diamonds_machine_learning_regression

June 7, 2020

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as st
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from xgboost import XGBRFRegressor
```

[2]:
```
diamonds = pd.read_csv("dataset/diamonds.csv", index_col=0)
```

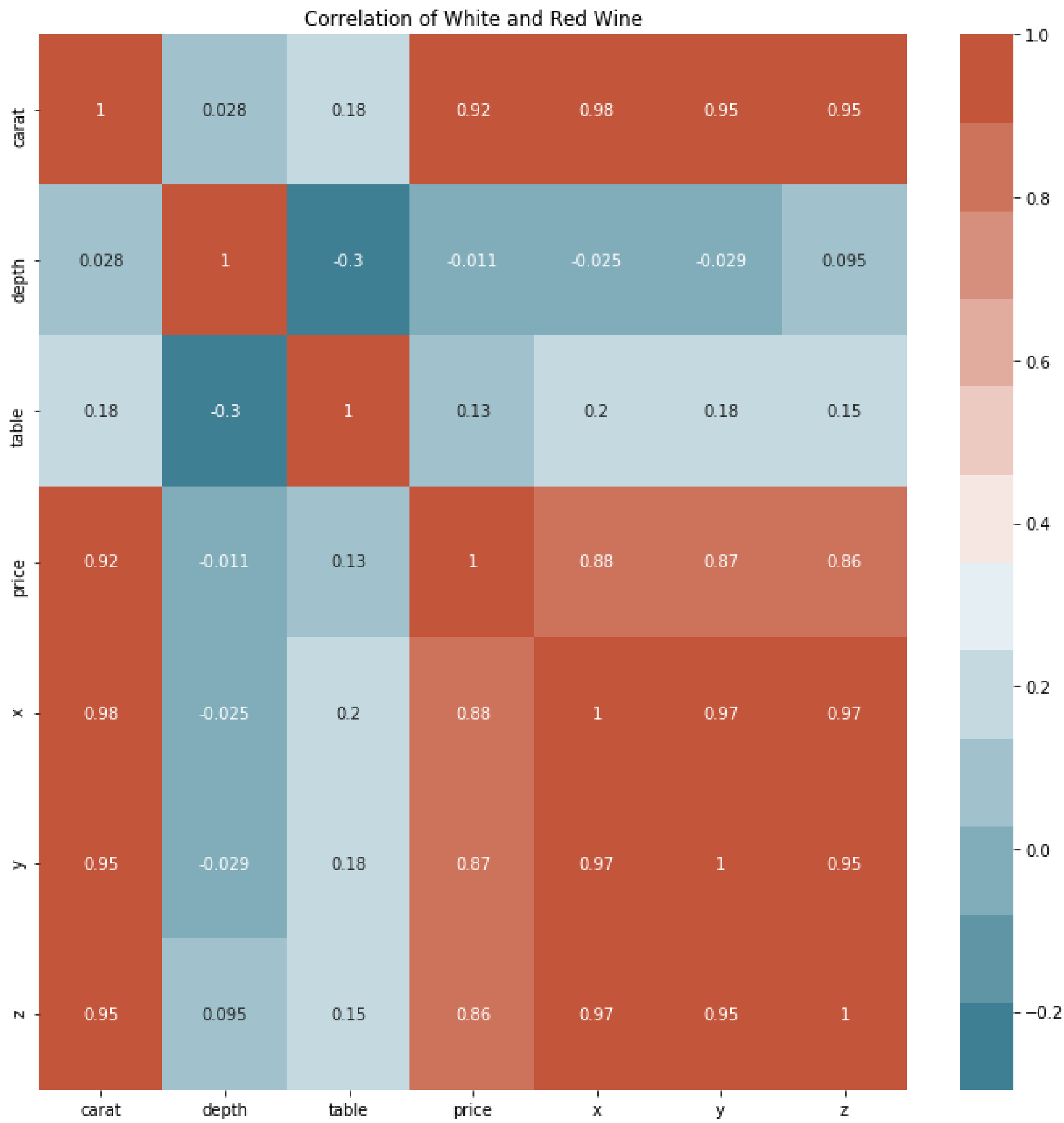
[3]:
```
diamonds
```

[3]:

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
…	…	…	…	…	…	…	…	…	…	…
53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56

```
53939    0.86     Premium     H      SI2    61.0    58.0    2757   6.15   6.12   3.74
53940    0.75       Ideal     D      SI2    62.2    55.0    2757   5.83   5.87   3.64

[53940 rows x 10 columns]
```

[4]:
```python
plt.figure(figsize=(12, 12))
df_corr = diamonds.corr()
sns.heatmap(df_corr, cmap=sns.diverging_palette(220, 20, n=12), annot=True)
plt.title("Correlation of White and Red Wine")
plt.show()
```



[5]:
```
diamonds.loc[diamonds['cut'] == 'Fair', 'cut'] = 1
diamonds.loc[diamonds['cut'] == 'Good', 'cut'] = 2
diamonds.loc[diamonds['cut'] == 'Very Good', 'cut'] = 3
diamonds.loc[diamonds['cut'] == 'Premium', 'cut'] = 4
diamonds.loc[diamonds['cut'] == 'Ideal', 'cut'] = 5
```

[6]:
```
diamonds.loc[diamonds['color'] == 'J', 'color'] = 1
diamonds.loc[diamonds['color'] == 'I', 'color'] = 2
diamonds.loc[diamonds['color'] == 'H', 'color'] = 3
diamonds.loc[diamonds['color'] == 'G', 'color'] = 4
diamonds.loc[diamonds['color'] == 'F', 'color'] = 5
diamonds.loc[diamonds['color'] == 'E', 'color'] = 6
diamonds.loc[diamonds['color'] == 'D', 'color'] = 7
```

[7]:
```
diamonds.loc[diamonds['clarity'] == 'I1', 'clarity'] = 1
diamonds.loc[diamonds['clarity'] == 'SI2', 'clarity'] = 2
diamonds.loc[diamonds['clarity'] == 'SI1', 'clarity'] = 3
diamonds.loc[diamonds['clarity'] == 'VS2', 'clarity'] = 4
diamonds.loc[diamonds['clarity'] == 'VS1', 'clarity'] = 5
diamonds.loc[diamonds['clarity'] == 'VVS2', 'clarity'] = 6
diamonds.loc[diamonds['clarity'] == 'VVS1', 'clarity'] = 7
diamonds.loc[diamonds['clarity'] == 'IF', 'clarity'] = 8
```

[8]:
```
X = diamonds[['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'x',␣
↪'y','z']]
y = diamonds[["price"]].values
```

[9]:
```
scalar = StandardScaler()
X = scalar.fit_transform(X)
```

[10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2,␣
↪random_state= 42)
```

[11]:
```
Model = []
RMSE = []
MAE = []
MSE = []
R_Square = []
adj_rsquared = []
CV = []
```

[12]:
```
names = ["Linear Regression", "Ridge Regression", "Lasso Regression",
         "Decision Tree Regressor", "Random Forest Regressor", "Gradient␣
↪Boosting Regressor",
         "Adaboost Regressor", "BaggingRegressor",␣
↪"ExtraTreesRegressor","XGBRegressor", "XGBRFRegressor"]
models = [LinearRegression(), Ridge(), Lasso(), DecisionTreeRegressor(),
```

```
          RandomForestRegressor(), GradientBoostingRegressor(),
          AdaBoostRegressor(), BaggingRegressor(),␣
↪ExtraTreesRegressor(),XGBRegressor(), XGBRFRegressor()]
```

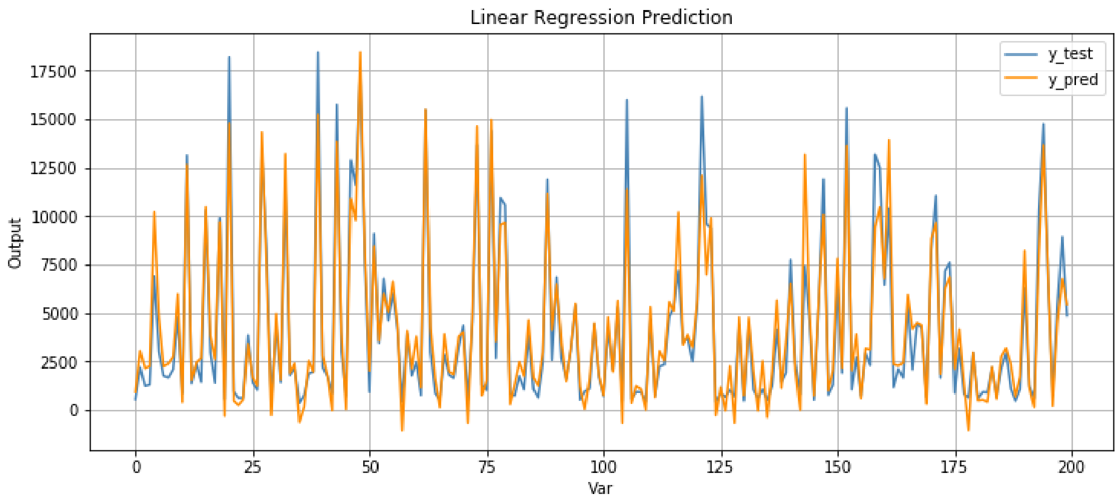
[13]:
```
def evaluate(true, predicted, variable_of_model):
    MAE.append(metrics.mean_absolute_error(true, predicted))
    MSE.append(metrics.mean_squared_error(true, predicted))
    RMSE.append(np.sqrt(metrics.mean_squared_error(true, predicted)))
    R_Square.append(metrics.r2_score(true, predicted))
    n= X_test.shape[0]
    p= X_test.shape[1] - 1
    adj_rsquared.append(1 - (1 - R_Square[-1]) * ((n - 1)/(n-p-1)))
    cv_accuracies = cross_val_score(estimator = variable_of_model, X = X_train,␣
↪y = y_train.ravel(), cv = 5,verbose = 1)
    CV.append(cv_accuracies.mean())
```

[14]:
```
def print_evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    n= X_test.shape[0]
    p= X_test.shape[1] - 1
    adj_rsquared = 1 - (1 - r2_square) * ((n - 1)/(n-p-1))
    print("MAE:", mae)
    print("MSE:", mse)
    print("RMSE:", rmse)
    print("R2 Square", r2_square)
    print("adj R Square", adj_rsquared)
```

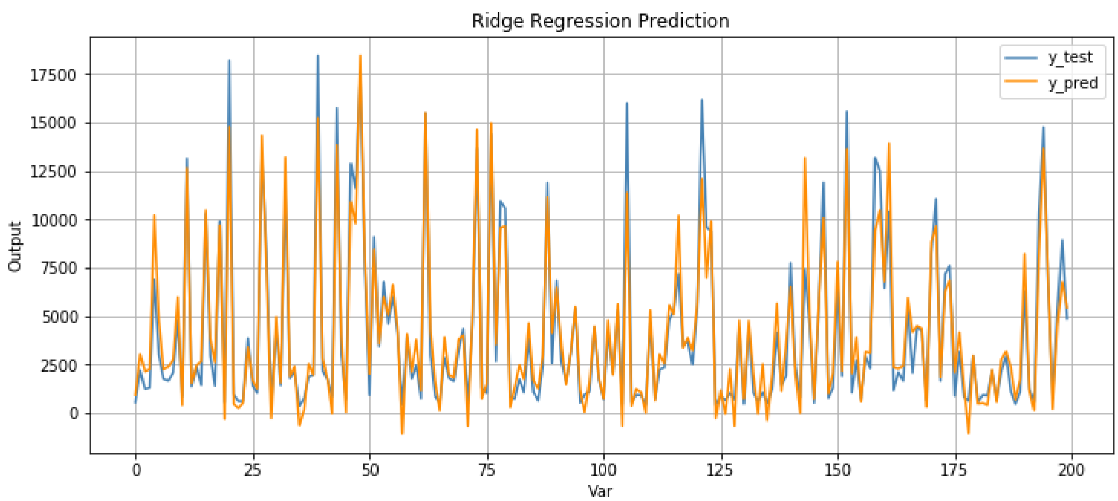
[15]:
```
def pred_vis(name, y_test_vis, y_pred_vis):
    if y_test_vis.shape[0] > 200:
        y_test_vis = y_test_vis[:200]
        y_pred_vis = y_pred_vis[:200]

    y_test_m_vis = y_test_vis
    plt.figure(figsize=(12,5))
    plt.title("{} Prediction" .format(name))
    plt.plot(y_test_m_vis, c="steelblue", alpha=1)
    plt.plot(y_pred_vis, c="darkorange", alpha=2)
    legend_list = ["y_test", "y_pred"]
    plt.xlabel("Var")
    plt.ylabel("Output")
    plt.legend(legend_list, loc=1, fontsize="10")
    plt.grid(True)
    plt.show()
```

[16]:
```
def fit_and_predict(name, model):
    variable_of_model = model
    variable_of_model.fit(X_train, y_train.ravel())
    pred = variable_of_model.predict(X_test)
    pred_vis(name, y_test, pred)
    evaluate(y_test, pred, variable_of_model)
```

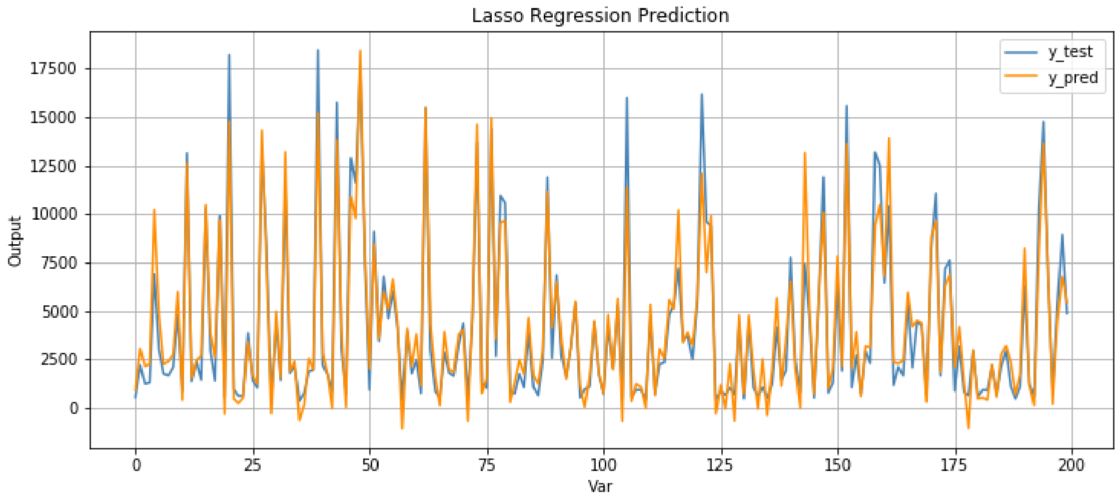
[17]:
```
for name, model in zip(names, models):
    fit_and_predict(name, model)
```



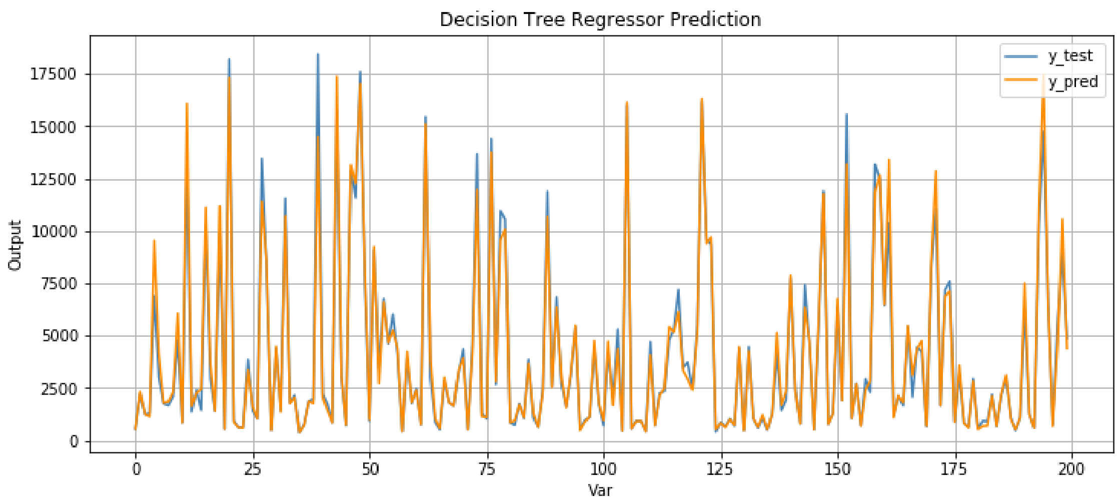

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
```



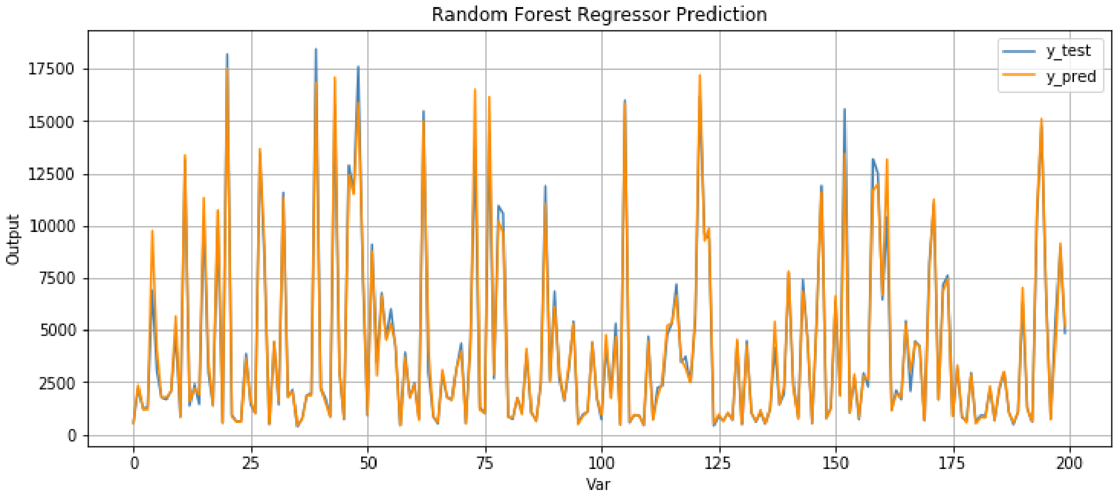
```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
```



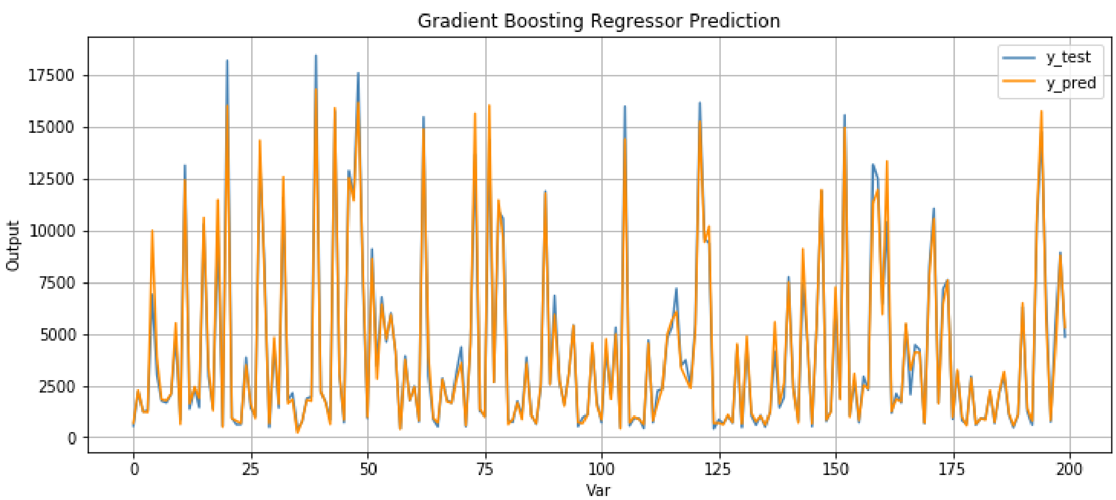

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.4s finished
```



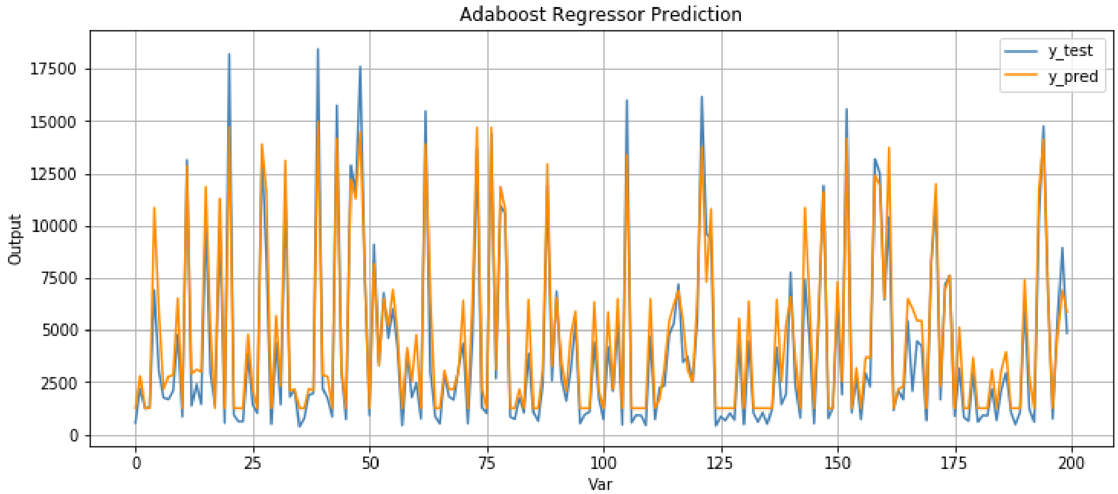

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    2.6s finished
```



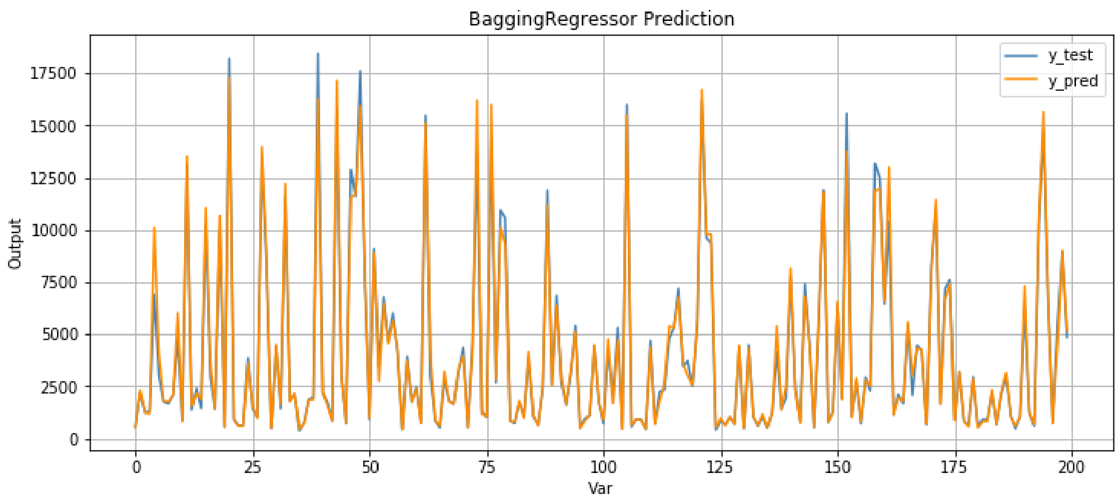

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  2.4min finished
```



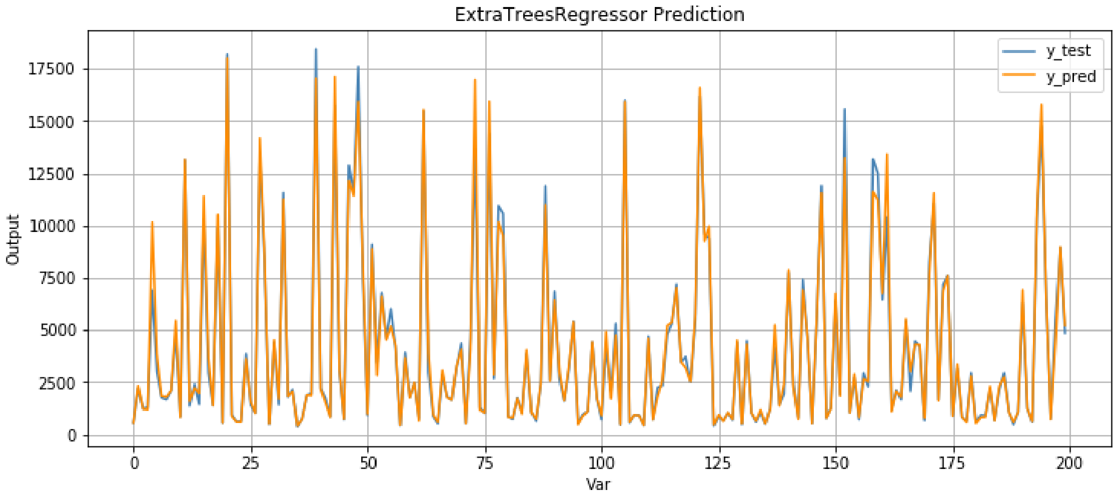

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   46.9s finished
```



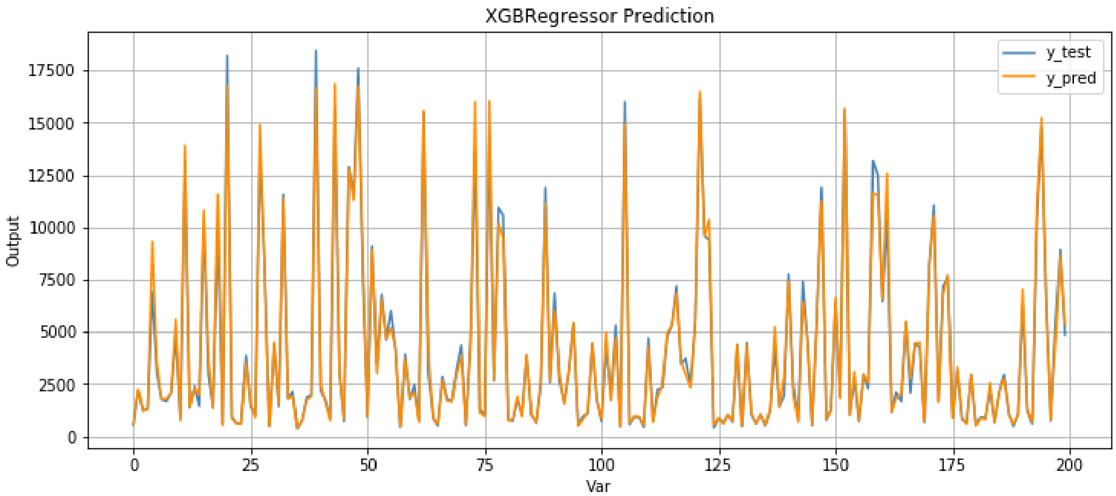

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   30.0s finished
```



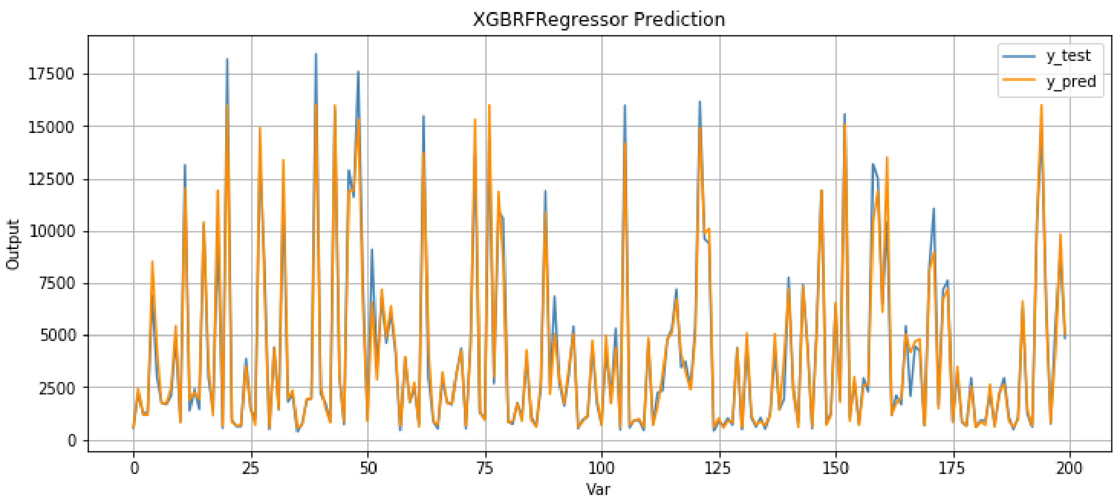

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   18.8s finished
```




```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  2.0min finished
```




```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   17.3s finished
```




```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   13.4s finished
```

[18]:
```
evaluation_dataframe = pd.DataFrame({"Model": names,
                                     "MAE": MAE,
                                     "MSE": MSE,
                                     "RMSE": RMSE,
                                     "R Squared": R_Square,
                                     "adj R Squared": adj_rsquared,
                                     "Cross Validation": CV})
```

[19]:
```
evaluation_dataframe = evaluation_dataframe.sort_values("adj R Squared")
```

[20]:
```
evaluation_dataframe
```

[20]:
```
                           Model         MAE           MSE         RMSE  \
2               Lasso Regression  806.528249  1.499715e+06  1224.628491
0              Linear Regression  805.274366  1.499637e+06  1224.596542
1               Ridge Regression  805.364487  1.499618e+06  1224.588886
6             Adaboost Regressor  891.226303  1.360682e+06  1166.482571
10                XGBRFRegressor  436.879141  5.911960e+05   768.892728
3        Decision Tree Regressor  348.536893  5.225054e+05   722.845352
5    Gradient Boosting Regressor  336.630566  3.712598e+05   609.310892
7               BaggingRegressor  280.343445  3.273823e+05   572.173327
8            ExtraTreesRegressor  263.479970  2.954383e+05   543.542346
9                   XGBRegressor  274.684170  2.936233e+05   541.870230
4        Random Forest Regressor  266.231085  2.923981e+05   540.738519

    R Squared  adj R Squared  Cross Validation
2    0.905659       0.905589          0.907182
```

0	0.905664	0.905594	0.907122
1	0.905666	0.905596	0.907124
6	0.914405	0.914342	0.921999
10	0.962810	0.962783	0.961185
3	0.967131	0.967107	0.964216
5	0.976646	0.976628	0.975199
7	0.979406	0.979391	0.979515
8	0.981415	0.981401	0.981022
9	0.981529	0.981516	0.980844
4	0.981607	0.981593	0.981088

[ ]:

```
```

[ ]:

```
```

[ ]:

```
```

[ ]:

```
```