linear_regression_v1_8

October 21, 2022

1 Linear regression

[ ]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
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                    Lasso

   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       hMean     hSigma
0      39.623303  105.368375  41.572939
1      40.179745  112.399458  41.795584
2      40.533218  114.021526  42.145582
3      41.752678  112.612830  43.575351
4      44.442097  114.839424  46.302008
…          …          …        …
42054  57.702652   87.260572  61.485334
42055  55.840861   94.175906  59.006132
42056  54.355753  100.534577  56.921028
42057  52.787629  102.891159  55.083532
42058  52.025453  105.292067  53.994155

[42059 rows x 8 columns]
```

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

```
[ ]: df_train = df[(df.Year >= 2012) & (df.Year <= 2017)]
     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"])
```

1.3 Divide dataset to X and Y

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

     folds = KFold(n_splits = 5, shuffle = True, random_state = 100)
     clf = cross_val_score(pipeline, X_train, y_train, scoring='r2', cv=folds)
```

```
[ ]: clf
```

```
[ ]: array([0.1593821 , 0.15823139, 0.15456937, 0.15749307, 0.17837916])
```

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

```
[ ]: Pipeline(steps=[('scaler', StandardScaler()), ('clf', LinearRegression())])
```

1.5 Test Model

```
[ ]: y_pred = pipeline.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=len(X_train.columns) + 1))
```

```
R^2:  0.11826044004406694
mse:  260848.0265376896
rmse:  361.4301195288088
mae:  304.6086277309594
mape:  6.8141953706081656e+16
Error estandar:  [5.16774324e-01 6.29999958e+02]
```

```
[ ]: residuals = y_test - y_pred
     residuals
```

```
[ ]:           Stage   Discharge
     28811  0.106292 -210.874905
     28812 -0.580412 -957.347858
     28813 -0.451868 -784.631805
     28814  0.071584  -87.625149
     28815 -0.424021 -711.479812
     …           …           …
     42054 -0.324914 -471.784825
```

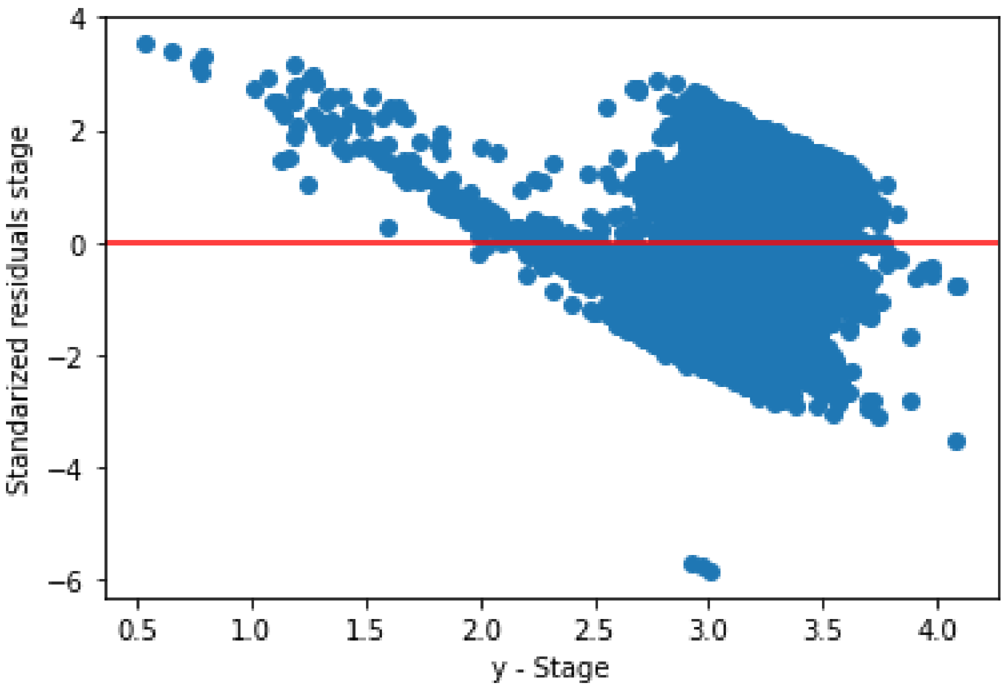
```
42055 -0.413741 -614.366084
42056 -0.497440 -750.010401
42057 -0.561150 -846.566605
42058 -0.656136 -989.026133

[13248 rows x 2 columns]
```

```
[ ]: resid = np.array(residuals["Stage"])
     norm_resid = resid / resid.std()

     plt.scatter([i[0] for i in y_pred], norm_resid)
     plt.axhline(y = 0.0, color = 'r', linestyle = '-')
     plt.xlabel("y - Stage")
     plt.ylabel("Standarized residuals stage")
```

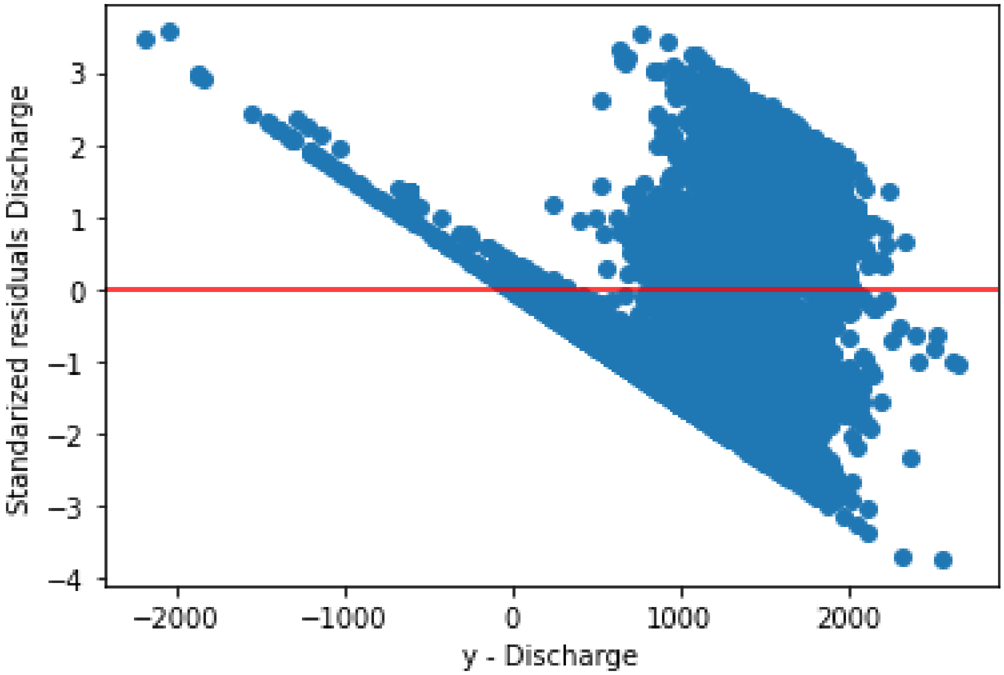
```
[ ]: Text(0, 0.5, 'Standarized residuals stage')
```



```
[ ]: resid = np.array(residuals["Discharge"])
     norm_resid = resid / resid.std()

     plt.scatter([i[1] for i in y_pred], norm_resid)
     plt.axhline(y = 0.0, color = 'r', linestyle = '-')
     plt.xlabel("y - Discharge")
     plt.ylabel("Standarized residuals Discharge")
```

```
[ ]: Text(0, 0.5, 'Standarized residuals Discharge')
```



```
[ ]:
```