_epoch_end(self, epoch, logs=None):
        logs = logs or {}
        current = logs.get(self.monitor)
        if current is None:
            return

        # Check if current metric is better than best
        if (self.mode == 'min' and current < self.best) or \
           (self.mode == 'max' and current > self.best):
            if self.verbose:
                print(f'\nEpoch {epoch}: {self.monitor} improved from
{self.best:.4f} to {current:.4f}')
            self.best = current
            if self.save_best_only:
                self.model.save_model(self.filepath)  # Save the
entire model
```

Ensemble Learning and Submission Preparation

- **Ensemble Learning**: The model uses a **Voting Regressor**, which combines the predictions from LightGBM, XGBoost, and CatBoost. This approach is beneficial as it leverages the strengths of multiple models, reducing overfitting and improving overall model performance.
- **Out-of-Fold (OOF) Predictions**: During cross-validation, out-of-fold predictions are generated for the training set, which helps in model evaluation without data leakage.
- **Kappa Optimizer**: The Kappa Optimizer ensures that the predicted values are as close to the actual values as possible by adjusting the thresholds used to convert raw model outputs into class labels.
- **Test Set Predictions**: After the model is trained and thresholds are optimized, the test dataset is processed, and predictions are generated using the ensemble model. These predictions are converted into the appropriate format for submission.

- **Submission File Creation**: The predictions are saved in a CSV file following the required format for submission (e.g., for a Kaggle competition), which includes columns like `id` and `sii` (Severity Impairment Index).

Final Results and Performance Metrics

- **Train and Validation Scores**: After training across multiple folds, the mean Quadratic Weighted Kappa (QWK) score is calculated for both the training and validation datasets, providing an indicator of model performance.
- **Optimized QWK Score**: The final optimized QWK score after threshold tuning is displayed, showcasing the model's ability to predict the severity levels effectively.
- **Test Predictions**: The test set predictions are evaluated, and a breakdown of the predicted severity levels (None, Mild, Moderate, Severe) is shown, along with their respective counts.

```python
# Create model instances
Light = LGBMRegressor(**Params, random_state=SEED, verbose=-1,
n_estimators=300)
XGB_Model = XGBRegressor(**XGB_Params)
CatBoost_Model = CatBoostRegressor(**CatBoost_Params)
TabNet_Model = TabNetWrapper(**TabNet_Params) # New

voting_model = VotingRegressor(estimators=[
    ('lightgbm', Light),
    ('xgboost', XGB_Model),
    ('catboost', CatBoost_Model),
    ('tabnet', TabNet_Model)
])

Submission1 = TrainML(voting_model, test)

Submission1
```

```
Training Folds: 100%|██████████| 5/5 [03:05<00:00, 37.14s/it]

Mean Train QWK --> 0.7520
Mean Validation QWK ---> 0.4738
----> || Optimized QWK SCORE ::  0.524
```

	id	sii
0	00008ff9	0
1	000fd460	0
2	00105258	1
3	00115b9f	1
4	0016bb22	1
5	001f3379	1
6	0038ba98	1
7	0068a485	0

8	0069fbed	1
9	0083e397	1
10	0087dd65	1
11	00abe655	0
12	00ae59c9	1
13	00af6387	1
14	00bd4359	1
15	00c0cd71	1
16	00d56d4b	0
17	00d9913d	1
18	00e6167c	0
19	00ebc35d	1

```
train = pd.read_csv('/kaggle/input/child-mind-institute-problematic-
internet-use/train.csv')
test = pd.read_csv('/kaggle/input/child-mind-institute-problematic-
internet-use/test.csv')
sample = pd.read_csv('/kaggle/input/child-mind-institute-problematic-
internet-use/sample_submission.csv')

def process_file(filename, dirname):
    df = pd.read_parquet(os.path.join(dirname, filename, 'part-
0.parquet'))
    df.drop('step', axis=1, inplace=True)
    return df.describe().values.reshape(-1), filename.split('=')[1]

def load_time_series(dirname) -> pd.DataFrame:
    ids = os.listdir(dirname)

    with ThreadPoolExecutor() as executor:
        results = list(tqdm(executor.map(lambda fname:
process_file(fname, dirname), ids), total=len(ids)))

    stats, indexes = zip(*results)

    df = pd.DataFrame(stats, columns=[f"stat_{i}" for i in
range(len(stats[0]))])
    df['id'] = indexes
    return df

train_ts = load_time_series("/kaggle/input/child-mind-institute-
problematic-internet-use/series_train.parquet")
test_ts = load_time_series("/kaggle/input/child-mind-institute-
problematic-internet-use/series_test.parquet")

time_series_cols = train_ts.columns.tolist()
time_series_cols.remove("id")

train = pd.merge(train, train_ts, how="left", on='id')
test = pd.merge(test, test_ts, how="left", on='id')
```

```
train = train.drop('id', axis=1)
test = test.drop('id', axis=1)

featuresCols = ['Basic_Demos-Enroll_Season', 'Basic_Demos-Age',
'Basic_Demos-Sex',
                'CGAS-Season', 'CGAS-CGAS_Score', 'Physical-Season',
'Physical-BMI',
                'Physical-Height', 'Physical-Weight', 'Physical-
Waist_Circumference',
                'Physical-Diastolic_BP', 'Physical-HeartRate',
'Physical-Systolic_BP',
                'Fitness_Endurance-Season', 'Fitness_Endurance-
Max_Stage',
                'Fitness_Endurance-Time_Mins', 'Fitness_Endurance-
Time_Sec',
                'FGC-Season', 'FGC-FGC_CU', 'FGC-FGC_CU_Zone', 'FGC-
FGC_GSND',
                'FGC-FGC_GSND_Zone', 'FGC-FGC_GSD', 'FGC-
FGC_GSD_Zone', 'FGC-FGC_PU',
                'FGC-FGC_PU_Zone', 'FGC-FGC_SRL', 'FGC-FGC_SRL_Zone',
'FGC-FGC_SRR',
                'FGC-FGC_SRR_Zone', 'FGC-FGC_TL', 'FGC-FGC_TL_Zone',
'BIA-Season',
                'BIA-BIA_Activity_Level_num', 'BIA-BIA_BMC', 'BIA-
BIA_BMI',
                'BIA-BIA_BMR', 'BIA-BIA_DEE', 'BIA-BIA_ECW', 'BIA-
BIA_FFM',
                'BIA-BIA_FFMI', 'BIA-BIA_FMI', 'BIA-BIA_Fat', 'BIA-
BIA_Frame_num',
                'BIA-BIA_ICW', 'BIA-BIA_LDM', 'BIA-BIA_LST', 'BIA-
BIA_SMM',
                'BIA-BIA_TBW', 'PAQ_A-Season', 'PAQ_A-PAQ_A_Total',
'PAQ_C-Season',
                'PAQ_C-PAQ_C_Total', 'SDS-Season', 'SDS-
SDS_Total_Raw',
                'SDS-SDS_Total_T', 'PreInt_EduHx-Season',
                'PreInt_EduHx-computerinternet_hoursday', 'sii']

featuresCols += time_series_cols

train = train[featuresCols]
train = train.dropna(subset='sii')

cat_c = ['Basic_Demos-Enroll_Season', 'CGAS-Season', 'Physical-
Season',
          'Fitness_Endurance-Season', 'FGC-Season', 'BIA-Season',
          'PAQ_A-Season', 'PAQ_C-Season', 'SDS-Season', 'PreInt_EduHx-
Season']
```

```
def update(df):
    global cat_c
    for c in cat_c:
        df[c] = df[c].fillna('Missing')
        df[c] = df[c].astype('category')
    return df

train = update(train)
test = update(test)

def create_mapping(column, dataset):
    unique_values = dataset[column].unique()
    return {value: idx for idx, value in enumerate(unique_values)}

for col in cat_c:
    mapping = create_mapping(col, train)
    mappingTe = create_mapping(col, test)

    train[col] = train[col].replace(mapping).astype(int)
    test[col] = test[col].replace(mappingTe).astype(int)

def quadratic_weighted_kappa(y_true, y_pred):
    return cohen_kappa_score(y_true, y_pred, weights='quadratic')

def threshold_Rounder(oof_non_rounded, thresholds):
    return np.where(oof_non_rounded < thresholds[0], 0,
                    np.where(oof_non_rounded < thresholds[1], 1,
                             np.where(oof_non_rounded < thresholds[2],
2, 3)))

def evaluate_predictions(thresholds, y_true, oof_non_rounded):
    rounded_p = threshold_Rounder(oof_non_rounded, thresholds)
    return -quadratic_weighted_kappa(y_true, rounded_p)

def TrainML(model_class, test_data):
    X = train.drop(['sii'], axis=1)
    y = train['sii']

    SKF = StratifiedKFold(n_splits=n_splits, shuffle=True,
random_state=SEED)

    train_S = []
    test_S = []

    oof_non_rounded = np.zeros(len(y), dtype=float)
    oof_rounded = np.zeros(len(y), dtype=int)
    test_preds = np.zeros((len(test_data), n_splits))

    for fold, (train_idx, test_idx) in enumerate(tqdm(SKF.split(X, y),
desc="Training Folds", total=n_splits)):
```

```
        X_train, X_val = X.iloc[train_idx], X.iloc[test_idx]
        y_train, y_val = y.iloc[train_idx], y.iloc[test_idx]

        model = clone(model_class)
        model.fit(X_train, y_train)

        y_train_pred = model.predict(X_train)
        y_val_pred = model.predict(X_val)

        oof_non_rounded[test_idx] = y_val_pred
        y_val_pred_rounded = y_val_pred.round(0).astype(int)
        oof_rounded[test_idx] = y_val_pred_rounded

        train_kappa = quadratic_weighted_kappa(y_train,
y_train_pred.round(0).astype(int))
        val_kappa = quadratic_weighted_kappa(y_val,
y_val_pred_rounded)

        train_S.append(train_kappa)
        test_S.append(val_kappa)

        test_preds[:, fold] = model.predict(test_data)

        print(f"Fold {fold+1} - Train QWK: {train_kappa:.4f},
Validation QWK: {val_kappa:.4f}")
        clear_output(wait=True)

    print(f"Mean Train QWK --> {np.mean(train_S):.4f}")
    print(f"Mean Validation QWK ---> {np.mean(test_S):.4f}")

    KappaOPtimizer = minimize(evaluate_predictions,
                              x0=[0.5, 1.5, 2.5], args=(y,
oof_non_rounded),
                              method='Nelder-Mead')
    assert KappaOPtimizer.success, "Optimization did not converge."

    oof_tuned = threshold_Rounder(oof_non_rounded, KappaOPtimizer.x)
    tKappa = quadratic_weighted_kappa(y, oof_tuned)

    print(f"----> || Optimized QWK SCORE :: {Fore.CYAN}{Style.BRIGHT}
{tKappa:.3f}{Style.RESET_ALL}")

    tpm = test_preds.mean(axis=1)
    tpTuned = threshold_Rounder(tpm, KappaOPtimizer.x)

    submission = pd.DataFrame({
        'id': sample['id'],
        'sii': tpTuned
    })
```

```
    return submission

# Model parameters for LightGBM
Params = {
    'learning_rate': 0.046,
    'max_depth': 12,
    'num_leaves': 478,
    'min_data_in_leaf': 13,
    'feature_fraction': 0.893,
    'bagging_fraction': 0.784,
    'bagging_freq': 4,
    'lambda_l1': 10,  # Increased from 6.59
    'lambda_l2': 0.01  # Increased from 2.68e-06
}


# XGBoost parameters
XGB_Params = {
    'learning_rate': 0.05,
    'max_depth': 6,
    'n_estimators': 200,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'reg_alpha': 1,  # Increased from 0.1
    'reg_lambda': 5,  # Increased from 1
    'random_state': SEED
}


CatBoost_Params = {
    'learning_rate': 0.05,
    'depth': 6,
    'iterations': 200,
    'random_seed': SEED,
    'cat_features': cat_c,
    'verbose': 0,
    'l2_leaf_reg': 10  # Increase this value
}

# Create model instances
Light = LGBMRegressor(**Params, random_state=SEED, verbose=-1,
n_estimators=300)
XGB_Model = XGBRegressor(**XGB_Params)
CatBoost_Model = CatBoostRegressor(**CatBoost_Params)
TabNet_Model = TabNetWrapper(**TabNet_Params)  # New:TAbNet

# Combine models using Voting Regressor
voting_model = VotingRegressor(estimators=[
    ('lightgbm', Light),
    ('xgboost', XGB_Model),
```

```
    ('catboost', CatBoost_Model),
    ('tabnet', TabNet_Model)  # New:TabNet
])

# Train the ensemble model
Submission2 = TrainML(voting_model, test)

# Save submission
#Submission2.to_csv('submission.csv', index=False)
Submission2
```

```
Training Folds: 100%|██████████| 5/5 [02:40<00:00, 32.17s/it]

Mean Train QWK --> 0.6819
Mean Validation QWK ---> 0.3796



----> || Optimized QWK SCORE ::  0.460
```

	id	sii
0	00008ff9	1
1	000fd460	0
2	00105258	0
3	00115b9f	0
4	0016bb22	1
5	001f3379	1
6	0038ba98	0
7	0068a485	0
8	0069fbed	1
9	0083e397	1
10	0087dd65	0
11	00abe655	1
12	00ae59c9	1
13	00af6387	1
14	00bd4359	1
15	00c0cd71	1
16	00d56d4b	0
17	00d9913d	0
18	00e6167c	0
19	00ebc35d	1

```
train = pd.read_csv('/kaggle/input/child-mind-institute-problematic-internet-use/train.csv')
test = pd.read_csv('/kaggle/input/child-mind-institute-problematic-internet-use/test.csv')
sample = pd.read_csv('/kaggle/input/child-mind-institute-problematic-internet-use/sample_submission.csv')

featuresCols = ['Basic_Demos-Enroll_Season', 'Basic_Demos-Age',
'Basic_Demos-Sex',
```

```
                'CGAS-Season', 'CGAS-CGAS_Score', 'Physical-Season',
'Physical-BMI',
                'Physical-Height', 'Physical-Weight', 'Physical-
Waist_Circumference',
                'Physical-Diastolic_BP', 'Physical-HeartRate',
'Physical-Systolic_BP',
                'Fitness_Endurance-Season', 'Fitness_Endurance-
Max_Stage',
                'Fitness_Endurance-Time_Mins', 'Fitness_Endurance-
Time_Sec',
                'FGC-Season', 'FGC-FGC_CU', 'FGC-FGC_CU_Zone', 'FGC-
FGC_GSND',
                'FGC-FGC_GSND_Zone', 'FGC-FGC_GSD', 'FGC-
FGC_GSD_Zone', 'FGC-FGC_PU',
                'FGC-FGC_PU_Zone', 'FGC-FGC_SRL', 'FGC-FGC_SRL_Zone',
'FGC-FGC_SRR',
                'FGC-FGC_SRR_Zone', 'FGC-FGC_TL', 'FGC-FGC_TL_Zone',
'BIA-Season',
                'BIA-BIA_Activity_Level_num', 'BIA-BIA_BMC', 'BIA-
BIA_BMI',
                'BIA-BIA_BMR', 'BIA-BIA_DEE', 'BIA-BIA_ECW', 'BIA-
BIA_FFM',
                'BIA-BIA_FFMI', 'BIA-BIA_FMI', 'BIA-BIA_Fat', 'BIA-
BIA_Frame_num',
                'BIA-BIA_ICW', 'BIA-BIA_LDM', 'BIA-BIA_LST', 'BIA-
BIA_SMM',
                'BIA-BIA_TBW', 'PAQ_A-Season', 'PAQ_A-PAQ_A_Total',
'PAQ_C-Season',
                'PAQ_C-PAQ_C_Total', 'SDS-Season', 'SDS-
SDS_Total_Raw',
                'SDS-SDS_Total_T', 'PreInt_EduHx-Season',
                'PreInt_EduHx-computerinternet_hoursday', 'sii']

cat_c = ['Basic_Demos-Enroll_Season', 'CGAS-Season', 'Physical-
Season',
          'Fitness_Endurance-Season', 'FGC-Season', 'BIA-Season',
          'PAQ_A-Season', 'PAQ_C-Season', 'SDS-Season', 'PreInt_EduHx-
Season']

train_ts = load_time_series("/kaggle/input/child-mind-institute-
problematic-internet-use/series_train.parquet")
test_ts = load_time_series("/kaggle/input/child-mind-institute-
problematic-internet-use/series_test.parquet")

time_series_cols = train_ts.columns.tolist()
time_series_cols.remove("id")

train = pd.merge(train, train_ts, how="left", on='id')
test = pd.merge(test, test_ts, how="left", on='id')
```

```
train = train.drop('id', axis=1)
test = test.drop('id', axis=1)

featuresCols += time_series_cols

train = train[featuresCols]
train = train.dropna(subset='sii')

def update(df):
    global cat_c
    for c in cat_c:
        df[c] = df[c].fillna('Missing')
        df[c] = df[c].astype('category')
    return df

train = update(train)
test = update(test)

def create_mapping(column, dataset):
    unique_values = dataset[column].unique()
    return {value: idx for idx, value in enumerate(unique_values)}

for col in cat_c:
    mapping = create_mapping(col, train)
    mappingTe = create_mapping(col, test)

    train[col] = train[col].replace(mapping).astype(int)
    test[col] = test[col].replace(mappingTe).astype(int)

def quadratic_weighted_kappa(y_true, y_pred):
    return cohen_kappa_score(y_true, y_pred, weights='quadratic')

def threshold_Rounder(oof_non_rounded, thresholds):
    return np.where(oof_non_rounded < thresholds[0], 0,
                    np.where(oof_non_rounded < thresholds[1], 1,
                             np.where(oof_non_rounded < thresholds[2],
2, 3)))

def evaluate_predictions(thresholds, y_true, oof_non_rounded):
    rounded_p = threshold_Rounder(oof_non_rounded, thresholds)
    return -quadratic_weighted_kappa(y_true, rounded_p)

def TrainML(model_class, test_data):
    X = train.drop(['sii'], axis=1)
    y = train['sii']

    SKF = StratifiedKFold(n_splits=n_splits, shuffle=True,
random_state=SEED)

    train_S = []
```

```
    test_S = []

    oof_non_rounded = np.zeros(len(y), dtype=float)
    oof_rounded = np.zeros(len(y), dtype=int)
    test_preds = np.zeros((len(test_data), n_splits))

    for fold, (train_idx, test_idx) in enumerate(tqdm(SKF.split(X, y),
desc="Training Folds", total=n_splits)):
        X_train, X_val = X.iloc[train_idx], X.iloc[test_idx]
        y_train, y_val = y.iloc[train_idx], y.iloc[test_idx]

        model = clone(model_class)
        model.fit(X_train, y_train)

        y_train_pred = model.predict(X_train)
        y_val_pred = model.predict(X_val)

        oof_non_rounded[test_idx] = y_val_pred
        y_val_pred_rounded = y_val_pred.round(0).astype(int)
        oof_rounded[test_idx] = y_val_pred_rounded

        train_kappa = quadratic_weighted_kappa(y_train,
y_train_pred.round(0).astype(int))
        val_kappa = quadratic_weighted_kappa(y_val,
y_val_pred_rounded)

        train_S.append(train_kappa)
        test_S.append(val_kappa)

        test_preds[:, fold] = model.predict(test_data)

        print(f"Fold {fold+1} - Train QWK: {train_kappa:.4f},
Validation QWK: {val_kappa:.4f}")
        clear_output(wait=True)

    print(f"Mean Train QWK --> {np.mean(train_S):.4f}")
    print(f"Mean Validation QWK ---> {np.mean(test_S):.4f}")

    KappaOPtimizer = minimize(evaluate_predictions,
                              x0=[0.5, 1.5, 2.5], args=(y,
oof_non_rounded),
                              method='Nelder-Mead')
    assert KappaOPtimizer.success, "Optimization did not converge."

    oof_tuned = threshold_Rounder(oof_non_rounded, KappaOPtimizer.x)
    tKappa = quadratic_weighted_kappa(y, oof_tuned)

    print(f"----> || Optimized QWK SCORE :: {Fore.CYAN}{Style.BRIGHT}
{tKappa:.3f}{Style.RESET_ALL}")
```

```python
    tpm = test_preds.mean(axis=1)
    tp_rounded = threshold_Rounder(tpm, KappaOPtimizer.x)

    return tp_rounded

imputer = SimpleImputer(strategy='median')

ensemble = VotingRegressor(estimators=[
    ('lgb', Pipeline(steps=[('imputer', imputer), ('regressor', 
LGBMRegressor(random_state=SEED))])),
    ('xgb', Pipeline(steps=[('imputer', imputer), ('regressor', 
XGBRegressor(random_state=SEED))])),
    ('cat', Pipeline(steps=[('imputer', imputer), ('regressor', 
CatBoostRegressor(random_state=SEED, silent=True))])),
    ('rf', Pipeline(steps=[('imputer', imputer), ('regressor', 
RandomForestRegressor(random_state=SEED))])),
    ('gb', Pipeline(steps=[('imputer', imputer), ('regressor', 
GradientBoostingRegressor(random_state=SEED))])),
    ('tabnet', Pipeline(steps=[('imputer', imputer), ('regressor', 
TabNetWrapper(**TabNet_Params))]))  # New:TabNet
])

Submission3 = TrainML(ensemble, test)

Submission3 = TrainML(ensemble, test)
Submission3 = pd.DataFrame({
    'id': sample['id'],
    'sii': Submission3
})

Submission3
```

```
Training Folds: 100%|██████████| 5/5 [03:55<00:00, 47.11s/it]

Mean Train QWK --> 0.8544
Mean Validation QWK ---> 0.3708



----> || Optimized QWK SCORE ::  0.443
```

	id	sii
0	00008ff9	2
1	000fd460	0
2	00105258	0
3	00115b9f	0
4	0016bb22	1
5	001f3379	1
6	0038ba98	0
7	0068a485	0
8	0069fbed	2

```
9   0083e397    0
10  0087dd65    1
11  00abe655    0
12  00ae59c9    2
13  00af6387    1
14  00bd4359    2
15  00c0cd71    2
16  00d56d4b    0
17  00d9913d    0
18  00e6167c    0
19  00ebc35d    1
```

```
sub1 = Submission1
sub2 = Submission2
sub3 = Submission3

sub1 = sub1.sort_values(by='id').reset_index(drop=True)
sub2 = sub2.sort_values(by='id').reset_index(drop=True)
sub3 = sub3.sort_values(by='id').reset_index(drop=True)

combined = pd.DataFrame({
    'id': sub1['id'],
    'sii_1': sub1['sii'],
    'sii_2': sub2['sii'],
    'sii_3': sub3['sii']
})

def majority_vote(row):
    return row.mode()[0]

combined['final_sii'] = combined[['sii_1', 'sii_2',
'sii_3']].apply(majority_vote, axis=1)

final_submission = combined[['id',
'final_sii']].rename(columns={'final_sii': 'sii'})

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

print("Majority voting completed and saved to 'Final_Submission.csv'")
```

```
Majority voting completed and saved to 'Final_Submission.csv'
```

```
final_submission
```

	id	sii
0	00008ff9	0
1	000fd460	0
2	00105258	0
3	00115b9f	0
4	0016bb22	1
5	001f3379	1

```
6   0038ba98    0
7   0068a485    0
8   0069fbed    1
9   0083e397    1
10  0087dd65    1
11  00abe655    0
12  00ae59c9    1
13  00af6387    1
14  00bd4359    1
15  00c0cd71    1
16  00d56d4b    0
17  00d9913d    0
18  00e6167c    0
19  00ebc35d    1
```