FinalProject

June 15, 2022

[ ]:
```
import pandas as pd
import numpy as np
import sklearn as sk
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder,StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.kernel_ridge import KernelRidge
from sklearn.metrics import accuracy_score,mean_squared_error,r2_score
from sklearn.linear_model import Ridge,RidgeCV,Lasso,LassoCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
import tensorflow as tf
import keras as ks
from keras.layers import Dense
from keras.utils import np_utils
from keras.models import Sequential
```

[ ]:
```
def compare(a,b):
    for i,k in zip(a,b):
        print(i,k)

def compare_plot(pred,test,fig_name):
    score = r2_score(test,pred)
    plt.scatter(np.arange(len(pred[1:-1:150])),pred[1:-1:150],marker="x",label␣
 ↪= "Predicted Data",color="blue")
    plt.scatter(np.arange(len(test[1:-1:150])),test[1:-1:150],marker="P",label␣
 ↪= "Real Data",color ="darkorange")

    ax =plt.gca()
    ax.axes.xaxis.set_ticks([])

    plt.legend()
    plt.ylabel("Atomization Energy (eV)",rotation=90)
    plt.xlabel("Target Variable Data Points")
    plt.title(r"$R^2$ ="+" "+str(score)[:6]+"\n"+str(fig_name))
```

```
    plt.savefig(str(fig_name)+".png")
    plt.show()

minmax = MinMaxScaler()
standard = StandardScaler()
original_data = pd.read_csv("roboBohr.csv")
original_data
```

[ ]:

	Unnamed: 0	0	1	2	3	4	\
0	0	73.516695	17.817765	12.469551	12.458130	12.454607	
1	1	73.516695	20.649126	18.527789	17.891535	17.887995	
2	2	73.516695	17.830377	12.512263	12.404775	12.394493	
3	3	73.516695	17.875810	17.871259	17.862402	17.850920	
4	4	73.516695	17.883818	17.868256	17.864221	17.818540	
…	…	…	…	…	…	…	
16237	16268	73.516695	20.753166	18.624076	17.872009	17.851690	
16238	16269	73.516695	20.724740	18.579933	17.741621	14.716676	
16239	16270	53.358707	20.820797	19.150234	19.148721	15.135514	
16240	16271	53.358707	15.707759	15.707644	13.653838	13.653570	
16241	16272	53.358707	15.708752	15.708094	13.653893	13.653176	

	5	6	7	8	…	1267	1268	1269	\
0	12.447345	12.433065	12.426926	12.387474	…	0.0	0.0	0.5	
1	17.871731	17.852586	17.729842	15.864270	…	0.0	0.0	0.0	
2	12.391564	12.324461	12.238106	10.423249	…	0.0	0.0	0.0	
3	17.850440	12.558105	12.557645	12.517583	…	0.0	0.0	0.0	
4	12.508657	12.490519	12.450098	10.597068	…	0.0	0.0	0.0	
…	…	…	…	…	…	…	…	…	
16237	17.851254	17.742176	14.655754	12.706683	…	0.0	0.0	0.0	
16238	13.697829	13.697558	13.653512	13.652942	…	0.0	0.0	0.0	
16239	15.123685	12.942704	12.938162	12.488633	…	0.0	0.0	0.0	
16240	13.653314	13.652591	13.652585	13.652550	…	0.0	0.0	0.0	
16241	13.653120	13.652930	13.652528	13.652322	…	0.0	0.0	0.0	

	1270	1271	1272	1273	1274	pubchem_id	Eat
0	0.0	0.0	0.0	0.0	0.0	25004	-19.013763
1	0.0	0.0	0.0	0.0	0.0	25005	-10.161019
2	0.0	0.0	0.0	0.0	0.0	25006	-9.376619
3	0.0	0.0	0.0	0.0	0.0	25009	-13.776438
4	0.0	0.0	0.0	0.0	0.0	25011	-8.537140
…	…	…	…	…	…	…	…
16237	0.0	0.0	0.0	0.0	0.0	74976	-8.876123
16238	0.0	0.0	0.0	0.0	0.0	74977	-13.105268
16239	0.0	0.0	0.0	0.0	0.0	74978	-16.801464
16240	0.0	0.0	0.0	0.0	0.0	74979	-13.335088
16241	0.0	0.0	0.0	0.0	0.0	74980	-13.336696

[16242 rows x 1278 columns]

[ ]:
```
data = original_data.drop(columns=["Unnamed: 0","pubchem_id"])
data
```

[ ]:

	0	1	2	3	4	5	\
0	73.516695	17.817765	12.469551	12.458130	12.454607	12.447345	
1	73.516695	20.649126	18.527789	17.891535	17.887995	17.871731	
2	73.516695	17.830377	12.512263	12.404775	12.394493	12.391564	
3	73.516695	17.875810	17.871259	17.862402	17.850920	17.850440	
4	73.516695	17.883818	17.868256	17.864221	17.818540	12.508657	
…	…	…	…	…	…	…	
16237	73.516695	20.753166	18.624076	17.872009	17.851690	17.851254	
16238	73.516695	20.724740	18.579933	17.741621	14.716676	13.697829	
16239	53.358707	20.820797	19.150234	19.148721	15.135514	15.123685	
16240	53.358707	15.707759	15.707644	13.653838	13.653570	13.653314	
16241	53.358707	15.708752	15.708094	13.653893	13.653176	13.653120	

	6	7	8	9	…	1266	1267	1268	\
0	12.433065	12.426926	12.387474	12.365984	…	0.0	0.0	0.0	
1	17.852586	17.729842	15.864270	15.227643	…	0.0	0.0	0.0	
2	12.324461	12.238106	10.423249	8.698826	…	0.0	0.0	0.0	
3	12.558105	12.557645	12.517583	12.444141	…	0.0	0.0	0.0	
4	12.490519	12.450098	10.597068	10.595914	…	0.0	0.0	0.0	
…	…	…	…	…	…	…	…	…	
16237	17.742176	14.655754	12.706683	12.557785	…	0.0	0.0	0.0	
16238	13.697558	13.653512	13.652942	13.652387	…	0.0	0.0	0.0	
16239	12.942704	12.938162	12.488633	12.488061	…	0.0	0.0	0.0	
16240	13.652591	13.652585	13.652550	12.743890	…	0.0	0.0	0.0	
16241	13.652930	13.652528	13.652322	12.743688	…	0.0	0.0	0.0	

	1269	1270	1271	1272	1273	1274	Eat
0	0.5	0.0	0.0	0.0	0.0	0.0	-19.013763
1	0.0	0.0	0.0	0.0	0.0	0.0	-10.161019
2	0.0	0.0	0.0	0.0	0.0	0.0	-9.376619
3	0.0	0.0	0.0	0.0	0.0	0.0	-13.776438
4	0.0	0.0	0.0	0.0	0.0	0.0	-8.537140
…	…	…	…	…	…	…	…
16237	0.0	0.0	0.0	0.0	0.0	0.0	-8.876123
16238	0.0	0.0	0.0	0.0	0.0	0.0	-13.105268
16239	0.0	0.0	0.0	0.0	0.0	0.0	-16.801464
16240	0.0	0.0	0.0	0.0	0.0	0.0	-13.335088
16241	0.0	0.0	0.0	0.0	0.0	0.0	-13.336696

[16242 rows x 1276 columns]

[ ]:
```
X = data.iloc[:,:-1]
y = data.iloc[:,-1].to_numpy()
```

[ ]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
↪20,random_state=42)

X_train_std = standard.fit_transform(X_train)
X_test_std = standard.fit_transform(X_test)

X_train_mm = minmax.fit_transform(X_train)
X_test_mm = minmax.fit_transform(X_test)
```

[ ]:
```
alphas = [0.1,0.2,0.5,1,10,20,50,100,200,500]
alphas_log = np.logspace(-4, 4, 14)
```

1 Kernel Ridge

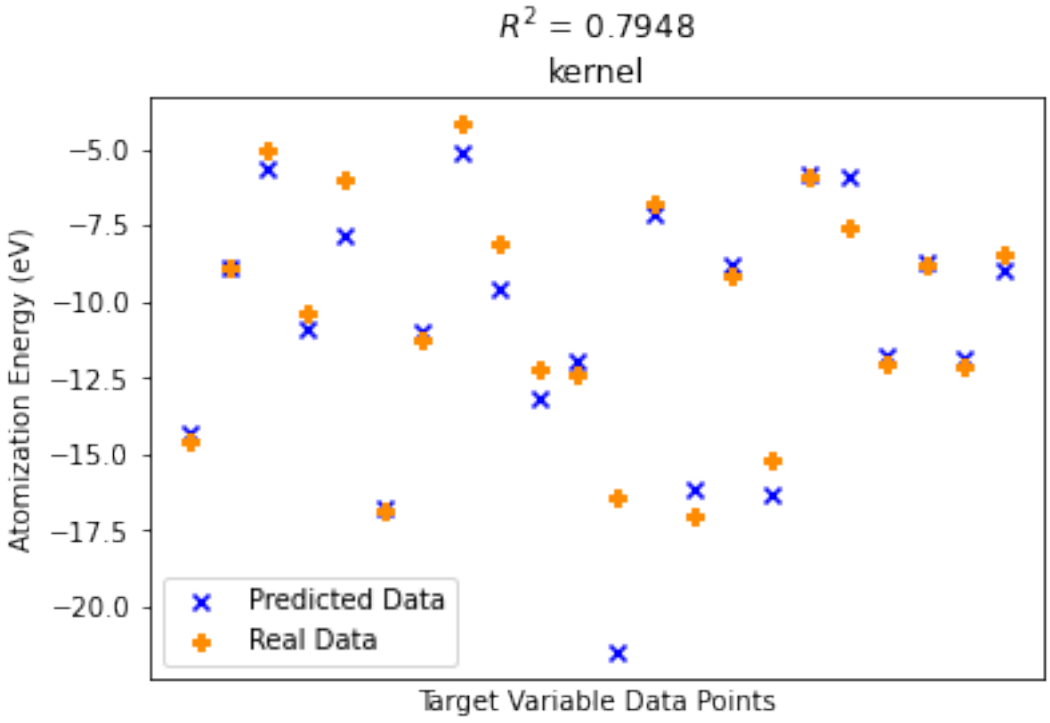
1.1 Original Data

[ ]:
```
def Kernel(X_train,y_train,X_test,y_test):
    global y_pred_kr
    model_kernel = KernelRidge(alpha=20.0)
    model_kernel.fit(X_train,y_train)
    y_pred_kr = model_kernel.predict(X_test)
    kr_mse = mean_squared_error(y_test,y_pred_kr)
    kr_score = r2_score(y_test,y_pred_kr)
    print("Score=",kr_score,"\n","Error=",kr_mse)
# 150, 1.80
# 2000, 1.65
# 3000, 1.6590
```

[ ]:
```
Kernel(X_train,y_train,X_test,y_test)
```

```
Score= 0.7948075781967459
 Error= 2.7851104680719776
```

[ ]:
```
compare_plot(y_pred_kr,y_test,"kernel")
```

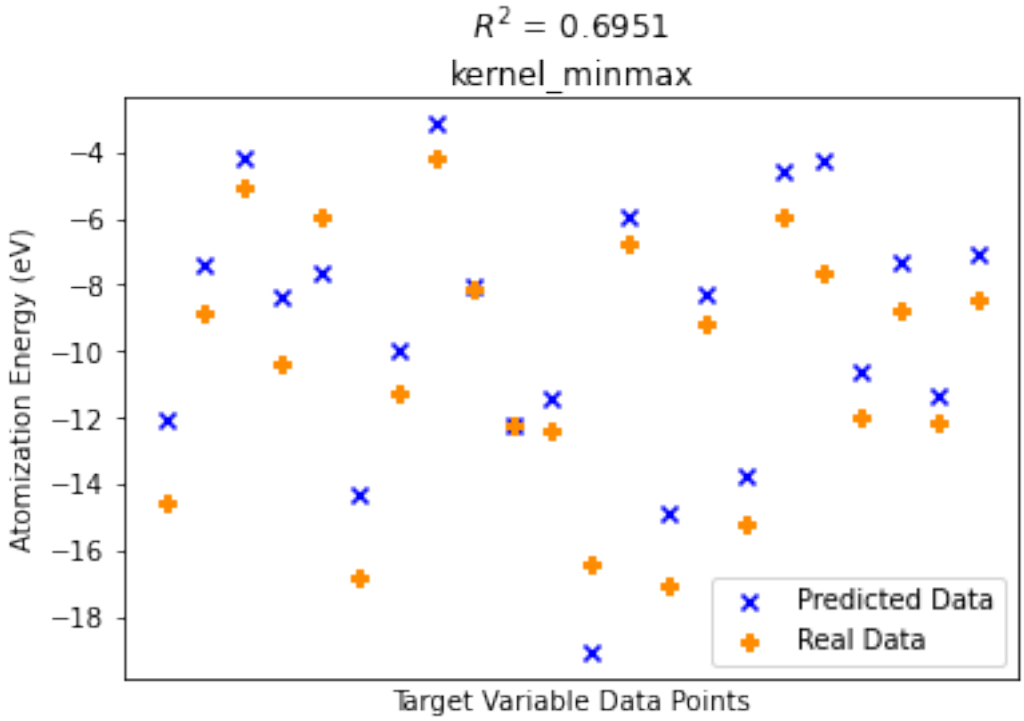


1.2 MinMax Data

[ ]:
```
Kernel(X_train_mm,y_train,X_test_mm,y_test)
```

```
Score= 0.6951060090029014
 Error= 4.138376254423622
```

[ ]:
```
compare_plot(y_pred_kr,y_test,"kernel_minmax")
```

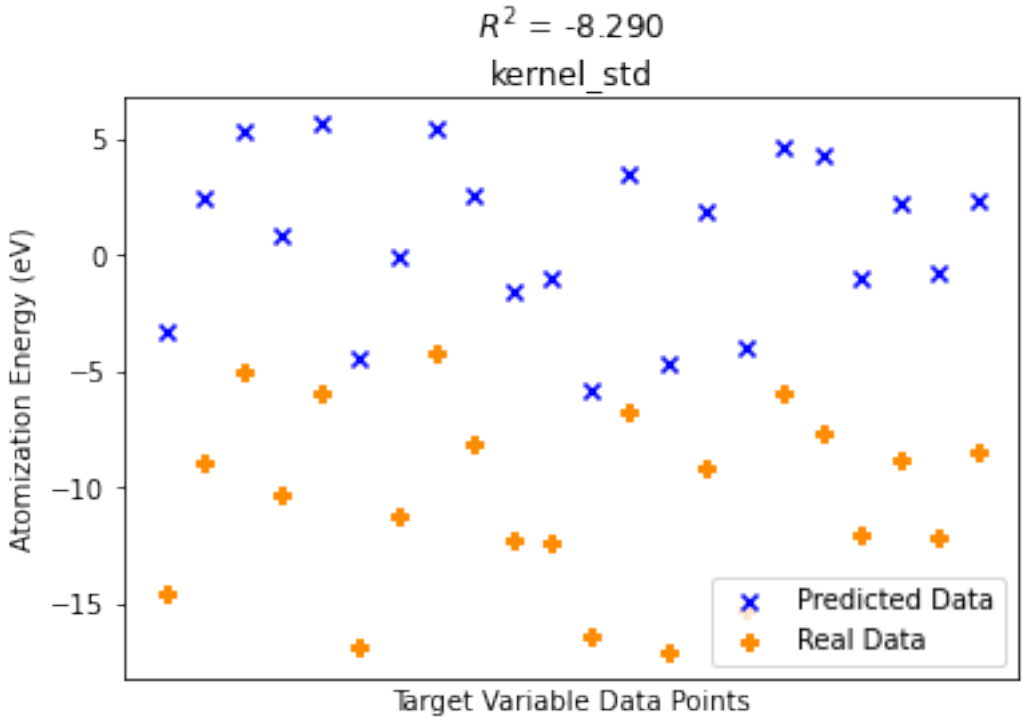


1.3 Std. Scaled Data

[ ]:
```
Kernel(X_train_std,y_train,X_test_std,y_test)
```

```
Score= -8.290268705239127
 Error= 126.09834415313995
```

[ ]:
```
compare_plot(y_pred_kr,y_test,"kernel_std")
```



2 Ridge Regression

2.1 Original Data

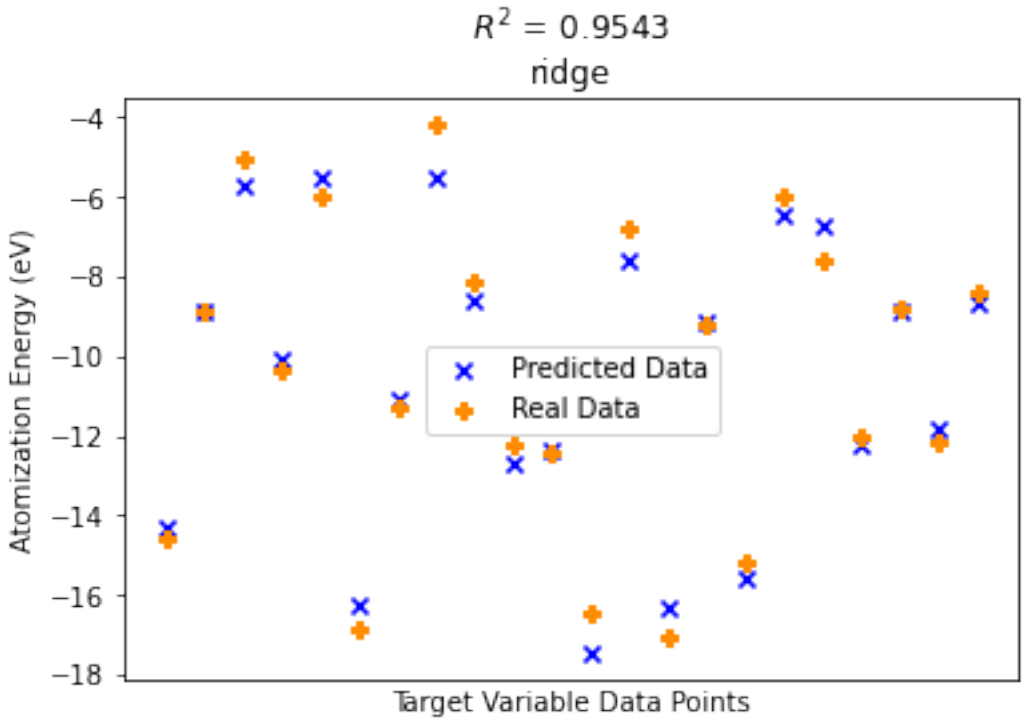
[ ]:
```
def RidgeExp(X_train,X_test,alpha_val):
    global y_pred_ridge
    model_ridge = Ridge(alpha=alpha_val)
    model_ridge.fit(X_train,y_train)
    y_pred_ridge = model_ridge.predict(X_test)
    ridge_error = mean_squared_error(y_test,y_pred_ridge)
    ridge_score = r2_score(y_test,y_pred_ridge)
    print("Score=",ridge_score,"\n","Error=",ridge_error)
```

[ ]:
```
#for i in alphas_log:
#    print(i)
#    RidgeExp(i)

# Selected Alpha Values : 587.8016072274924
#                         2424.462017082331

# 587.80 is the best.
```

[ ]:
```
RidgeExp(X_train,X_test,587.80)
```

```
Score= 0.9543415684921968
 Error= 0.619729395611205
```

[ ]:
```
compare_plot(y_pred_ridge,y_test,"ridge")
```

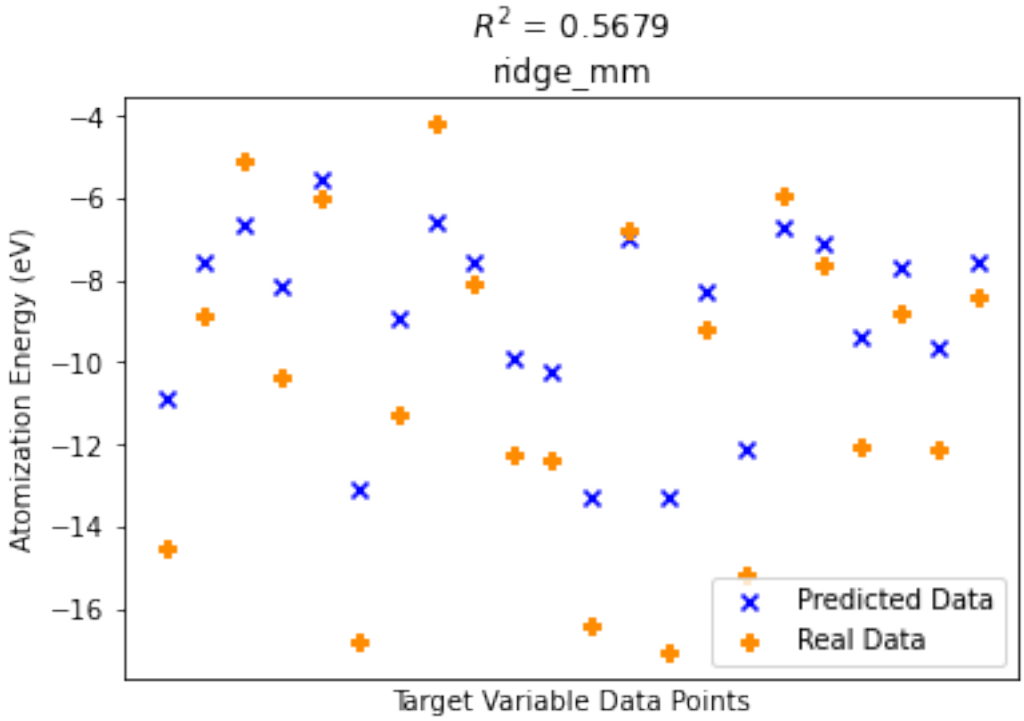


2.2 MinMax Data

[ ]:
```
RidgeExp(X_train_mm,X_test_mm,587.80)
```

```
Score= 0.5679596804313025
 Error= 5.864154270832137
```

[ ]:
```
compare_plot(y_pred_ridge,y_test,"ridge_mm")
```

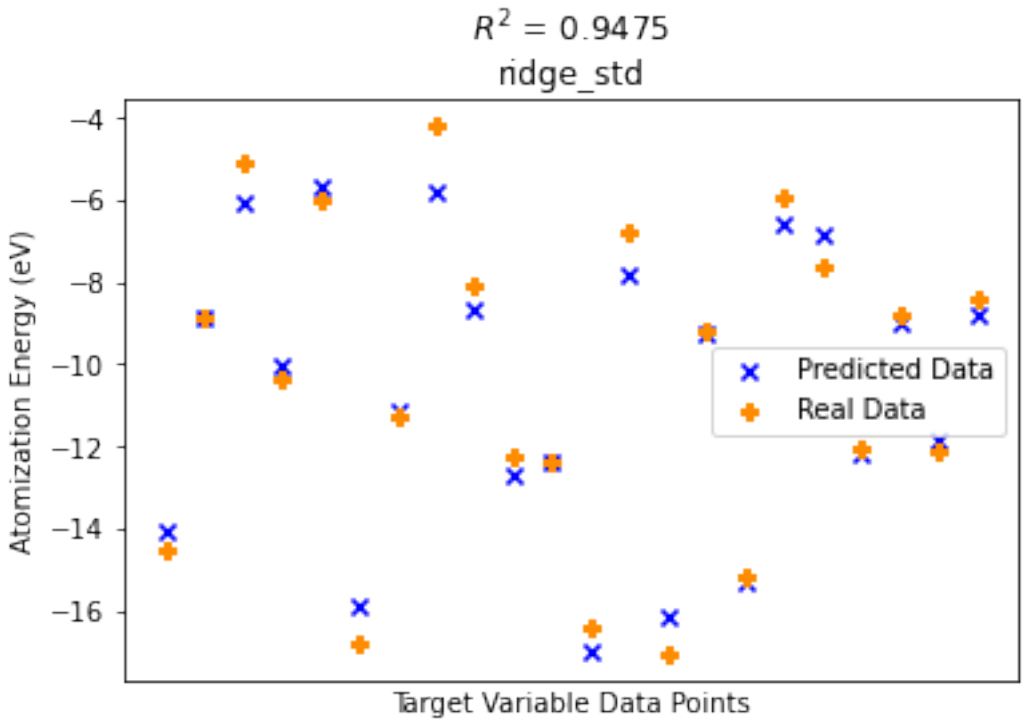


2.3 Std. Scaled Data

[ ]:
```
RidgeExp(X_train_std,X_test_std,587.80)
```

```
Score= 0.9475026016200853
 Error= 0.7125558170692959
```

[ ]:
```
compare_plot(y_pred_ridge,y_test,"ridge_std")
```



3 Lasso

3.1 Original Data

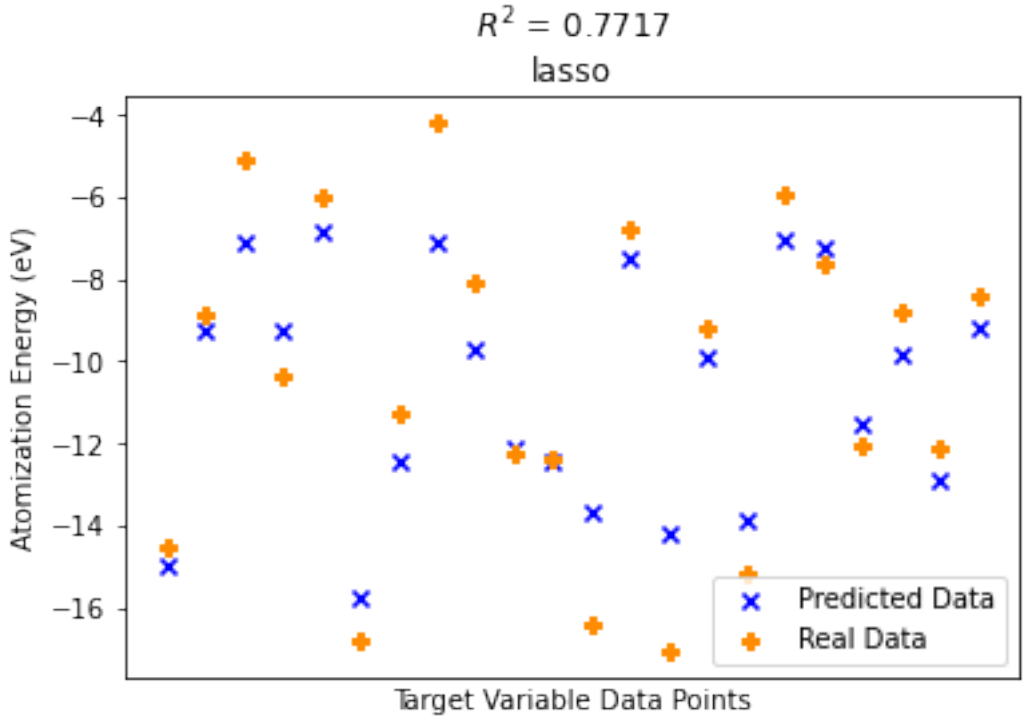
[ ]:
```
def LassoExp(X_train,X_test,alpha_val):
    global y_pred_lasso
    model_lasso = Lasso(max_iter = 10**4,alpha=alpha_val)
    model_lasso.fit(X_train,y_train)
    y_pred_lasso = model_lasso.predict(X_test)
    lasso_error = mean_squared_error(y_test,y_pred_lasso)
    lasso_score = r2_score(y_test,y_pred_lasso)
    print("Score=",lasso_score,"\n","Error=",lasso_error)
```

[ ]:
```
for i in alphas_log[4:8]:
    print("Alpha=",i)
    LassoExp(X_train,X_test,i)
```

```
Alpha= 0.028942661247167517
Score= 0.9348479513516612
 Error= 0.8843194651740565
Alpha= 0.1193776641714437
Score= 0.8646421456723128
 Error= 1.8372344052027028
```

```
Alpha= 0.49238826317067413
Score= 0.7874386541010026
 Error= 2.885130086034092
Alpha= 2.030917620904739
Score= 0.7717876346213297
 Error= 3.0975639459484268
```

[ ]:
```
compare_plot(y_pred_lasso,y_test,"lasso")
```

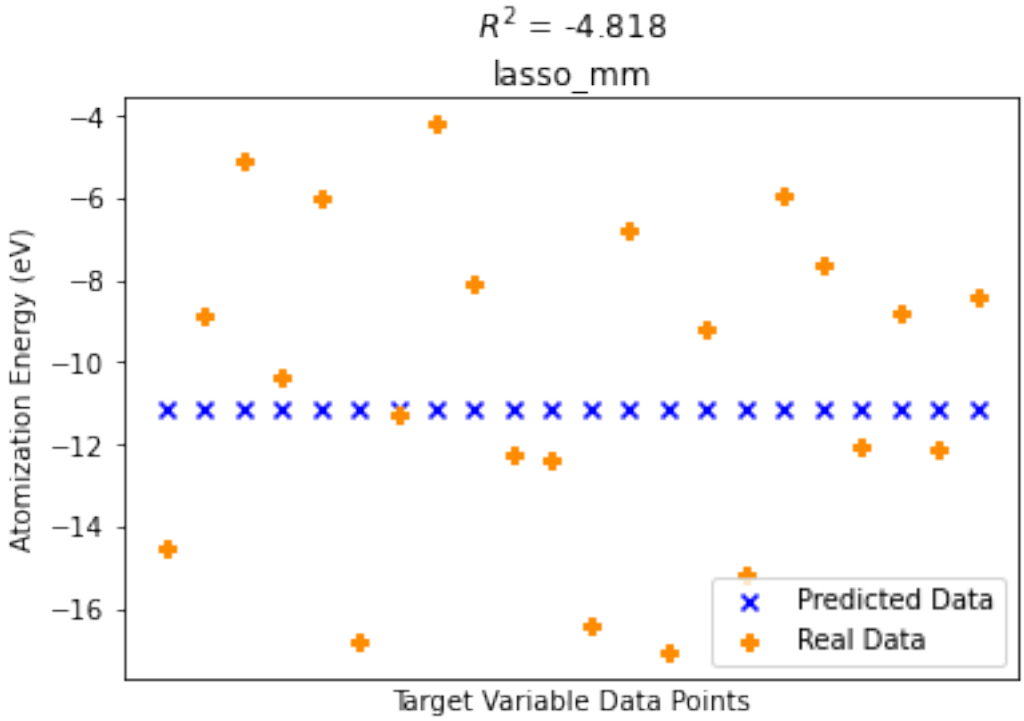


3.2 MinMax Data

[ ]:
```
LassoExp(X_train_mm,X_test_mm,i)
```

```
Score= -4.818961356178875e-06
 Error= 13.57322977591333
```

[ ]:
```
compare_plot(y_pred_lasso,y_test,"lasso_mm")
```

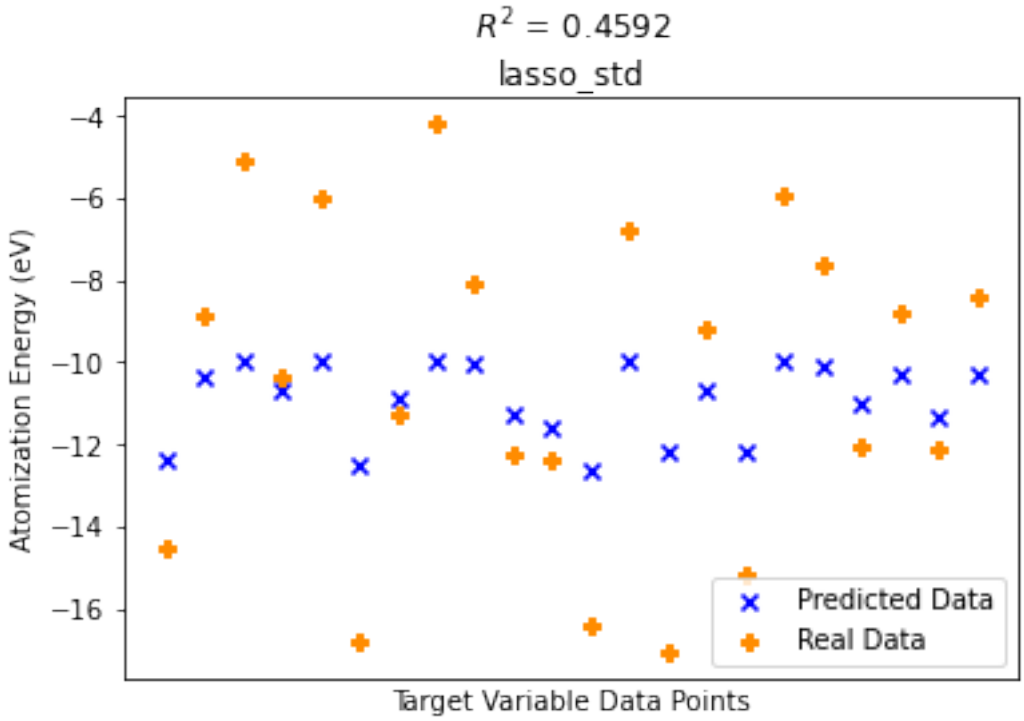


3.3 Std. Scaled Data

[ ]:
```
LassoExp(X_train_std,X_test_std,i)
```

```
Score= 0.4592490429408119
 Error= 7.339701621970921
```

[ ]:
```
compare_plot(y_pred_lasso,y_test,"lasso_std")
```



4 Ridge with Cross-Validation

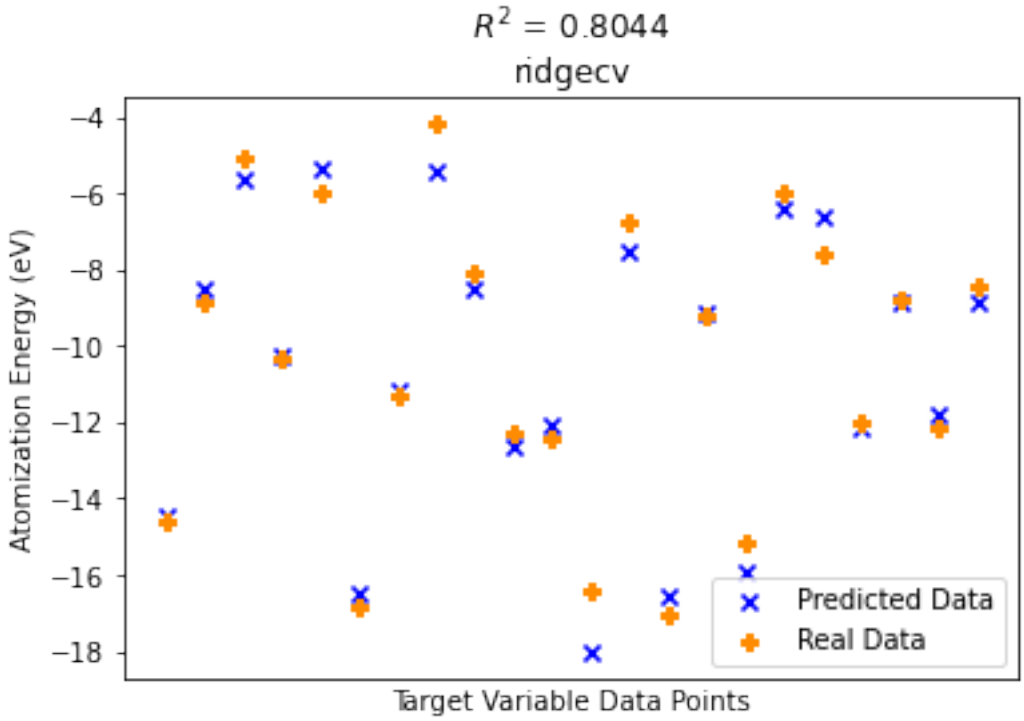
4.1 Original Data

[ ]:
```
def RidgeCVExp(X_train,y_train,X_test,y_test):
    global y_pred_ridgecv
    model_ridgecv = RidgeCV(alphas=alphas_log)
    model_ridgecv.fit(X_train,y_train)
    y_pred_ridgecv = model_ridgecv.predict(X_test)
    ridgecv_error = mean_squared_error(y_test,y_pred_ridgecv)
    ridgecv_score = r2_score(y_test,y_pred_ridgecv)
    print("Score=",ridgecv_score,"\n","Error=",ridgecv_error)
```

[ ]:
```
RidgeCVExp(X_train,y_train,X_test,y_test)
```

```
Score= 0.8044872411515771
 Error= 2.6537268117654205
```

[ ]:
```
compare_plot(y_pred_ridgecv,y_test,"ridgecv")
```

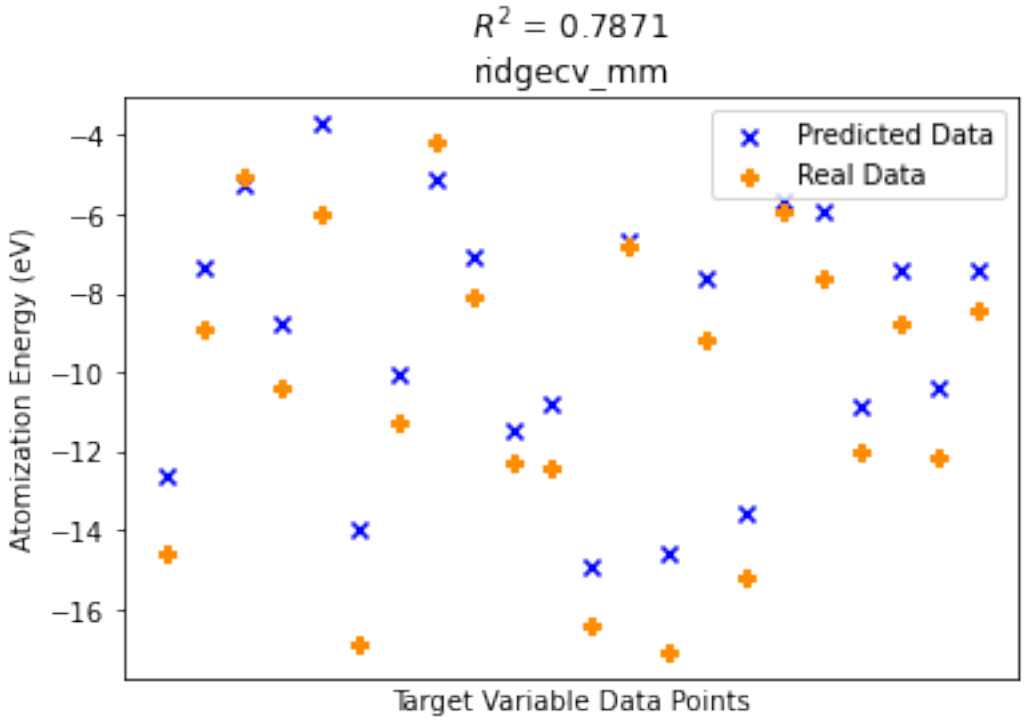


4.2 MinMax Data

[ ]:
```
RidgeCVExp(X_train_mm,y_train,X_test_mm,y_test)
```

```
Score= 0.7871903839655123
 Error= 2.888499897390609
```

[ ]:
```
compare_plot(y_pred_ridgecv,y_test,"ridgecv_mm")
```

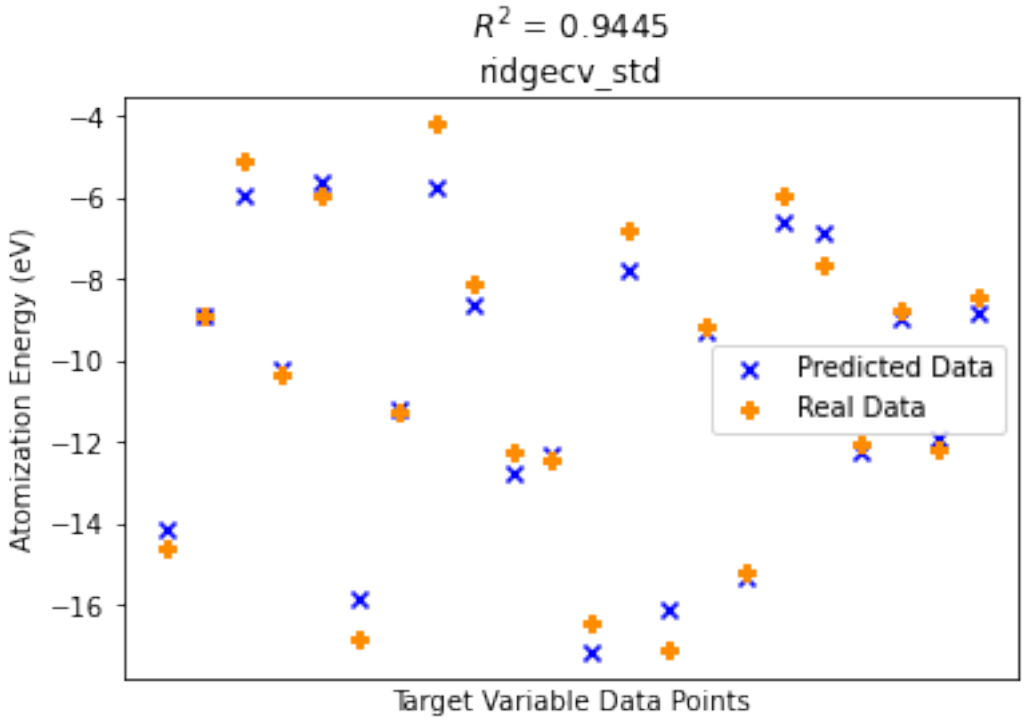


4.3 Std. Scaled Data

[ ]:
```
RidgeCVExp(X_train_std,y_train,X_test_std,y_test)
```

```
Score= 0.9445815914783816
 Error= 0.7522031678413623
```

[ ]:
```
compare_plot(y_pred_ridgecv,y_test,"ridgecv_std")
```



5 LassoCV

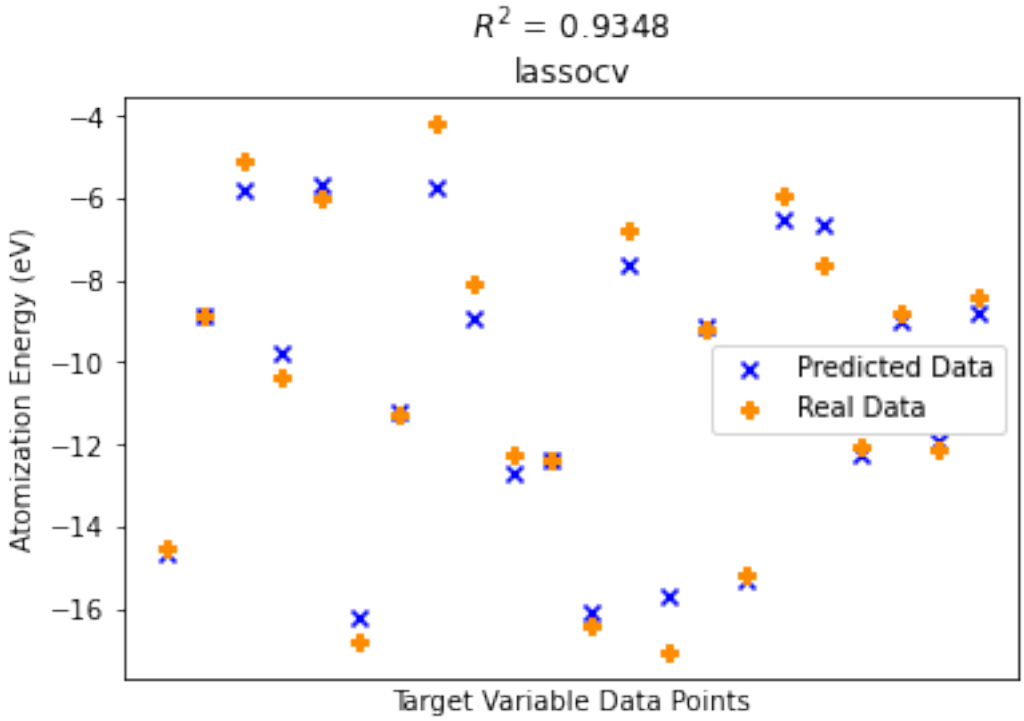
5.1 Original Data

[ ]:
```
def LassoCVExp(X_train,y_train,X_test,y_test):
    global y_pred_lassocv
    model_lassocv = LassoCV(max_iter=10**6, alphas=alphas_log)
    model_lassocv.fit(X_train,y_train)
    y_pred_lassocv = model_lassocv.predict(X_test)
    lassocv_error = mean_squared_error(y_test,y_pred_lassocv)
    lassocv_score = r2_score(y_test,y_pred_lassocv)
    print("Score=",lassocv_score,"\n","Error=",lassocv_error)
```

[ ]:
```
LassoCVExp(X_train,y_train,X_test,y_test)
```

```
Score= 0.9348479513516612
 Error= 0.8843194651740565
```

[ ]:
```
compare_plot(y_pred_lassocv,y_test,"lassocv")
```

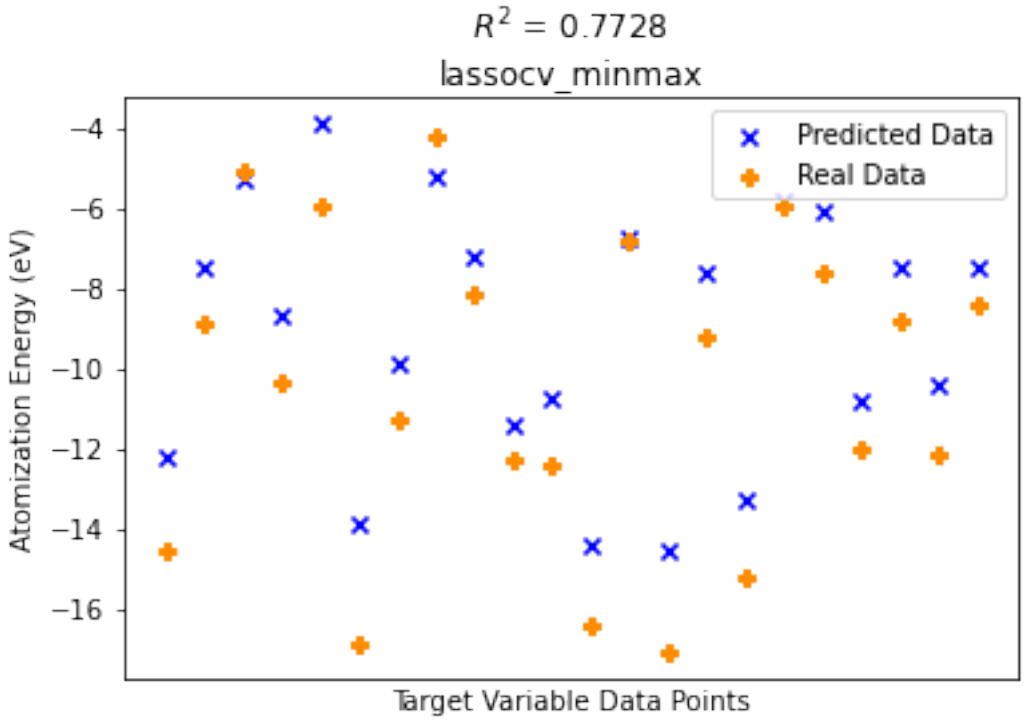


5.2 MinMaxData

[ ]:
```
LassoCVExp(X_train_mm,y_train,X_test_mm,y_test)
```

```
Score= 0.7728560483875612
 Error= 3.083062190287018
```

[ ]:
```
compare_plot(y_pred_lassocv,y_test,"lassocv_minmax")
```

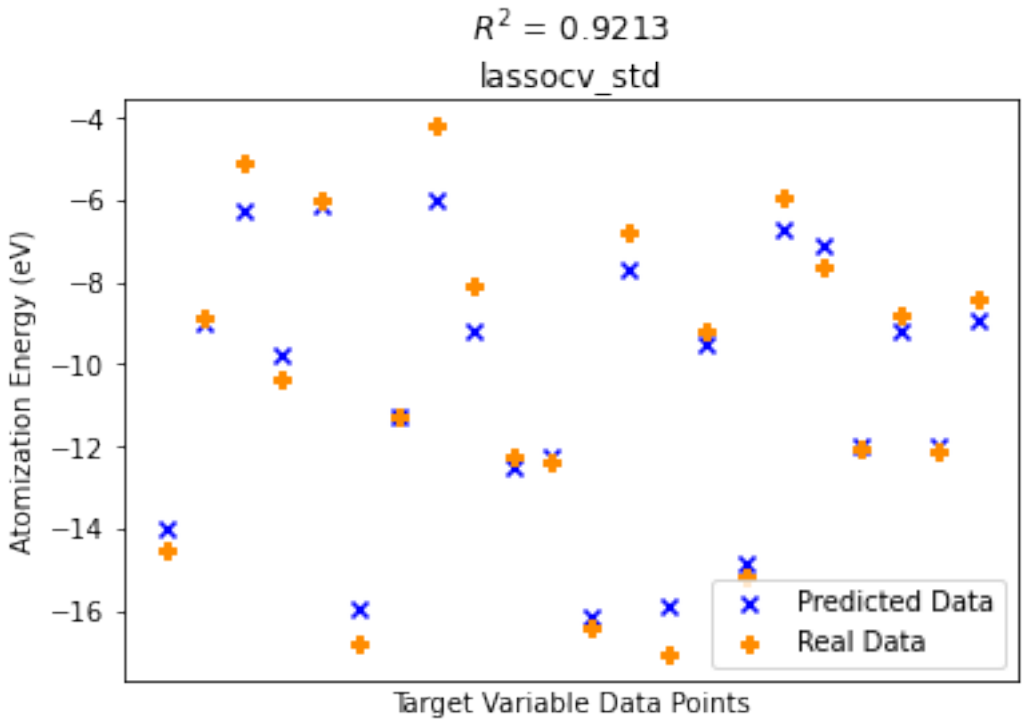


5.3 Std. Scaled Data

[ ]:
```
LassoCVExp(X_train_std,y_train,X_test_std,y_test)
```

```
Score= 0.9213305599607838
 Error= 1.0677932403403554
```

[ ]:
```
compare_plot(y_pred_lassocv,y_test,"lassocv_std")
```



6 Support Vector Regression

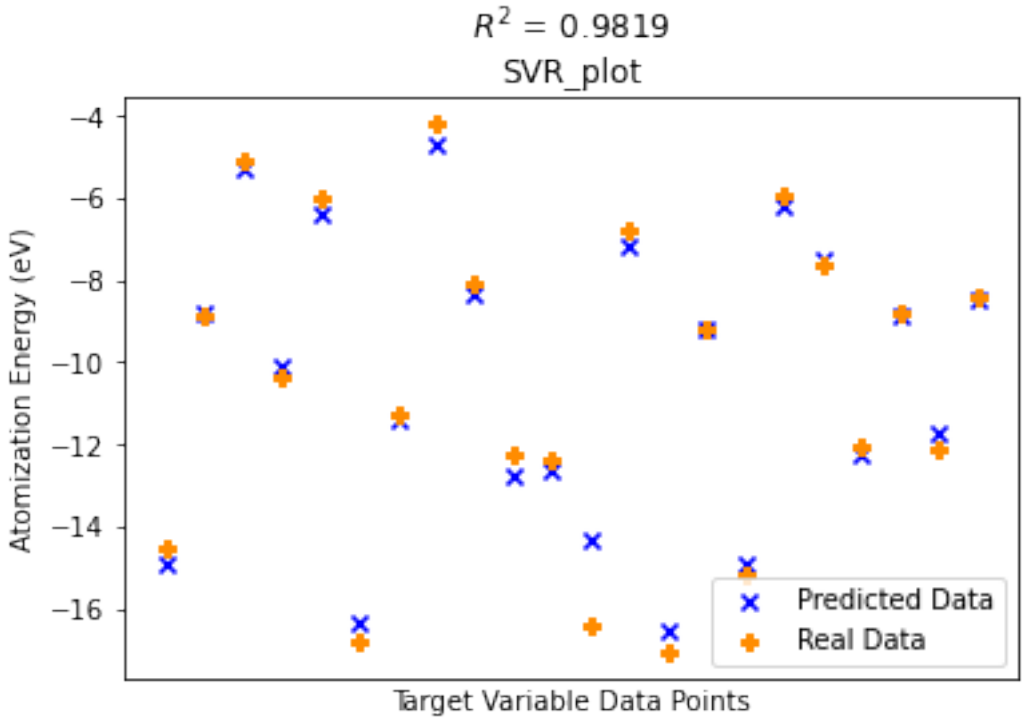
[ ]:
```
def SVRExp(X_train,X_test):
    global pred_svm
    model_svr = SVR(kernel="rbf")
    model_svr.fit(X_train,y_train)
    pred_svm = model_svr.predict(X_test)
    score = r2_score(pred_svm,y_test)
    print(score)
```

6.1 Original Data

[ ]:
```
SVRExp(X_train,X_test)
```

```
0.9804528258306725
```

[ ]:
```
compare_plot(pred_svm,y_test,"SVR_plot")
```

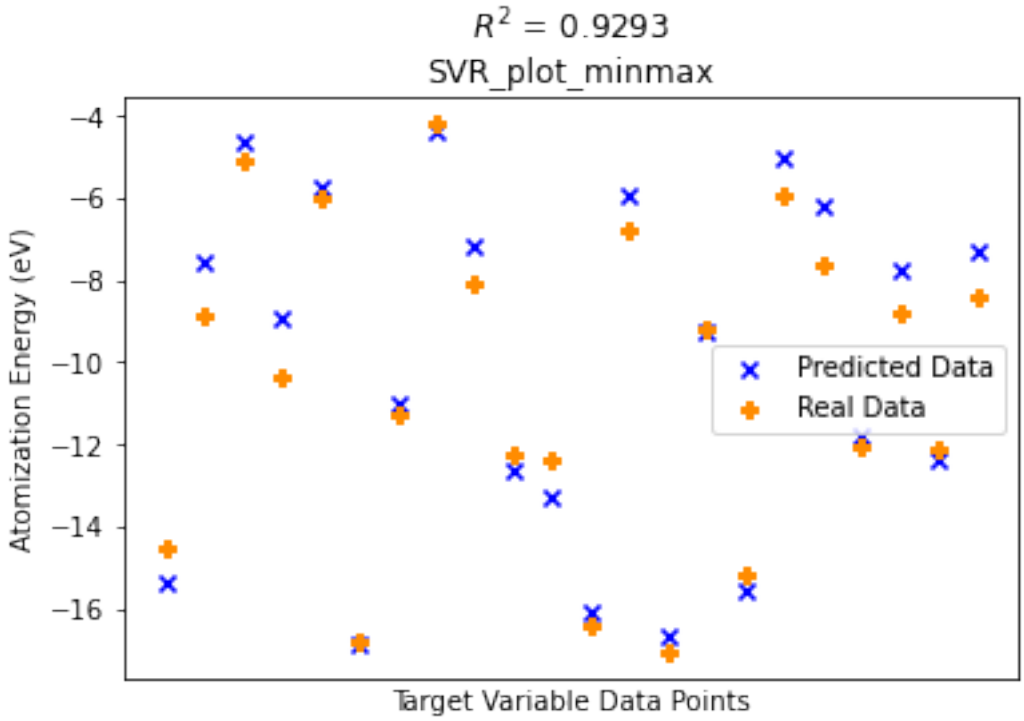


6.2 MinMax

[ ]:
```
SVRExp(X_train_mm,X_test_mm)
```

```
0.9417416135823121
```

[ ]:
```
compare_plot(pred_svm,y_test,"SVR_plot_minmax")
```

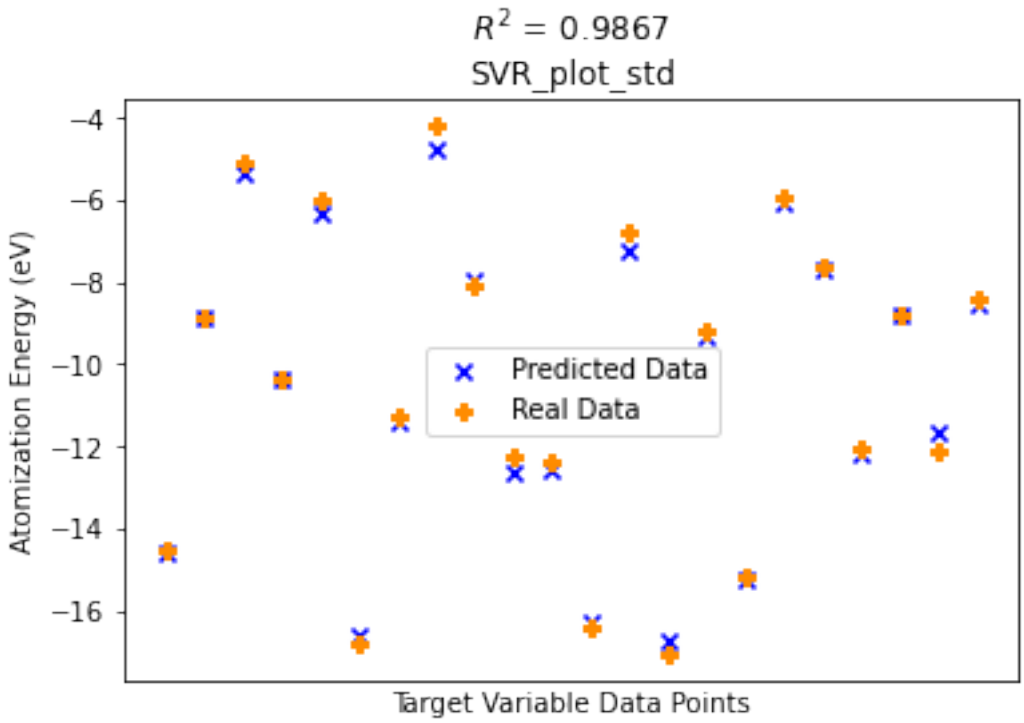


6.3 Std. Scaled Data

[ ]:
```
SVRExp(X_train_std,X_test_std)
```

```
0.9859692065461165
```

[ ]:
```
compare_plot(pred_svm,y_test,"SVR_plot_std")
```



7 KNN

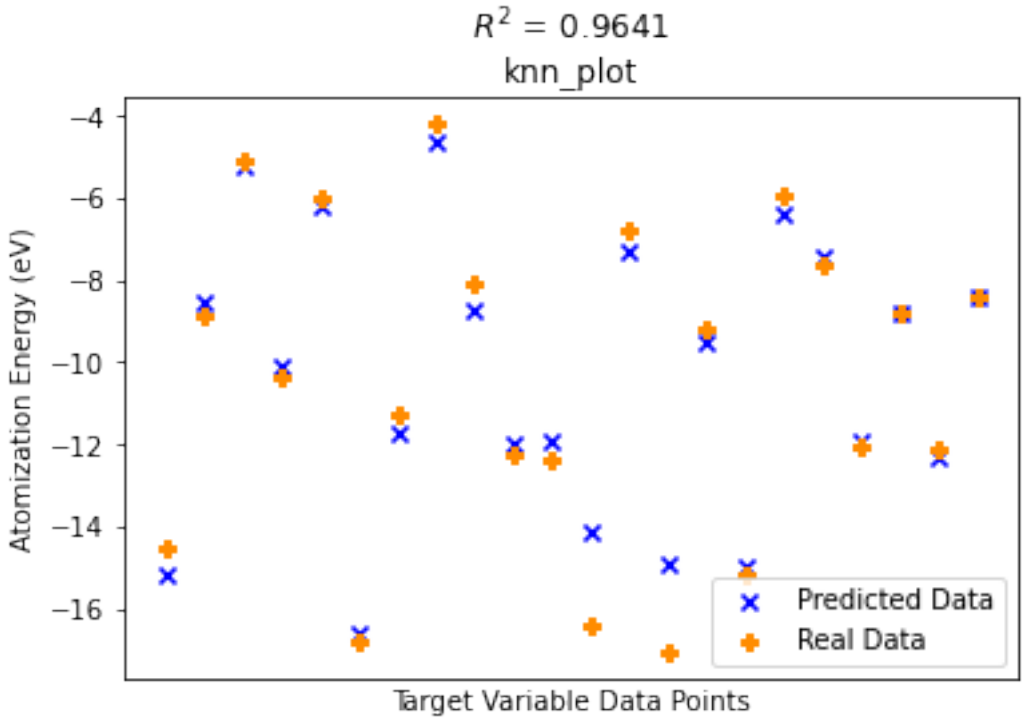
[ ]:
```
def KNNexp(X_train,y_train,X_test,y_test,N):
    global pred
    model_KNN = KNeighborsRegressor(n_neighbors=N)
    model_KNN.fit(X_train,y_train)
    pred = model_KNN.predict(X_test)
    score = r2_score(pred,y_test)
    print("Score = ",score)
```

7.1 Original Data

[ ]:
```
KNNexp(X_train,y_train,X_test,y_test,10)
```

```
Score =  0.961560999094339
```

[ ]:
```
compare_plot(pred,y_test,"knn_plot")
```

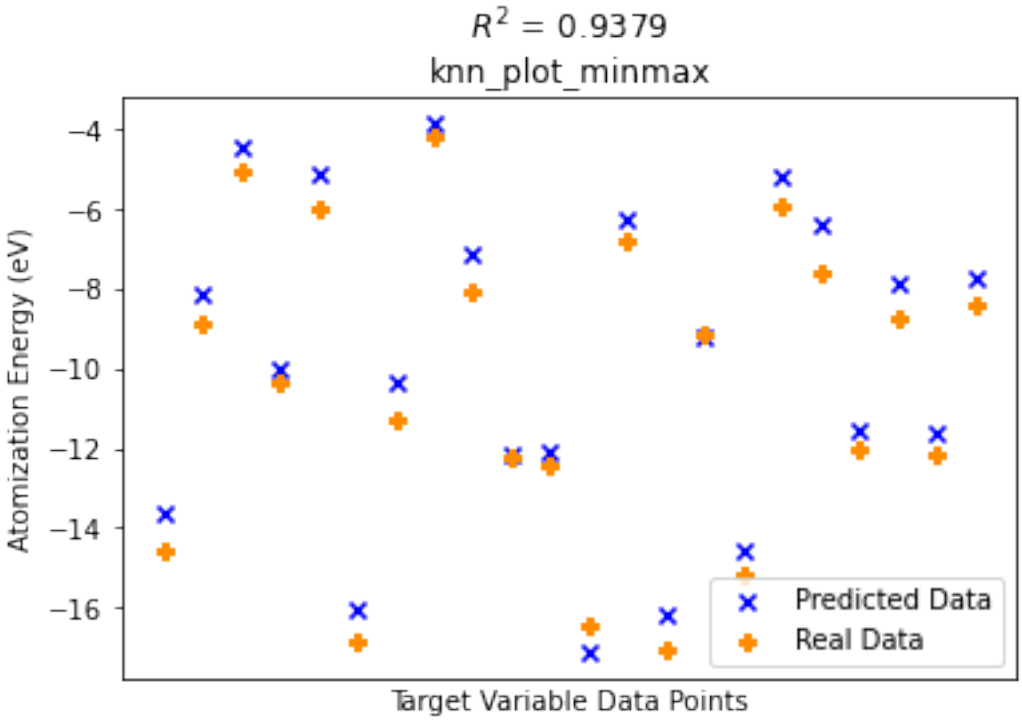


7.2 MinMax Data

[ ]:
```
KNNexp(X_train_mm,y_train,X_test_mm,y_test,10)
```

```
Score =  0.943327389430883
```

[ ]:
```
compare_plot(pred,y_test,"knn_plot_minmax")
```

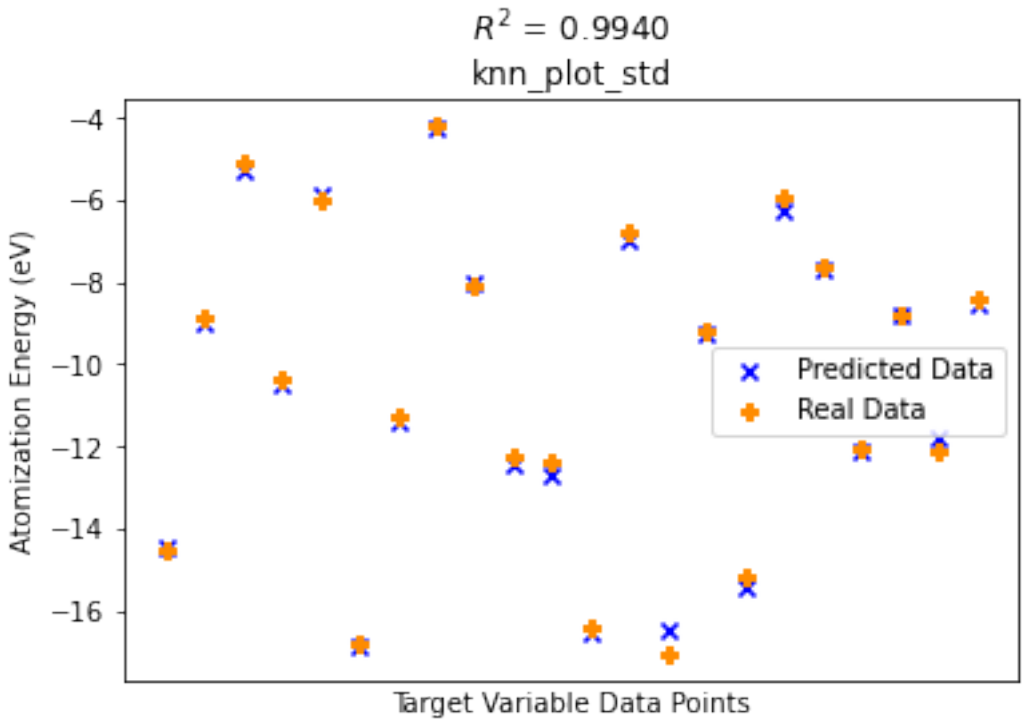


7.3 Std. Scaled Data

[ ]:
```
KNNexp(X_train_std,y_train,X_test_std,y_test,10)
```

```
Score =  0.993944181370908
```

[ ]:
```
compare_plot(pred,y_test,"knn_plot_std")
```



8 Decision Trees

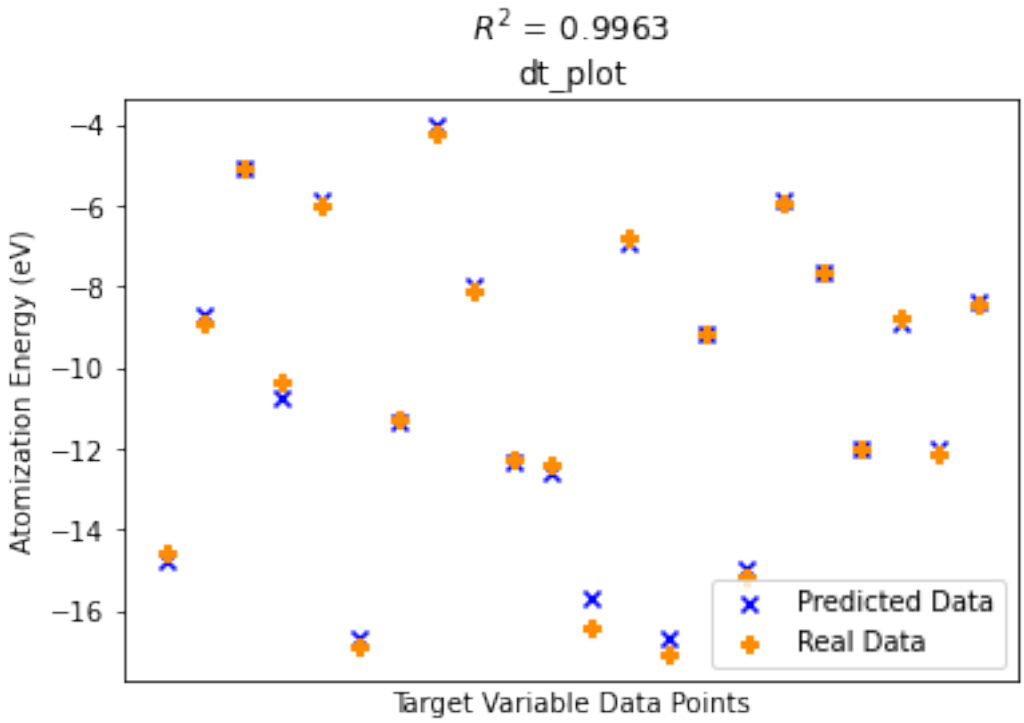
[ ]:
```
def DTexp(X_train,y_train,X_test,y_test,depth):
    global pred
    model_dt = DecisionTreeRegressor(max_depth=depth)
    model_dt.fit(X_train,y_train)
    pred = model_dt.predict(X_test)
    print(r2_score(y_test,pred))
```

[ ]:
```
def DTexp_minsample(X_train,y_train,X_test,y_test,depth):
    model_dt = DecisionTreeRegressor(min_samples_split=5)
    model_dt.fit(X_train,y_train)
    pred = model_dt.predict(X_test)
    score = r2_score(pred,y_test)
    return pred
```

[ ]:
```
DTexp(X_train,y_train,X_test,y_test,10)
```

```
0.9963001262473653
```

[ ]:
```
compare_plot(pred,y_test,"dt_plot")
```

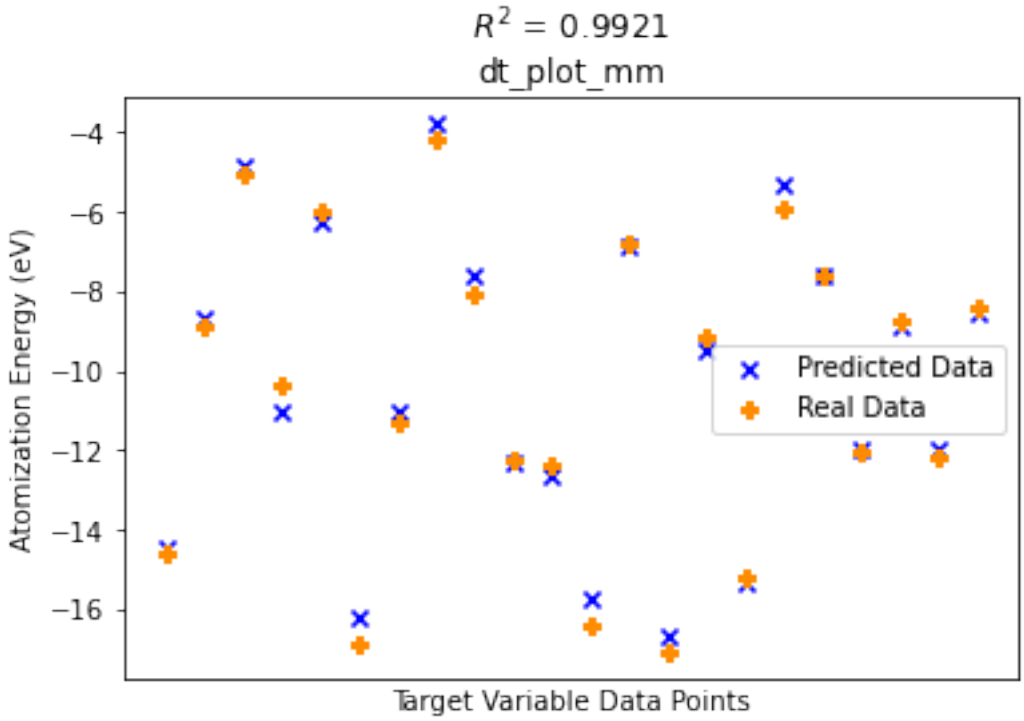


8.1 MinMax Data

[ ]:
```
DTexp(X_train_mm,y_train,X_test_mm,y_test,10)
```

```
0.9921822695528284
```

[ ]:
```
compare_plot(pred,y_test,"dt_plot_mm")
```

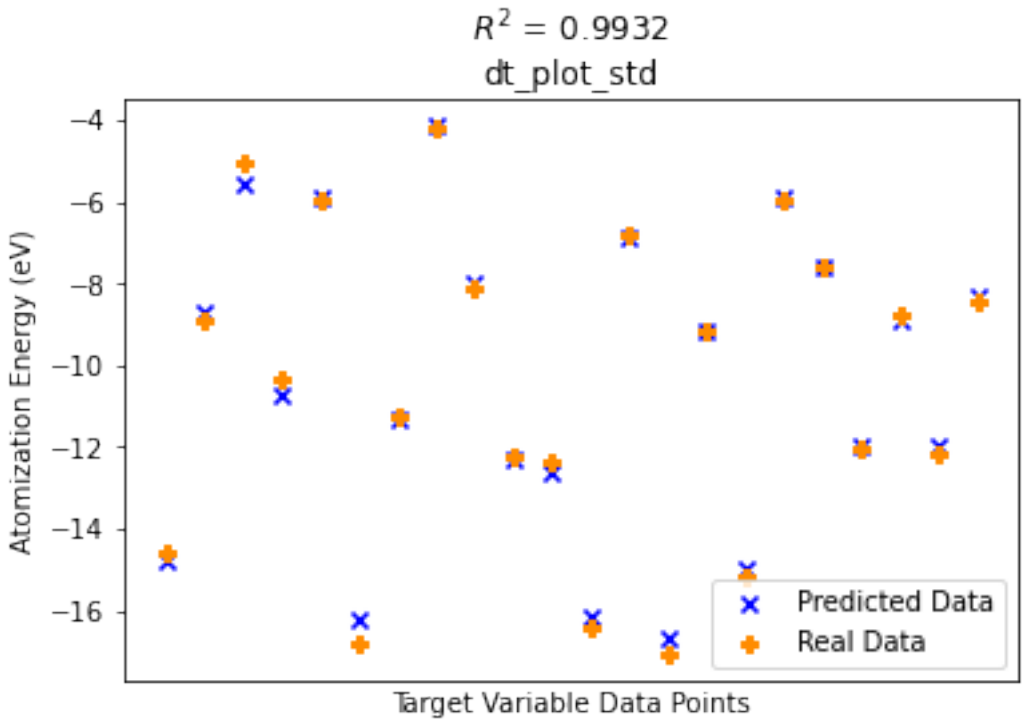


8.2 Std. Scaled Data

[ ]:
```
DTexp(X_train_std,y_train,X_test_std,y_test,10)
```

```
0.9932596730940714
```

[ ]:
```
compare_plot(pred,y_test,"dt_plot_std")
```



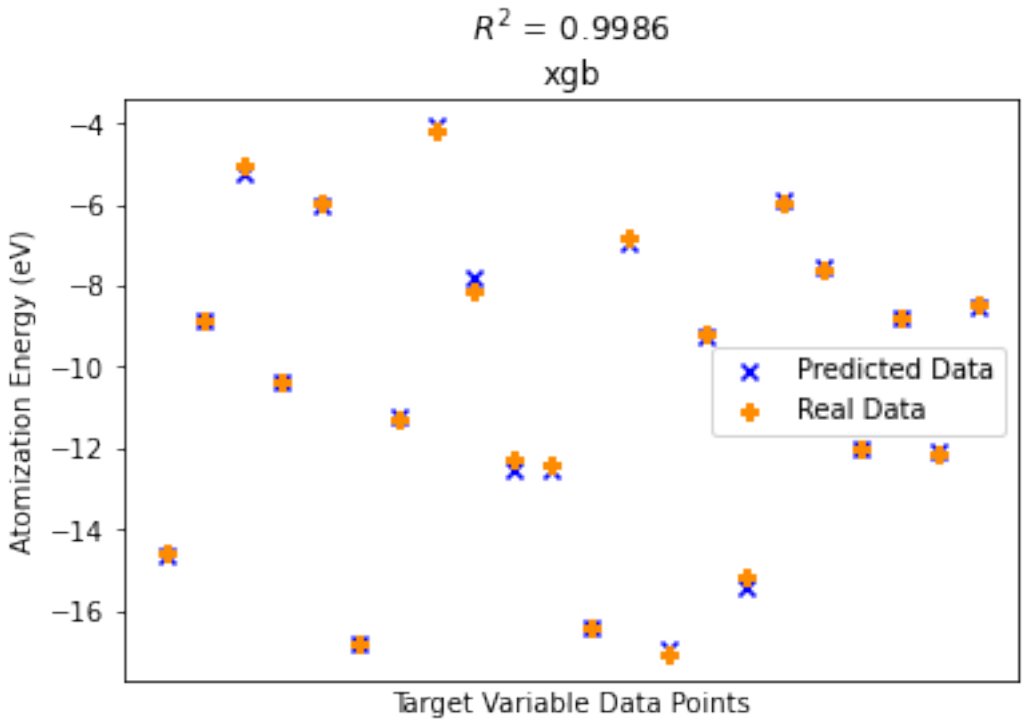
9 XGBoost

[ ]:
```
def XGB(X_train,X_test):
    global pred
    model = XGBRegressor()
    model.fit(X_train,y_train)
    pred = model.predict(X_test)
    score = r2_score(y_test,pred)
    print(score)
```

[ ]:
```
XGB(X_train,X_test)
```

```
[18:20:02] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
0.9986067621590289
```

[ ]:
```
compare_plot(pred,y_test,"xgb")
```

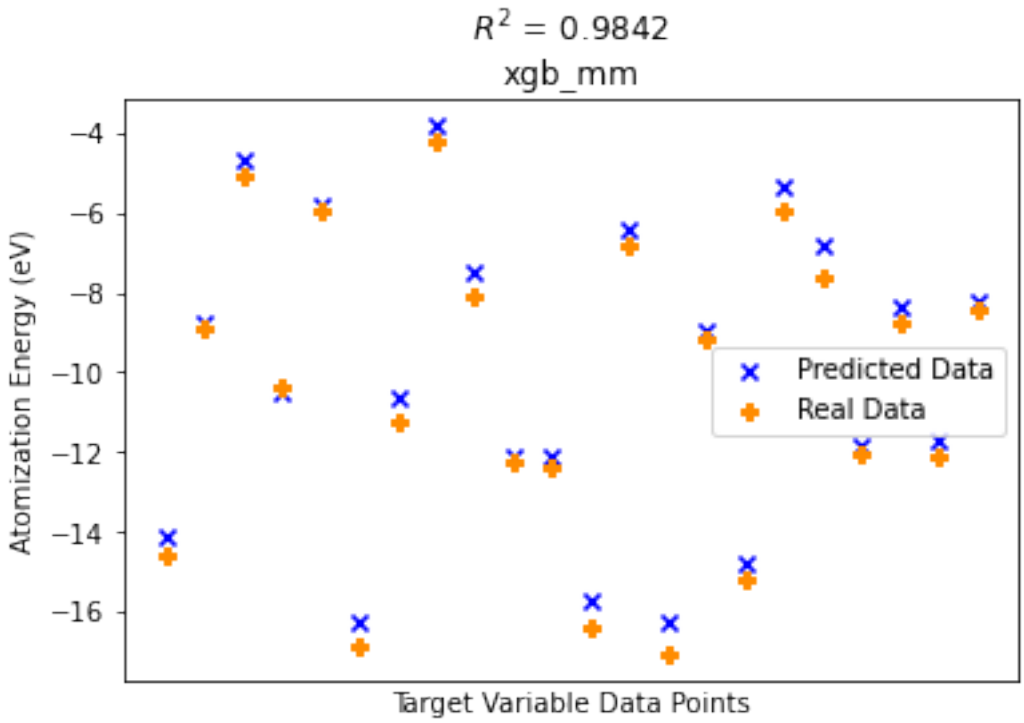


9.1 MinMax Data

[ ]:
```
XGB(X_train_mm,X_test_mm)
```

```
[18:20:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
0.9842195266684683
```

[ ]:
```
compare_plot(pred,y_test,"xgb_mm")
```

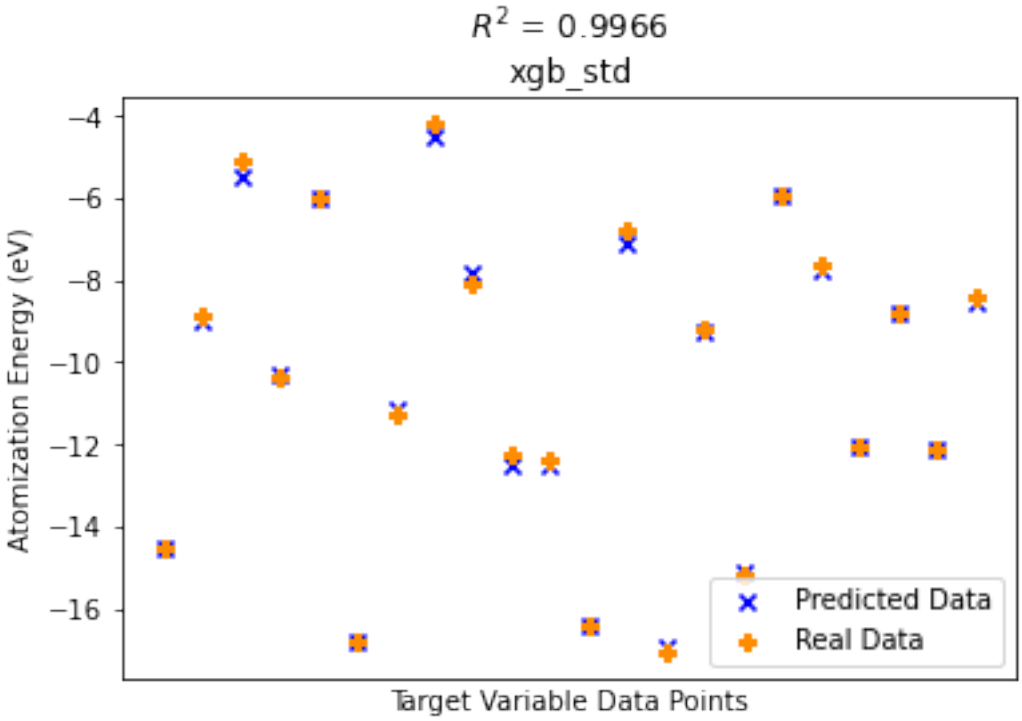


9.2 Std. Scaled Data

[ ]:
```
XGB(X_train_std,X_test_std)
```

```
[18:21:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
0.9966656233857084
```

[ ]:
```
compare_plot(pred,y_test,"xgb_std")
```


$R^2 = 0.9966$
xgb_std
Atomization Energy (eV)
−4
−6
−8
−10
−12
−14
−16
Predicted Data
Real Data
Target Variable Data Points