MULTILINGUAL DEPRESSION DETECTION USING SPEECH-BASED PRE-TRAINED MODEL

EERIK SVEN PUUDIST, FEDOR STOMAKHIN
RIO SCHULTS, ZUZANNA MAKOWSKA

EXPLAINABLE AUTOMATED MACHINE LEARNING - AUTUMN 2023

CONTENT

01 PROBLEM DESCRIPTION

02 DATASETS

03 FEATURE EXTRACTION

04 AUTOML - HYPEROPT

PROJECT DESCRIPTION

MULTILINGUAL DEPRESSION DETECTION USING SPEECH-BASED PRE-TRAINED MODEL

DATASETS

ANDROID

reading

interview

Spanish

5 seconds

silence removed

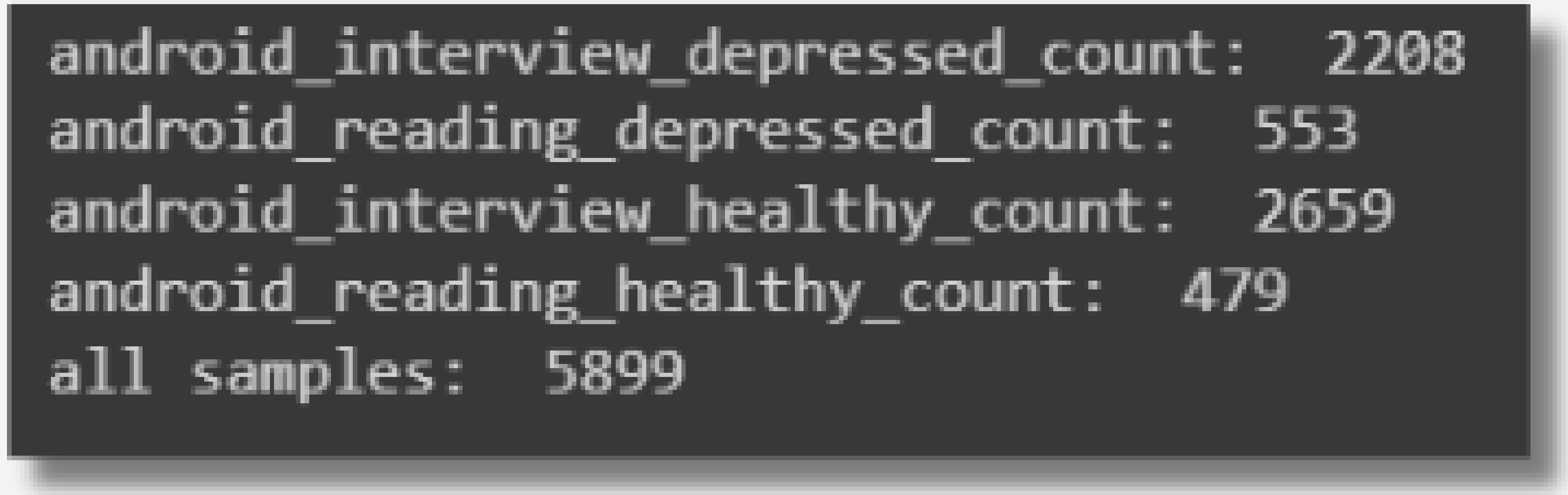
labeled

EDAIC

test, train, validation

English

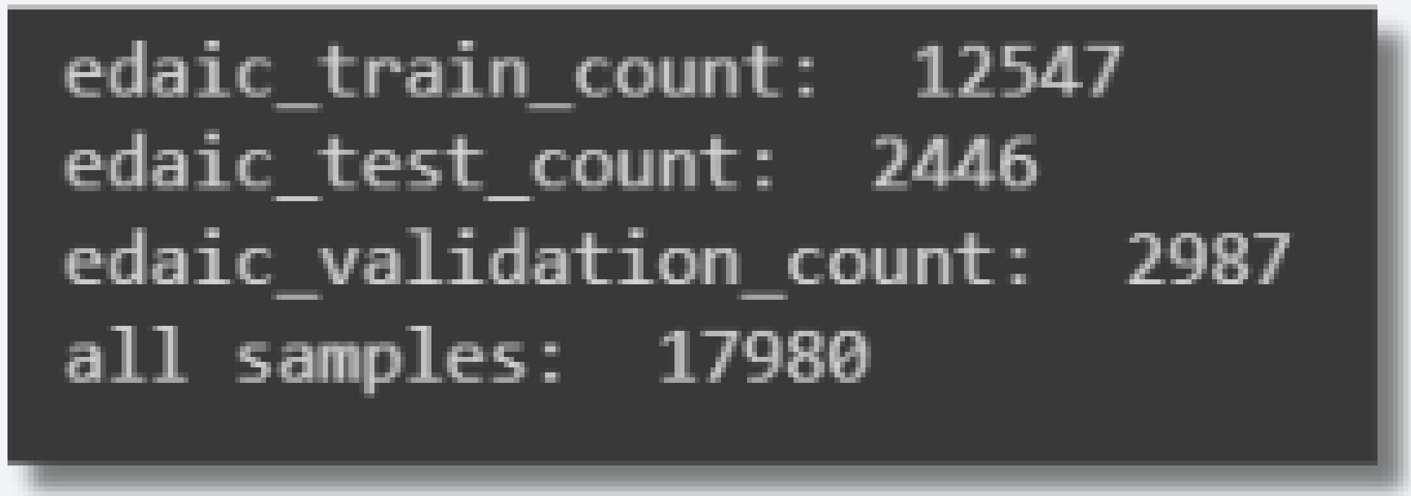
ANDROID



```
android_interview_depressed_count:  2208
android_reading_depressed_count:  553
android_interview_healthy_count:  2659
android_reading_healthy_count:  479
all samples:  5899
```

https://drive.google.com/uc?id=1OQ87c6vEKkTuLu2Z3jYz0P6-pvCytojz

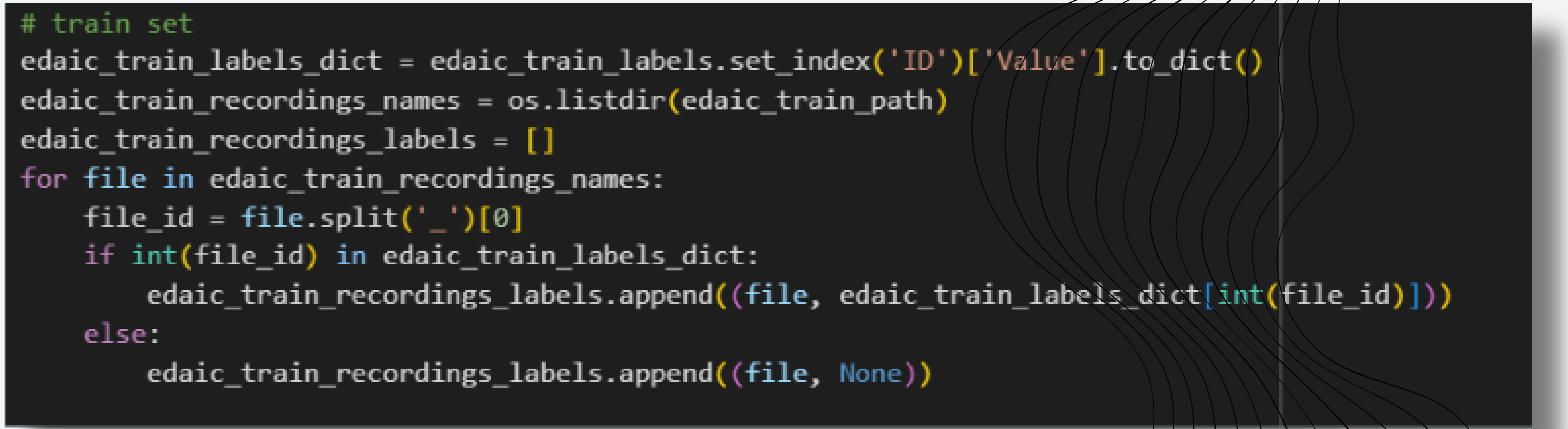
EDAIC



```
edaic_train_count:  12547
edaic_test_count:  2446
edaic_validation_count:  2987
all samples:  17980
```

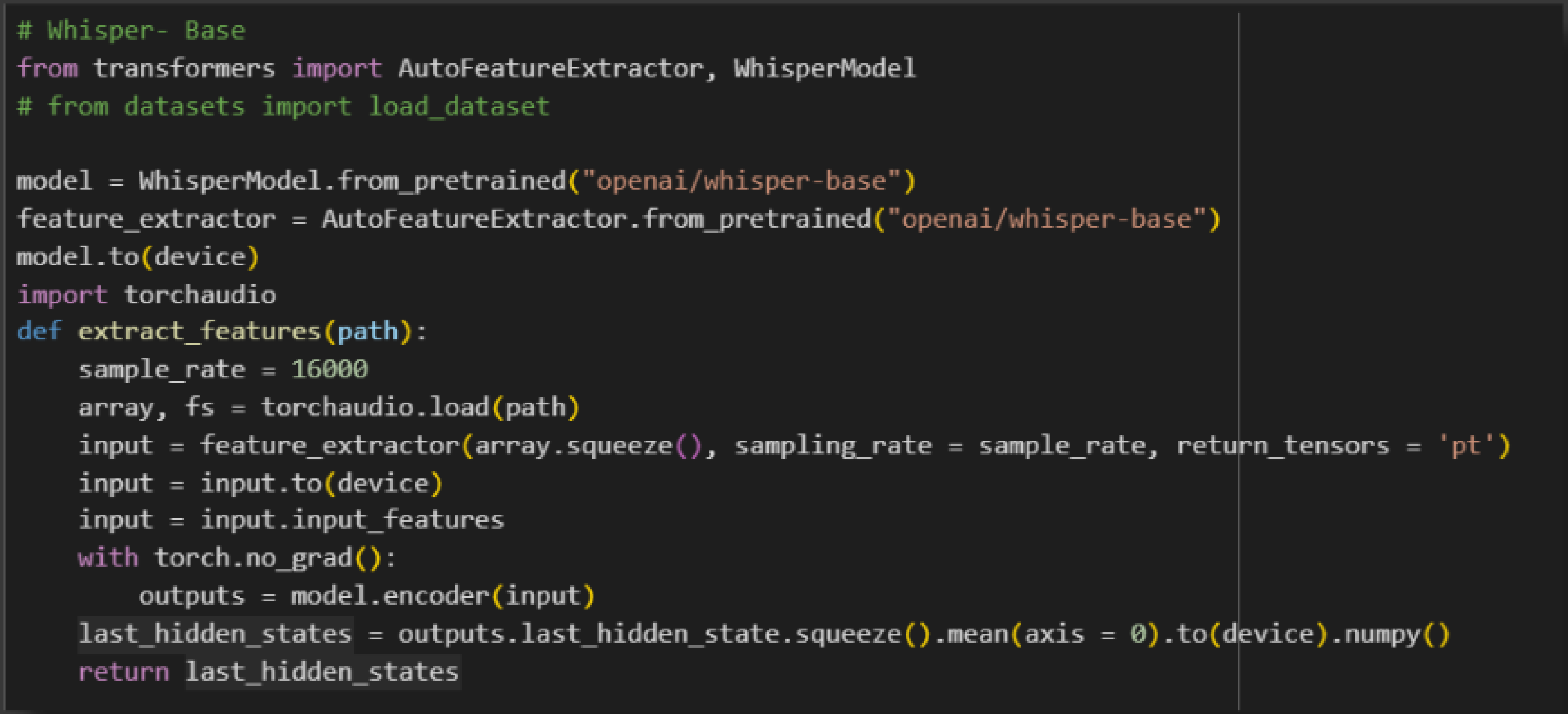
https://drive.google.com/uc?id=1PT9Iij7DJOB1s4iOT4gT3jZxpoZqSpzU

EDAIC - LABELS

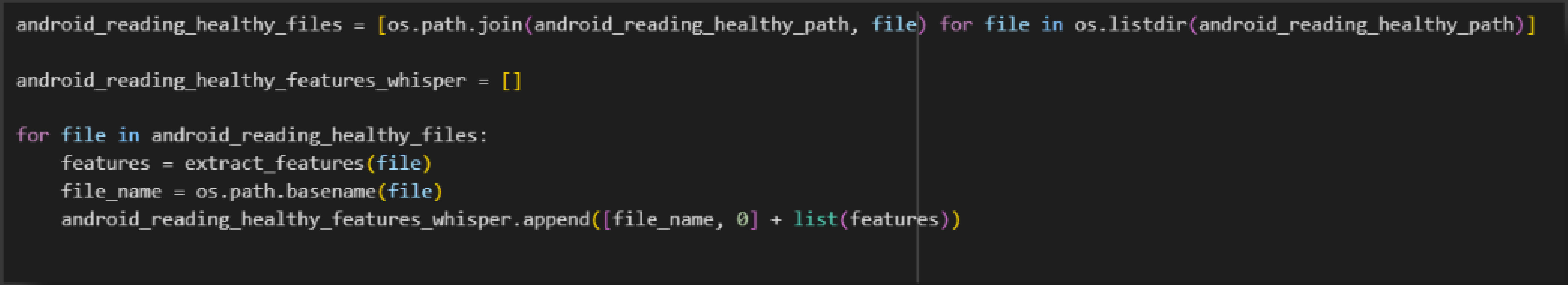


```python
# train set
edaic_train_labels_dict = edaic_train_labels.set_index('ID')['Value'].to_dict()
edaic_train_recordings_names = os.listdir(edaic_train_path)
edaic_train_recordings_labels = []
for file in edaic_train_recordings_names:
    file_id = file.split('_')[0]
    if int(file_id) in edaic_train_labels_dict:
        edaic_train_recordings_labels.append((file, edaic_train_labels_dict[int(file_id)]))
    else:
        edaic_train_recordings_labels.append((file, None))
```

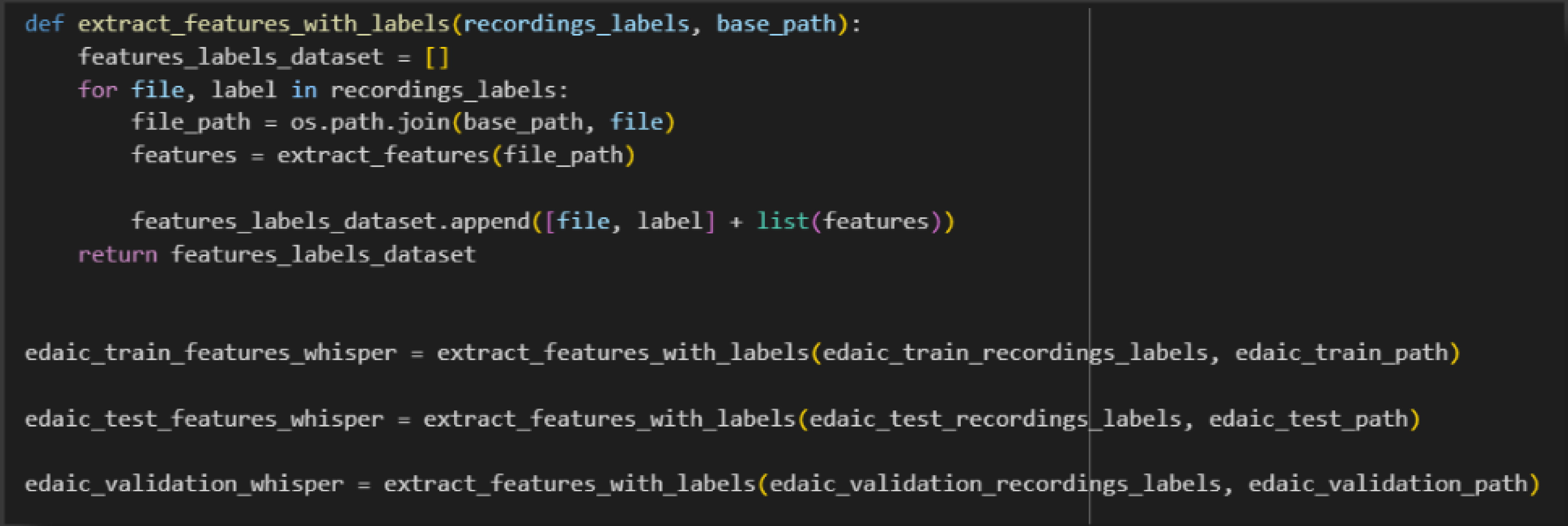
WHISPER



```python
# Whisper- Base
from transformers import AutoFeatureExtractor, WhisperModel
# from datasets import load_dataset

model = WhisperModel.from_pretrained("openai/whisper-base")
feature_extractor = AutoFeatureExtractor.from_pretrained("openai/whisper-base")
model.to(device)
import torchaudio
def extract_features(path):
    sample_rate = 16000
    array, fs = torchaudio.load(path)
    input = feature_extractor(array.squeeze(), sampling_rate = sample_rate, return_tensors = 'pt')
    input = input.to(device)
    input = input.input_features
    with torch.no_grad():
        outputs = model.encoder(input)
    last_hidden_states = outputs.last_hidden_state.squeeze().mean(axis = 0).to(device).numpy()
    return last_hidden_states
```



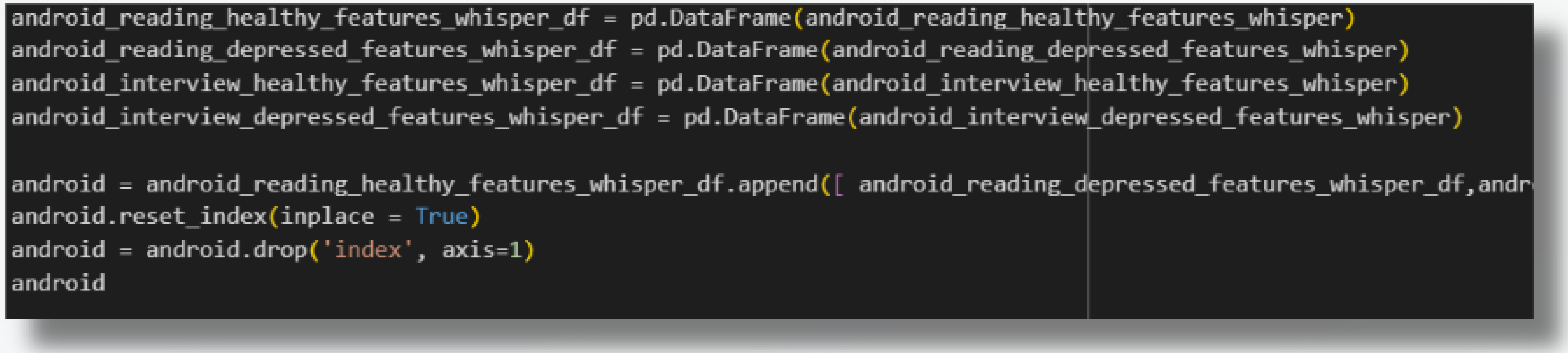
```
android_reading_healthy_files = [os.path.join(android_reading_healthy_path, file) for file in os.listdir(android_reading_healthy_path)]

android_reading_healthy_features_whisper = []

for file in android_reading_healthy_files:
    features = extract_features(file)
    file_name = os.path.basename(file)
    android_reading_healthy_features_whisper.append([file_name, 0] + list(features))
```



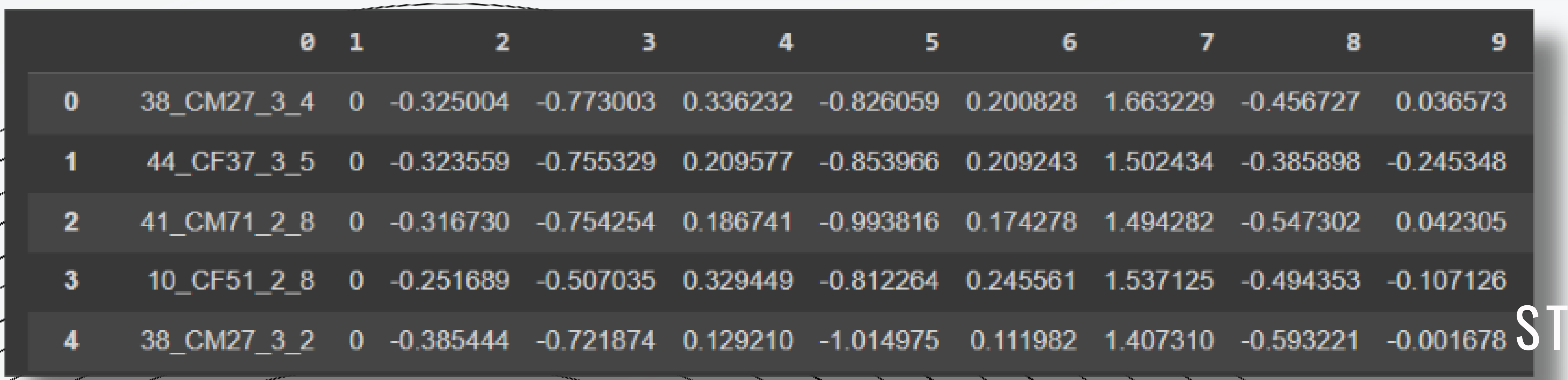
```python
def extract_features_with_labels(recordings_labels, base_path):
    features_labels_dataset = []
    for file, label in recordings_labels:
        file_path = os.path.join(base_path, file)
        features = extract_features(file_path)

        features_labels_dataset.append([file, label] + list(features))
    return features_labels_dataset


edaic_train_features_whisper = extract_features_with_labels(edaic_train_recordings_labels, edaic_train_path)

edaic_test_features_whisper = extract_features_with_labels(edaic_test_recordings_labels, edaic_test_path)

edaic_validation_whisper = extract_features_with_labels(edaic_validation_recordings_labels, edaic_validation_path)
```

BASELINE

ANDROID - DATA PREPARATION



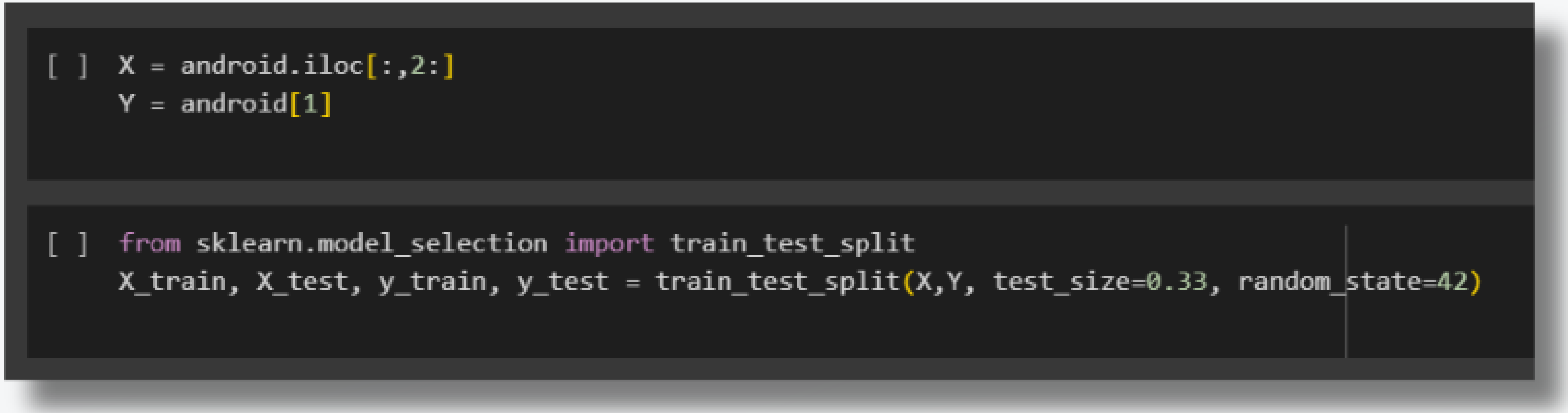
```
android_reading_healthy_features_whisper_df = pd.DataFrame(android_reading_healthy_features_whisper)
android_reading_depressed_features_whisper_df = pd.DataFrame(android_reading_depressed_features_whisper)
android_interview_healthy_features_whisper_df = pd.DataFrame(android_interview_healthy_features_whisper)
android_interview_depressed_features_whisper_df = pd.DataFrame(android_interview_depressed_features_whisper)

android = android_reading_healthy_features_whisper_df.append([ android_reading_depressed_features_whisper_df,andr
android.reset_index(inplace = True)
android = android.drop('index', axis=1)
android
```



	0	1	2	3	4	5	6	7	8	9
0	38_CM27_3_4	0	-0.325004	-0.773003	0.336232	-0.826059	0.200828	1.663229	-0.456727	0.036573
1	44_CF37_3_5	0	-0.323559	-0.755329	0.209577	-0.853966	0.209243	1.502434	-0.385898	-0.245348
2	41_CM71_2_8	0	-0.316730	-0.754254	0.186741	-0.993816	0.174278	1.494282	-0.547302	0.042305
3	10_CF51_2_8	0	-0.251689	-0.507035	0.329449	-0.812264	0.245561	1.537125	-0.494353	-0.107126
4	38_CM27_3_2	0	-0.385444	-0.721874	0.129210	-1.014975	0.111982	1.407310	-0.593221	-0.001678

STRATEGY N°3

ANDROID - SPLITTING THE DATA

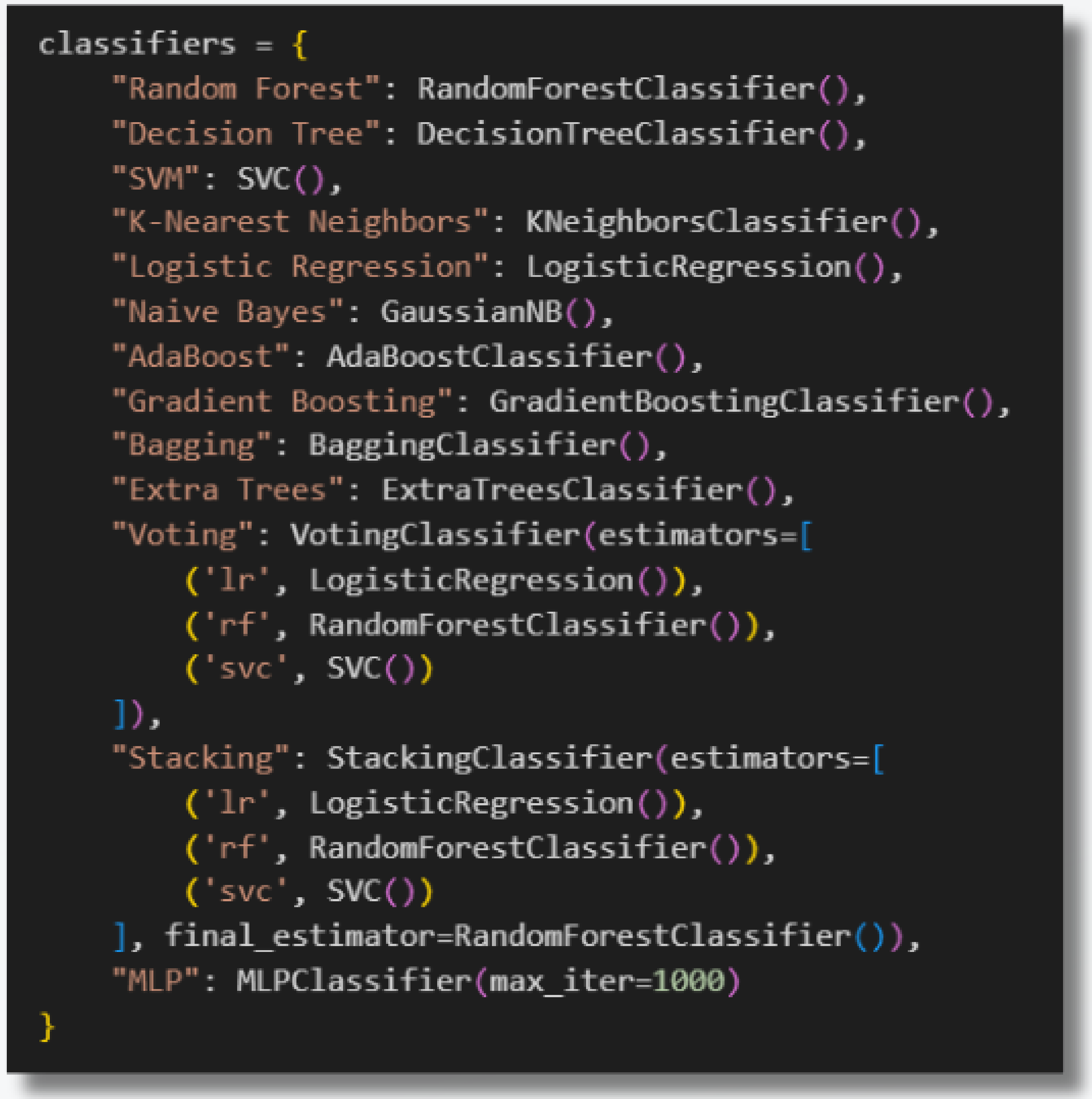


```
X = android.iloc[:,2:]
Y = android[1]
```

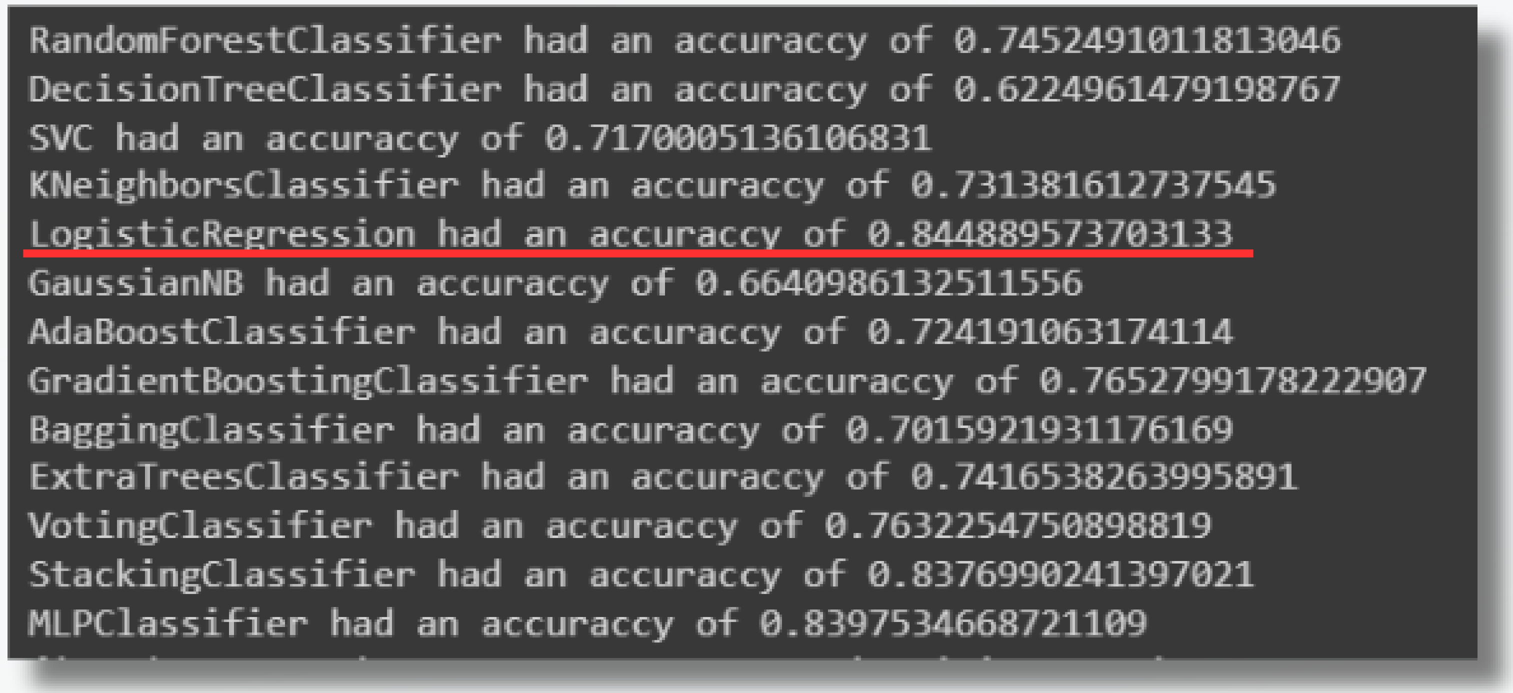
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.33, random_state=42)
```

STRATEGY N°3

ANDROID - BASELINE

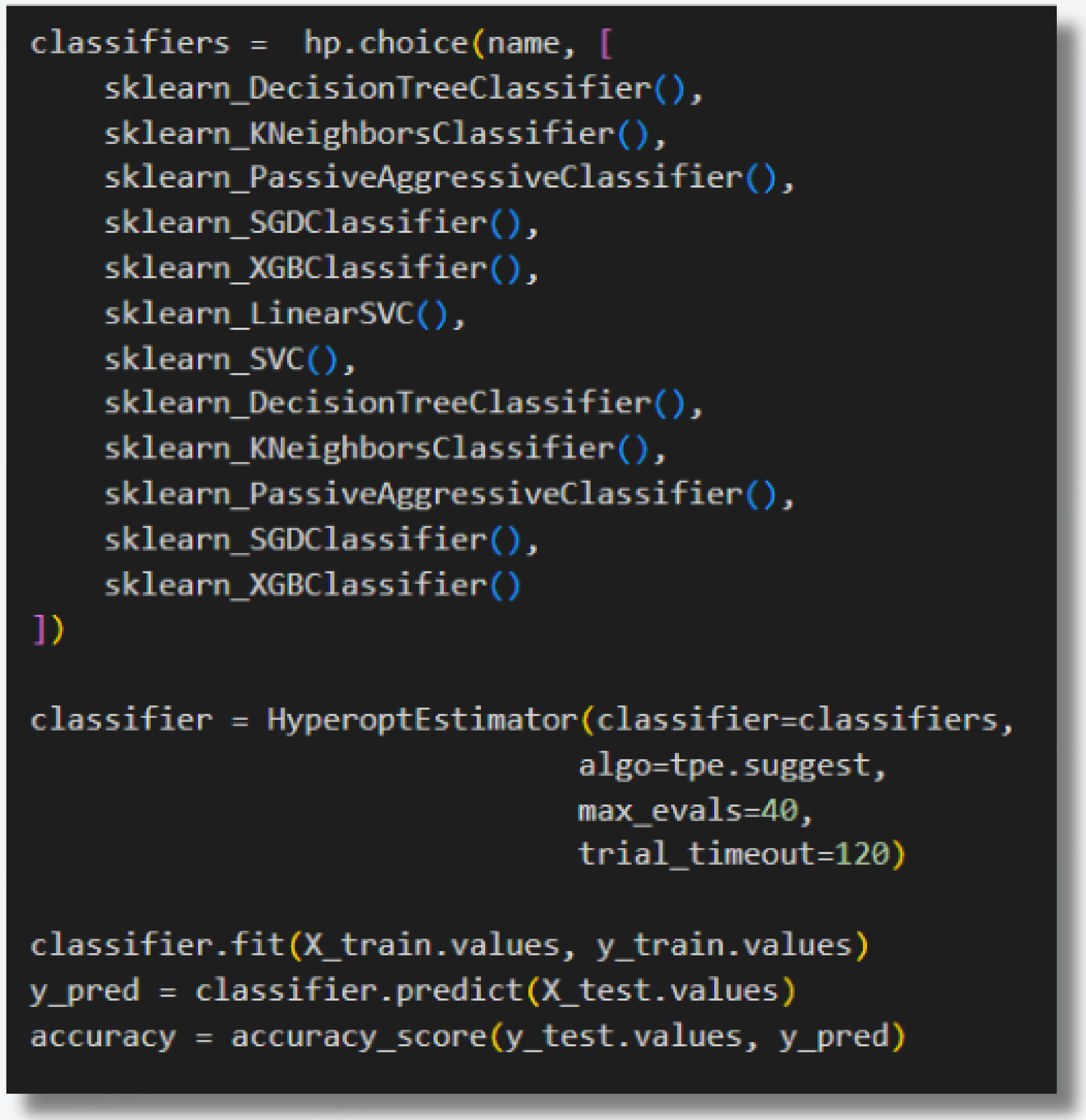


```
classifiers = {
    "Random Forest": RandomForestClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "SVM": SVC(),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Logistic Regression": LogisticRegression(),
    "Naive Bayes": GaussianNB(),
    "AdaBoost": AdaBoostClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "Bagging": BaggingClassifier(),
    "Extra Trees": ExtraTreesClassifier(),
    "Voting": VotingClassifier(estimators=[
        ('lr', LogisticRegression()),
        ('rf', RandomForestClassifier()),
        ('svc', SVC())
    ]),
    "Stacking": StackingClassifier(estimators=[
        ('lr', LogisticRegression()),
        ('rf', RandomForestClassifier()),
        ('svc', SVC())
    ], final_estimator=RandomForestClassifier()),
    "MLP": MLPClassifier(max_iter=1000)
}
```



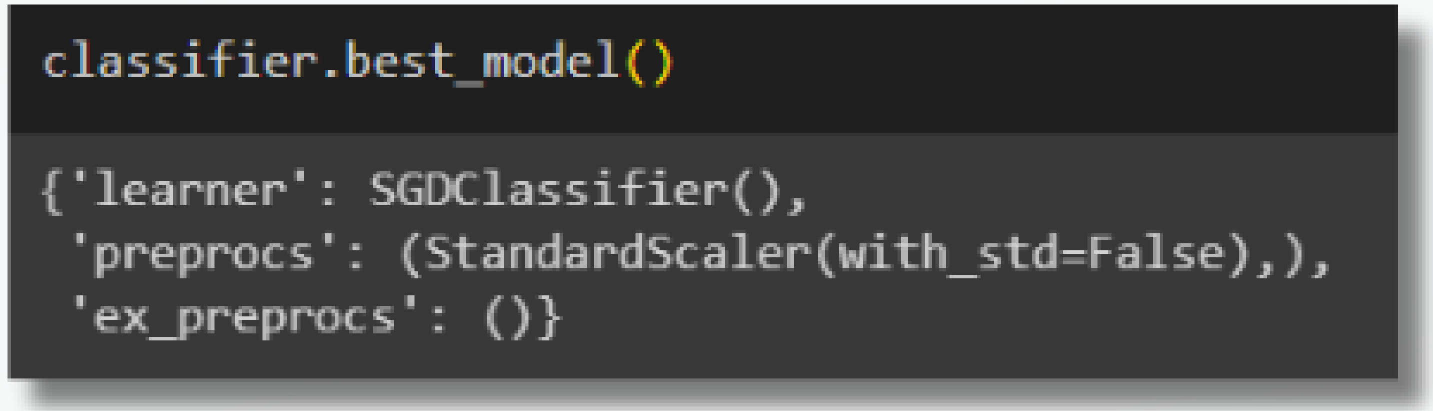
```
RandomForestClassifier had an accuraccy of 0.7452491011813046
DecisionTreeClassifier had an accuraccy of 0.6224961479198767
SVC had an accuraccy of 0.7170005136106831
KNeighborsClassifier had an accuraccy of 0.731381612737545
LogisticRegression had an accuraccy of 0.844889573703133
GaussianNB had an accuraccy of 0.6640986132511556
AdaBoostClassifier had an accuraccy of 0.724191063174114
GradientBoostingClassifier had an accuraccy of 0.7652799178222907
BaggingClassifier had an accuraccy of 0.7015921931176169
ExtraTreesClassifier had an accuraccy of 0.7416538263995891
VotingClassifier had an accuraccy of 0.7632254750898819
StackingClassifier had an accuraccy of 0.8376990241397021
MLPClassifier had an accuraccy of 0.8397534668721109
```

ANDROID - HYPEROPT



```
classifiers =  hp.choice(name, [
    sklearn_DecisionTreeClassifier(),
    sklearn_KNeighborsClassifier(),
    sklearn_PassiveAggressiveClassifier(),
    sklearn_SGDClassifier(),
    sklearn_XGBClassifier(),
    sklearn_LinearSVC(),
    sklearn_SVC(),
    sklearn_DecisionTreeClassifier(),
    sklearn_KNeighborsClassifier(),
    sklearn_PassiveAggressiveClassifier(),
    sklearn_SGDClassifier(),
    sklearn_XGBClassifier()
])

classifier = HyperoptEstimator(classifier=classifiers,
                              algo=tpe.suggest,
                              max_evals=40,
                              trial_timeout=120)

classifier.fit(X_train.values, y_train.values)
y_pred = classifier.predict(X_test.values)
accuracy = accuracy_score(y_test.values, y_pred)
```

accuracy: **0.8628659476117103**



```
classifier.best_model()
```

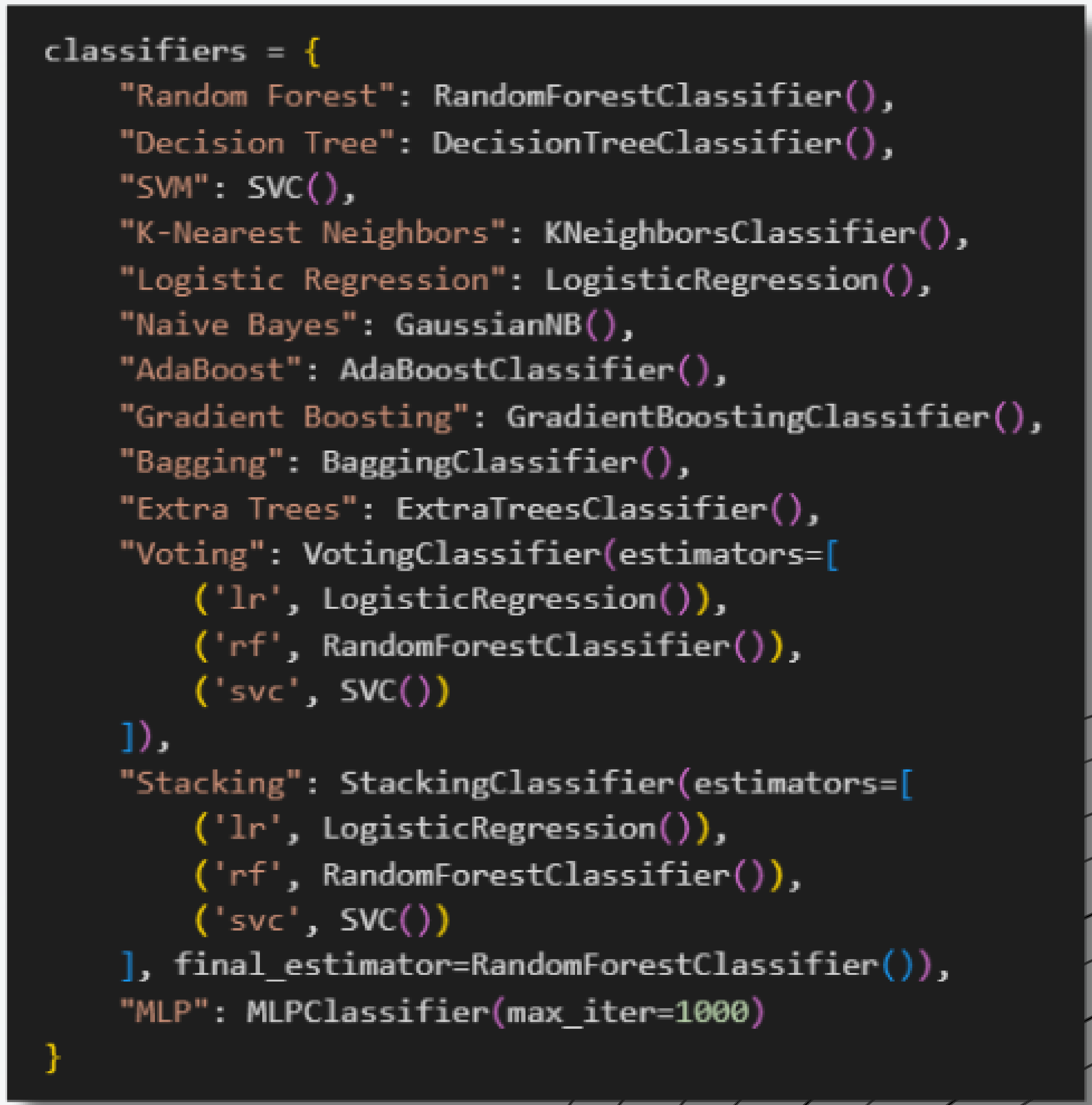
```
{'learner': SGDClassifier(),
 'preprocs': (StandardScaler(with_std=False),),
 'ex_preprocs': ()}
```

EDAIC - BASELINE

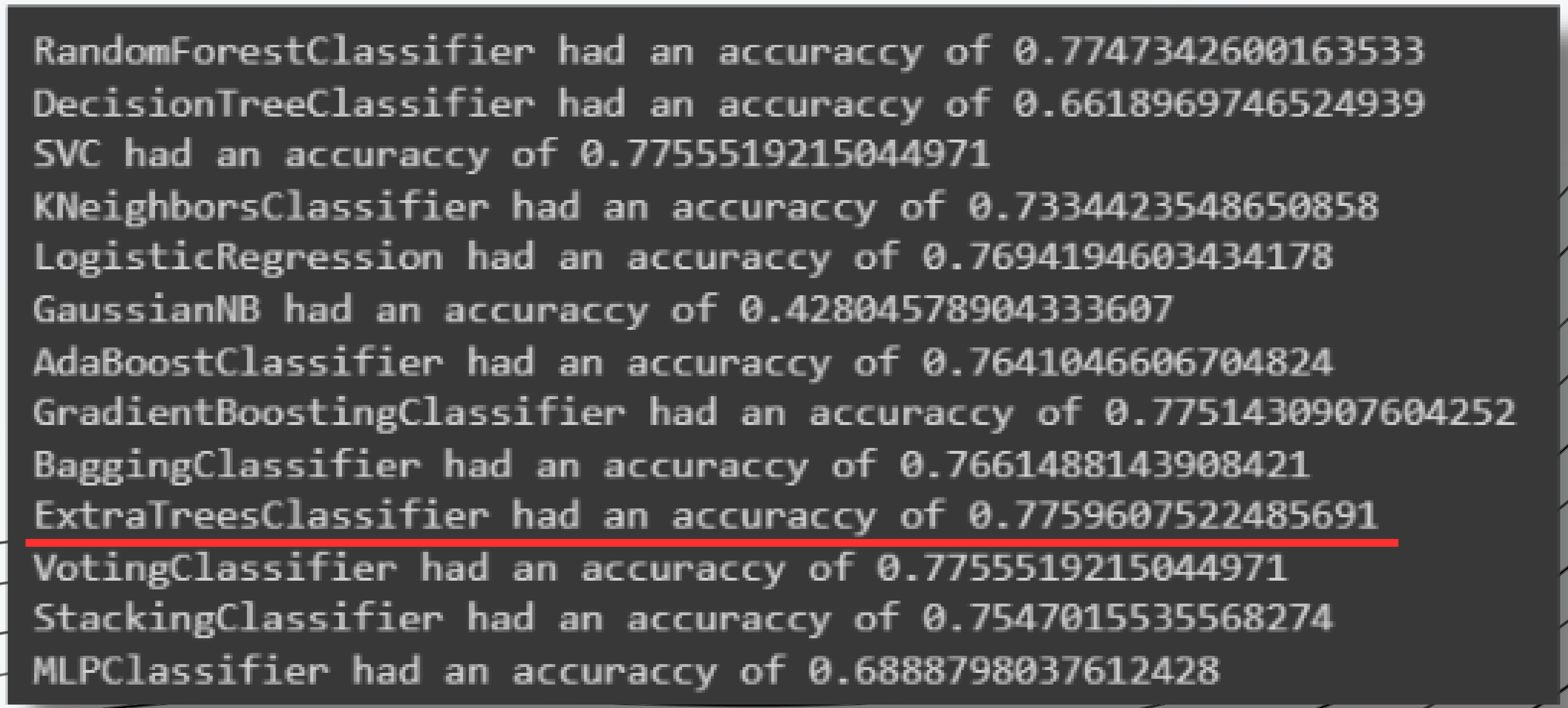
train - test - validation → already prepared in datasets

2 baselines → train/test, train/validation

EDAIC - TRAIN/TEST BASELINE

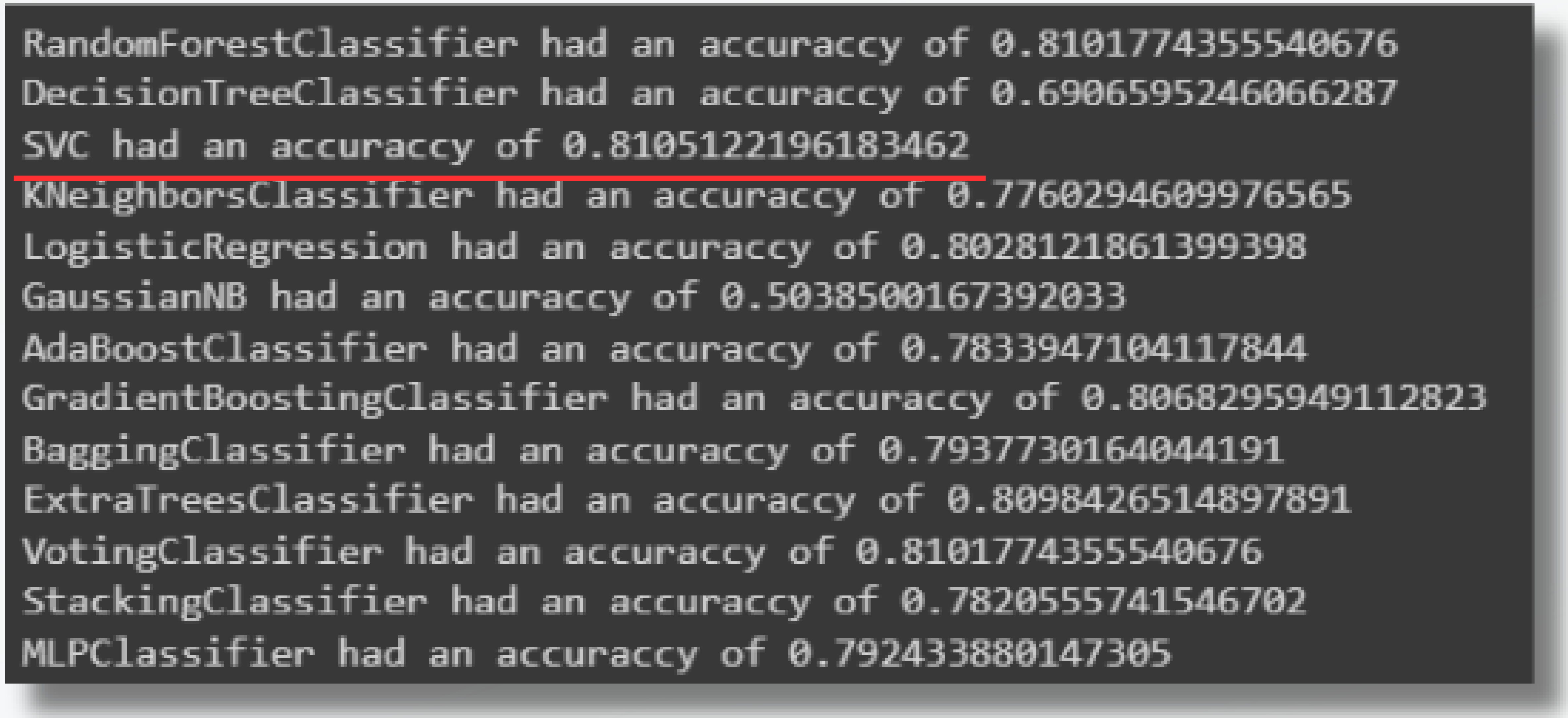


```python
classifiers = {
    "Random Forest": RandomForestClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "SVM": SVC(),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Logistic Regression": LogisticRegression(),
    "Naive Bayes": GaussianNB(),
    "AdaBoost": AdaBoostClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "Bagging": BaggingClassifier(),
    "Extra Trees": ExtraTreesClassifier(),
    "Voting": VotingClassifier(estimators=[
        ('lr', LogisticRegression()),
        ('rf', RandomForestClassifier()),
        ('svc', SVC())
    ]),
    "Stacking": StackingClassifier(estimators=[
        ('lr', LogisticRegression()),
        ('rf', RandomForestClassifier()),
        ('svc', SVC())
    ], final_estimator=RandomForestClassifier()),
    "MLP": MLPClassifier(max_iter=1000)
}
```



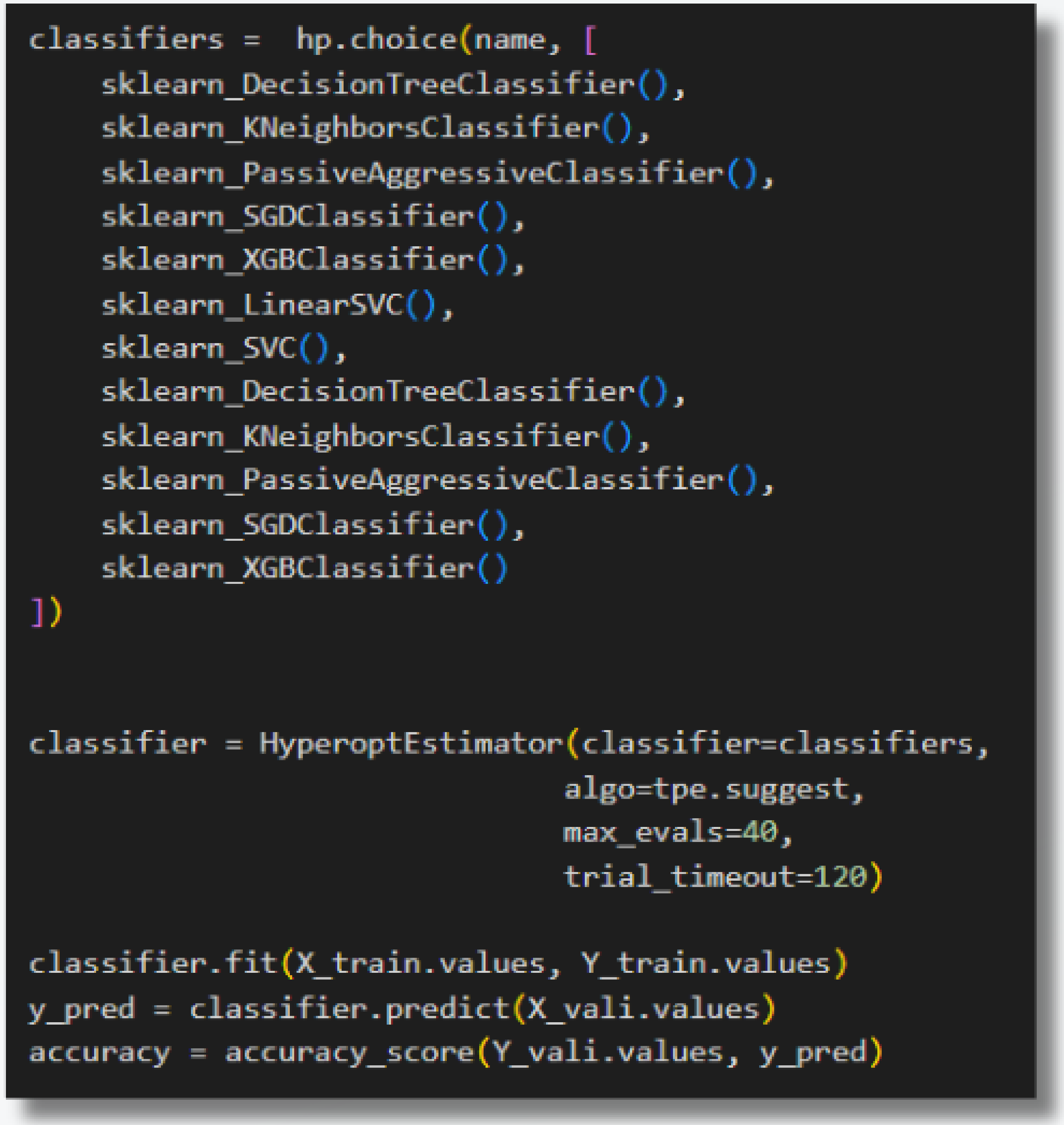
```
RandomForestClassifier had an accuraccy of 0.7747342600163533
DecisionTreeClassifier had an accuraccy of 0.6618969746524939
SVC had an accuraccy of 0.7755519215044971
KNeighborsClassifier had an accuraccy of 0.7334423548650858
LogisticRegression had an accuraccy of 0.7694194603434178
GaussianNB had an accuraccy of 0.42804578904333607
AdaBoostClassifier had an accuraccy of 0.7641046606704824
GradientBoostingClassifier had an accuraccy of 0.7751430907604252
BaggingClassifier had an accuraccy of 0.7661488143908421
ExtraTreesClassifier had an accuraccy of 0.7759607522485691
VotingClassifier had an accuraccy of 0.7755519215044971
StackingClassifier had an accuraccy of 0.7547015535568274
MLPClassifier had an accuraccy of 0.6888798037612428
```

EDAIC - TRAIN/VALIDATION BASELINE



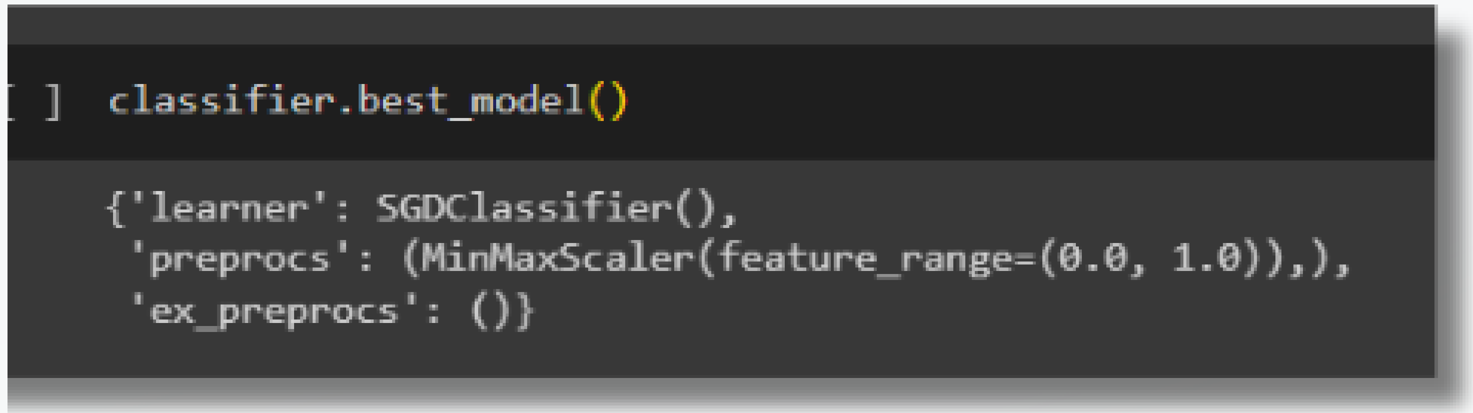
```
RandomForestClassifier had an accuraccy of 0.8101774355540676
DecisionTreeClassifier had an accuraccy of 0.6906595246066287
SVC had an accuraccy of 0.8105122196183462
KNeighborsClassifier had an accuraccy of 0.7760294609976565
LogisticRegression had an accuraccy of 0.8028121861399398
GaussianNB had an accuraccy of 0.5038500167392033
AdaBoostClassifier had an accuraccy of 0.7833947104117844
GradientBoostingClassifier had an accuraccy of 0.8068295949112823
BaggingClassifier had an accuraccy of 0.7937730164044191
ExtraTreesClassifier had an accuraccy of 0.8098426514897891
VotingClassifier had an accuraccy of 0.8101774355540676
StackingClassifier had an accuraccy of 0.7820555741546702
MLPClassifier had an accuraccy of 0.792433880147305
```

EDAIC - HYPEROPT



```
classifiers =  hp.choice(name, [
    sklearn_DecisionTreeClassifier(),
    sklearn_KNeighborsClassifier(),
    sklearn_PassiveAggressiveClassifier(),
    sklearn_SGDClassifier(),
    sklearn_XGBClassifier(),
    sklearn_LinearSVC(),
    sklearn_SVC(),
    sklearn_DecisionTreeClassifier(),
    sklearn_KNeighborsClassifier(),
    sklearn_PassiveAggressiveClassifier(),
    sklearn_SGDClassifier(),
    sklearn_XGBClassifier()
])


classifier = HyperoptEstimator(classifier=classifiers,
                              algo=tpe.suggest,
                              max_evals=40,
                              trial_timeout=120)

classifier.fit(X_train.values, Y_train.values)
y_pred = classifier.predict(X_vali.values)
accuracy = accuracy_score(Y_vali.values, y_pred)
```

accuracy: **0.7787077335118848**



```
[ ]  classifier.best_model()

     {'learner': SGDClassifier(),
      'preprocs': (MinMaxScaler(feature_range=(0.0, 1.0)),),
      'ex_preprocs': ()}
```

THANK YOU