In [ ]:
```
# install dependencies
%pip install -q \
  matplotlib \
  pandas \
  pycaret \
  'pycaret[analysis]' \
  seaborn
```

```
WARNING: visions 0.7.5 does not provide the extra 'type-image-path'
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
# global parameters
DATA_DIR = '../datasets/swell/final'
TEST_DATA_NAME = 'test'
DO_SAVE_RESULTS = True
DO_COMPARE_MODELS = True
DO_PLOT_DATA = True
```

In [ ]:
```
# set up the environment
import os
os.environ['PYCARET_CUSTOM_LOGGING_LEVEL'] = 'CRITICAL'

import pandas as pd
pd.set_option('display.max_columns', 128)
```

In [ ]:
```
# prepare the data
from pathlib import Path
from pycaret.datasets import get_data
from zipfile import ZipFile

DATA = {
  name: None
  for name in ['train', TEST_DATA_NAME]
}

for data_name in DATA.keys():
  data_path = Path(DATA_DIR).joinpath(data_name)
  # extract the compressed data files
  ZipFile(data_path.with_suffix('.zip'), 'r').extract(
    str(data_path.with_suffix('.csv')), '..'
  )
  print(f'Data file "{data_name}" has been extracted successfully')
  # load the data
  print(f'Loading data file "{data_name}"')
  DATA[data_name] = get_data(dataset=f'{data_path}')
```

```
Data file "train" has been extracted successfully
Loading data file "train"
```

	MEAN_RR	MEDIAN_RR	SDRR	RMSSD	SDSD	SDRR_RMSSD	HR	pNN
0	885.157845	853.763730	140.972741	15.554505	15.553371	9.063146	69.499952	11.1333
1	939.425371	948.357865	81.317742	12.964439	12.964195	6.272369	64.363150	5.6000
2	898.186047	907.006860	84.497236	16.305279	16.305274	5.182201	67.450066	13.0666
3	881.757865	893.460030	90.370537	15.720468	15.720068	5.748591	68.809562	11.8000
4	809.625331	811.184865	62.766242	19.213819	19.213657	3.266724	74.565728	20.2000

```
Data file "test" has been extracted successfully
Loading data file "test"
```

	MEAN_RR	MEDIAN_RR	SDRR	RMSSD	SDSD	SDRR_RMSSD	HR	pNN
0	721.901897	727.267280	74.722315	12.361264	12.361069	6.044877	84.121868	4.9333
1	843.538633	844.407930	58.499429	19.298880	19.298795	3.031234	71.478642	21.0000
2	958.523868	966.671125	132.849110	21.342715	21.342653	6.224565	63.874293	24.1333
3	824.838669	842.485905	117.822094	11.771814	11.771248	10.008830	74.330531	4.7333
4	756.707933	747.941620	143.968457	13.357748	13.356388	10.777899	82.092049	5.9333

In [ ]:
```
# set column specifications
TARGET_NAME = 'condition'
IGNORE_NAMES = ['datasetId']
```

In [ ]:
```
# establish an experiment
from pycaret.classification import ClassificationExperiment

exp = ClassificationExperiment()
exp.setup(
  data=DATA['train'],
  test_data=DATA[TEST_DATA_NAME],
  target=TARGET_NAME,
  ignore_features=IGNORE_NAMES,
  index=False,
  session_id=123,
  remove_multicollinearity=True,
  multicollinearity_threshold=0.999,
  imputation_type=None,
)
exp.dataset_transformed.head(5)
```

	Description	Value
0	Session id	123
1	Target	condition
2	Target type	Multiclass
3	Target mapping	interruption: 0, no stress: 1, time pressure: 2
4	Original data shape	(410322, 36)
5	Transformed data shape	(410322, 29)
6	Transformed train set shape	(369289, 29)
7	Transformed test set shape	(41033, 29)
8	Ignore features	1
9	Numeric features	34
10	Preprocess	True
11	Imputation type	None
12	Remove multicollinearity	True
13	Multicollinearity threshold	0.999000
14	Fold Generator	StratifiedKFold
15	Fold Number	10
16	CPU Jobs	-1
17	Use GPU	False
18	Log Experiment	False
19	Experiment Name	clf-default-name
20	USI	f1d2

Out[ ]:

	MEAN_RR	MEDIAN_RR	RMSSD	SDRR_RMSSD	HR	pNN25	pNN50	
0	885.157837	853.763733	15.554504	9.063146	69.499954	11.133333	0.533333	199.061
1	939.425354	948.357849	12.964439	6.272368	64.363152	5.600000	0.000000	114.634
2	898.186035	907.006836	16.305279	5.182201	67.450066	13.066667	0.200000	118.939
3	881.757874	893.460022	15.720469	5.748590	68.809563	11.800000	0.133333	127.318
4	809.625305	811.184875	19.213820	3.266724	74.565727	20.200001	0.200000	87.718

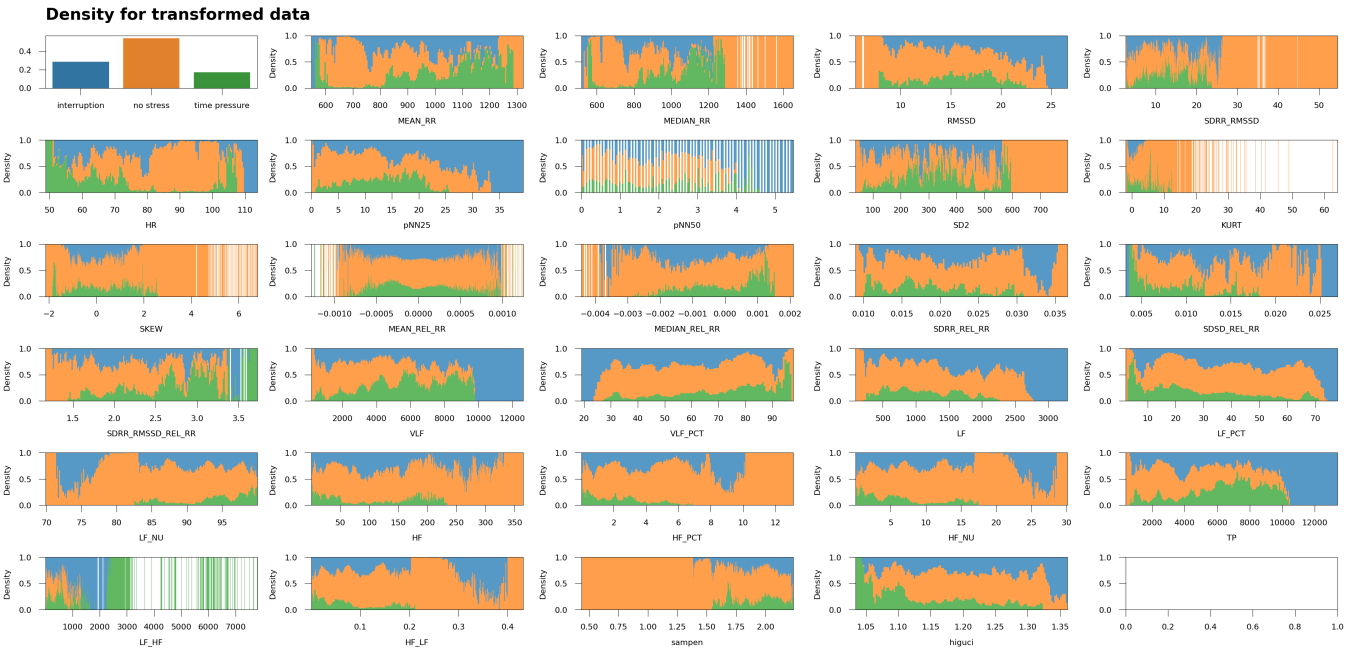
In [ ]:
```
('Removed columns: ', list(
  set(IGNORE_NAMES) |
  (set(exp.dataset.columns) - set(exp.dataset_transformed.columns))
))
```

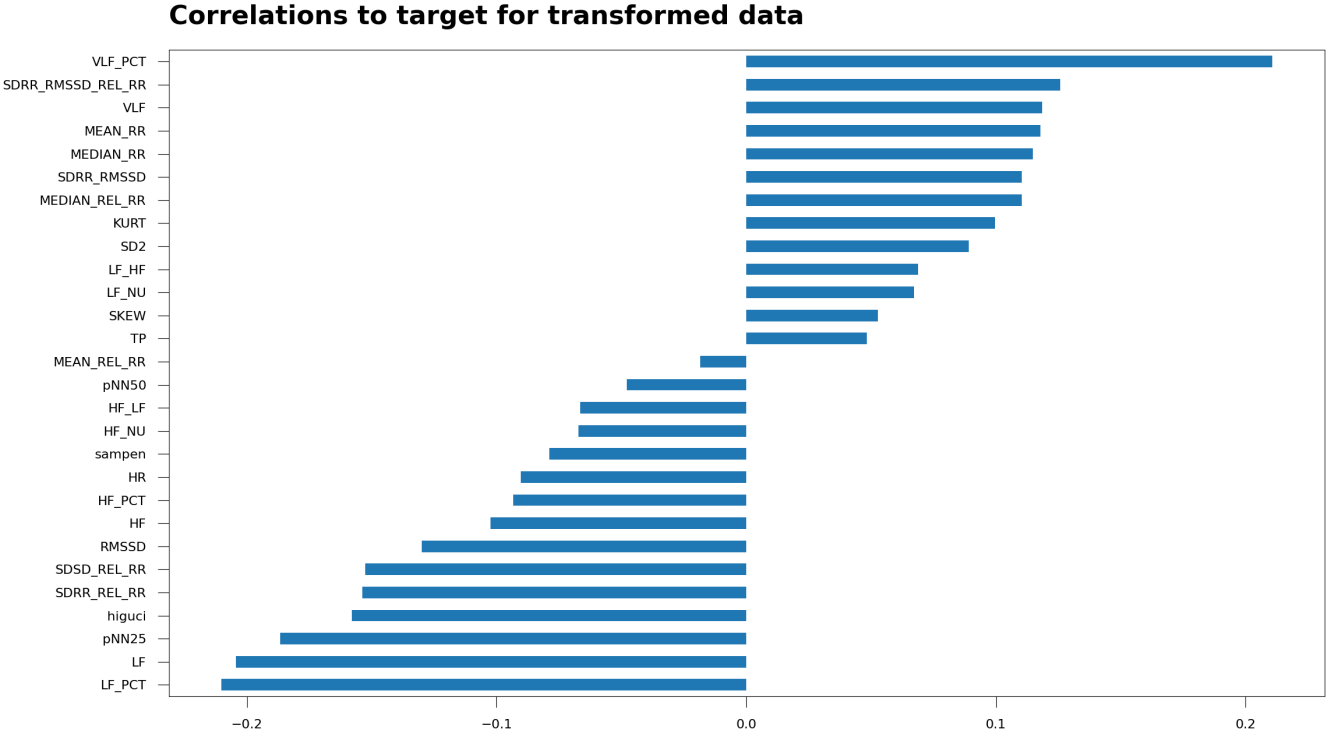
Out[ ]:
```
('Removed columns: ',
 ['SKEW_REL_RR',
  'datasetId',
  'SD1',
  'SDSD',
  'KURT_REL_RR',
  'SDRR',
  'RMSSD_REL_RR'])
```

In [ ]:
```
# show the distributions of the data
```

```python
# DO_PLOT_DATA = True
if DO_PLOT_DATA:
  # set plot parameters
  from pathlib import Path
  import matplotlib.pyplot as plt
  import seaborn as sns

  # reset old global plot parameters
  plt.rcdefaults()

  # adjustable global plot parameters
  COLORMAP = sns.color_palette()
  DPI = 400
  OUTLINE_WIDTH = 0.2
  plt.rcParams['axes.grid'] = False
  plt.rcParams['axes.linewidth'] = OUTLINE_WIDTH
  plt.rcParams['figure.dpi'] = DPI
  plt.rcParams['font.size'] = 4
  plt.rcParams['xtick.major.width'] = OUTLINE_WIDTH
  plt.rcParams['xtick.minor.width'] = OUTLINE_WIDTH
  plt.rcParams['ytick.major.width'] = OUTLINE_WIDTH
  plt.rcParams['ytick.minor.width'] = OUTLINE_WIDTH
  plot_dir = Path(f'../images/{TEST_DATA_NAME}')
  plot_dir.mkdir(parents=True, exist_ok=True)

  from math import ceil

  # adjustable local plot parameters
  TITLE = 'Density for transformed data'
  SUBPLOT_SIZE = (750, 300)

  # setting local plot parameters
  plots_per_col = 5
  shape = (plots_per_col, ceil(exp.dataset_transformed.shape[1] / plots_per_col))
  figsize = tuple(pxs[0] * pxs[1] / DPI for pxs in zip(SUBPLOT_SIZE, shape))
  title_params = {
    'label': TITLE,
    'fontdict': {
      'fontsize': plt.rcParams['font.size'] * 2,
      'fontweight': 'bold',
    },
    'loc': 'left',
    'pad': plt.rcParams['font.size'] * 2,
  }

  # plot grid
  axs = plt.subplots(
    nrows=shape[1],
    ncols=shape[0],
    layout='constrained',
    figsize=figsize,
  )[1].flat

  # plot target distribution
  target_dist_data = exp.y.value_counts(normalize=True)
  ax = sns.barplot(
    x=target_dist_data.index,
    y=target_dist_data.values,
    ax=axs[0],
    palette=COLORMAP,
  )

  # plot data title
  axs[0].set_title(**title_params)

  # plot features distribution
```

```
for x, ax in zip(exp.X_transformed.columns, axs[1:]):
  sns.histplot(
    data=exp.dataset_transformed,
    x=x,
    ax=ax,
    hue=TARGET_NAME,
    legend=False,
    linewidth=0,
    multiple='fill',
    palette=COLORMAP,
    stat='density',
  )

# save the plot
plt.savefig(
  fname=plot_dir.joinpath(f'{TITLE}.png'),
  bbox_inches='tight',
)
plt.show()

# check correlation between target and features
# adjustable plot parameters
TITLE = 'Correlations to target for transformed data'
PLOT_SIZE = (2560, 1440)

# setting plot parameters
figsize = tuple(px / DPI for px in PLOT_SIZE)
title_params = {
  'label': TITLE,
  'fontdict': {
    'fontsize': plt.rcParams['font.size'] * 2,
    'fontweight': 'bold',
  },
  'loc': 'left',
  'pad': plt.rcParams['font.size'] * 2,
}

# plot correlation to target
target_corr_data = (exp.dataset_transformed
  .corr()[TARGET_NAME]
  .drop(TARGET_NAME)
  .sort_values())
ax = target_corr_data.plot.barh(figsize=figsize)
ax.set_title(**title_params)

# save the plot
plt.savefig(
  fname=plot_dir.joinpath(f'{TITLE}.png'),
  bbox_inches='tight',
)
plt.show()

# reset plot parameters
plt.rcdefaults()
```





In [ ]:

```
# compare models with AUROC

# DO_COMPARE_MODELS = True
if DO_COMPARE_MODELS:
  exp.compare_models(
    exclude=[ # excludes slow and unsuitable models
      'ada',
      'catboost',
      'gbc',
      'knn',
      'lr',
      'ridge',
      'rf',
      'svm',
    ],
    sort='auc',
    cross_validation=False,
  )
  None
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
et	Extra Trees Classifier	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	4.7500
xgboost	Extreme Gradient Boosting	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	4.1100
lightgbm	Light Gradient Boosting Machine	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	5.7900
dt	Decision Tree Classifier	0.9997	0.9997	0.9997	0.9997	0.9997	0.9994	0.9994	15.1500
qda	Quadratic Discriminant Analysis	0.6271	0.8479	0.6271	0.7081	0.6277	0.4415	0.4707	1.6900
lda	Linear Discriminant Analysis	0.6255	0.7425	0.6255	0.6113	0.5972	0.3084	0.3273	2.3200
nb	Naive Bayes	0.5411	0.6898	0.5411	0.5587	0.5415	0.2505	0.2544	1.3100
dummy	Dummy Classifier	0.5400	0.0000	0.5400	0.2916	0.3787	0.0000	0.0000	1.2200

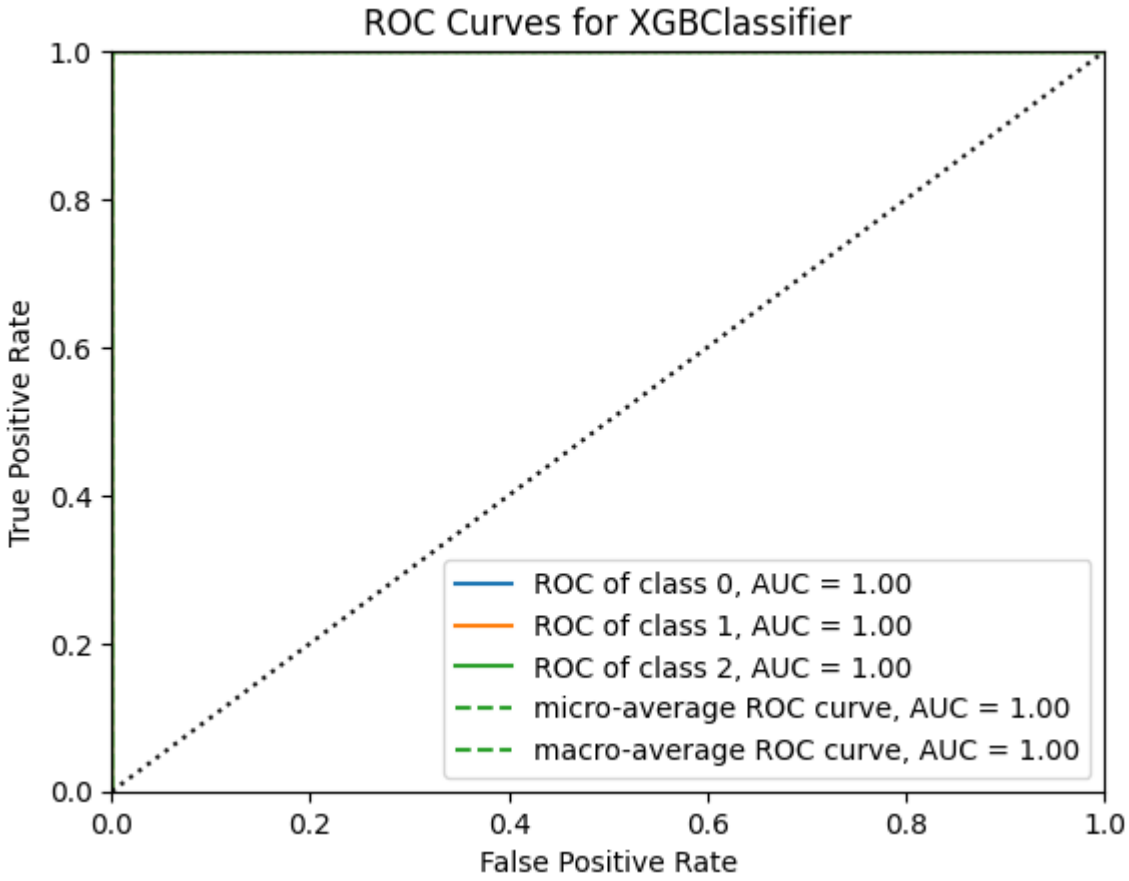
In [ ]:
```
# assign the best model id manually
BEST_MODEL_ID = 'xgboost'

best_model = exp.create_model(
  estimator=BEST_MODEL_ID,
  cross_validation=False,
)
best_model
```

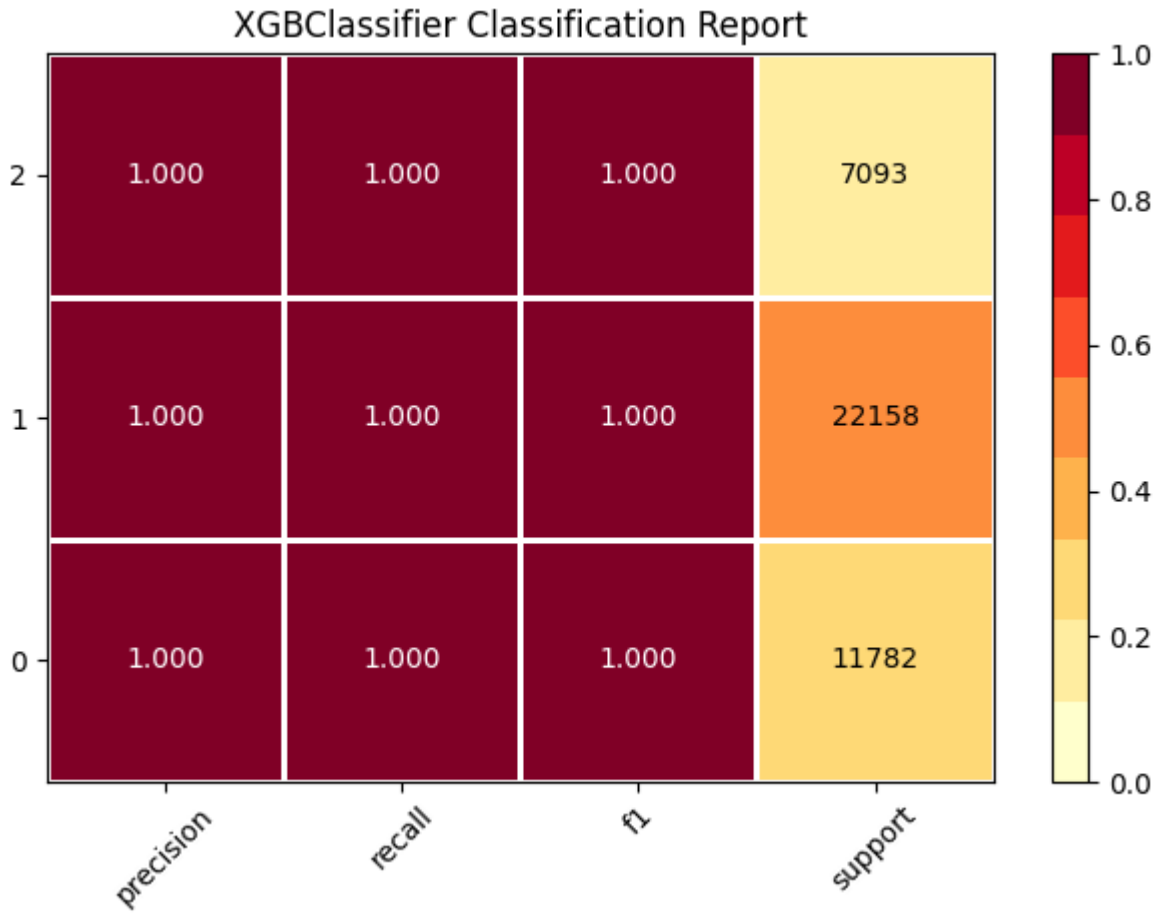
	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
Test	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

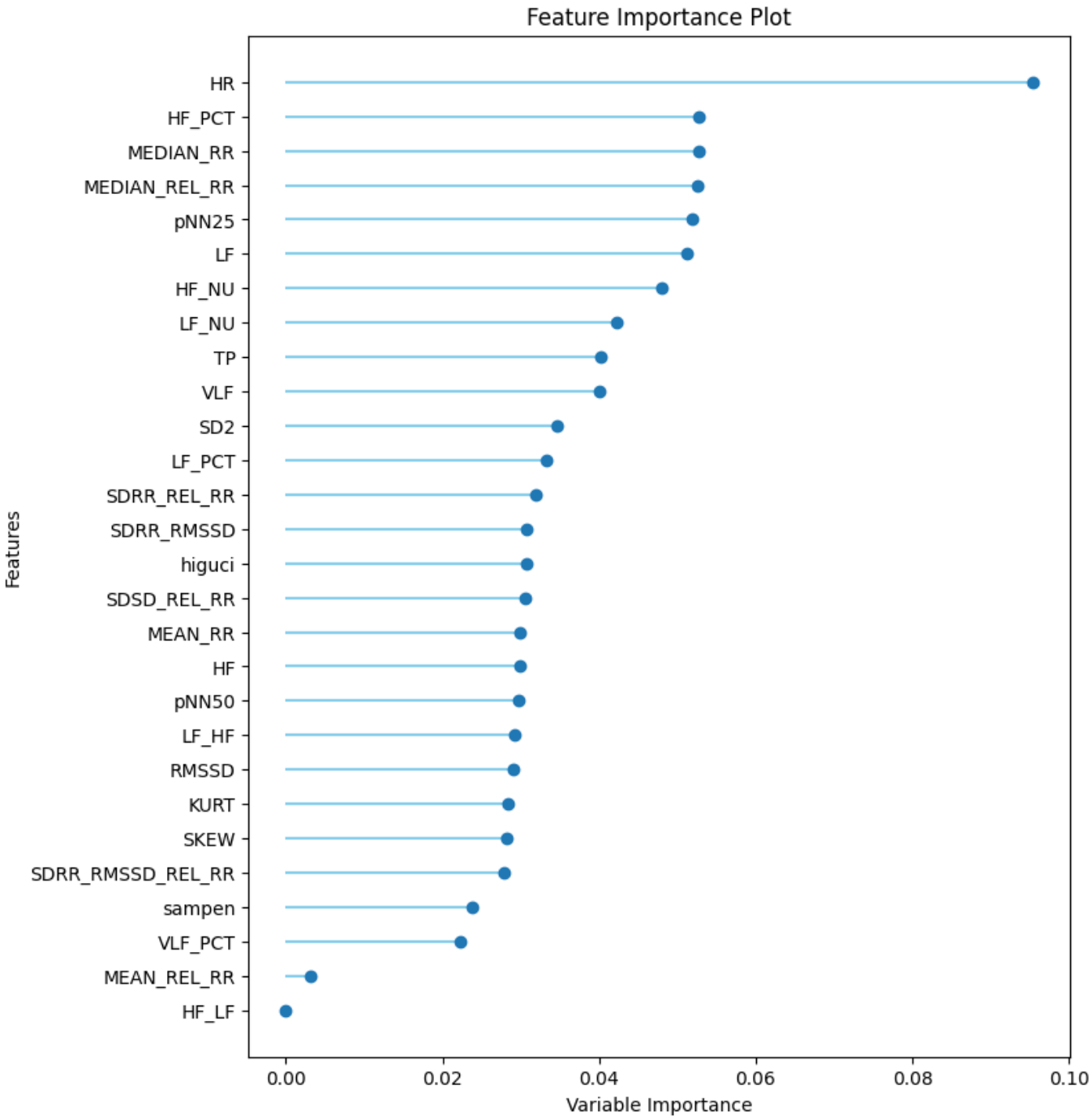
Out[ ]:
```
XGBClassifier(base_score=None, booster='gbtree', callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device='cpu', early_stopping_rounds=Non
e,
              enable_categorical=False, eval_metric=None, feature_types=Non
e,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=Non
e,
```

In [ ]:
```
plot_types = [
  'pipeline',
  'auc',
  'class_report',
  'feature_all',
  'parameter',
]
for plot_type in plot_types:
  exp.plot_model(
    estimator=best_model,
```

```
    plot=plot_type,
)
```






Feature Importance Plot
HR
HF_PCT
MEDIAN_RR
MEDIAN_REL_RR
pNN25
LF
HF_NU
LF_NU
TP
VLF
SD2
LF_PCT
SDRR_REL_RR
SDRR_RMSSD
higuci
SDSD_REL_RR
MEAN_RR
HF
pNN50
LF_HF
RMSSD
KURT
SKEW
SDRR_RMSSD_REL_RR
sampen
VLF_PCT
MEAN_REL_RR
HF_LF
Features
0.00
0.02
0.04
0.06
0.08
0.10
Variable Importance

	Parameters
objective	multi:softprob
base_score	None
booster	gbtree
callbacks	None
colsample_bylevel	None
colsample_bynode	None
colsample_bytree	None
device	cpu
early_stopping_rounds	None
enable_categorical	False
eval_metric	None
feature_types	None
gamma	None
grow_policy	None
importance_type	None
interaction_constraints	None
learning_rate	None
max_bin	None
max_cat_threshold	None
max_cat_to_onehot	None
max_delta_step	None
max_depth	None
max_leaves	None
min_child_weight	None
missing	nan
monotone_constraints	None
multi_strategy	None
n_estimators	None
n_jobs	-1
num_parallel_tree	None
random_state	123
reg_alpha	None
reg_lambda	None
sampling_method	None
scale_pos_weight	None
subsample	None
tree_method	auto

	Parameters
validate_parameters	None
verbosity	0

In [ ]:
```
# show hold-out predictions
predictions = exp.predict_model(
  estimator=best_model,
  raw_score=True,
)
display(predictions[filter(
  lambda name: name.startswith('prediction_'),
  predictions.columns,
)].sample(
  n=15,
  random_state=123,
))
predictions = None
```

	prediction_label	prediction_score_interruption	prediction_score_no stress	prediction_score_tim pressur
380118	interruption	0.9997	0.0003	0.000
369536	no stress	0.0002	0.9997	0.000
399645	no stress	0.0001	0.9999	0.000
375878	no stress	0.0009	0.9987	0.000
385576	no stress	0.0001	0.9999	0.000
402199	time pressure	0.0001	0.0000	0.999
391050	interruption	0.9996	0.0004	0.000
397656	no stress	0.0004	0.9982	0.001
398352	no stress	0.0002	0.9994	0.000
398153	no stress	0.0025	0.9968	0.000
394973	time pressure	0.0002	0.0000	0.999
400474	interruption	0.9998	0.0002	0.000
398507	no stress	0.0003	0.9996	0.000
385040	interruption	0.9986	0.0006	0.000
395651	interruption	0.9960	0.0008	0.003

In [ ]:
```
# save the experiment and model

# DO_SAVE_RESULTS = True
if DO_SAVE_RESULTS:
  from pathlib import Path

  result_dir = Path(f'../models/{TEST_DATA_NAME}')
  result_dir.mkdir(
    parents=True,
    exist_ok=True,
  )
  exp.save_experiment(
    path_or_file=result_dir.joinpath('experiment.pkl'),
  )
  exp.save_model(
```

```
    model=best_model,
    model_name=result_dir.joinpath('model'),
  )
```

```
Transformation Pipeline and Model Successfully Saved
```