In [1]:
```
from pycaret.datasets import get_data
df = get_data('diamond')
```

	Carat Weight	Cut	Color	Clarity	Polish	Symmetry	Report	Price
0	1.10	Ideal	H	SI1	VG	EX	GIA	5169
1	0.83	Ideal	H	VS1	ID	ID	AGSL	3470
2	0.85	Ideal	H	SI1	EX	EX	GIA	3183
3	0.91	Ideal	E	SI1	VG	VG	GIA	4370
4	0.83	Ideal	G	SI1	EX	EX	GIA	3171

In [2]:
```
%matplotlib inline
```

In [3]:
```
df.head(7)
```

Out[3]:

	Carat Weight	Cut	Color	Clarity	Polish	Symmetry	Report	Price
0	1.10	Ideal	H	SI1	VG	EX	GIA	5169
1	0.83	Ideal	H	VS1	ID	ID	AGSL	3470
2	0.85	Ideal	H	SI1	EX	EX	GIA	3183
3	0.91	Ideal	E	SI1	VG	VG	GIA	4370
4	0.83	Ideal	G	SI1	EX	EX	GIA	3171
5	1.53	Ideal	E	SI1	ID	ID	AGSL	12791
6	1.00	Very Good	D	SI1	VG	G	GIA	5747

In [4]:
```
data = df.sample(frac = 0.9, random_state = 786)
```

In [5]:
```
data
```

Out[5]:

	Carat Weight	Cut	Color	Clarity	Polish	Symmetry	Report	Price
5333	1.50	Good	E	VS2	VG	G	GIA	12702
3749	1.08	Ideal	D	VS2	EX	EX	GIA	8481
2376	2.25	Good	I	VS2	G	VG	GIA	18623
3756	2.27	Very Good	G	VS2	VG	VG	GIA	27252
4248	1.20	Ideal	F	VS1	EX	EX	GIA	9641
...	...	...	...	...	...	...	...	...
1248	0.80	Ideal	E	VS1	VG	VG	GIA	3714
5299	1.02	Very Good	E	SI1	VG	EX	GIA	5424
4979	1.03	Ideal	F	VS2	VG	VG	GIA	6997
933	2.01	Good	H	VS1	VG	VG	GIA	18988
1653	2.01	Very Good	H	VS2	VG	G	GIA	19044

5400 rows × 8 columns

In [6]:
```
data_test = df.drop(data.index)
```

In [7]:
```
data_test
```

Out[7]:

	Carat Weight	Cut	Color	Clarity	Polish	Symmetry	Report	Price
5	1.53	Ideal	E	SI1	ID	ID	AGSL	12791
7	1.50	Fair	F	SI1	VG	VG	GIA	10450
11	1.01	Good	E	SI1	G	G	GIA	5161
13	2.51	Very Good	G	VS2	VG	VG	GIA	34361
18	1.01	Good	I	SI1	VG	VG	GIA	4238
...	...	...	...	...	...	...	...	...
5973	1.02	Very Good	F	SI1	G	G	GIA	5430
5975	2.05	Good	G	VS1	G	VG	GIA	26297
5976	2.01	Very Good	E	VS2	EX	VG	GIA	27002
5995	1.03	Ideal	D	SI1	EX	EX	GIA	6250
5999	2.19	Ideal	E	VS1	EX	EX	GIA	30507

600 rows × 8 columns

In [8]:
```
data.reset_index(drop = True,inplace = True)
```

In [9]:
```
data
```

Out[9]:

	Carat Weight	Cut	Color	Clarity	Polish	Symmetry	Report	Price
0	1.50	Good	E	VS2	VG	G	GIA	12702
1	1.08	Ideal	D	VS2	EX	EX	GIA	8481
2	2.25	Good	I	VS2	G	VG	GIA	18623
3	2.27	Very Good	G	VS2	VG	VG	GIA	27252
4	1.20	Ideal	F	VS1	EX	EX	GIA	9641
...	...	...	...	...	...	...	...	...
5395	0.80	Ideal	E	VS1	VG	VG	GIA	3714
5396	1.02	Very Good	E	SI1	VG	EX	GIA	5424
5397	1.03	Ideal	F	VS2	VG	VG	GIA	6997
5398	2.01	Good	H	VS1	VG	VG	GIA	18988
5399	2.01	Very Good	H	VS2	VG	G	GIA	19044

5400 rows × 8 columns

In [10]:
```
data_test.reset_index(drop=True, inplace=True)
```

In [11]:
```
data_test
```

Out[11]:

	Carat Weight	Cut	Color	Clarity	Polish	Symmetry	Report	Price
0	1.53	Ideal	E	SI1	ID	ID	AGSL	12791
1	1.50	Fair	F	SI1	VG	VG	GIA	10450
2	1.01	Good	E	SI1	G	G	GIA	5161
3	2.51	Very Good	G	VS2	VG	VG	GIA	34361
4	1.01	Good	I	SI1	VG	VG	GIA	4238
...	...	...	...	...	...	...	...	...
595	1.02	Very Good	F	SI1	G	G	GIA	5430
596	2.05	Good	G	VS1	G	VG	GIA	26297
597	2.01	Very Good	E	VS2	EX	VG	GIA	27002
598	1.03	Ideal	D	SI1	EX	EX	GIA	6250
599	2.19	Ideal	E	VS1	EX	EX	GIA	30507

600 rows × 8 columns

In [12]:
```
print('Data for Modeling: ' + str(data.shape))
print('Unseen Data For Predictions ' + str(data_test.shape))
```

```
Data for Modeling: (5400, 8)
Unseen Data For Predictions (600, 8)
```

In [13]:
```
from pycaret.regression import *
```

In [14]:
```
exp_reg102 = setup(data = data, target = 'Price', session_id=123,
                   normalize = True, transformation = True, transform_target =
                   remove_multicollinearity = True, multicollinearity_threshold
                   bin_numeric_features = ['Carat Weight'],
                   remove_outliers = True)
```

	Description	Value
0	Session id	123
1	Target	Price
2	Target type	Regression
3	Original data shape	(5400, 8)
4	Transformed data shape	(5211, 28)
5	Transformed train set shape	(3590, 28)
6	Transformed test set shape	(1621, 28)
7	Ordinal features	1
8	Numeric features	1
9	Categorical features	6
10	Preprocess	True
11	Imputation type	simple
12	Numeric imputation	mean
13	Categorical imputation	mode
14	Maximum one-hot encoding	25
15	Encoding method	None
16	Remove multicollinearity	True
17	Multicollinearity threshold	0.950000
18	Remove outliers	True
19	Outliers threshold	0.050000
20	Transformation	True
21	Transformation method	yeo-johnson
22	Normalize	True
23	Normalize method	zscore
24	Transform target	True
25	Transform target method	yeo-johnson
26	Fold Generator	KFold
27	Fold Number	10
28	CPU Jobs	-1
29	Use GPU	False
30	Log Experiment	False
31	Experiment Name	reg-default-name
32	USI	2389

In [15]:
```
best = compare_models()
```

	Model	MAE	MSE	RMSE	R2
gbr	Gradient Boosting Regressor	0.0067	0.0001	0.0087	0.8565
lightgbm	Light Gradient Boosting Machine	0.0066	0.0001	0.0087	0.8554
rf	Random Forest Regressor	0.0069	0.0001	0.0093	0.8345
ridge	Ridge Regression	0.0081	0.0001	0.0099	0.8146
br	Bayesian Ridge	0.0081	0.0001	0.0099	0.8146
huber	Huber Regressor	0.0078	0.0001	0.0100	0.8093
et	Extra Trees Regressor	0.0073	0.0001	0.0100	0.8092
dt	Decision Tree Regressor	0.0073	0.0001	0.0101	0.8053
ada	AdaBoost Regressor	0.0084	0.0001	0.0102	0.7995
omp	Orthogonal Matching Pursuit	0.0091	0.0001	0.0116	0.7427
knn	K Neighbors Regressor	0.0115	0.0002	0.0154	0.5414
lasso	Lasso Regression	0.0192	0.0005	0.0229	-0.0027
en	Elastic Net	0.0192	0.0005	0.0229	-0.0027
llar	Lasso Least Angle Regression	0.0192	0.0005	0.0229	-0.0027
dummy	Dummy Regressor	0.0192	0.0005	0.0229	-0.0027
lar	Least Angle Regression	0.0105	0.0007	0.0175	-0.2686
par	Passive Aggressive Regressor	0.0284	0.0013	0.0356	-1.4665
lr	Linear Regression	150792324.9108	85950869443310845952.0000	2931737871.0217	-1468918309

In [16]:
```
fit_best_model = create_model('gbr')
```

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	0.0067	0.0001	0.0085	0.8509	0.0024	0.0026
1	0.0068	0.0001	0.0088	0.8613	0.0025	0.0026
2	0.0072	0.0001	0.0093	0.8492	0.0026	0.0028
3	0.0062	0.0001	0.0078	0.8657	0.0022	0.0024
4	0.0067	0.0001	0.0087	0.8419	0.0024	0.0026
5	0.0071	0.0001	0.0090	0.8606	0.0025	0.0027
6	0.0068	0.0001	0.0087	0.8521	0.0024	0.0027
7	0.0049	0.0001	0.0071	0.8833	0.0020	0.0020
8	0.0067	0.0001	0.0087	0.8566	0.0024	0.0026
9	0.0079	0.0001	0.0102	0.8437	0.0028	0.0030
Mean	0.0067	0.0001	0.0087	0.8565	0.0024	0.0026
Std	0.0007	0.0000	0.0008	0.0115	0.0002	0.0003

In [17]:
```
tuned_fit_model = tune_model(fit_best_model)
```

Fold	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	0.0066	0.0001	0.0086	0.8463	0.0024	0.0026
1	0.0069	0.0001	0.0089	0.8579	0.0025	0.0027
2	0.0072	0.0001	0.0093	0.8486	0.0026	0.0028
3	0.0062	0.0001	0.0078	0.8676	0.0022	0.0024
4	0.0067	0.0001	0.0087	0.8409	0.0025	0.0026
5	0.0071	0.0001	0.0090	0.8603	0.0025	0.0027
6	0.0069	0.0001	0.0088	0.8483	0.0025	0.0027
7	0.0051	0.0001	0.0072	0.8792	0.0021	0.0020
8	0.0067	0.0001	0.0088	0.8524	0.0025	0.0026
9	0.0080	0.0001	0.0103	0.8405	0.0028	0.0031
Mean	0.0067	0.0001	0.0087	0.8542	0.0025	0.0026
Std	0.0007	0.0000	0.0008	0.0116	0.0002	0.0002

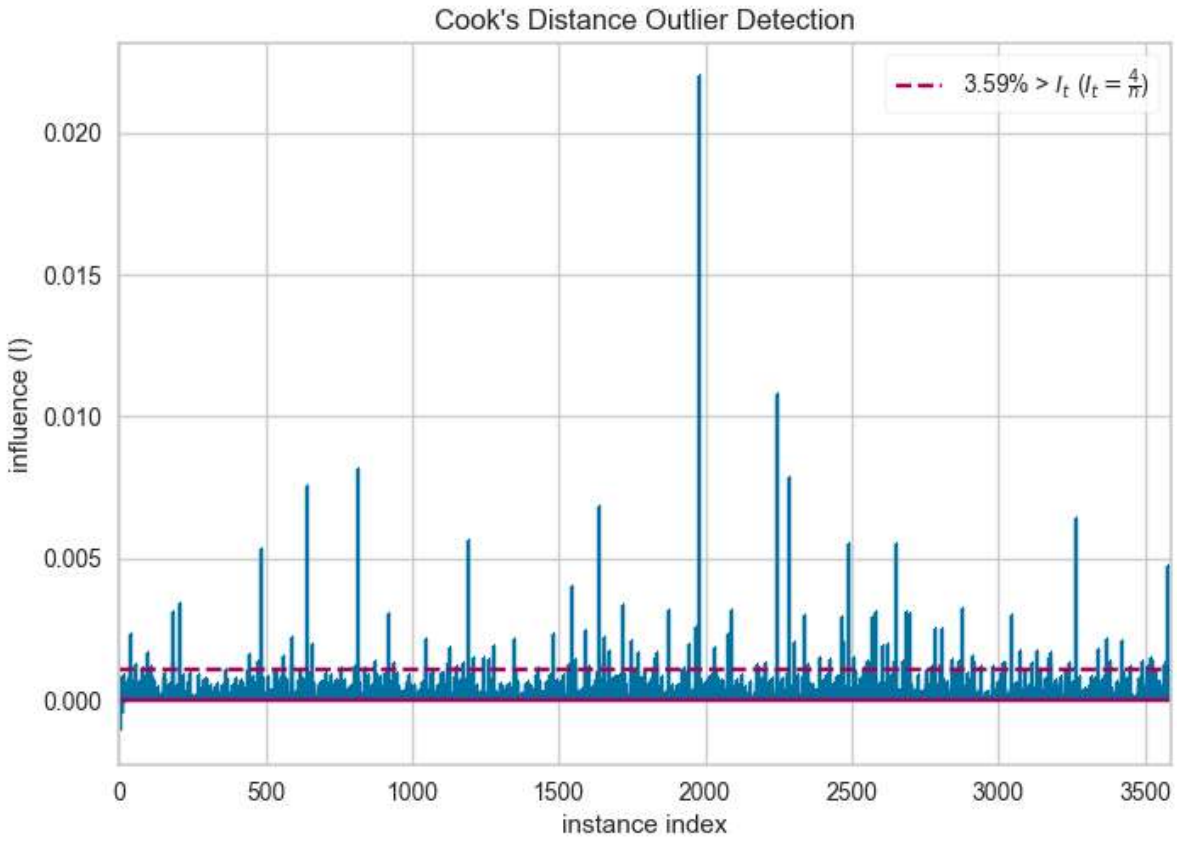
```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Original model was better than the tuned model, hence it will be returned. NO
TE: The display metrics are for the tuned model (not the original one).
```

In [18]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
fig = plt.gcf()
fig.set_size_inches(5, 3.5, forward=True)

plot_model(tuned_fit_model, plot= 'cooks', save=True )
```

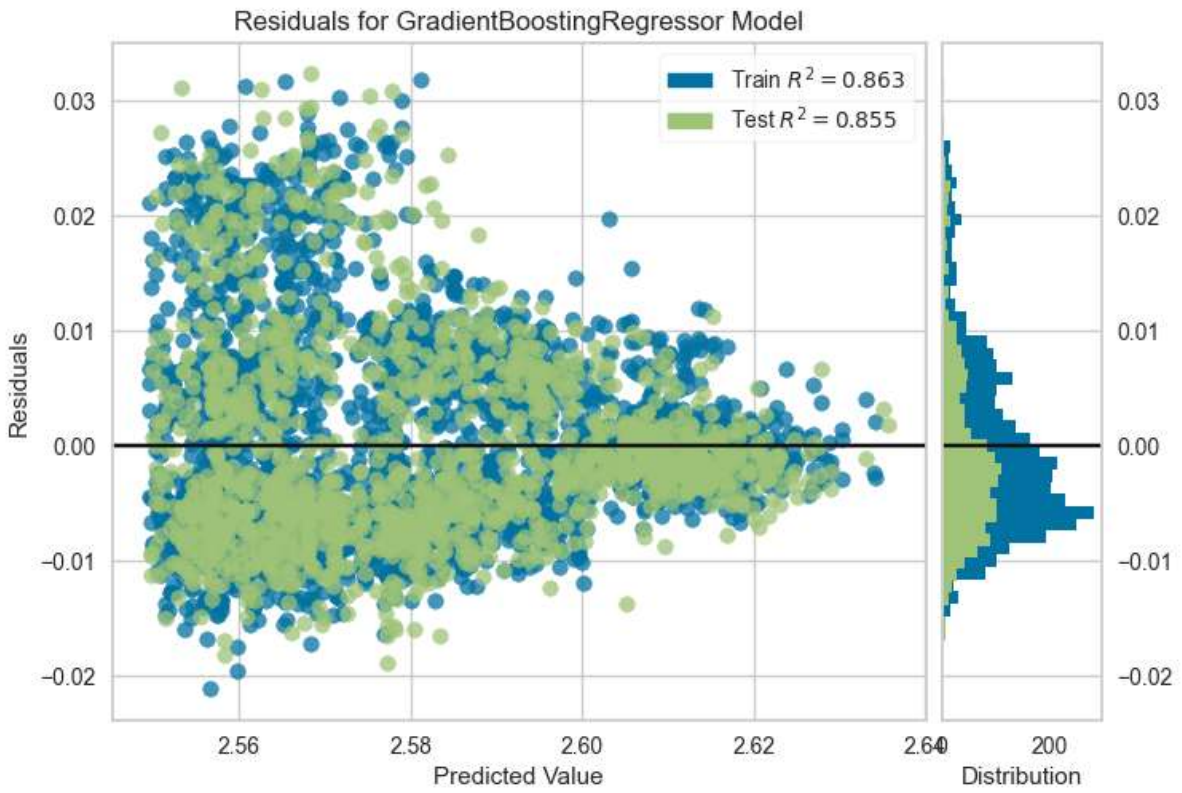
In [19]:
```
plot_model(tuned_fit_model, plot= 'cooks')
```




```
import matplotlib.pyplot as plt
fig = plt.gcf()
fig.set_size_inches(4, 3.5, forward=True)

plot_model(tuned_fit_model, save=True, plot = 'residuals' )
```

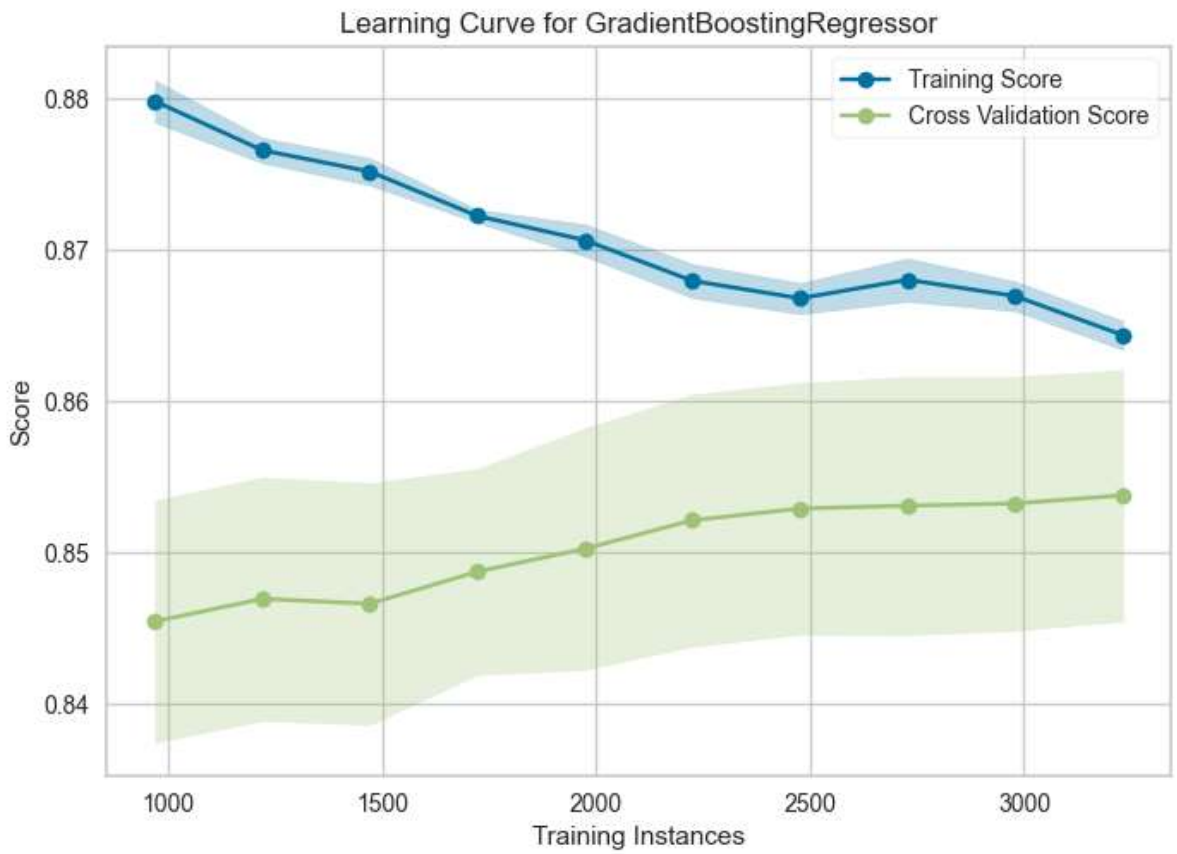
In [20]:
```
plot_model(tuned_fit_model, plot = 'residuals' )
```



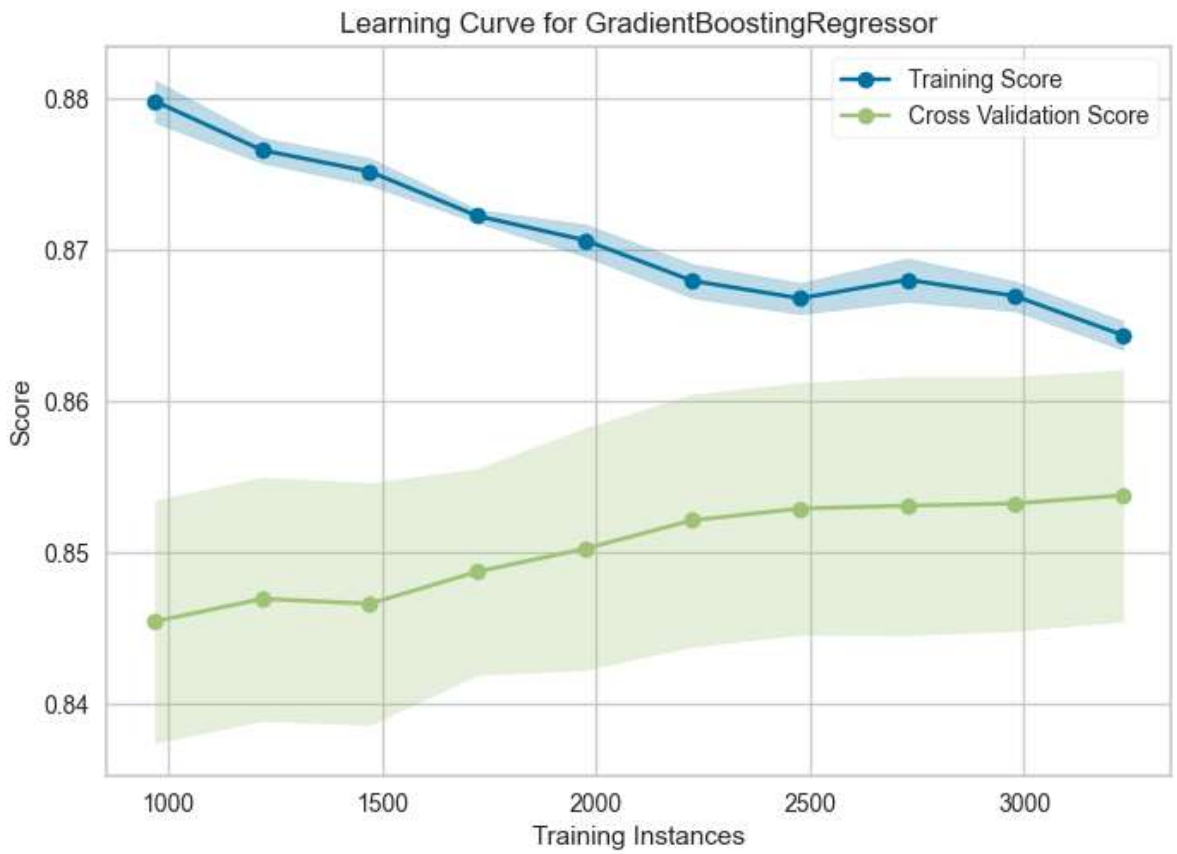

```
import matplotlib.pyplot as plt
fig = plt.gcf()
fig.set_size_inches(5, 3.5, forward=True)

plot_model(fit_best_model, plot = 'learning', save=True )
```

In [21]:
```
plot_model(tuned_fit_model, plot = 'learning')
```



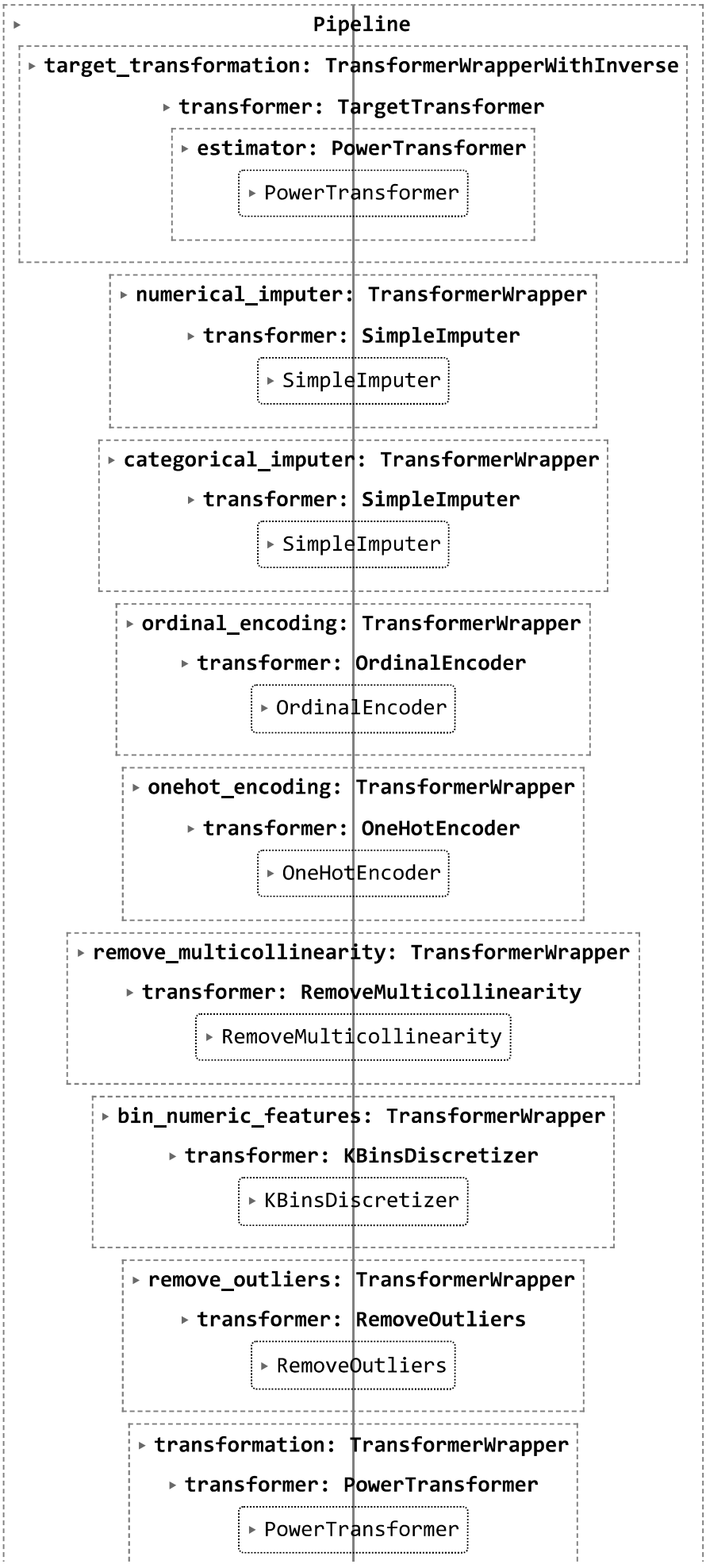
In [22]:
```
plot_model(best, plot = 'learning')
```

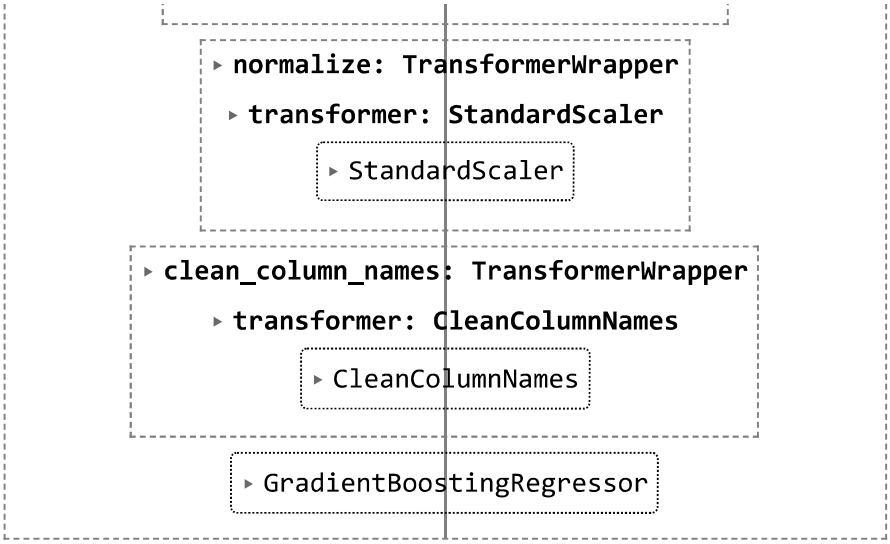


In [23]:
```
final_model = finalize_model(tuned_fit_model)
```

In [24]:
```
final_model
```

Out[24]:


Pipeline
target_transformation: TransformerWrapperWithInverse
transformer: TargetTransformer
estimator: PowerTransformer
PowerTransformer
numerical_imputer: TransformerWrapper
transformer: SimpleImputer
SimpleImputer
categorical_imputer: TransformerWrapper
transformer: SimpleImputer
SimpleImputer
ordinal_encoding: TransformerWrapper
transformer: OrdinalEncoder
OrdinalEncoder
onehot_encoding: TransformerWrapper
transformer: OneHotEncoder
OneHotEncoder
remove_multicollinearity: TransformerWrapper
transformer: RemoveMulticollinearity
RemoveMulticollinearity
bin_numeric_features: TransformerWrapper
transformer: KBinsDiscretizer
KBinsDiscretizer
remove_outliers: TransformerWrapper
transformer: RemoveOutliers
RemoveOutliers
transformation: TransformerWrapper
transformer: PowerTransformer
PowerTransformer



In [25]:
```
print(final_model)
```

```
Pipeline(memory=Memory(location=None),
         steps=[('target_transformation',
                 TransformerWrapperWithInverse(transformer=TargetTransformer
(estimator=PowerTransformer(standardize=False)))),
                ('numerical_imputer',
                 TransformerWrapper(include=['Carat Weight'],
                                    transformer=SimpleImputer())),
                ('categorical_imputer',
                 TransformerWrapper(include=['Cut', 'Color', 'Clarity',
                                             'P...
                 TransformerWrapper(transformer=RemoveOutliers(random_state=1
23))),
                ('transformation',
                 TransformerWrapper(transformer=PowerTransformer(standardize=
False))),
                ('normalize', TransformerWrapper(transformer=StandardScaler
())),
                ('clean_column_names',
                 TransformerWrapper(transformer=CleanColumnNames())),
                ('actual_estimator',
                 GradientBoostingRegressor(random_state=123))])
```

In [26]:
```
save_model(final_model,'regresion_best_model_ejercicio')
```

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

Out[26]:
```
(Pipeline(memory=Memory(location=None),
          steps=[('target_transformation',
                  TransformerWrapperWithInverse(transformer=TargetTransformer
(estimator=PowerTransformer(standardize=False)))),
                 ('numerical_imputer',
                  TransformerWrapper(include=['Carat Weight'],
                                     transformer=SimpleImputer())),
                 ('categorical_imputer',
                  TransformerWrapper(include=['Cut', 'Color', 'Clarity',
                                              'P...
                  TransformerWrapper(transformer=RemoveOutliers(random_state=
123))),
                 ('transformation',
                  TransformerWrapper(transformer=PowerTransformer(standardize
=False))),
                 ('normalize', TransformerWrapper(transformer=StandardScaler
())),
                 ('clean_column_names',
                  TransformerWrapper(transformer=CleanColumnNames())),
                 ('actual_estimator',
                  GradientBoostingRegressor(random_state=123))]),
 'regresion_best_model_ejercicio.pkl')
```