MLPRegressor_v1_seg_2

November 24, 2022

1 MLPRegressor

[ ]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.neural_network import MLPRegressor
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import r2_score, mean_absolute_percentage_error,␣
↪mean_absolute_error, mean_squared_error
from statsmodels.tools.eval_measures import stde
```

1.1 Read the etl info results

[ ]:
```
df_info = pd.read_csv('../dataset_clean/options_csv_v1_etl.csv')
df_info
```

[ ]:
```
   remove_time_features  generic_features  remove_atypical_values  \
0                 False              True                   False

   feature_combination  remove_feature_selection  \
0                False                     False

   remove_invalid_correlated_features
0                               False
```

1.2 Read the dataset

[ ]:
```
df = pd.read_csv('../dataset_clean/PlatteRiverWeir_features_v1_clean.csv')
df
```

[ ]:
```
                SensorTime          CaptureTime  Stage  Discharge   grayMean  \
0      2012-06-09 13:15:00  2012-06-09T13:09:07   2.99      916.0  97.405096
```

```
1      2012-06-09 13:15:00  2012-06-09T13:10:29   2.99      916.0  104.066757
2      2012-06-09 13:45:00  2012-06-09T13:44:01   2.96      873.0  105.636831
3      2012-06-09 14:45:00  2012-06-09T14:44:30   2.94      846.0  104.418949
4      2012-06-09 15:45:00  2012-06-09T15:44:59   2.94      846.0  106.763541
…                    …                    …     …         …           …
42054  2019-10-11 09:00:00  2019-10-11T08:59:53   2.54      434.0   82.872720
42055  2019-10-11 10:00:00  2019-10-11T09:59:52   2.54      434.0   89.028383
42056  2019-10-11 11:00:00  2019-10-11T10:59:52   2.54      434.0   94.722097
42057  2019-10-11 12:00:00  2019-10-11T11:59:53   2.54      434.0   96.693270
42058  2019-10-11 12:45:00  2019-10-11T12:59:52   2.54      434.0   98.738399

       graySigma  entropyMean  entropySigma       hMean     hSigma  \
0      39.623303     0.203417      0.979825  105.368375  41.572939
1      40.179745     0.206835      1.002624  112.399458  41.795584
2      40.533218     0.204756      0.994246  114.021526  42.145582
3      41.752678     0.202428      0.983170  112.612830  43.575351
4      44.442097     0.202661      0.989625  114.839424  46.302008
…            …            …             …           …          …
42054  57.702652     0.221708      1.076393   87.260572  61.485334
42055  55.840861     0.233168      1.124774   94.175906  59.006132
42056  54.355753     0.240722      1.151833  100.534577  56.921028
42057  52.787629     0.244789      1.171987  102.891159  55.083532
42058  52.025453     0.252812      1.213278  105.292067  53.994155

            sMean    sSigma       vMean     vSigma
0      124.520218  4.111846  132.405971  14.983367
1      124.317679  4.270429  133.070221  15.334166
2      124.304621  4.310293  133.294541  15.502448
3      124.369736  4.120586  133.458381  15.190064
4      124.283191  4.088480  133.573595  14.801143
…             …         …           …          …
42054  127.807813  2.564157  124.073149  13.757842
42055  127.336000  2.585121  124.882812  13.234735
42056  126.958768  2.774867  126.145409  13.408480
42057  126.679956  2.998683  127.508063  13.863205
42058  126.328075  3.258103  128.788256  14.353808

[42059 rows x 14 columns]
```

[ ]:
```python
df['SensorTime'] = pd.to_datetime(df['SensorTime'])
df['Year'] = df['SensorTime'].dt.year
```

[ ]:
```python
df.dtypes
```

[ ]:
```
SensorTime      datetime64[ns]
CaptureTime             object
Stage                  float64
```

```
Discharge              float64
grayMean               float64
graySigma              float64
entropyMean            float64
entropySigma           float64
hMean                  float64
hSigma                 float64
sMean                  float64
sSigma                 float64
vMean                  float64
vSigma                 float64
Year                     int64
dtype: object
```

```
[ ]: df = df[(df.Stage > 0) & (df.Discharge > 0)]
```

1.3 Divide dataset to X and Y

```
[ ]: np.random.seed(0)

     df_train = df[(df.Year >= 2012) & (df.Year <= 2017)]
     df_train = df_train.iloc[np.random.permutation(len(df_train))]

     df_test = df[(df.Year >= 2018) & (df.Year <= 2019)]
```

```
[ ]: df_train = df_train.drop(columns=["Year", "SensorTime", "CaptureTime"])
     df_test = df_test.drop(columns=["Year", "SensorTime", "CaptureTime"])
```

```
[ ]: y_train = df_train["Stage"]
     X_train = df_train.drop(columns=["Stage", "Discharge"])

     y_test = df_test["Stage"]
     X_test = df_test.drop(columns=["Stage", "Discharge"])
```

```
[ ]: #X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
      ↪random_state=0)
```

1.4 Train model

```
[ ]: pipeline = Pipeline([
         ('scaler', StandardScaler()),
         ('clf', MLPRegressor(shuffle=False, max_iter=2000))
     ])
```

```
#param_grid = {'clf__hidden_layer_sizes': [(10), (10, 20), (10, 5, 15), (20,␣
↪30, 10, 15)], 'clf__alpha': np.arange(1e-3, 1, 0.001),␣
↪'clf__learning_rate_init': np.arange(1e-3, 0.1, 0.001), 'clf__activation':␣
↪['tanh', 'relu']}

param_grid = {'clf__hidden_layer_sizes': [(256, 256, 128, 128, 64), (512, 256),␣
↪(128, 64, 64, 32), (512, 256, 128, 128)], 'clf__alpha': np.arange(1e-3, 0.1,␣
↪0.001), 'clf__activation': ['tanh', 'relu']}

clf = RandomizedSearchCV(pipeline, param_distributions=param_grid, n_iter=10,␣
↪n_jobs=8, verbose=3, scoring="neg_mean_squared_error")
```

[ ]:
```
clf.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV 3/5] END clf__activation=tanh, clf__alpha=0.095,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.247 total time=  38.8s
[CV 1/5] END clf__activation=tanh, clf__alpha=0.095,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.244 total time=  43.6s
[CV 2/5] END clf__activation=tanh, clf__alpha=0.095,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.241 total time=  44.0s
[CV 4/5] END clf__activation=tanh, clf__alpha=0.095,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.245 total time=  37.0s
[CV 5/5] END clf__activation=tanh, clf__alpha=0.095,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.228 total time=  39.2s
[CV 3/5] END clf__activation=tanh, clf__alpha=0.019000000000000003,
clf__hidden_layer_sizes=(256, 256, 128, 128, 64);, score=-0.235 total time=
2.0min
[CV 4/5] END clf__activation=tanh, clf__alpha=0.019000000000000003,
clf__hidden_layer_sizes=(256, 256, 128, 128, 64);, score=-0.254 total time=
2.1min
[CV 5/5] END clf__activation=tanh, clf__alpha=0.019000000000000003,
clf__hidden_layer_sizes=(256, 256, 128, 128, 64);, score=-0.231 total time=
2.4min
[CV 2/5] END clf__activation=tanh, clf__alpha=0.019000000000000003,
clf__hidden_layer_sizes=(256, 256, 128, 128, 64);, score=-0.250 total time=
2.7min
[CV 1/5] END clf__activation=tanh, clf__alpha=0.019000000000000003,
clf__hidden_layer_sizes=(256, 256, 128, 128, 64);, score=-0.270 total time=
3.0min
[CV 1/5] END clf__activation=tanh, clf__alpha=0.064,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.222 total time= 6.4min
[CV 2/5] END clf__activation=tanh, clf__alpha=0.064,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.223 total time= 6.1min
[CV 3/5] END clf__activation=tanh, clf__alpha=0.064,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.237 total time= 6.0min
[CV 4/5] END clf__activation=tanh, clf__alpha=0.064,
```

```
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.238 total time= 5.9min
[CV 2/5] END clf__activation=tanh, clf__alpha=0.026000000000000002,
clf__hidden_layer_sizes=(512, 256);, score=-0.231 total time= 5.3min
[CV 5/5] END clf__activation=tanh, clf__alpha=0.064,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.234 total time= 6.1min
[CV 1/5] END clf__activation=tanh, clf__alpha=0.026000000000000002,
clf__hidden_layer_sizes=(512, 256);, score=-0.234 total time= 5.8min
[CV 3/5] END clf__activation=tanh, clf__alpha=0.026000000000000002,
clf__hidden_layer_sizes=(512, 256);, score=-0.232 total time= 5.4min
[CV 1/5] END clf__activation=tanh, clf__alpha=0.005,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.263 total time= 1.3min
[CV 2/5] END clf__activation=tanh, clf__alpha=0.005,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.259 total time= 1.3min
[CV 3/5] END clf__activation=tanh, clf__alpha=0.005,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.276 total time= 1.4min
[CV 4/5] END clf__activation=tanh, clf__alpha=0.005,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.266 total time= 1.3min
[CV 5/5] END clf__activation=tanh, clf__alpha=0.005,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.260 total time= 1.3min
[CV 5/5] END clf__activation=tanh, clf__alpha=0.026000000000000002,
clf__hidden_layer_sizes=(512, 256);, score=-0.241 total time= 4.2min
[CV 4/5] END clf__activation=tanh, clf__alpha=0.026000000000000002,
clf__hidden_layer_sizes=(512, 256);, score=-0.240 total time= 5.5min
[CV 1/5] END clf__activation=tanh, clf__alpha=0.014000000000000002,
clf__hidden_layer_sizes=(512, 256);, score=-0.224 total time= 5.9min
[CV 2/5] END clf__activation=tanh, clf__alpha=0.014000000000000002,
clf__hidden_layer_sizes=(512, 256);, score=-0.235 total time= 5.6min
[CV 1/5] END clf__activation=tanh, clf__alpha=0.035,
clf__hidden_layer_sizes=(512, 256);, score=-0.236 total time= 5.2min
[CV 4/5] END clf__activation=tanh, clf__alpha=0.014000000000000002,
clf__hidden_layer_sizes=(512, 256);, score=-0.234 total time= 5.5min
[CV 3/5] END clf__activation=tanh, clf__alpha=0.014000000000000002,
clf__hidden_layer_sizes=(512, 256);, score=-0.226 total time= 6.4min
[CV 5/5] END clf__activation=tanh, clf__alpha=0.014000000000000002,
clf__hidden_layer_sizes=(512, 256);, score=-0.224 total time= 6.2min
[CV 2/5] END clf__activation=tanh, clf__alpha=0.035,
clf__hidden_layer_sizes=(512, 256);, score=-0.240 total time= 5.3min
[CV 3/5] END clf__activation=tanh, clf__alpha=0.035,
clf__hidden_layer_sizes=(512, 256);, score=-0.235 total time= 5.1min
[CV 1/5] END clf__activation=relu, clf__alpha=0.098,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.225 total time= 3.9min
[CV 1/5] END clf__activation=relu, clf__alpha=0.079,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.219 total time=  49.3s
[CV 4/5] END clf__activation=tanh, clf__alpha=0.035,
clf__hidden_layer_sizes=(512, 256);, score=-0.250 total time= 4.7min
[CV 5/5] END clf__activation=tanh, clf__alpha=0.035,
clf__hidden_layer_sizes=(512, 256);, score=-0.238 total time= 4.9min
[CV 2/5] END clf__activation=relu, clf__alpha=0.079,
```

```
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.225 total time=  49.3s
[CV 3/5] END clf__activation=relu, clf__alpha=0.079,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.252 total time=  54.4s
[CV 5/5] END clf__activation=relu, clf__alpha=0.079,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.218 total time=  22.7s
[CV 4/5] END clf__activation=relu, clf__alpha=0.079,
clf__hidden_layer_sizes=(128, 64, 64, 32);, score=-0.226 total time=  53.8s
[CV 2/5] END clf__activation=relu, clf__alpha=0.098,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.209 total time= 5.6min
[CV 4/5] END clf__activation=relu, clf__alpha=0.098,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.223 total time= 5.6min
[CV 3/5] END clf__activation=relu, clf__alpha=0.098,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.246 total time= 6.0min
[CV 3/5] END clf__activation=tanh, clf__alpha=0.068,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.236 total time= 4.2min
[CV 5/5] END clf__activation=relu, clf__alpha=0.098,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.202 total time= 7.1min
[CV 1/5] END clf__activation=tanh, clf__alpha=0.068,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.223 total time= 4.6min
[CV 2/5] END clf__activation=tanh, clf__alpha=0.068,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.223 total time= 4.7min
[CV 4/5] END clf__activation=tanh, clf__alpha=0.068,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.242 total time= 4.4min
[CV 5/5] END clf__activation=tanh, clf__alpha=0.068,
clf__hidden_layer_sizes=(512, 256, 128, 128);, score=-0.230 total time= 4.0min
```

[ ]:
```
RandomizedSearchCV(estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                             ('clf',
                                              MLPRegressor(max_iter=2000,
                                                           shuffle=False))]),
                   n_jobs=8,
                   param_distributions={'clf__activation': ['tanh', 'relu'],
                                        'clf__alpha': array([0.001, 0.002,
0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009,
       0.01 , 0.011, 0.012, 0.013, 0.014, 0.015, 0.016, 0.017, 0.018,
       0.019, 0.02 , 0.021, 0.022, 0.023, 0.024…
       0.064, 0.065, 0.066, 0.067, 0.068, 0.069, 0.07 , 0.071, 0.072,
       0.073, 0.074, 0.075, 0.076, 0.077, 0.078, 0.079, 0.08 , 0.081,
       0.082, 0.083, 0.084, 0.085, 0.086, 0.087, 0.088, 0.089, 0.09 ,
       0.091, 0.092, 0.093, 0.094, 0.095, 0.096, 0.097, 0.098, 0.099]),
                                        'clf__hidden_layer_sizes': [(256, 256,
                                                                     128, 128,
                                                                     64),
                                                                    (512, 256),
                                                                    (128, 64,
                                                                     64, 32),
                                                                    (512, 256,
```

```
                                                                      128,
                                                                      128)]},
                         scoring='neg_mean_squared_error', verbose=3)
```

[ ]:
```
clf.best_score_
```

[ ]: -0.22091953882505555

[ ]:
```
clf.best_params_
```

[ ]:
```
{'clf__hidden_layer_sizes': (512, 256, 128, 128),
 'clf__alpha': 0.098,
 'clf__activation': 'relu'}
```

1.5 Test model

[ ]:
```
clf.score(X_test, y_test)
```

[ ]: -0.4255222840231655

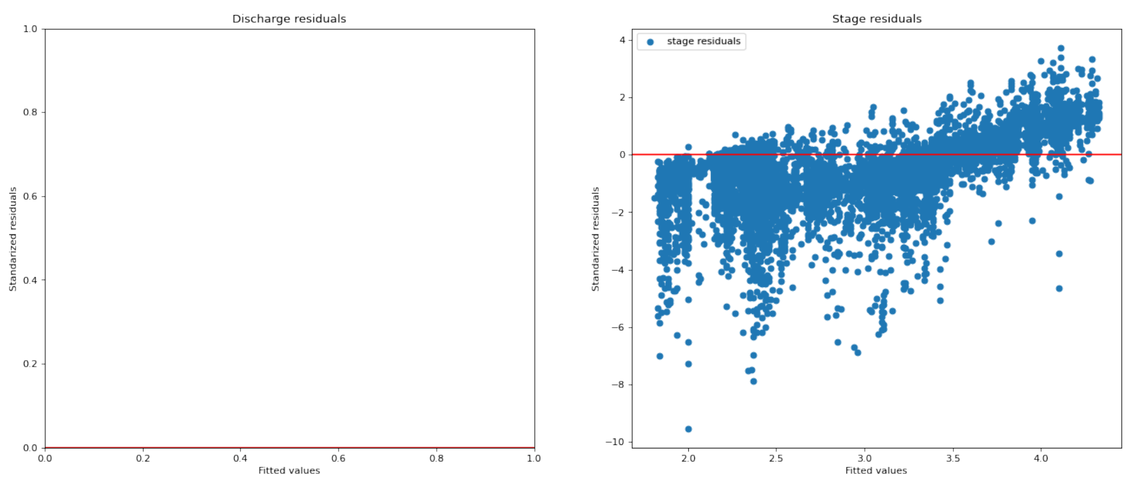
[ ]:
```
y_pred = clf.predict(X_test)
```

[ ]:
```
print("R^2: ", r2_score(y_test, y_pred))
print("mse: ", mean_squared_error(y_test, y_pred))
print("rmse: ", mean_squared_error(y_test, y_pred, squared=False))
print("mae: ", mean_absolute_error(y_test, y_pred))
print("mape: ", mean_absolute_percentage_error(y_test, y_pred))
print("Error estandar: ", stde(y_test.squeeze(),
      y_pred.squeeze(), ddof=2))
```

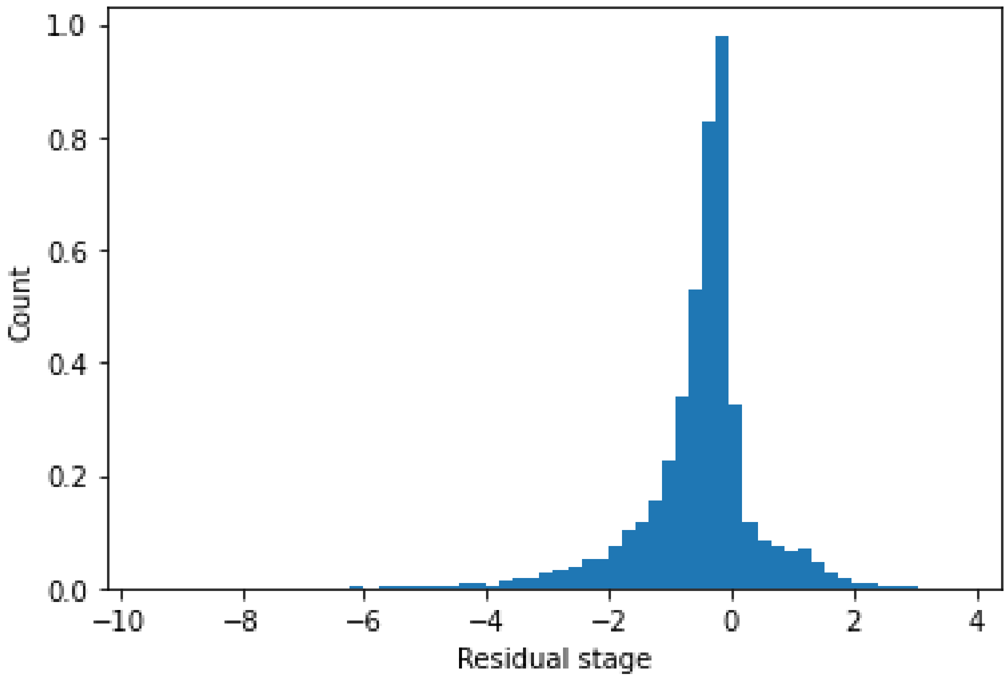
```
R^2:  -0.08957704724707871
mse:  0.4255222840231655
rmse:  0.6523206910892567
mae:  0.43109842602881954
mape:  0.16322167249582156
Error estandar:  0.581351761528604
```

[ ]:
```
residuals = y_test - y_pred
residuals_std = residuals/residuals.std()

y_real_stage = y_test
residual_stage = residuals

#y_real_discharge = np.array([i[-1] for i in y_test])
#residual_discharge = np.array([i[-1] for i in residuals])
```

```
figure, ax = plt.subplots(ncols=2, figsize=(20, 8), dpi=80)

ax[1].scatter(y_real_stage, residual_stage / residual_stage.std(), label="stage␣
↪residuals")
#ax[0].scatter(y_real_discharge, residual_discharge / residual_discharge.std(),␣
↪label="discharge residuals")
ax[1].axhline(y=0.0, color='r', linestyle='-')
ax[0].axhline(y=0.0, color='r', linestyle='-')

ax[1].set_title("Stage residuals")
ax[0].set_title("Discharge residuals")

ax[1].set_xlabel("Fitted values")
ax[0].set_xlabel("Fitted values")
ax[1].set_ylabel("Standarized residuals")
ax[0].set_ylabel("Standarized residuals")

plt.legend()
plt.show()
```



[ ]:
```
plt.hist(residual_stage / residual_stage.std(), density=True, bins = 60)
plt.ylabel('Count')
plt.xlabel('Residual stage');
plt.show()
```

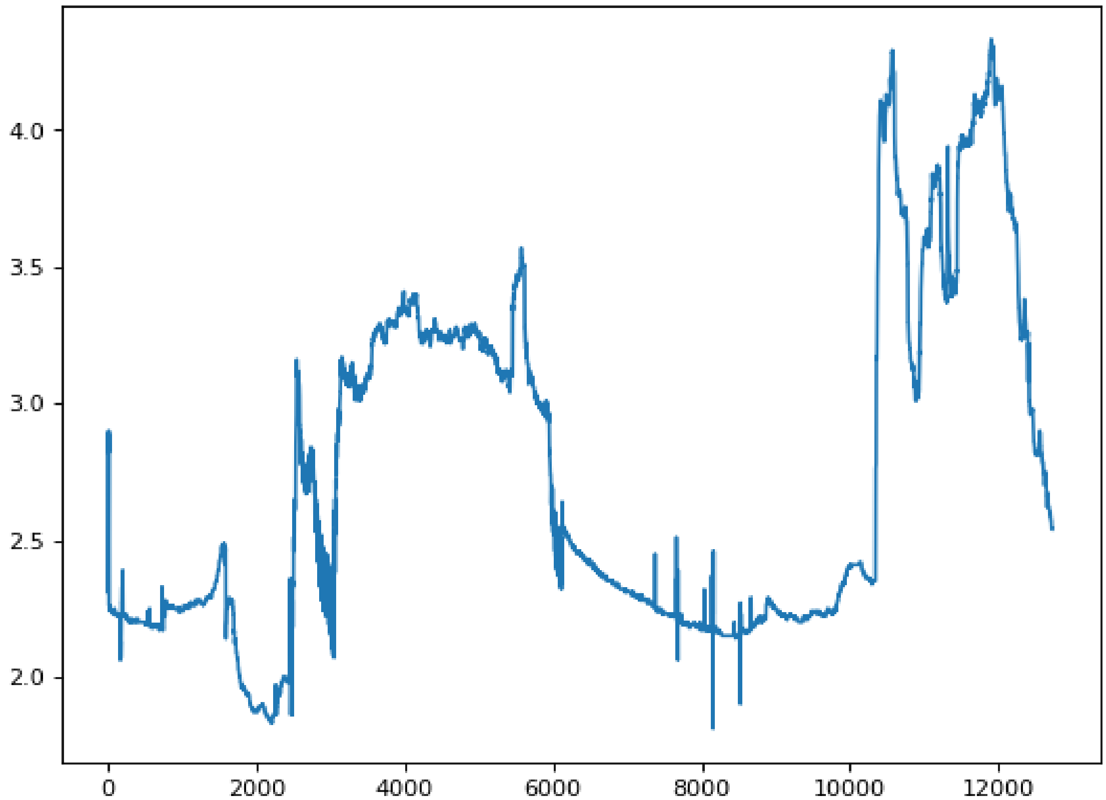


[ ]:
```
"""plt.hist(residual_discharge / residual_discharge.std(), density=True, bins =
 ↪60)
plt.ylabel('Count')
plt.xlabel('Residual discharge');
plt.show()"""
```

[ ]:
```
"plt.hist(residual_discharge / residual_discharge.std(), density=True, bins =
60)\nplt.ylabel('Count')\nplt.xlabel('Residual discharge');\nplt.show()"
```

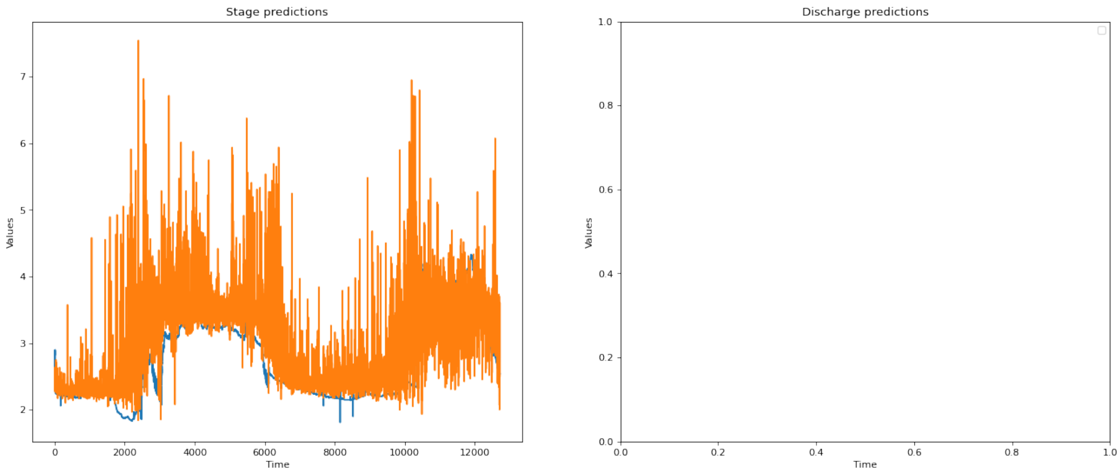
[ ]:
```
plt.figure(figsize=(8, 6), dpi=80)
plt.plot(np.arange(len(y_test)), y_test, label="Stage real")
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f0d54aef700>]
```



[ ]:
```
figure, ax = plt.subplots(ncols=2, figsize=(20, 8), dpi=80)

ax[0].plot(np.arange(len(y_test)), y_test, label="Stage real")
ax[0].plot(np.arange(len(y_test)), y_pred, label="Stage pred")

ax[0].set_title("Stage predictions")
ax[1].set_title("Discharge predictions")

ax[1].set_ylabel("Values")
ax[0].set_ylabel("Values")
ax[1].set_xlabel("Time")
ax[0].set_xlabel("Time")

plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```


Stage predictions
Values
Time
0
2000
4000
6000
8000
10000
12000
2
3
4
5
6
7
Discharge predictions
Values
Time
0.0
0.2
0.4
0.6
0.8
1.0