MULTI-TARGET REGRESSOR MODELS

June 4, 2020

1 MULTI-TARGET REGRESSION

1.1 INTRODUCTION

When multiple dependent variables exist in a regression model, this task is called as multi-target regression. In this case, a multi-output regressor is employed to learn the mapping from input features to output variables jointly. In this study, multi-target regression technique is implemented for quality prediction in a mining process to estimate the amount of silica and iron concentrates in the ore at the end of the process.

In this study, two inter-dependent single target regression tasks are transformed into a multiple output regression problem for quality prediction in a mining process.

In the pervious models have been conducted to estimate silica concentrate with or without taking iron concentrate as input parameter. In this aspect, the problem is a single-target regression problem. However, this study that focuses on the estimation of both iron and silica concentrates simultaneously as output variables. We compared different multi-target regressors that use Random Forest, AdaBoost, XGBOOST ,RIDGE and Decision Tree algorithms separately in the background. Coefficient of determination (R2) metric and MSE was used to evaluate predictive performance of the regression methods for the mentioned data.

1.2 METHODS TO IMPLEMENT MTR

Problem transformation methods

1. These methods are mainly based on transforming the multi-output regression problem into single-target problems, then building a model for each target, and finally concatenating all the d predictions. The main drawback of these methods is that the relationships among the targets are ignored, and the targets are predicted independently, which may affect the overall quality of the predictions.

2. **Regressor chains (RC) method**

It is inspired by the recent multi-label chain classifiers 31. RC is another problem transformation method, based on the idea of chaining single-target models. The training of RC consists of selecting a random chain (i.e., permutation) of the set of target variables, then building a separate regression model for each target following the order of the selected chain.

3. **Single traget model**

output variables are estimated independently and potential relations between them cannot be exploited

1.3 RELATED WORKS

https://ieeexplore.ieee.org/abstract/document/8907120Y

The paper focus on inherent multiregressor models and concluded to it is best to predict silica and iron concentrate at the same time.

1.4 NEW METHODS

https://machinelearningmastery.com/multi-output-regression-models-with-python/

My work focus on following implementation:

1. To see whether the %silica concentrate can be predicted without iron concentrate and result showed us it is not good to predict silica concentrate withou iron concentrate . Hence, to solve the problem we can implement the multitarget regression method to predict both target variables at same time.
2. To try differnt models which is not inherent multitarget regression models like Randomforest,Ridge,Xgboost
3. To finalize the best model with R2 as well as MSE metric.

2 RIDGE REGRESSOR

2.1 INTRODUCTION

2.2 DATA MODELING

[0]:
```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
```

[0]:
```
#code snippet taken from microsoft malware detection case study notebook-- to␣
↪get the idea to get the graphs
#https://classroom.appliedcourse.com/classrooms/jEARG7xb/assignments/q2AJp9B5/
↪users/jEARG7xb
df=pd.read_csv('/content/drive/My Drive/preprocessed_time')
y = df.iloc[:,23:25]
X = df.drop(['% Silica Concentrate','% Iron Concentrate','index','datetime␣
↪hours'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
↪2,random_state=30)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,test_size=0.
↪20,random_state=30)

scale_features_std = StandardScaler()
```

```
features_train = scale_features_std.fit_transform(X_train)
features_test = scale_features_std.transform(X_test)

features_cv = scale_features_std.transform(X_cv)

print(X_train.shape,features_train.shape,X_test.shape,features_test.
 ↪shape,features_cv.shape,y_train.shape,y_cv.shape,y_test.shape)
```

```
(471969, 21) (471969, 21) (147491, 21) (147491, 21) (117993, 21) (471969, 2)
(117993, 2) (147491, 2)
```

2.3 USING SKLEARN MULTIOUTPUT REGRESSOR METHOD

[0]:
```
#https://machinelearningmastery.com/multi-output-regression-models-with-python/
#https://stackoverflow.com/questions/50132322/
 ↪how-does-multiple-target-ridge-regression-work-in-scikit-learn

from sklearn.multioutput import MultiOutputRegressor
```

[0]:
```
model = MultiOutputRegressor(Ridge(alpha=1000,random_state=0))
model.fit(features_train, y_train)
```

[0]:
```
MultiOutputRegressor(estimator=Ridge(alpha=1000, copy_X=True,
                                     fit_intercept=True, max_iter=None,
                                     normalize=False, random_state=0,
                                     solver='auto', tol=0.001),
                     n_jobs=None)
```

2.4 PREDICTION

[0]:
```
y_pred=model.predict(features_cv)
y_pred_train=model.predict(features_train)
y_pred_test=model.predict(features_test)
```

2.5 EVALUATION METRIC

[0]:
```
mse_train=mean_squared_error(y_train,y_pred_train)
mse_cv=mean_squared_error(y_cv,y_pred)
mse_test=mean_squared_error(y_test,y_pred_test)
```

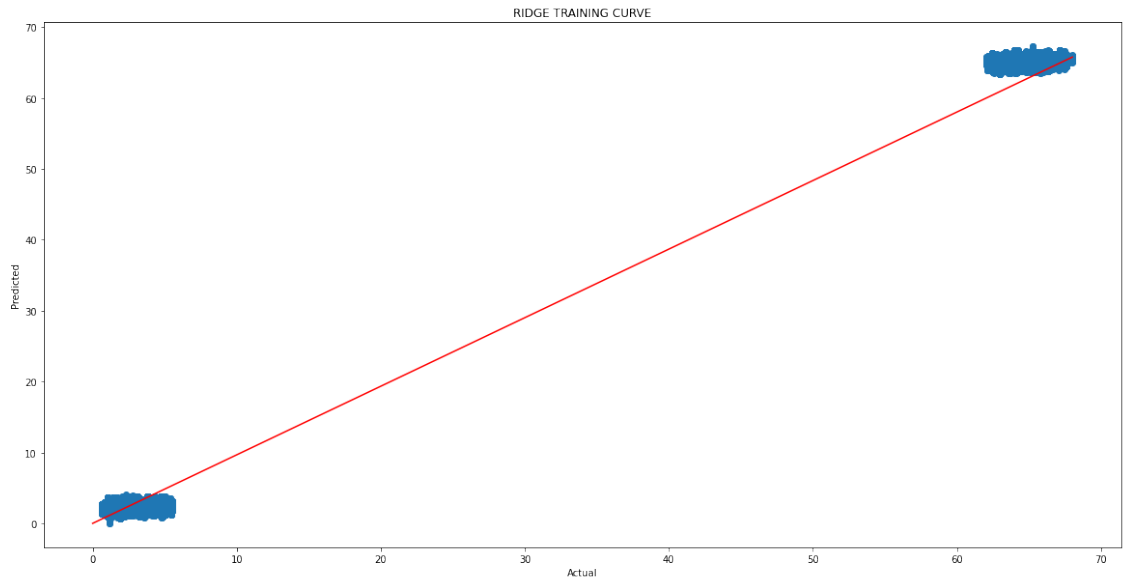
[0]:
```
r2_train=r2_score(y_train,y_pred_train)
r2_cv=r2_score(y_cv,y_pred)
r2_test=r2_score(y_test,y_pred_test)
```

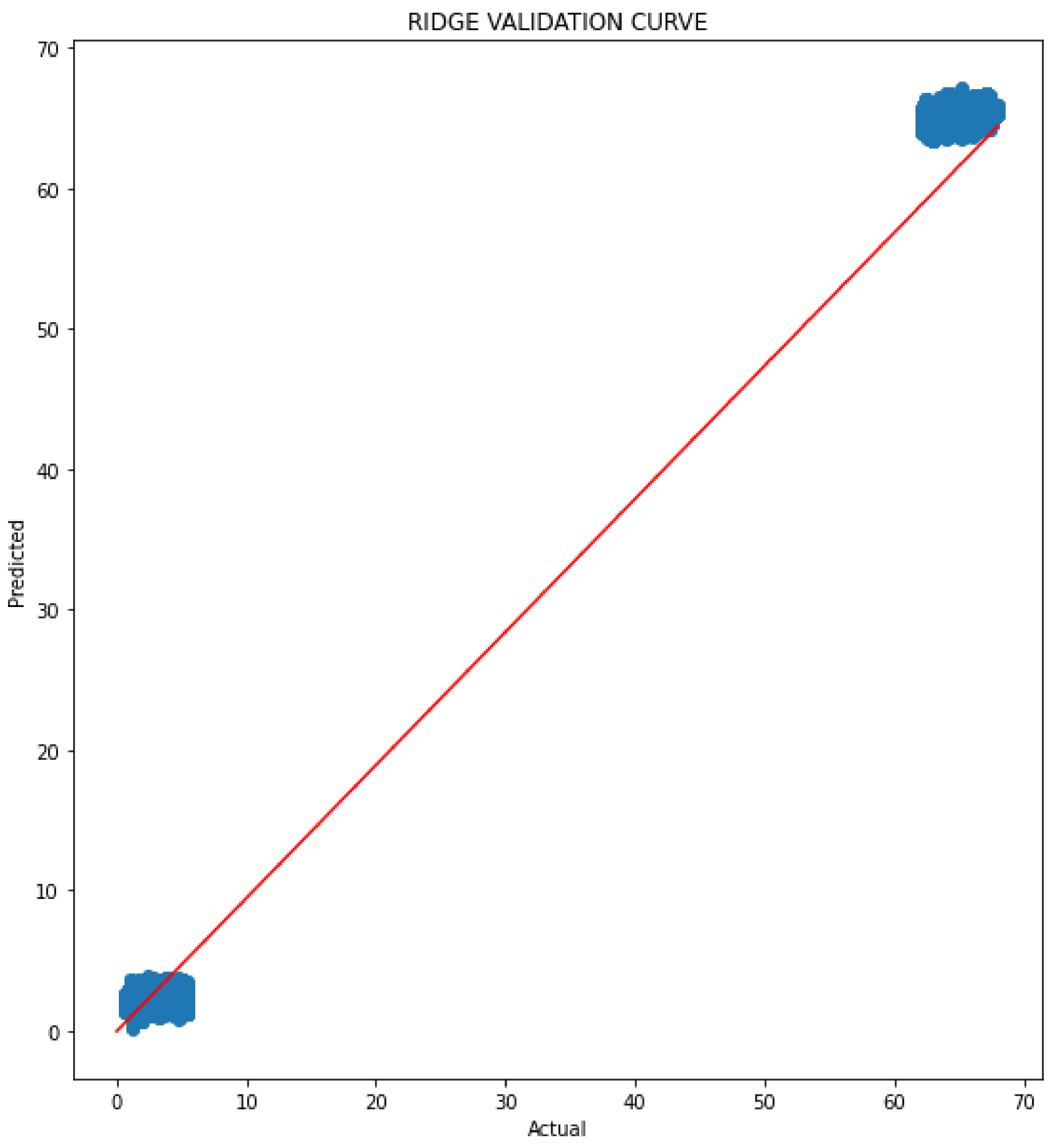
[0]:
```
r2_train,r2_cv,r2_test,mse_cv,mse_test,mse_train,mse_cv
```

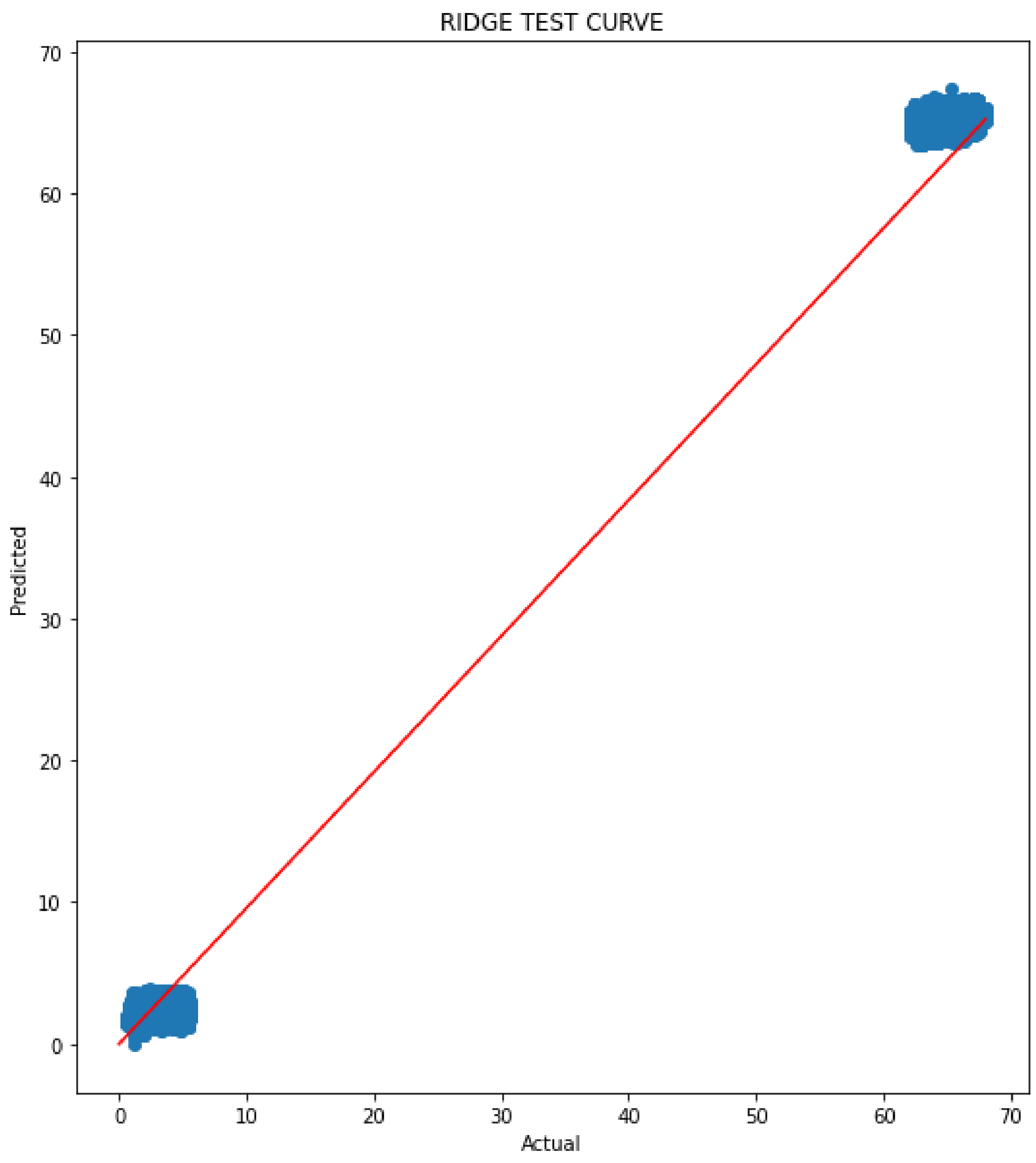
[0]:
```
(0.1457070772811201,
 0.14798745903277455,
 0.1466686774031905,
1.0722436207597437,
1.0729684678403668,
1.076168464619778,
1.0722436207597437)
```

2.6 PLOTS UDING MTR METHOD

[0]:
```
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111)
ax.set(title="RIDGE TRAINING CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_train, y_pred_train)
ax.plot([0,max(y_train['% Iron Concentrate'])], [0,max(y_pred_train[0])],␣
 ↪color='r')
fig.show()
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(121)
ax.set(title="RIDGE VALIDATION CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_cv, y_pred)
ax.plot([0,max(y_cv['% Iron Concentrate'])], [0,max(y_pred[0])], color='r')
fig.show()
fig = plt.figure(figsize=(30, 10))
ax = fig.add_subplot(131)
ax.set(title="RIDGE TEST CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_test, y_pred_test)
ax.plot([0,max(y_test['% Iron Concentrate'])], [0,max(y_pred_test[0])],␣
 ↪color='r')
fig.show()
```




RIDGE VALIDATION CURVE
Predicted
Actual
0
10
20
30
40
50
60
70



2.7 USING REGRESSION CHAIN METHOD

[0]:
```
from sklearn.multioutput import RegressorChain
```

[0]:
```
model = Ridge(alpha =1000, random_state=123)
wrapper1 = RegressorChain(model,cv=5)
wrapper1.fit(features_train,y_train)
```

[0]:
```
RegressorChain(base_estimator=Ridge(alpha=1000, copy_X=True, fit_intercept=True,
                                    max_iter=None, normalize=False,
                                      random_state=123, solver='auto',
                                      tol=0.001),
                  cv=5, order=None, random_state=None)
```

2.8 PREDICTION

[0]:
```
y_pred_cv=wrapper1.predict(features_cv)
y_pred_train=wrapper1.predict(features_train)
y_pred_test=wrapper1.predict(features_test)
```

2.9 EVALUATION METRIC

[0]:
```
r2_cv=r2_score(y_pred_cv,y_cv)
r2_train=r2_score(y_pred_train,y_train)
r2_test=r2_score(y_pred_test,y_test)
```

[0]:
```
mse_train=mean_squared_error(y_train,y_pred_train)
mse_cv=mean_squared_error(y_cv,y_pred)
mse_test=mean_squared_error(y_test,y_pred_test)
```

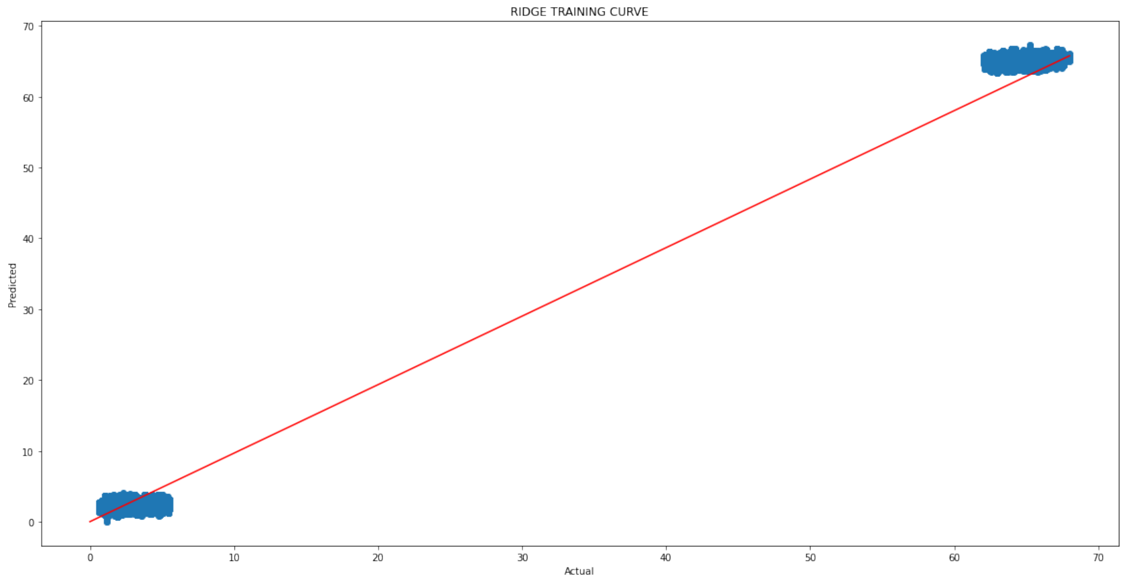
[0]:
```
r2_train,r2_cv,r2_test,mse_train,mse_cv,mse_test
```

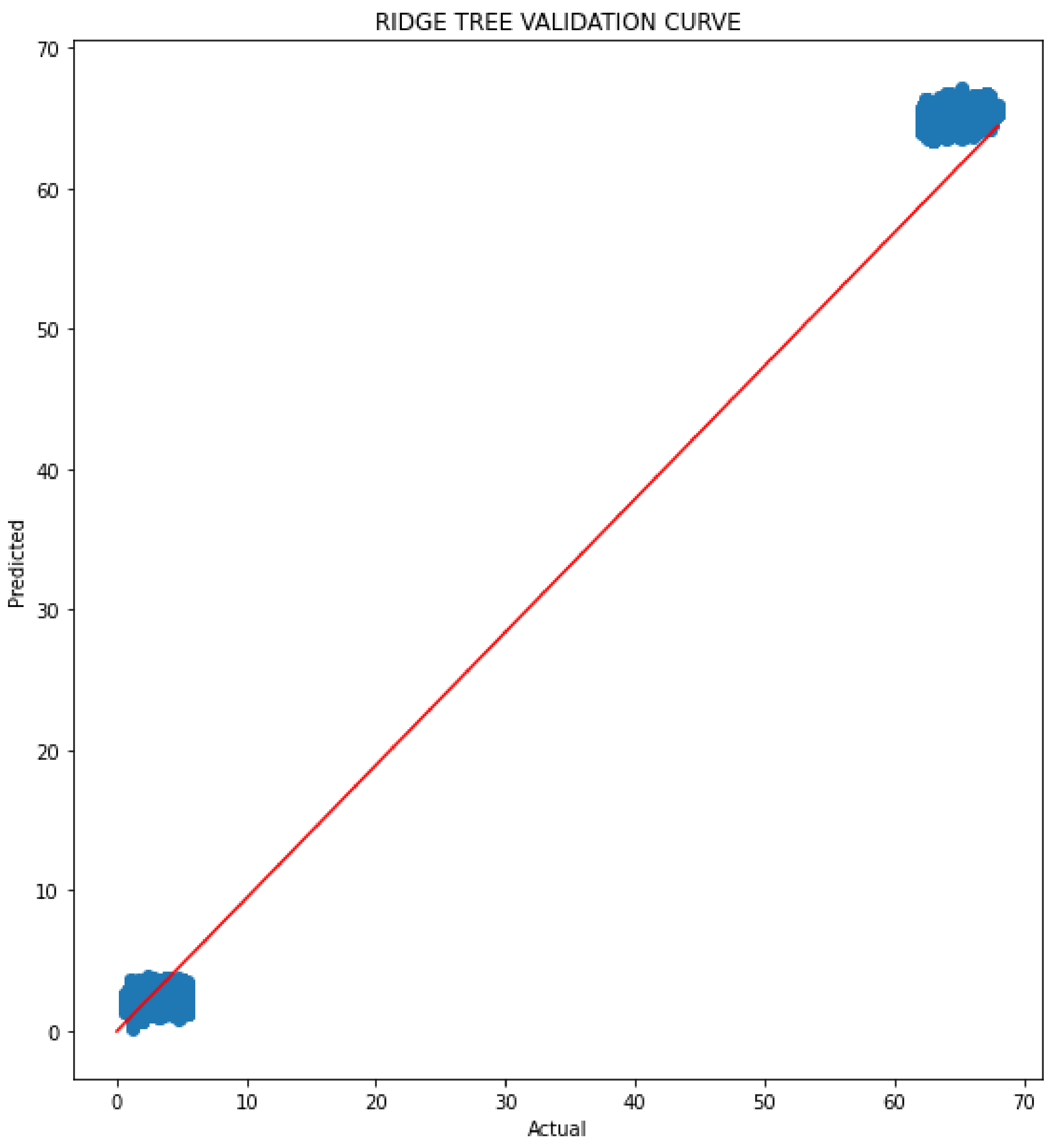
[0]:
```
(0.1457070772811201,
 0.14798745903277455,
 0.1466686774031905,
 1.0761683118743428,
 1.0722436207597437,
 1.0729682609689142)
```

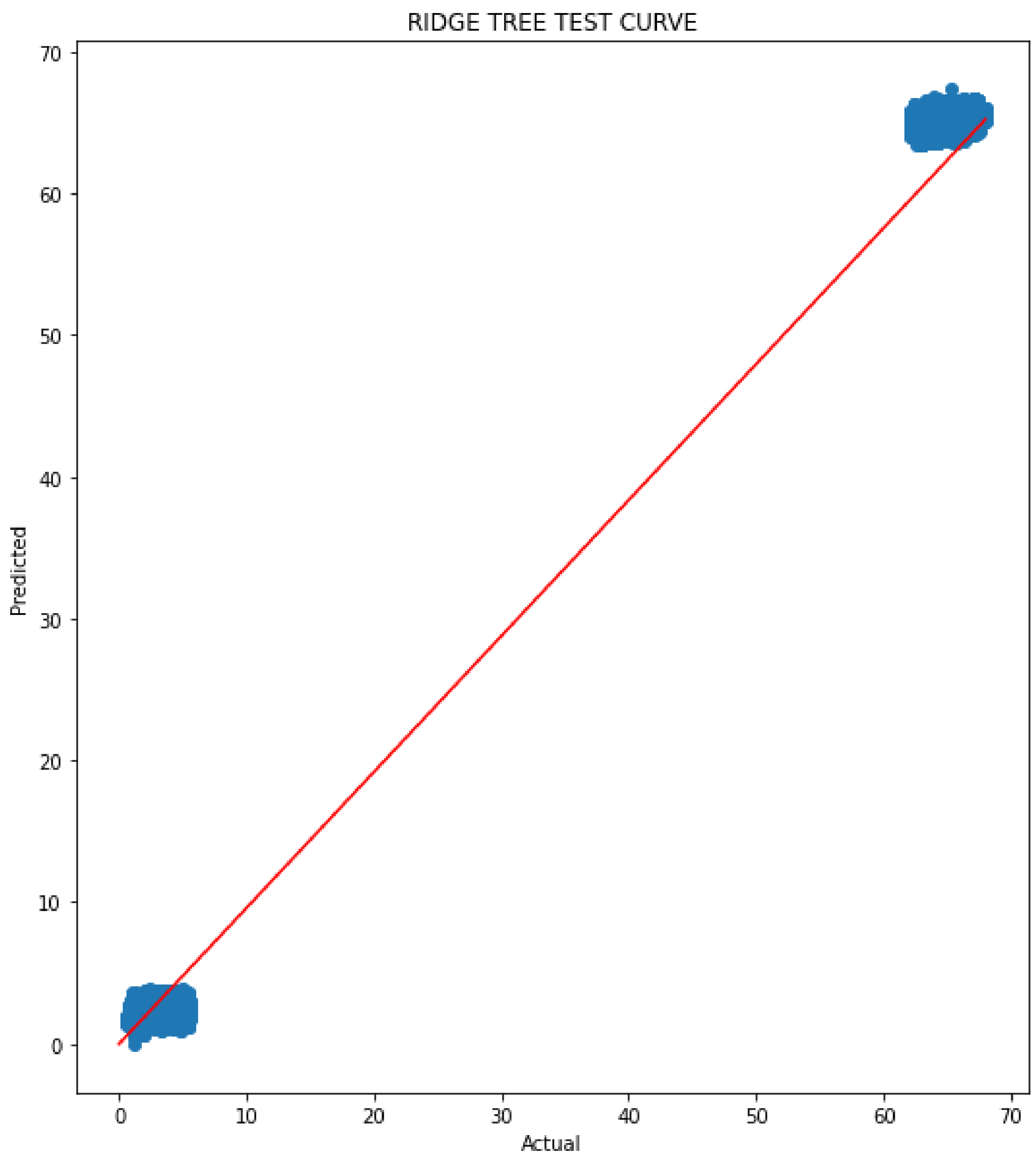
2.10 PLOTS

[0]:
```
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111)
ax.set(title="RIDGE TRAINING CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_train, y_pred_train)
ax.plot([0,max(y_train['% Iron Concentrate'])], [0,max(y_pred_train[0])],␣
 ↪color='r')
fig.show()
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(121)
ax.set(title="RIDGE TREE VALIDATION CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_cv, y_pred_cv)
ax.plot([0,max(y_cv['% Iron Concentrate'])], [0,max(y_pred[0])], color='r')
fig.show()
fig = plt.figure(figsize=(30, 10))
ax = fig.add_subplot(131)
ax.set(title="RIDGE TREE TEST CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_test, y_pred_test)
ax.plot([0,max(y_test['% Iron Concentrate'])], [0,max(y_pred_test[0])],␣
 ↪color='r')

fig.show()
```




RIDGE TREE VALIDATION CURVE
Predicted
Actual
0
10
20
30
40
50
60
70



3 SINGLE TARGET REGRESSSION METHOD(NAIVE METHOD)

3.1 MODEL TO PREDICT IRON CONCENTRATE

[0]:
```
y = df['% Iron Concentrate']
X = df.drop(['% Iron Concentrate','index','datetime hours'], axis=1)

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
↪2,random_state=30)

X_train, X_cv, y_train, y_cv1 = train_test_split(X_train, y_train,test_size=0.
↪20,random_state=30)

from sklearn.preprocessing import StandardScaler
scale_features_std = StandardScaler()
features_train = scale_features_std.fit_transform(X_train)
features_test = scale_features_std.transform(X_test)

features_cv = scale_features_std.transform(X_cv)

print(X_train.shape,features_train.shape,X_test.shape,features_test.
↪shape,features_cv.shape,y_train.shape)

model=Ridge(alpha =1000, random_state=0)
model.fit(features_train,y_train)

y_pred_train_iron=model.predict(features_train)
y_pred_cv_iron=model.predict(features_cv)
y_pred_test_iron=model.predict(features_test)

r2_cv_iron=r2_score(y_pred_cv_iron,y_cv1)
r2_train_iron=r2_score(y_pred_train_iron,y_train)
r2_test_iron=r2_score(y_pred_test_iron,y_test)

mse_train_iron=mean_squared_error(y_train,y_pred_train_iron)
mse_test_iron=mean_squared_error(y_test,y_pred_test_iron)
mse_cv_iron=mean_squared_error(y_cv1,y_pred_cv_iron)

r2_train_iron,r2_cv_iron,r2_test_iron,mse_train_iron,mse_cv_iron,mse_test_iron
```

```
(471969, 22) (471969, 22) (147491, 22) (147491, 22) (117993, 22) (471969,)
```

[0]:
```
(0.514535485002991,
 0.5123168325856021,
 0.5157302218318192,
 0.40752759926814774,
 0.4091846624430369,
 0.40599913412920674)
```

3.2 MODEL TO PREDICT SILICA CONCENTRATE

[0]:
```
y = df['% Silica Concentrate']
X = df.drop(['% Silica Concentrate','index','datetime hours'], axis=1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪2,random_state=30)

X_train, X_cv, y_train, y_cv2 = train_test_split(X_train, y_train,test_size=0.
 ↪20,random_state=30)

from sklearn.preprocessing import StandardScaler
scale_features_std = StandardScaler()
features_train = scale_features_std.fit_transform(X_train)
features_test = scale_features_std.transform(X_test)

features_cv = scale_features_std.transform(X_cv)

print(X_train.shape,features_train.shape,X_test.shape,features_test.
 ↪shape,features_cv.shape,y_train.shape)


model.fit(features_train,y_train)

y_pred_train_silica=model.predict(features_train)
y_pred_cv_silica=model.predict(features_cv)
y_pred_test_silica=model.predict(features_test)


r2_cv_silica=r2_score(y_pred_cv_silica,y_cv2)
r2_train_silica=r2_score(y_pred_train_silica,y_train)
r2_test_silica=r2_score(y_pred_test_silica,y_test)

mse_train_silica=mean_squared_error(y_train,y_pred_train_silica)
mse_test_silica=mean_squared_error(y_test,y_pred_test_silica)
mse_cv_silica=mean_squared_error(y_cv2,y_pred_cv_silica)

r2_train_silica,r2_cv_silica,r2_test_silica,mse_train_silica,mse_cv_silica,mse_test_silica
```

```
(471969, 22) (471969, 22) (147491, 22) (147491, 22) (117993, 22) (471969,)
```

[0]:
```
(0.5259529825712774,
 0.526453891816218,
 0.5288178307374647,
 0.4068563070630419,
 0.4067576473714793,
 0.4035943750209711)
```

3.3 CONCATENATION OF TWO PREDICTIONS

[0]:
```
import numpy as np

y_pred_train_iron=y_pred_train_iron.reshape(-1,1)
y_pred_cv_iron=y_pred_cv_iron.reshape(-1,1)
y_pred_test_iron=y_pred_test_iron.reshape(-1,1)
y_pred_train_silica=y_pred_train_silica.reshape(-1,1)
y_pred_cv_silica=y_pred_cv_silica.reshape(-1,1)
y_pred_test_silica=y_pred_test_silica.reshape(-1,1)


y_pred_train=np.concatenate((y_pred_train_iron,y_pred_train_silica),axis=1)
y_pred_cv=np.concatenate((y_pred_cv_iron,y_pred_cv_silica),axis=1)
y_pred_test=np.concatenate((y_pred_test_iron,y_pred_test_silica),axis=1)

r2_cv=r2_score(y_pred_cv,y_cv)
r2_train=r2_score(y_pred_train,y_train)
r2_test=r2_score(y_pred_test,y_test)

mse_train=mean_squared_error(y_train,y_pred_train)
mse_test=mean_squared_error(y_test,y_pred_test)
mse_cv=mean_squared_error(y_cv,y_pred_cv)

r2_train,r2_cv,r2_test,mse_train,mse_cv,mse_test
```

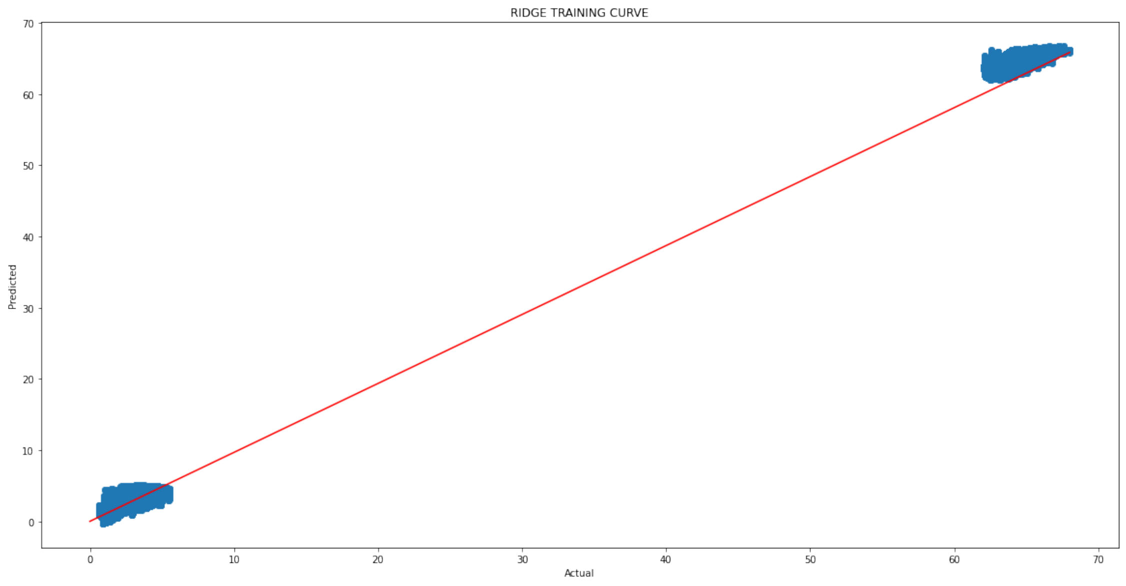
[0]:
```
(0.5202442337871346,
 0.5193853622009061,
 0.5222740262846326,
 0.4071919531656044,
 0.40797115490725677,
 0.4047967545750942)
```

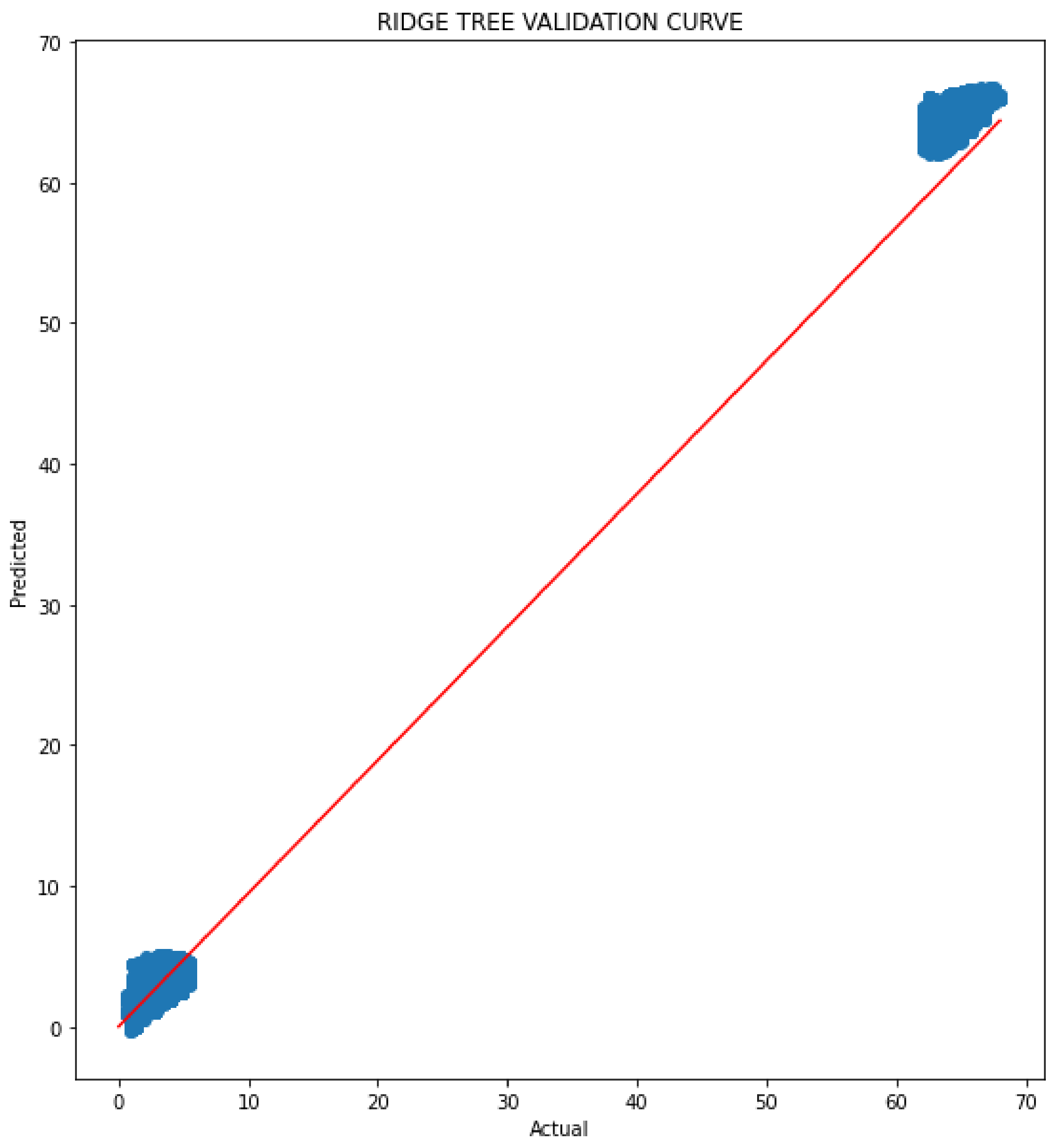
3.4 PLOTS

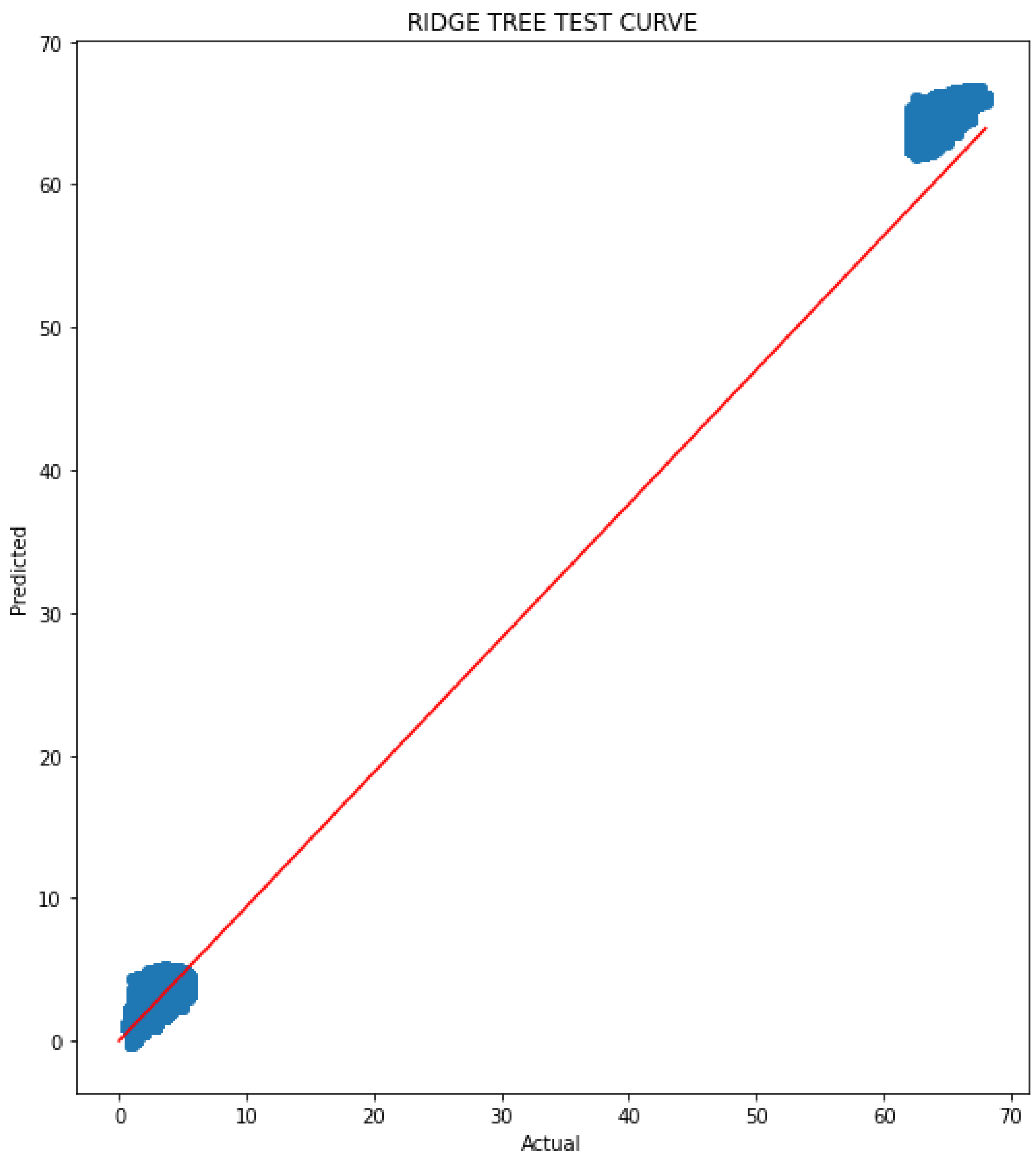
[0]:
```
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111)
ax.set(title="RIDGE TRAINING CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_train, y_pred_train)
ax.plot([0,max(y_train['% Iron Concentrate'])], [0,max(y_pred_train[0])],␣
 ↪color='r')
fig.show()
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(121)
ax.set(title="RIDGE TREE VALIDATION CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_cv, y_pred_cv)
ax.plot([0,max(y_cv['% Iron Concentrate'])], [0,max(y_pred[0])], color='r')
fig.show()
```

```
fig = plt.figure(figsize=(30, 10))
ax = fig.add_subplot(131)
ax.set(title="RIDGE TREE TEST CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_test, y_pred_test)
ax.plot([0,max(y_test['% Iron Concentrate'])], [0,max(y_pred_test[0])],␣
 ↪color='r')

fig.show()
```




RIDGE TREE VALIDATION CURVE
Predicted
Actual
0
10
20
30
40
50
60
70



3.5 REPORT

[0]:
```
#https://pypi.org/project/PrettyTable/-- for representation of data
#https://stackoverflow.com/questions/36423259/
 ↪how-to-use-pretty-table-in-python-to-print-out-data-from-multiple-lists-- to␣
 ↪add the rows in table
from prettytable import PrettyTable
x = PrettyTable(border=True, header=True, padding_width=15)
```

```
x.field_names = ["MODEL1",␣
↪"R2_TRAIN","R2_CV","R2_TEST","MSE_TRAIN","MSE_CV","MSE_TEST"]
x.add_row(["RIDGE MTR",0.1457070772811201,0.14798745903277455,
 0.1466686774031905,
 1.0761683118743428,
 1.0722436207597437,
 1.0729682609689142])
x.add_row(["RIDGE RIGRESSION CHAIN",0.1457070772811201,0.14798745903277455,
 0.1466686774031905,
 1.0761683118743428,
 1.0722436207597437,
 1.0729682609689142])

x.add_row(["RIDGE(NAIVE METHOD)",0.5202442337871346,0.5193853622009061,
 0.5222740262846326,
 0.4071919531656044,
 0.40797115490725677,
 0.4047967545750942])
print(x)
```

```
+----------------------------------------------------+--------------------------
----------------------+--------------------------------------------------+-------
-----------------------------------------+--------------------------------------
----------+--------------------------------------------------+------------------
-----------------------------+
|                        MODEL1                        |
R2_TRAIN                   |                      R2_CV                      |
R2_TEST                    |                   MSE_TRAIN                    |
MSE_CV                    |                    MSE_TEST                    |
+----------------------------------------------------+--------------------------
----------------------+--------------------------------------------------+-------
-----------------------------------------+--------------------------------------
----------+--------------------------------------------------+------------------
-----------------------------+
|                      RIDGE MTR                       |
0.1457070772811201                 |               0.14798745903277455
|                0.1466686774031905                |
1.0761683118743428                 |                1.0722436207597437
|                1.0729682609689142                |
|                RIDGE RIGRESSION CHAIN                |
0.1457070772811201                 |               0.14798745903277455
|                0.1466686774031905                |
1.0761683118743428                 |                1.0722436207597437
|                1.0729682609689142                |
|                 RIDGE(NAIVE METHOD)                  |
0.5202442337871346                 |                0.5193853622009061
|                0.5222740262846326                |
0.4071919531656044              |              0.40797115490725677
|              0.4047967545750942              |
+-----------------------------------------------+-------------------------
---------------------+-------------------------------------------------+-------
--------------------------------------------+-----------------------------------
----------+-----------------------------------------------+-------------------
-----------------------------+
```

3.6 ANALYSIS BETWEEN THREE METHODS

1. SINGLE TARGET MODEL:

The main drawback of these methods is that the relationships among the targets are ignored, and the targets are predicted independently, which may affect the overall quality of the predictions.

2. REGRESSOR CHAIN AND MTR :

The methods tend to give same result when implemented on ridge regression.

3.7 PLOTS INFERENCES

1. NAIVE METHOD

It is simply not a perefect method for analysis and it has better result because it doent take into count of target dependencies.

2. Regressor chain and MTR

In train curve, the points tend to overlay on the line and in both iron and silica the points are overfalling on regression line
In test and validation curve, the models are able to predict iron concetrate and are not able to predict the silica as the samw way in iron.

3.8 METRIC ANALYSIS

4 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html?

5 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared

R2 SCORE
Acoording to literature, the r2 score is good when it is closer to 1 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.
TRAIN R2 is so closer to 0 and TEST r2 is also to 0 and hence inorder to get better result , we must try other models
MSE VALUE
A non-negative floating point value (the best value is 0.0), or an array of floating point values, one for each individual target.
The train,cv,test loss and r2 score is not very good.

5.1 CONCLUSION

NAIVE METHOD must not be used in prediction and neither MTR and Regression with ridge performance are not satisfactory . Hence, to predict two target variables at same time we need to not ridge.

6 XGBOOST

6.1 DATA MODELING

[0]:
```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np
```

[0]:
```
import pandas as pd

df=pd.read_csv('/content/drive/My Drive/preprocessed_time')

y = df.iloc[:,23:25]
X = df.drop(['% Silica Concentrate','% Iron Concentrate','index','datetime␣
 ↪hours'], axis=1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪2,random_state=30)

X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,test_size=0.
 ↪20,random_state=30)

from sklearn.preprocessing import StandardScaler
scale_features_std = StandardScaler()
features_train = scale_features_std.fit_transform(X_train)
features_test = scale_features_std.transform(X_test)

features_cv = scale_features_std.transform(X_cv)

print(X_train.shape,features_train.shape,X_test.shape,features_test.
 ↪shape,features_cv.shape,y_train.shape,y_cv.shape,y_test.shape)
```

```
(471969, 21) (471969, 21) (147491, 21) (147491, 21) (117993, 21) (471969, 2)
(117993, 2) (147491, 2)
```

6.2 MODEL USING MULTIOUTPUTREGRESSOR

[0]:
```
import xgboost as xgb
from sklearn.multioutput import MultiOutputRegressor
```

[0]:
```
clf=xgb.XGBRegressor(max_depth=10 ,learning_rate=0.01,n_estimators=1000␣
 ↪,verbose=2,subsample=0.1,colsample_bytree=1)
```

[0]:
```
model=MultiOutputRegressor(clf)
model.fit(features_train,y_train)
```

```
[01:46:32] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[02:15:55] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
```

[0]:
```
MultiOutputRegressor(estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                            colsample_bylevel=1,
                                            colsample_bynode=1,
                                            colsample_bytree=1, gamma=0,
                                            importance_type='gain',
                                            learning_rate=0.01,
                                            max_delta_step=0, max_depth=10,
                                            min_child_weight=1, missing=None,
                                            n_estimators=1000, n_jobs=1,
                                            nthread=None,
                                            objective='reg:linear',
                                            random_state=0, reg_alpha=0,
                                            reg_lambda=1, scale_pos_weight=1,
                                            seed=None, silent=None,
                                            subsample=0.1, verbose=2,
                                            verbosity=1),
                     n_jobs=None)
```

7 PREDICTION

[0]:
```
y_pred=model.predict(features_cv)
y_pred_train=model.predict(features_train)
y_pred_test=model.predict(features_test)
```

8 EVALUATION METRIC

[0]:
```
mse_train=mean_squared_error(y_train,y_pred_train)
mse_cv=mean_squared_error(y_cv,y_pred)
mse_test=mean_squared_error(y_test,y_pred_test)
```

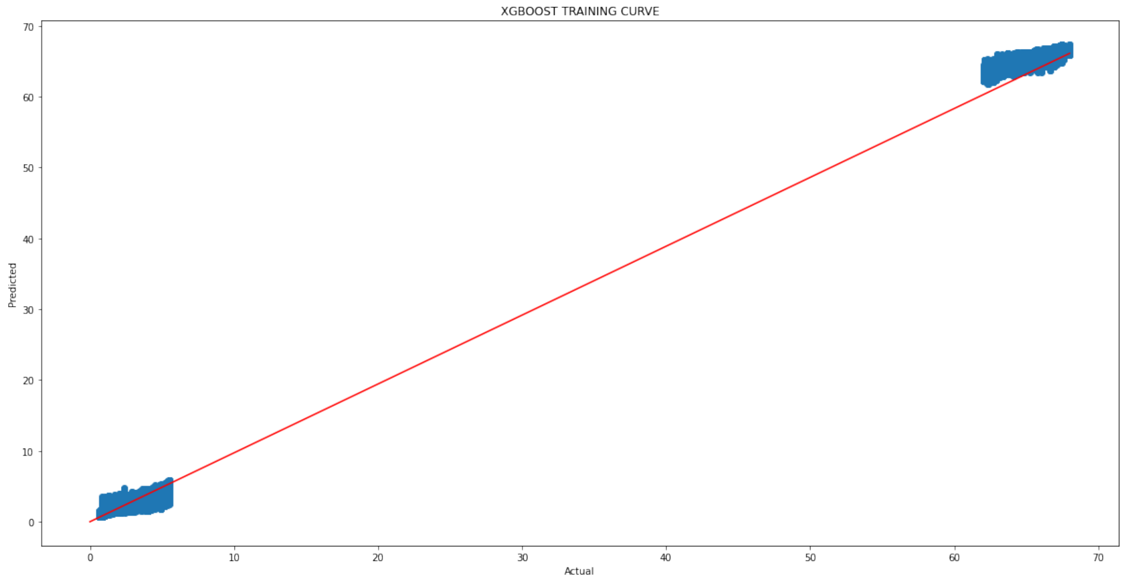
```
[0]: r2_train=r2_score(y_train,y_pred_train)
     r2_cv=r2_score(y_cv,y_pred)
     r2_test=r2_score(y_test,y_pred_test)
```

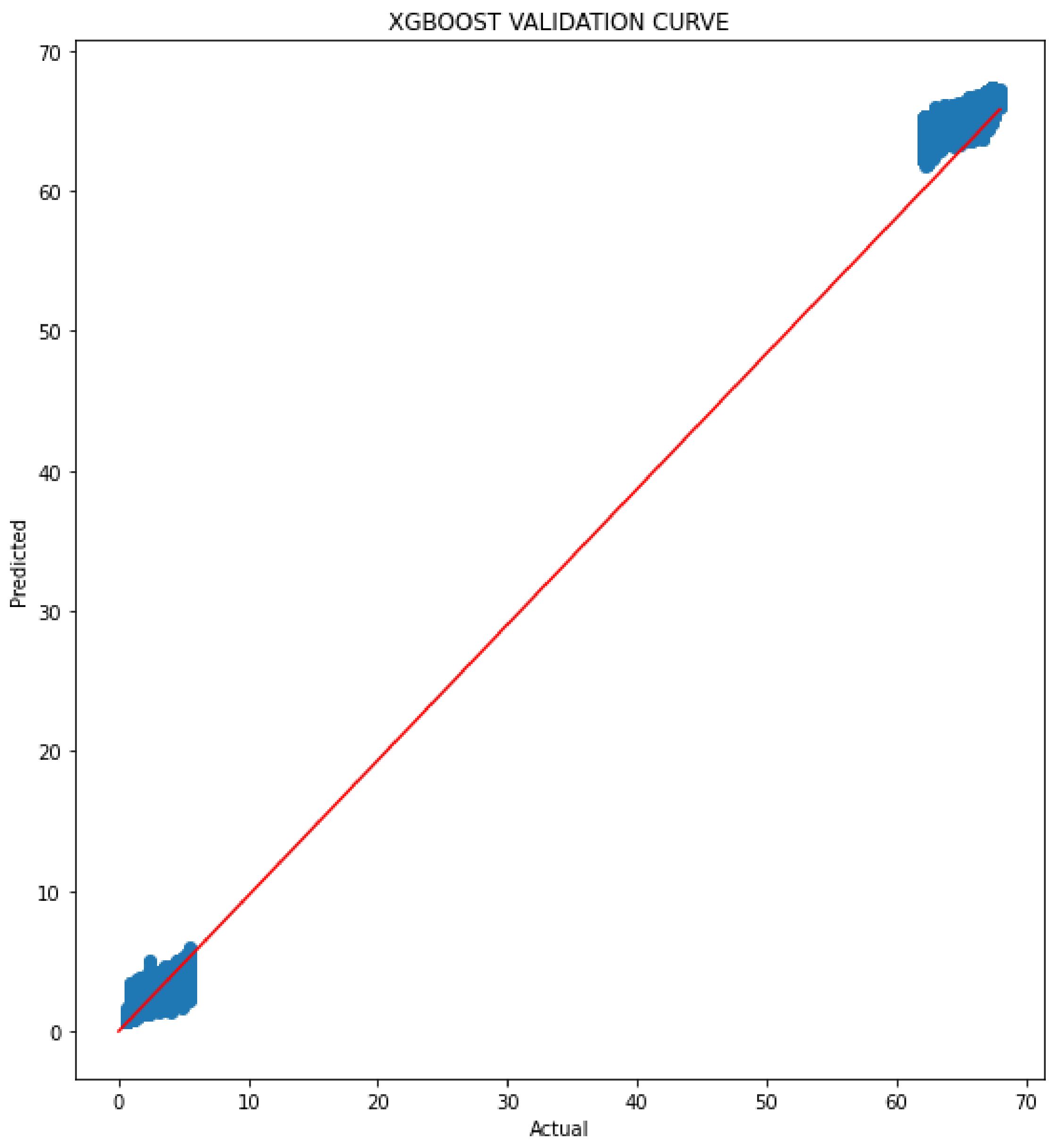
```
[0]: r2_train,r2_cv,r2_test,mse_cv,mse_test,mse_train,mse_cv
```

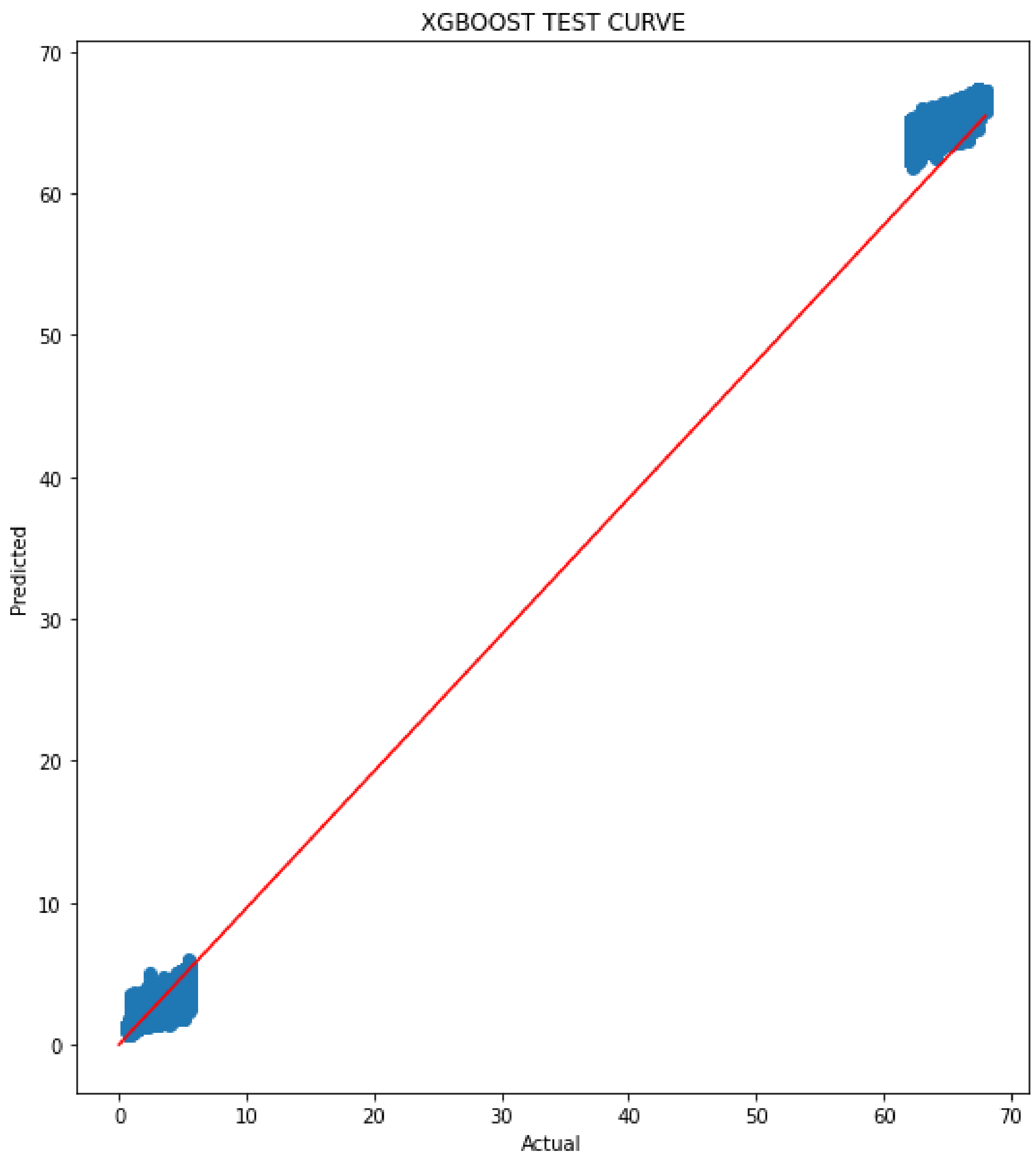
```
[0]: (0.790067573655429,
      0.7760975535975633,
      0.7750269140015339,
      0.2816141156963654,
      0.2827106725163956,
      0.2641687929200151,
      0.2816141156963654)
```

9 PLOTS

```
[0]: fig = plt.figure(figsize=(20, 10))
     ax = fig.add_subplot(111)
     ax.set(title="XGBOOST TRAINING CURVE", xlabel="Actual", ylabel="Predicted")
     ax.scatter(y_train, y_pred_train)
     ax.plot([0,max(y_train['% Iron Concentrate'])], [0,max(y_pred_train[0])],␣
      ↪color='r')
     fig.show()
     fig = plt.figure(figsize=(20, 10))
     ax = fig.add_subplot(121)
     ax.set(title="XGBOOST VALIDATION CURVE", xlabel="Actual", ylabel="Predicted")
     ax.scatter(y_cv, y_pred)
     ax.plot([0,max(y_cv['% Iron Concentrate'])], [0,max(y_pred[0])], color='r')
     fig.show()
     fig = plt.figure(figsize=(30, 10))
     ax = fig.add_subplot(131)
     ax.set(title="XGBOOST TEST CURVE", xlabel="Actual", ylabel="Predicted")
     ax.scatter(y_test, y_pred_test)
     ax.plot([0,max(y_test['% Iron Concentrate'])], [0,max(y_pred_test[0])],␣
      ↪color='r')
     fig.show()
```


XGBOOST TRAINING CURVE
Predicted
Actual
0
10
20
30
40
50
60
70


XGBOOST VALIDATION CURVE
Predicted
Actual
0
10
20
30
40
50
60
70



10 MODEL USING REGRESSOR CHAIN

[0]:
```
from sklearn.multioutput import RegressorChain
```

[0]:
```
model=xgb.XGBRegressor(max_depth=10 ,learning_rate=0.01,n_estimators=1000␣
 ↪,verbose=2,subsample=0.1,colsample_bytree=1)
wrapper1 = RegressorChain(model)
wrapper1.fit(features_train,y_train)
```

```
[05:39:19] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
[06:09:35] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
```

[0]:
```
RegressorChain(base_estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           importance_type='gain',
                                           learning_rate=0.01, max_delta_step=0,
                                           max_depth=10, min_child_weight=1,
                                           missing=None, n_estimators=1000,
                                           n_jobs=1, nthread=None,
                                           objective='reg:linear',
                                           random_state=0, reg_alpha=0,
                                           reg_lambda=1, scale_pos_weight=1,
                                           seed=None, silent=None,
                                           subsample=0.1, verbose=2,
                                           verbosity=1),
               cv=None, order=None, random_state=None)
```

11 PREDICTION

[0]:
```
y_pred_cv=wrapper1.predict(features_cv)
y_pred_train=wrapper1.predict(features_train)
y_pred_test=wrapper1.predict(features_test)
```

12 EVALUATION METRIC

[0]:
```
r2_cv=r2_score(y_pred_cv,y_cv)
r2_train=r2_score(y_pred_train,y_train)
r2_test=r2_score(y_pred_test,y_test)
```

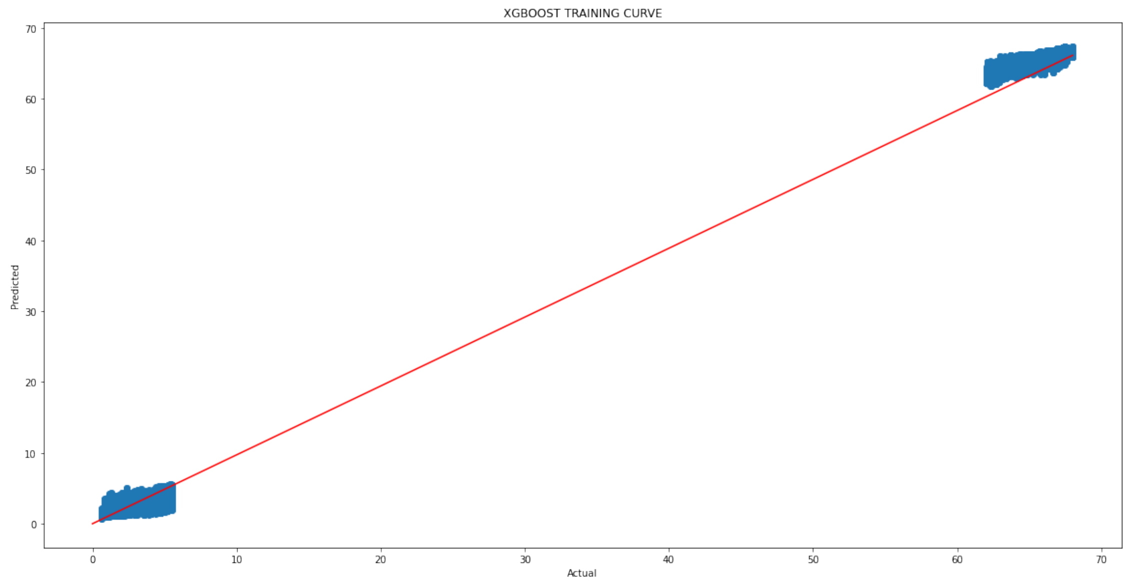
[0]:
```
mse_train=mean_squared_error(y_train,y_pred_train)
mse_cv=mean_squared_error(y_cv,y_pred)
mse_test=mean_squared_error(y_test,y_pred_test)
```

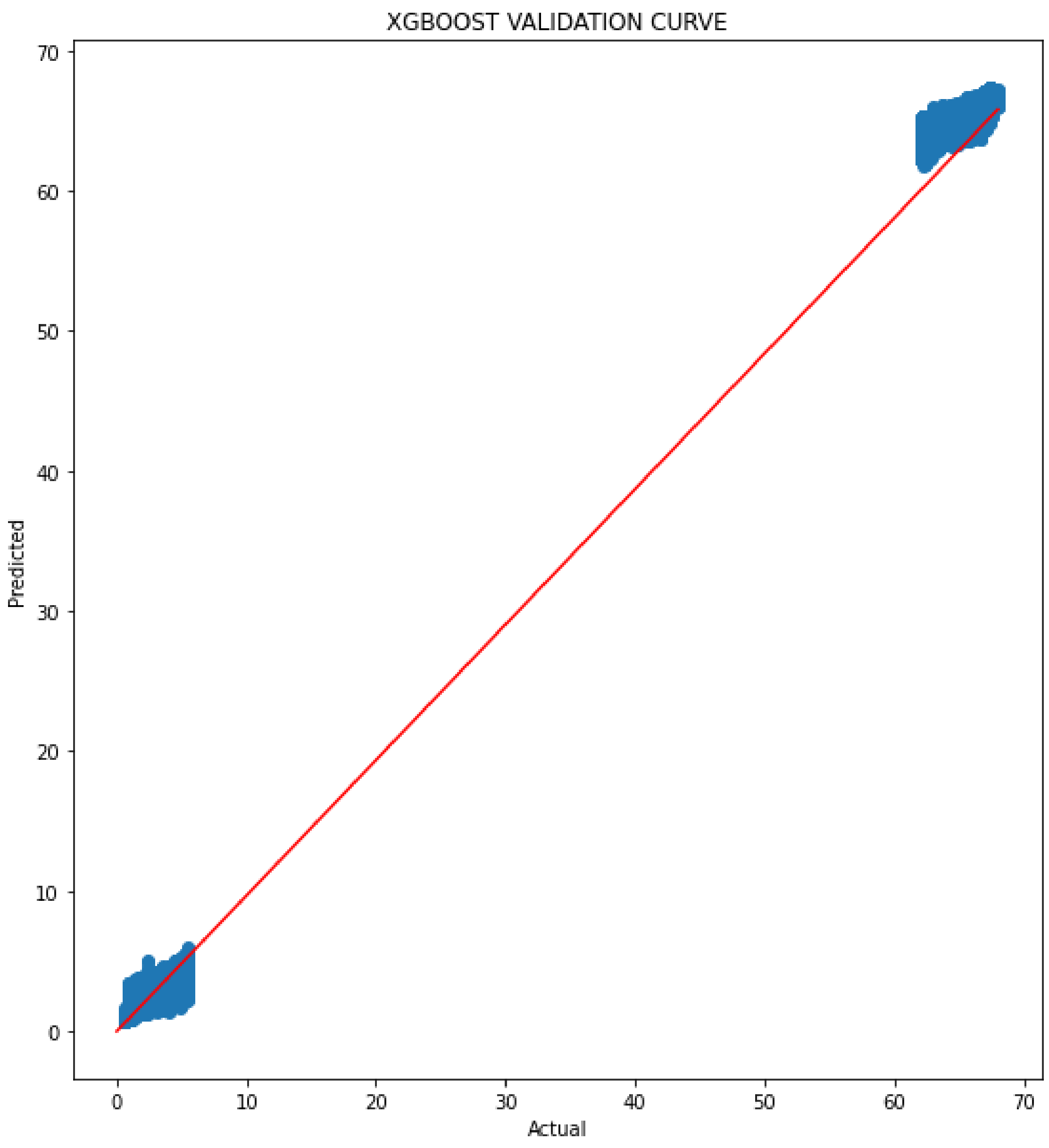
[0]:
```
r2_train,r2_cv,r2_test,mse_train,mse_cv,mse_test
```

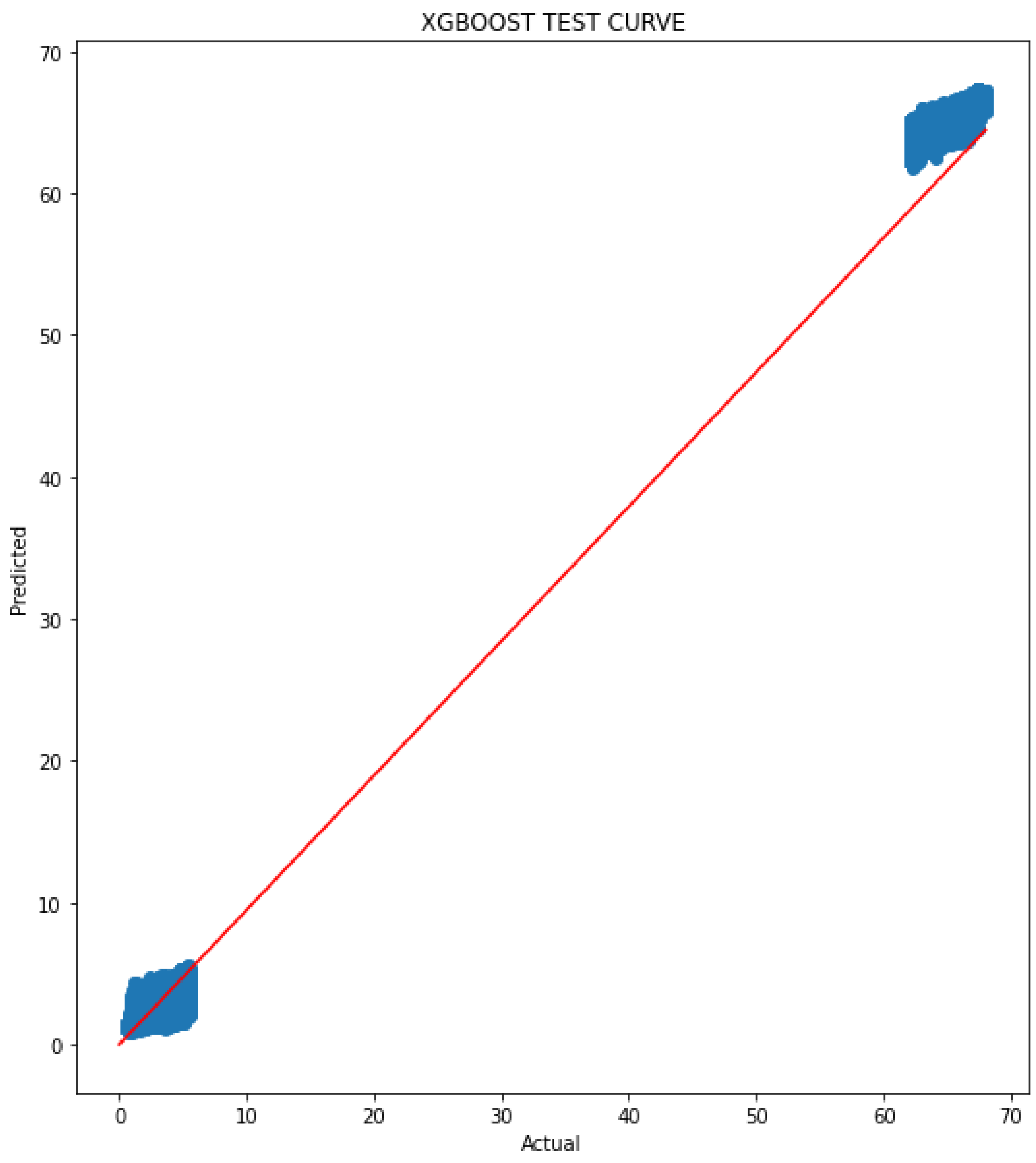
[0]:
```
(0.5341046012820809,
 0.5105948839081095,
 0.5101846872368989,
 0.3321947153953305,
 0.2816141156963654,
 0.3455795604174674)
```

13 PLOTS

[0]:
```
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111)
ax.set(title="XGBOOST TRAINING CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_train, y_pred_train)
ax.plot([0,max(y_train['% Iron Concentrate'])], [0,max(y_pred_train[0])],␣
 ↪color='r')
fig.show()
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(121)
ax.set(title="XGBOOST VALIDATION CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_cv, y_pred)
ax.plot([0,max(y_cv['% Iron Concentrate'])], [0,max(y_pred[0])], color='r')
fig.show()
fig = plt.figure(figsize=(30, 10))
ax = fig.add_subplot(131)
ax.set(title="XGBOOST TEST CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_test, y_pred_test)
ax.plot([0,max(y_test['% Iron Concentrate'])], [0,max(y_pred_test[0])],␣
 ↪color='r')
fig.show()
```




XGBOOST VALIDATION CURVE
Predicted
Actual
0
10
20
30
40
50
60
70



13.1 REPORT

[0]:
```
#https://pypi.org/project/PrettyTable/-- for representation of data
#https://stackoverflow.com/questions/36423259/
 ↪how-to-use-pretty-table-in-python-to-print-out-data-from-multiple-lists-- to␣
 ↪add the rows in table
from prettytable import PrettyTable
x = PrettyTable(border=True, header=True, padding_width=15)
```

```
x.field_names = ["MODEL1",␣
 ↪"R2_TRAIN","R2_CV","R2_TEST","MSE_TRAIN","MSE_CV","MSE_TEST"]
x.add_row(["XGBOOST MTR",0.790067573655429,
 0.7760975535975633,
 0.7750269140015339,
 0.2816141156963654,
 0.2827106725163956,
 0.2641687929200151,

])
x.add_row(["XGBOOST RIGRESSION CHAIN",0.5341046012820809,
 0.5105948839081095,
 0.5101846872368989,
 0.3321947153953305,
 0.2816141156963654,
 0.3455795604174674])
print(x)
```

```
+-------------------------------------------------------+------------------------
------------------------+-------------------------------------------------+------
------------------------------------------+-------------------------------------
-----------+-------------------------------------------------+-------------------
-----------------------------+
|                        MODEL1                         |
R2_TRAIN                    |                       R2_CV                     |
R2_TEST                     |                     MSE_TRAIN                   |
MSE_CV                     |                     MSE_TEST                    |
+-------------------------------------------------------+------------------------
------------------------+-------------------------------------------------+------
------------------------------------------+-------------------------------------
-----------+-------------------------------------------------+-------------------
-----------------------------+
|                     XGBOOST MTR                       |
0.790067573655429                |               0.7760975535975633
|                0.7750269140015339               |
0.2816141156963654               |               0.2827106725163956
|                0.2641687929200151               |
|                XGBOOST RIGRESSION CHAIN               |
0.5341046012820809               |               0.5105948839081095
|                0.5101846872368989               |
0.3321947153953305               |               0.2816141156963654
|                0.3455795604174674               |
+-------------------------------------------------------+------------------------
------------------------+-------------------------------------------------+------
------------------------------------------+-------------------------------------
-----------+-------------------------------------------------+-------------------
-----------------------------+
```

13.2 ANALYSIS BETWEEN METHODS

1. MTR AND REGRESSION CHAIN

The MTR has better performances than regression chain and the loss value is also low and plots are also better in MTR

13.3 PLOTS ANALYSIS

1. MTR

In train curve, the points tend to overlay on the line and in both iron and silica the points are overfalling on regression line

In test and validation curve, the models are able to predict iron concetrate and are able to predict the silica better in the same way iron concentrate

2. Regressor chain

In train curve, the points tend to overlay on the line and in both iron and silica the points are overfalling on regression line

In test curve, the models are able to predict iron concetrate and are not able to predict the silica better than iron concentrate.

13.4 METRIC ANALYSIS

14 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html

15 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_square

R2 SCORE

Acoording to literature, the r2 score is good when it is closer to 1 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

TRAIN R2 is so closer to 0 and TEST r2 is also to 0 and hence inorder to get better result , we must try other models

MSE VALUE

A non-negative floating point value (the best value is 0.0), or an array of floating point values, one for each individual target.

The MSE value and R2 score value is better than ridge and MSE value of MTR is better than REGRESSION CHAIN model.

15.1 CONCLUSION

MTR IS BETTER THAN REGRESSION CHAIN MODEL IN CASE OF XGBOOST XG-BOOST>RIDGE

16 DECISION TREE

16.1 Inherently Multioutput Regression Algorithms

Some regression machine learning algorithms support multiple outputs directly.

This includes most of the popular machine learning algorithms implemented in the scikit-learn library, such as:

LinearRegression (and related)
KNeighborsRegressor
DecisionTreeRegressor
RandomForestRegressor (and related)

16.2 DATA MODELING

[0]:
```
from sklearn.ensemble import RandomForestRegressor
```

[0]:
```
import pandas as pd
```

[0]:
```
df=pd.read_csv('/content/drive/My Drive/preprocessed_time')
```

[0]:
```
df[:2]
```

[0]:
```
                 index  ... % Silica Concentrate
0  2017-03-10 01:02:00  ...                 1.31
1  2017-03-10 01:02:20  ...                 1.31

[2 rows x 25 columns]
```

[0]:
```
y = df.iloc[:,23:25]
X = df.drop(['% Silica Concentrate','% Iron Concentrate','index','datetime␣
 ↪hours'], axis=1)
```

[0]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪2,random_state=30)
```

[0]:
```
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,test_size=0.
 ↪20,random_state=30)
```

[0]:
```
from sklearn.preprocessing import StandardScaler
scale_features_std = StandardScaler()
features_train = scale_features_std.fit_transform(X_train)
features_test = scale_features_std.transform(X_test)
```

[0]:
```
features_cv = scale_features_std.transform(X_cv)
```

[0]:
```
print(X_train.shape,features_train.shape,X_test.shape,features_test.
 ↪shape,features_cv.shape,y_train.shape,y_test.shape,y_cv.shape)
```

```
(471969, 21) (471969, 21) (147491, 21) (147491, 21) (117993, 21) (471969, 2)
(147491, 2) (117993, 2)
```

[0]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
```

16.3 MODEL FOR MULTIOUTPUT

[0]:
```
#https://machinelearningmastery.com/multi-output-regression-models-with-python/
dt2 = DecisionTreeRegressor()
se=dt2.fit(features_train,y_train)
parameters = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10,100, 500]}
cfl2=RandomizedSearchCV(dt2,param_distributions=parameters,verbose=10,n_jobs=-1,)
se2 = cfl2.fit(features_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   17.3s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:   33.6s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   43.5s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   51.3s
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  4.3min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  4.6min finished
```

[0]:
```
print(cfl2.best_estimator_)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=50,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=5,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

[0]:
```
best_tune_parameters=[{'max_depth':[50], 'min_samples_split':[5] }]
```

[0]:
```
clf=DecisionTreeRegressor (max_depth=50,min_samples_split=5)
clf.fit(features_train, y_train)
```

[0]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=50,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=5,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

16.4 PREDICTION

[0]:
```
y_pred=clf.predict(features_cv)
```

[0]:
```
y_pred_train=clf.predict(features_train)
y_pred_test=clf.predict(features_test)
```

[0]:
```
y_pred_train
```

[0]:
```
array([[66.68 ,  1.44 ],
       [64.7  ,  3.77 ],
       [63.8  ,  3.87 ],
       ...,
       [65.39 ,  2.18 ],
       [65.405,  1.79 ],
       [65.06 ,  2.13 ]])
```

16.5 EVALUATION METRIC

[0]:
```
R2 = r2_score(y_cv, y_pred)
print(R2)
```

```
0.9036198036618404
```

[0]:
```
mse_train=mean_squared_error(y_train,y_pred_train)
mse_test=mean_squared_error(y_test,y_pred_test)
mse_cv=mean_squared_error(y_cv,y_pred)
```

[0]:
```
r2_train=r2_score(y_train,y_pred_train)
```

[0]:
```
r2_train
```

[0]:
```
0.9969342900667011
```

[0]:
```
r2_test=r2_score(y_test,y_pred_test)
```

[0]:
```
r2_test
```

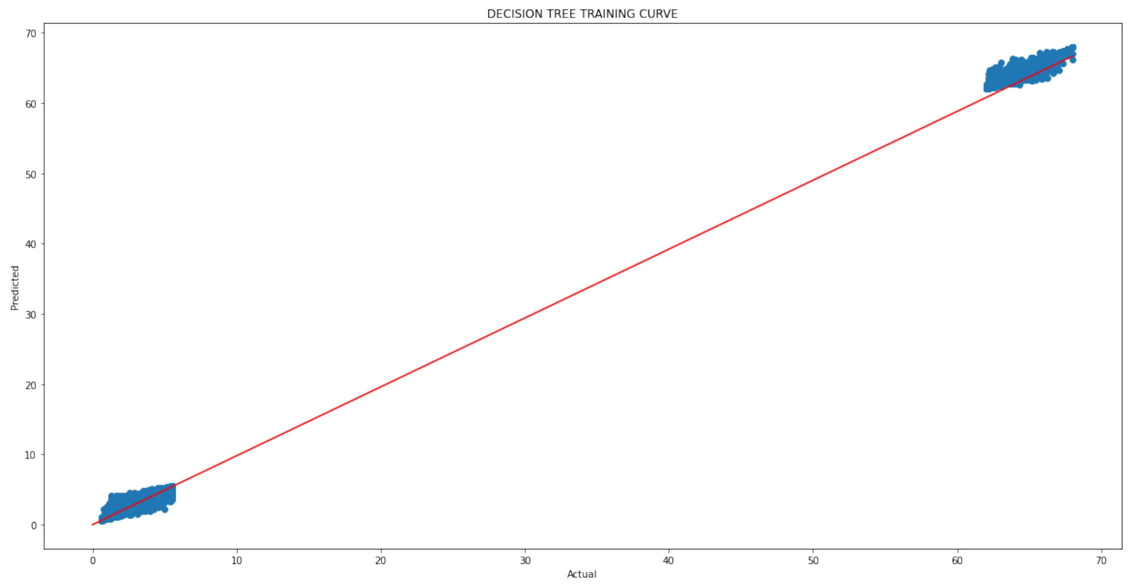
[0]:
```
0.9018280433286507
```

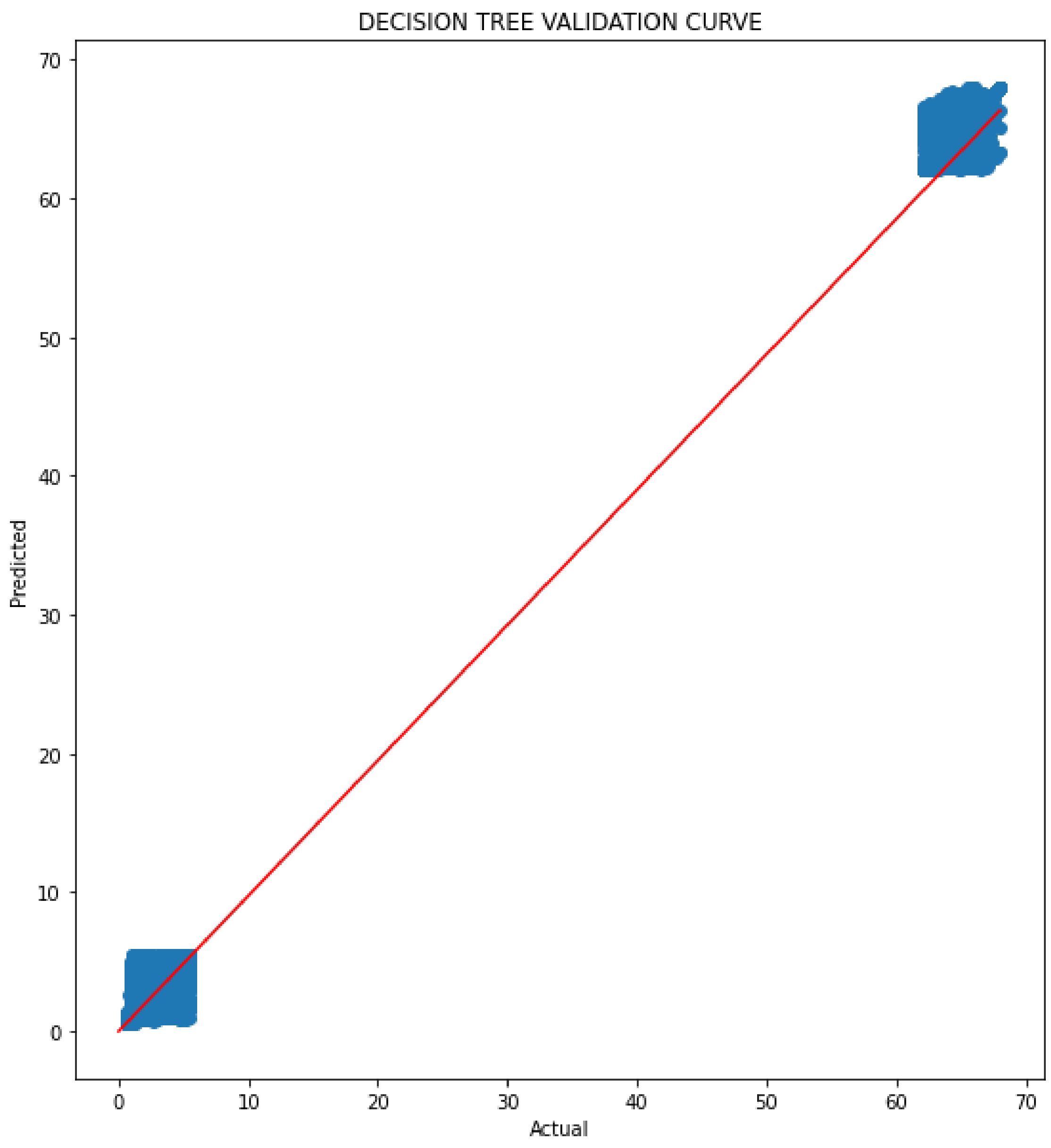
[0]:
```
r2_train,r2_test,mse_cv,mse_test,mse_train,mse_cv,R2
```

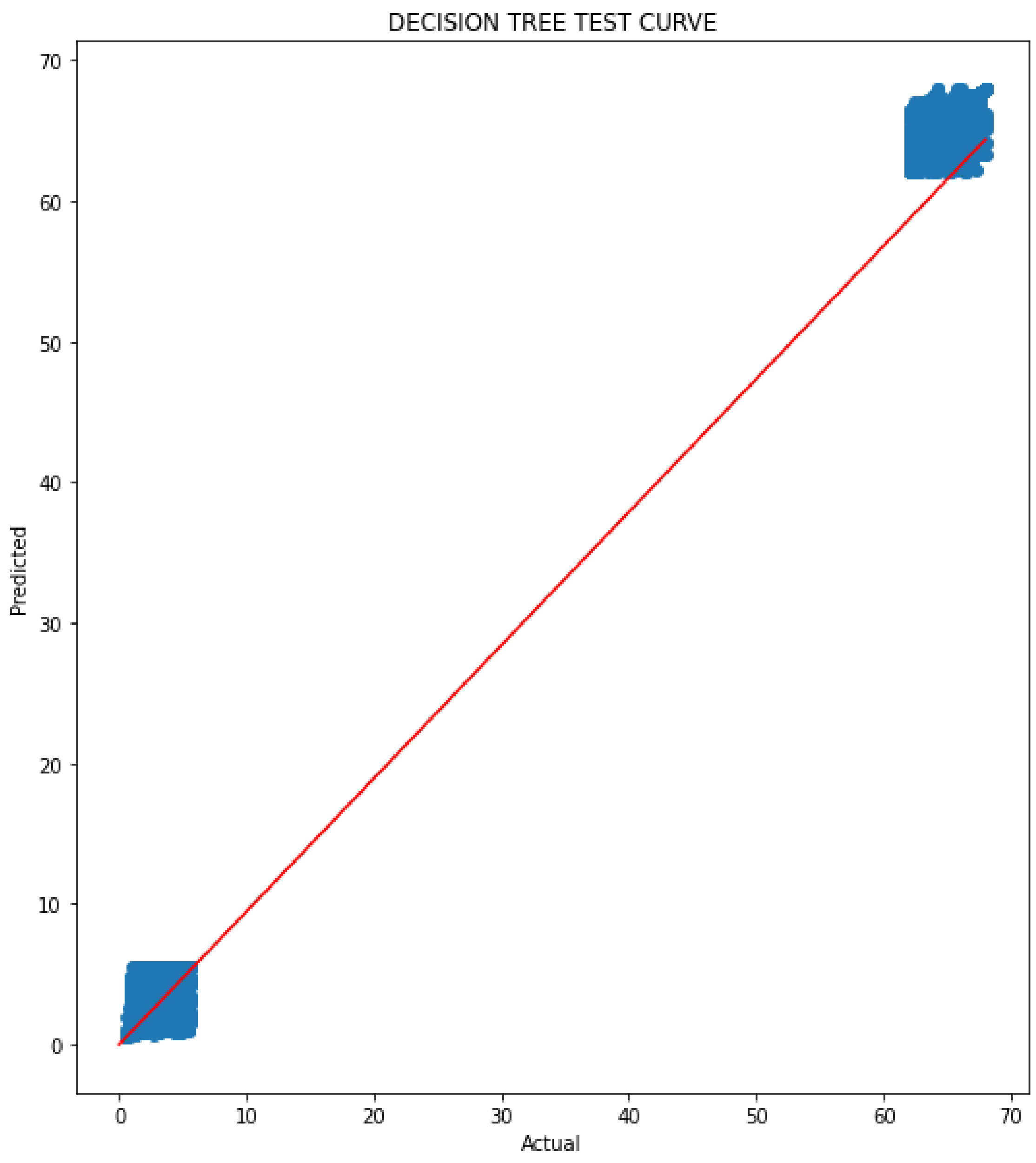
[0]:
```
(0.996940349079114,
 0.9010834860487484,
 0.12129044877301248,
 0.12437333275470228,
 0.0038556643517233397,
 0.12129044877301248,
 0.9036198036618404)
```

16.6 PLOTS

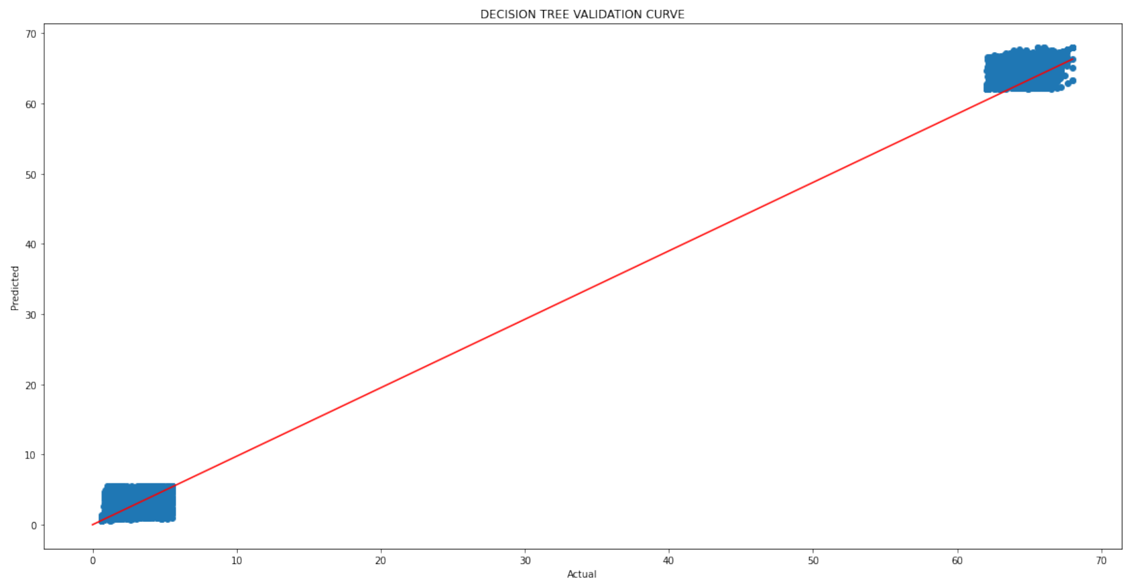
[0]:
```
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111)
ax.set(title="DECISION TREE TRAINING CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_train, y_pred_train)
ax.plot([0,max(y_train['% Iron Concentrate'])], [0,max(y_pred_train[0])],␣
 ↪color='r')
fig.show()
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(121)
ax.set(title="DECISION TREE VALIDATION CURVE", xlabel="Actual",␣
 ↪ylabel="Predicted")
ax.scatter(y_cv, y_pred)
ax.plot([0,max(y_cv['% Iron Concentrate'])], [0,max(y_pred[0])], color='r')
fig.show()
fig = plt.figure(figsize=(30, 10))
ax = fig.add_subplot(131)
ax.set(title="DECISION TREE TEST CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_test, y_pred_test)
ax.plot([0,max(y_test['% Iron Concentrate'])], [0,max(y_pred_test[0])],␣
 ↪color='r')
fig.show()
```




DECISION TREE VALIDATION CURVE
Predicted
Actual
0
10
20
30
40
50
60
70



[0]:
```
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111)
ax.set(title="DECISION TREE VALIDATION CURVE", xlabel="Actual",␣
 ↪ylabel="Predicted")
ax.scatter(y_cv, y_pred)
ax.plot([0,max(y_cv['% Iron Concentrate'])], [0,max(y_pred[0])], color='r')
fig.show()
```



16.7 MODEL USING REGRESSOR CHAIN

[0]:
```
from sklearn.multioutput import RegressorChain
```

[0]:
```
clf=DecisionTreeRegressor (max_depth=50,min_samples_split=5)
wrapper1 = RegressorChain(clf)
wrapper1.fit(features_train,y_train)
```

[0]:
```
RegressorChain(base_estimator=DecisionTreeRegressor(ccp_alpha=0.0,
                                                    criterion='mse',
                                                    max_depth=50,
                                                    max_features=None,
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=5,
min_weight_fraction_leaf=0.0,
                                                    presort='deprecated',
                                                    random_state=None,
                                                    splitter='best'),
               cv=None, order=None, random_state=None)
```

16.8 PREDICTION

[0]:
```
y_pred_cv=wrapper1.predict(features_cv)
y_pred_train=wrapper1.predict(features_train)
y_pred_test=wrapper1.predict(features_test)
```

16.9 EVALUATION METRIC

[0]:
```
r2_cv=r2_score(y_pred_cv,y_cv)
r2_train=r2_score(y_pred_train,y_train)
r2_test=r2_score(y_pred_test,y_test)
```

[0]:
```
mse_train=mean_squared_error(y_train,y_pred_train)
mse_test=mean_squared_error(y_test,y_pred_test)
mse_cv=mean_squared_error(y_cv,y_pred_cv)
```

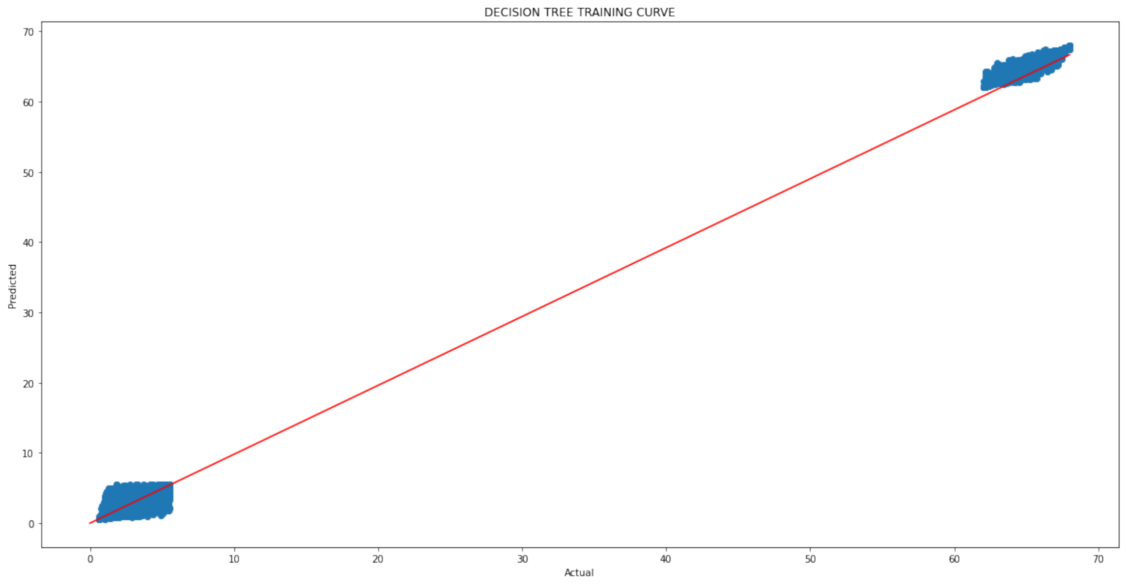
[0]:
```
r2_train,r2_cv,r2_test,mse_train,mse_cv,mse_test
```

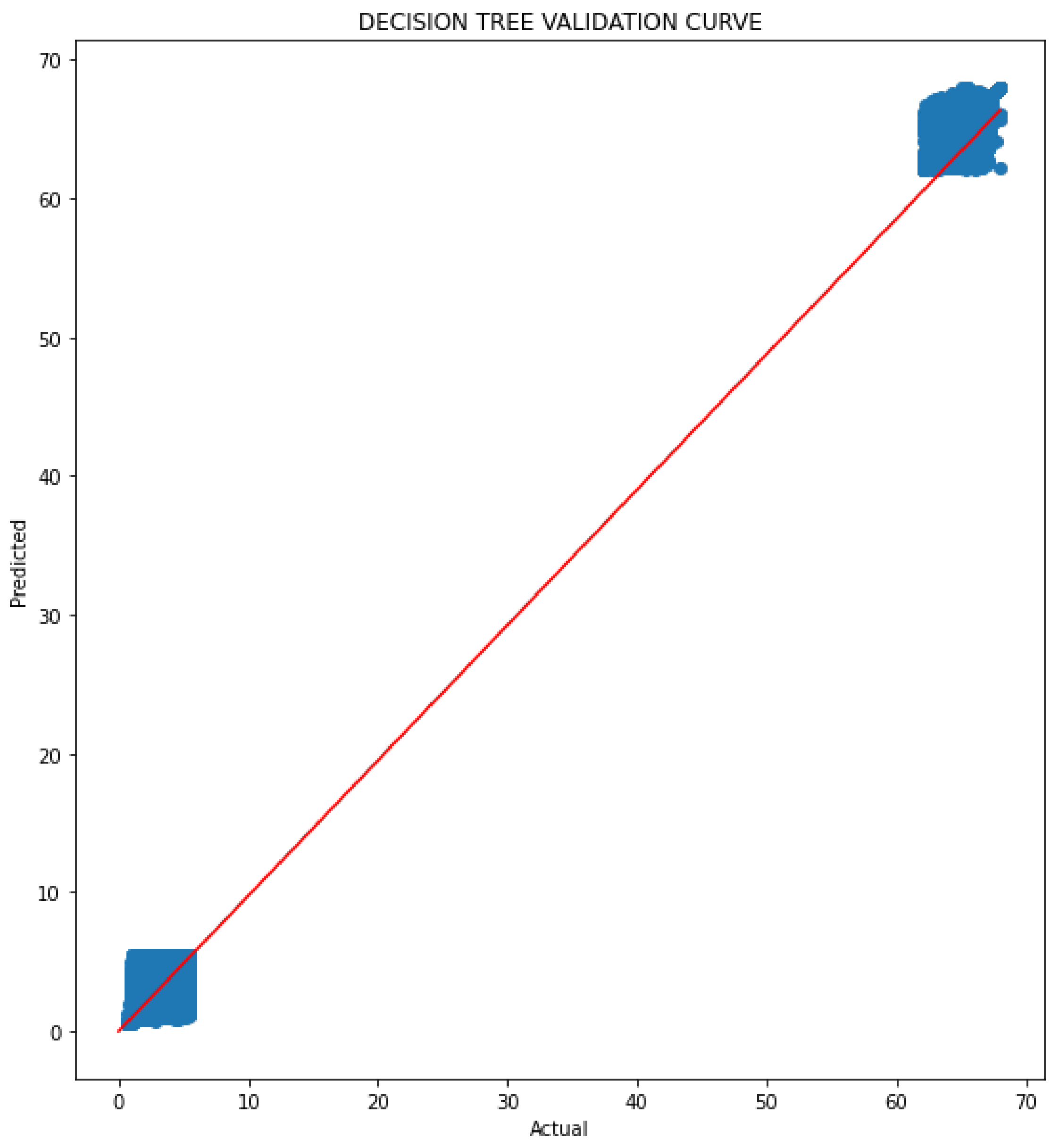
[0]:
```
(0.9908990814437093,
 0.8790498091768016,
 0.8774055851075784,
 0.011506126265385405,
 0.15191380344978905,
 0.1537079542189485)
```

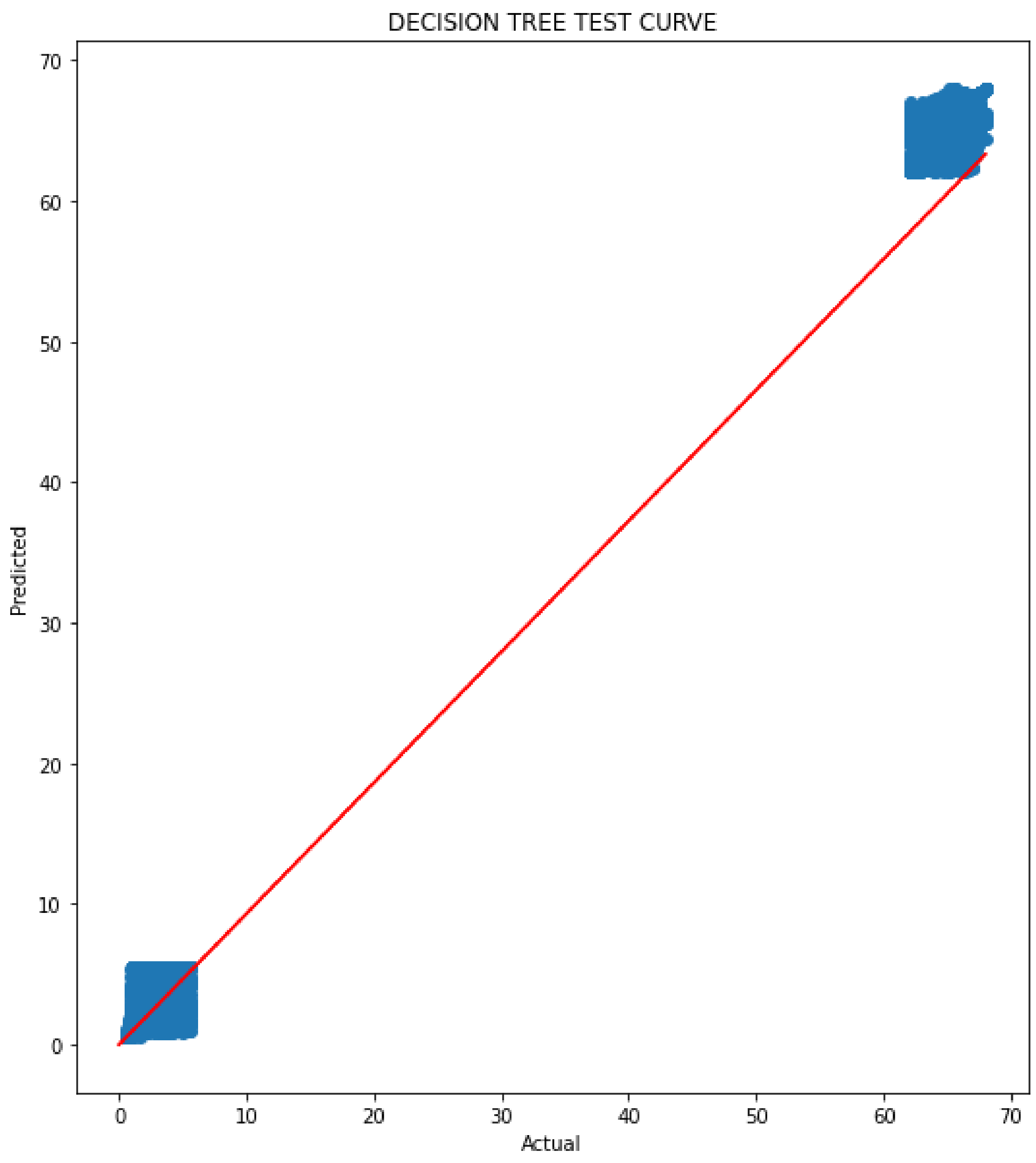
16.10 PLOTS

[0]:
```
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111)
ax.set(title="DECISION TREE TRAINING CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_train, y_pred_train)
ax.plot([0,max(y_train['% Iron Concentrate'])], [0,max(y_pred_train[0])],␣
 ↪color='r')
fig.show()
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(121)
ax.set(title="DECISION TREE VALIDATION CURVE", xlabel="Actual",␣
 ↪ylabel="Predicted")
ax.scatter(y_cv, y_pred_cv)
ax.plot([0,max(y_cv['% Iron Concentrate'])], [0,max(y_pred[0])], color='r')
fig.show()
fig = plt.figure(figsize=(30, 10))
ax = fig.add_subplot(131)
ax.set(title="DECISION TREE TEST CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_test, y_pred_test)
ax.plot([0,max(y_test['% Iron Concentrate'])], [0,max(y_pred_test[0])],␣
 ↪color='r')
```

```
ax.plot([0,max(y_test['% Iron Concentrate'])], [0,max(y_pred_test[0])],␣
 ↪color='r')
fig.show()
```




DECISION TREE VALIDATION CURVE
Predicted
Actual
0
10
20
30
40
50
60
70



16.11 REPORT

[3]:
```
#https://pypi.org/project/PrettyTable/-- for representation of data
#https://stackoverflow.com/questions/36423259/
 ↪how-to-use-pretty-table-in-python-to-print-out-data-from-multiple-lists-- to␣
 ↪add the rows in table
from prettytable import PrettyTable
x = PrettyTable(border=True, header=True, padding_width=15)
```

```
x.field_names = ["MODEL1",␣
 ↪"R2_TRAIN","R2_CV","R2_TEST","MSE_TRAIN","MSE_CV","MSE_TEST"]
x.add_row(["DECISION TREE MTR",0.996940349079114,
 0.9010834860487484,
 0.9036198036618404, 0.0038556643517233397,
 0.12129044877301248,
 0.12437333275470228
])
x.add_row(["DECISION TREE RIGRESSION CHAIN",0.9908990814437093,
 0.8790498091768016,
 0.8774055851075784,
 0.011506126265385405,
 0.15191380344978905,
 0.1537079542189485])

print(x)
```

```
+------------------------------------------------------------+------------------
------------------------------+-------------------------------------------------+
-------------------------------------------------+-------------------------------
--------------------+------------------------------------------------+---------
----------------------------------------+
|                           MODEL1                             |
R2_TRAIN                      |                    R2_CV                     |
R2_TEST                       |                    MSE_TRAIN
|                      MSE_CV                     |                     MSE_TEST
|
+------------------------------------------------------------+------------------
------------------------------+-------------------------------------------------+
-------------------------------------------------+-------------------------------
--------------------+------------------------------------------------+---------
----------------------------------------+
|                     DECISION TREE MTR                       |
0.996940349079114                |               0.9010834860487484
|               0.9036198036618404               |
0.0038556643517233397               |               0.12129044877301248
|               0.12437333275470228               |
|               DECISION TREE RIGRESSION CHAIN                 |
0.9908990814437093               |               0.8790498091768016
|               0.8774055851075784               |
0.011506126265385405               |               0.15191380344978905
|                0.1537079542189485               |
+------------------------------------------------------------+------------------
------------------------------+-------------------------------------------------+
-------------------------------------------------+-------------------------------
--------------------+------------------------------------------------+---------
----------------------------------------+
```

16.12 METRIC ANALYSIS

17 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html

18 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_square

R2 SCORE

Acoording to literature, the r2 score is good when it is closer to 1 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

TRAIN R2 OF INHERENT MODEL is so closer to 1 and TEST r2 OF INHERENT MODEL is also to 1 and hence inorder to get better result , we must try other models

MSE VALUE

A non-negative floating point value (the best value is 0.0), or an array of floating point values, one for each individual target.

The MSE value and R2 score value is better than ridge and MSE value of INHERENT MODEL is better than REGRESSION CHAIN model.

18.1 PLOTS ANALYSIS

1. INHERENT MTR MODEL

In train curve, the points tend to overlay on the line and in both iron and silica the points are overfalling on regression line

In test and validation curve, the models are able to predict iron concetrate and are able to predict the silica better in the same way for iron concentrate

2. Regressor chain

In train curve, the points tend to overlay on the line and in both iron and silica the points are overfalling on regression line

In test curve, the models are able to predict iron concetrate and are not able to predict the silica better than iron concentrate.

18.2 CONCLUSION

INHERENT MTR IS BETTER THAN REGRESSION CHAIN MODEL IN CASE OF DECISION TREE

DECISION TREE> XGBOOST> RIDGE

19 RANDOMFOREST

19.1 DATA MODELING

[0]:
```
from sklearn.ensemble import RandomForestRegressor
```

[0]:
```
df=pd.read_csv('/content/drive/My Drive/preprocessed_time')
```

[0]:
```
y = df.iloc[:,23:25]
X = df.drop(['% Silica Concentrate','% Iron Concentrate','index','datetime␣
 ↪hours'], axis=1)
```

[0]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪2,random_state=30)
```

[0]:
```
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,test_size=0.
 ↪20,random_state=30)
```

[0]:
```
from sklearn.preprocessing import StandardScaler
scale_features_std = StandardScaler()
features_train = scale_features_std.fit_transform(X_train)
features_test = scale_features_std.transform(X_test)
```

[0]:
```
features_cv = scale_features_std.transform(X_cv)
```

[0]:
```
print(X_train.shape,features_train.shape,X_test.shape,features_test.
 ↪shape,features_cv.shape,y_train.shape,y_cv.shape,y_test.shape)
```

```
(471969, 21) (471969, 21) (147491, 21) (147491, 21) (117993, 21) (471969, 2)
(117993, 2) (147491, 2)
```

19.2 MODEL FOR MULTIOUTPUT

[0]:
```
from sklearn.ensemble import RandomForestRegressor
```

[0]:
```
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}
```

[0]:
```
# Create a based model
rf = RandomForestRegressor()
# Instantiate the grid search model
grid_search = RandomizedSearchCV(estimator = rf,param_distributions= param_grid,
                          cv = 3,n_jobs=-1, verbose = 2)
```

[0]:
```
grid_search.fit(features_train,y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
/usr/local/lib/python3.6/dist-
packages/joblib/externals/loky/process_executor.py:706: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
```

```
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 111.6min finished
```

[0]:
```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100,
                                                   n_jobs=None, oob_score=False,
                                                   random_state=None, verbose=0,
                                                   warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'bootstrap': [True],
                                        'max_depth': [80, 90, 100, 110],
                                        'max_features': [2, 3],
                                        'min_samples_leaf': [3, 4, 5],
                                        'min_samples_split': [8, 10, 12],
                                        'n_estimators': [100, 200, 300, 1000]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=2)
```

[0]:
```
grid_search.best_params_
```

[0]:
```
{'bootstrap': True,
 'max_depth': 100,
 'max_features': 3,
 'min_samples_leaf': 5,
 'min_samples_split': 10,
 'n_estimators': 1000}
```

[0]:
```
model=RandomForestRegressor(n_estimators=1000,max_depth=100,min_samples_leaf=5,min_samples_spli
```

[0]:
```
model.fit(features_train,y_train)
```

[0]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=100, max_features=3, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=5,
                      min_samples_split=10, min_weight_fraction_leaf=0.0,
                      n_estimators=1000, n_jobs=None, oob_score=False,
                                random_state=None, verbose=0, warm_start=False)
```

19.3 PREDICTION

[0]:
```
y_pred=model.predict(features_cv)
y_pred_train=model.predict(features_train)
y_pred_test=model.predict(features_test)
```

19.4 EVALUATION METRIC

[0]:
```
mse_train=mean_squared_error(y_train,y_pred_train)
mse_test=mean_squared_error(y_test,y_pred_test)
mse_cv=mean_squared_error(y_cv,y_pred)

r2_train=r2_score(y_train,y_pred_train)

r2_test=r2_score(y_test,y_pred_test)
r2_cv=r2_score(y_cv,y_pred)
```

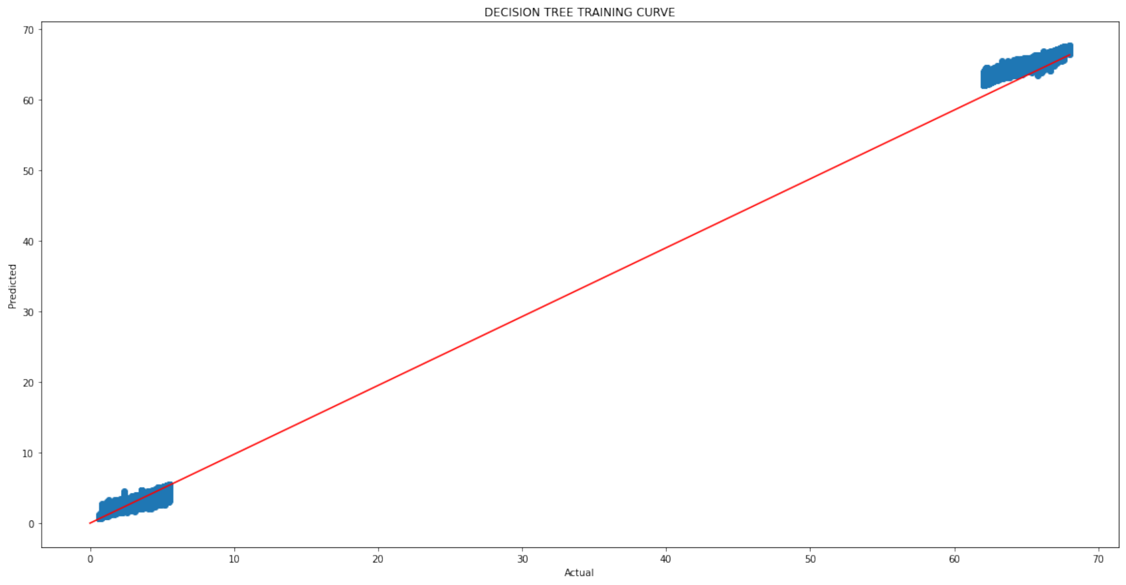
[0]:
```
r2_train,r2_cv,r2_test,mse_cv,mse_test,mse_train,mse_cv
```

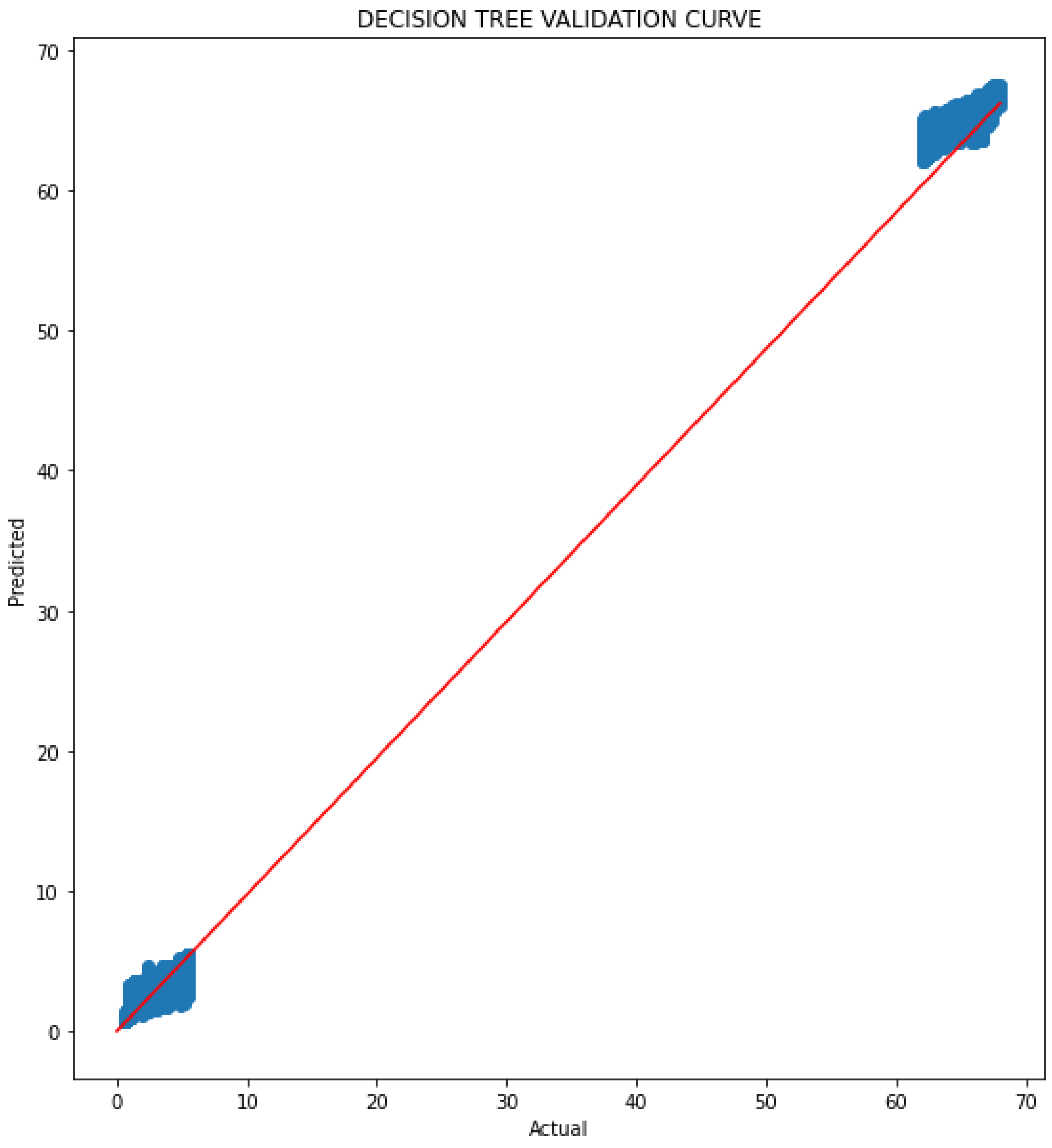
[0]:
```
(0.9404630125734039,
 0.8840832964855059,
 0.8836008488803795,
 0.14586763296600957,
 0.14634928647273932,
 0.0749937491528446,
 0.14586763296600957)
```

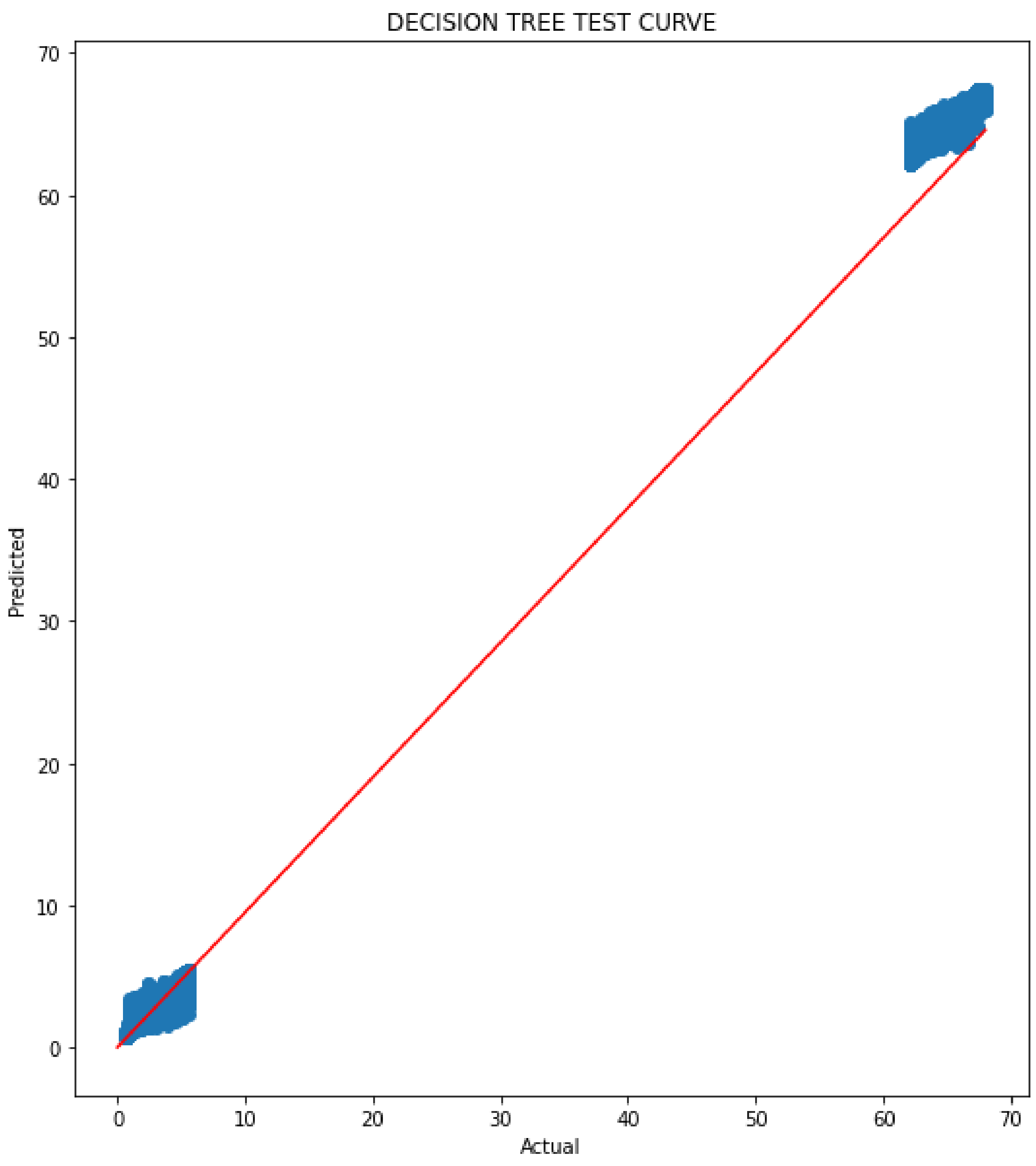
19.5 PLOTS

[0]:
```
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111)
ax.set(title="RANDOMFOREST TRAINING CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_train, y_pred_train)
ax.plot([0,max(y_train['% Iron Concentrate'])], [0,max(y_pred_train[0])],␣
 ↪color='r')
fig.show()
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(121)
ax.set(title="RANDOM FOREST VALIDATION CURVE", xlabel="Actual",␣
 ↪ylabel="Predicted")
ax.scatter(y_cv, y_pred_cv)
ax.plot([0,max(y_cv['% Iron Concentrate'])], [0,max(y_pred[0])], color='r')
fig.show()
fig = plt.figure(figsize=(30, 10))
ax = fig.add_subplot(131)
```

```
ax.set(title="RANDOM FOREST TEST CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_test, y_pred_test)
ax.plot([0,max(y_test['% Iron Concentrate'])], [0,max(y_pred_test[0])],␣
 ↪color='r')
ax.plot([0,max(y_test['% Iron Concentrate'])], [0,max(y_pred_test[0])],␣
 ↪color='r')
fig.show()
```




DECISION TREE VALIDATION CURVE
Predicted
Actual
0
10
20
30
40
50
60
70



19.6 REPORT

[4]:
```
#https://pypi.org/project/PrettyTable/-- for representation of data
#https://stackoverflow.com/questions/36423259/
 ↪how-to-use-pretty-table-in-python-to-print-out-data-from-multiple-lists-- to␣
 ↪add the rows in table
from prettytable import PrettyTable
x = PrettyTable(border=True, header=True, padding_width=15)
```

```
x.field_names = ["MODEL1",␣
 ↪"R2_TRAIN","R2_CV","R2_TEST","MSE_TRAIN","MSE_CV","MSE_TEST"]

x.add_row(["RANDOM FOREST",0.9404630125734039,
 0.8840832964855059,
 0.8836008488803795, 0.0749937491528446,
 0.14586763296600957,
 0.14634928647273932
])

print(x)
```

```
+-------------------------------------------+-----------------------------------
-------------+-----------------------------------------------+------------------
-------------------------------+-----------------------------------------------
+------------------------------------------------+-----------------------------
--------------------+
|                   MODEL1                  |                    R2_TRAIN
|                    R2_CV                      |                   R2_TEST
|                   MSE_TRAIN                   |                      MSE_CV
|                     MSE_TEST                   |
+-------------------------------------------+-----------------------------------
-------------+-----------------------------------------------+------------------
-------------------------------+-----------------------------------------------
+------------------------------------------------+-----------------------------
--------------------+
|               RANDOM FOREST               |               0.9404630125734039
|               0.8840832964855059              |
0.8836008488803795               |               0.0749937491528446
|               0.14586763296600957              |
0.14634928647273932               |
+-------------------------------------------+-----------------------------------
-------------+-----------------------------------------------+------------------
-------------------------------+-----------------------------------------------
+------------------------------------------------+-----------------------------
--------------------+
```

19.7 METRIC ANALYSIS

20 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html

21 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_square

R2 SCORE

Acoording to literature, the r2 score is good when it is closer to 1 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a Rˆ2 score of 0.0.

TRAIN R2 is so closer to 1 and TEST r2 is also to 1 and hence inorder to get better result , we must try other models

MSE VALUE

A non-negative floating point value (the best value is 0.0), or an array of floating point values, one for each individual target.

21.1 PLOTS ANALYSIS

1. INHERENT MTR MODEL

In train curve, the points tend to overlay on the line and in both iron and silica the points are overfalling on regression line

In test and validation curve, the models are able to predict iron concetrate and are able to predict the silica better in the same way for iron concentrate

21.2 CONCLUSION

INHERENT RANDOMFOREST MTR IS BETTER THAN OTHER MODELS

RANDOMFOREST> DECISION TREE> XGBOOST> RIDGE

22 ADABOOST

22.1 DATA MODELING

[0]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import RandomizedSearchCV
```

[0]:
```
import pandas as pd
```

[0]:
```
df=pd.read_csv('/content/drive/My Drive/preprocessed_time')
```

[0]:
```
y = df.iloc[:,23:25]
X = df.drop(['% Silica Concentrate','% Iron Concentrate','index','datetime␣
 ↪hours'], axis=1)
```

[0]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪2,random_state=30)
```

[0]:
```
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,test_size=0.
 ↪20,random_state=30)
```

[0]:
```
from sklearn.preprocessing import StandardScaler
scale_features_std = StandardScaler()
features_train = scale_features_std.fit_transform(X_train)
features_test = scale_features_std.transform(X_test)
```

[0]:
```
features_cv = scale_features_std.transform(X_cv)
```

[0]:
```
print(X_train.shape,features_train.shape,X_test.shape,features_test.
 ↪shape,features_cv.shape,y_train.shape,y_cv.shape,y_test.shape)
```

```
(471969, 21) (471969, 21) (147491, 21) (147491, 21) (117993, 21) (471969, 2)
(117993, 2) (147491, 2)
```

[0]:
```
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
```

[0]:
```
param_dist = {
 'n_estimators': [50, 100],
 'learning_rate' : [0.01,0.05,0.1,0.3,1],
 }

pre_gs_inst = RandomizedSearchCV(AdaBoostRegressor(),
 param_distributions = param_dist,
 cv=3,
 n_iter = 10,
 n_jobs=-1)
```

[0]:
```
pre_gs_inst.fit(features_train, y_train)
```

```
/usr/local/lib/python3.6/dist-
packages/joblib/externals/loky/process_executor.py:691: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

[0]:
```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=AdaBoostRegressor(base_estimator=None,
                                               learning_rate=1.0, loss='linear',
                                               n_estimators=50,
                                               random_state=None),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'learning_rate': [0.01, 0.05, 0.1, 0.3,
                                                          1],
                                        'n_estimators': [50, 100]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

22.2 MODEL USING MULTIOUTPUTREGRESSOR

[0]:
```
regr_1 = DecisionTreeRegressor(max_depth=50,min_samples_split=5)

regr_2 = AdaBoostRegressor(regr_1,n_estimators=50,learning_rate=1)
```

[0]:
```
from sklearn.multioutput import MultiOutputRegressor
```

[0]:
```
model=MultiOutputRegressor(regr_2)
model.fit(features_train,y_train)
```

[0]:
```
MultiOutputRegressor(estimator=AdaBoostRegressor(base_estimator=DecisionTreeRegr
essor(ccp_alpha=0.0,
      criterion='mse',
      max_depth=50,
      max_features=None,
      max_leaf_nodes=None,
      min_impurity_decrease=0.0,
      min_impurity_split=None,
      min_samples_leaf=1,
      min_samples_split=5,
      min_weight_fraction_leaf=0.0,
      presort='deprecated',
      random_state=None,
      splitter='best'),
                                                 learning_rate=1, loss='linear',
                                                 n_estimators=100,
                                                 random_state=None),
                     n_jobs=None)
```

22.3 PREDICTION

[0]:
```
y_pred=model.predict(features_cv)
y_pred_train=model.predict(features_train)
y_pred_test=model.predict(features_test)
```

22.4 EVALUATION METRIC

[0]:
```
mse_train=mean_squared_error(y_train,y_pred_train)
mse_cv=mean_squared_error(y_cv,y_pred)
mse_test=mean_squared_error(y_test,y_pred_test)
```

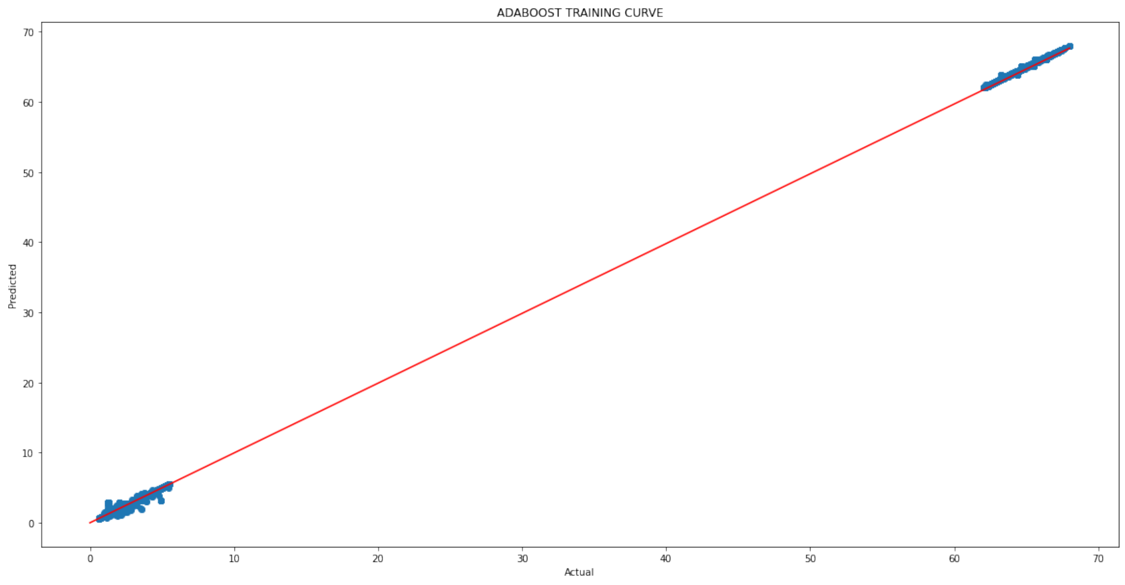
[0]:
```
r2_train=r2_score(y_train,y_pred_train)
r2_cv=r2_score(y_cv,y_pred)
r2_test=r2_score(y_test,y_pred_test)
```

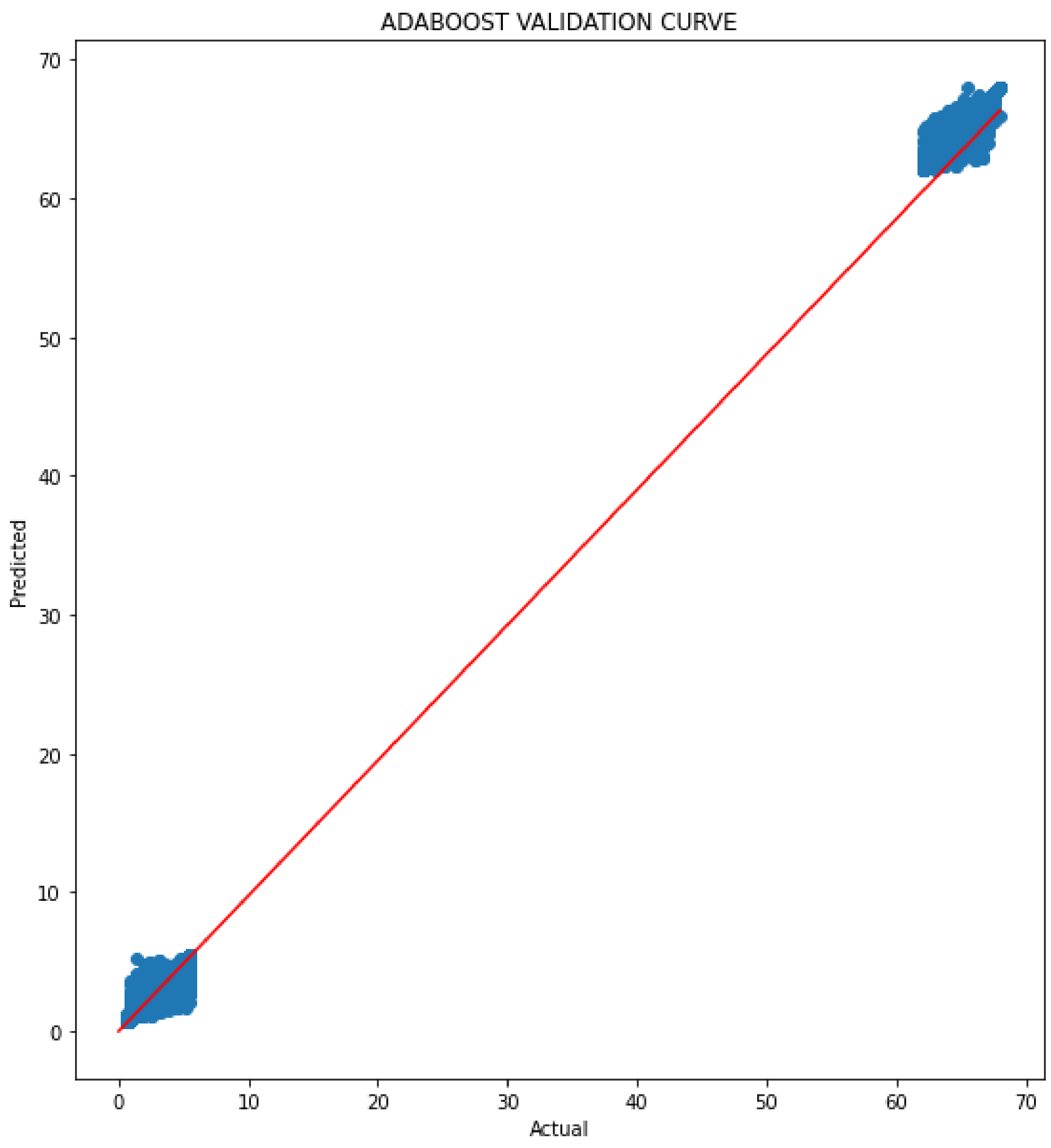
[0]:
```
r2_train,r2_cv,r2_test,mse_cv,mse_test,mse_train,mse_cv
```

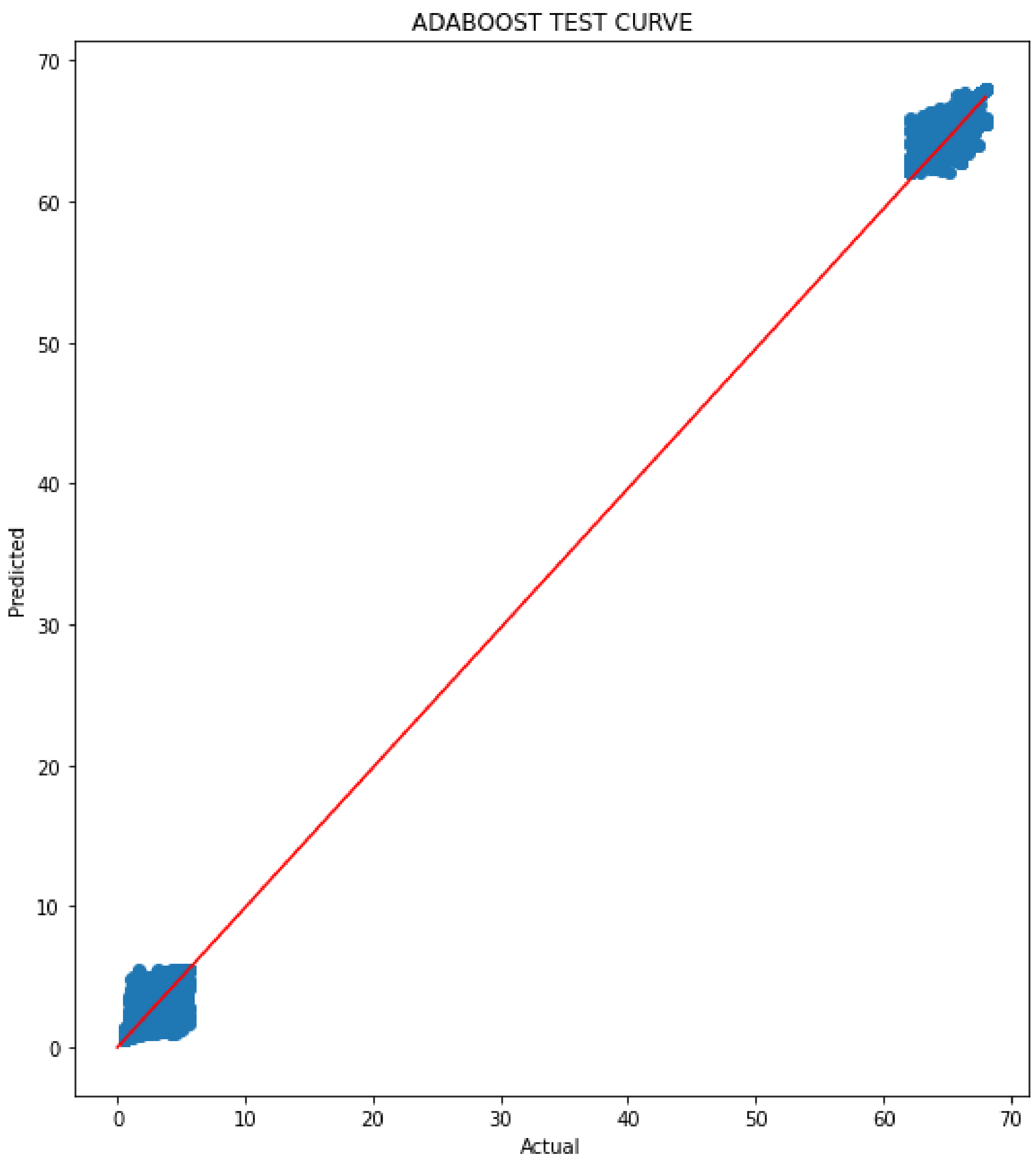
[0]:
```
(0.9989177977576809,
 0.9750738456318357,
 0.9745449086918032,
 0.03139462458532873,
 0.03203487819099303,
 0.0013715157924193923,
 0.03139462458532873)
```

22.5 PLOTS

[0]:
```
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111)
ax.set(title="ADABOOST TRAINING CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_train, y_pred_train)
ax.plot([0,max(y_train['% Iron Concentrate'])], [0,max(y_pred_train[0])],␣
 ↪color='r')
fig.show()
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(121)
ax.set(title="ADABOOST VALIDATION CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_cv, y_pred)
ax.plot([0,max(y_cv['% Iron Concentrate'])], [0,max(y_pred[0])], color='r')
fig.show()
fig = plt.figure(figsize=(30, 10))
ax = fig.add_subplot(131)
ax.set(title="ADABOOST TEST CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_test, y_pred_test)
ax.plot([0,max(y_test['% Iron Concentrate'])], [0,max(y_pred_test[0])],␣
 ↪color='r')
fig.show()
```




ADABOOST VALIDATION CURVE
Predicted
Actual
0
10
20
30
40
50
60
70



22.6 MODEL USING REGRESSOR CHAIN

[0]:
```
from sklearn.multioutput import RegressorChain
```

[0]:
```
model= AdaBoostRegressor(regr_1,n_estimators=50,learning_rate=1)
wrapper1 = RegressorChain(model)
wrapper1.fit(features_train,y_train)
```

[0]:
```
RegressorChain(base_estimator=AdaBoostRegressor(base_estimator=DecisionTreeRegre
ssor(ccp_alpha=0.0,
        criterion='mse',
        max_depth=50,
        max_features=None,
        max_leaf_nodes=None,
        min_impurity_decrease=0.0,
        min_impurity_split=None,
        min_samples_leaf=1,
        min_samples_split=5,
        min_weight_fraction_leaf=0.0,
        presort='deprecated',
        random_state=None,
        splitter='best'),
                                                    learning_rate=1, loss='linear',
                                                    n_estimators=50,
                                                    random_state=None),
                  cv=None, order=None, random_state=None)
```

22.7 PREDICTION

[0]:
```
y_pred_cv=wrapper1.predict(features_cv)
y_pred_train=wrapper1.predict(features_train)
y_pred_test=wrapper1.predict(features_test)
```

22.8 EVALUATION METRIC

[0]:
```
r2_cv=r2_score(y_pred_cv,y_cv)
r2_train=r2_score(y_pred_train,y_train)
r2_test=r2_score(y_pred_test,y_test)
```

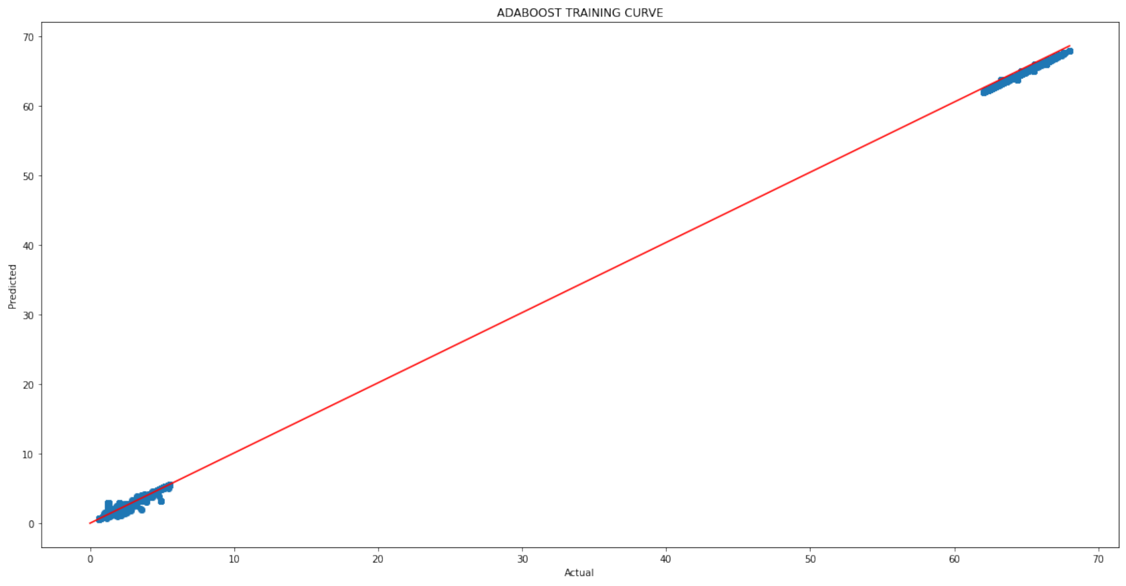
[0]:
```
mse_train=mean_squared_error(y_train,y_pred_train)
mse_cv=mean_squared_error(y_cv,y_pred)
mse_test=mean_squared_error(y_test,y_pred_test)
```

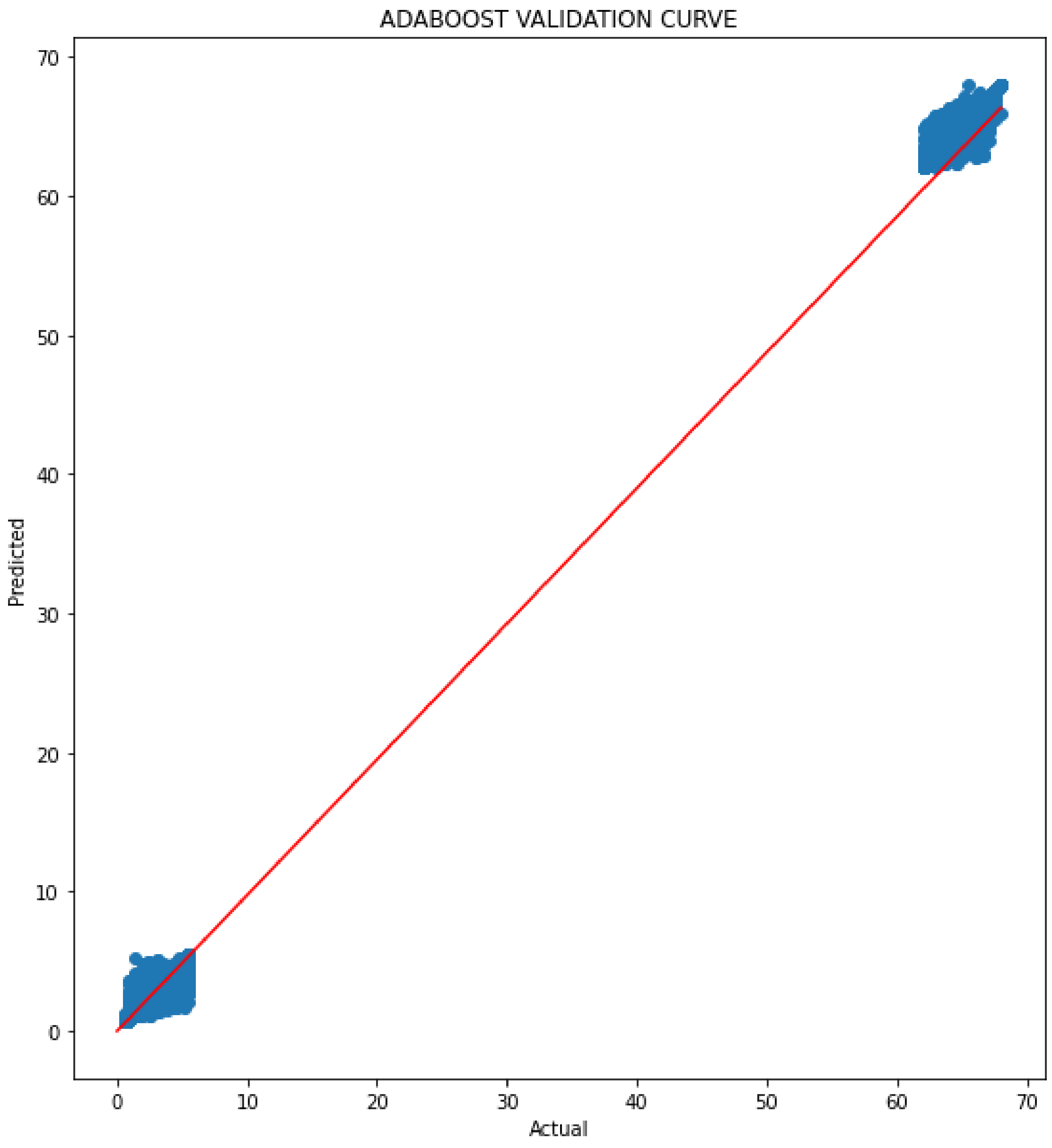
[0]:
```
r2_train,r2_cv,r2_test,mse_train,mse_cv,mse_test
```

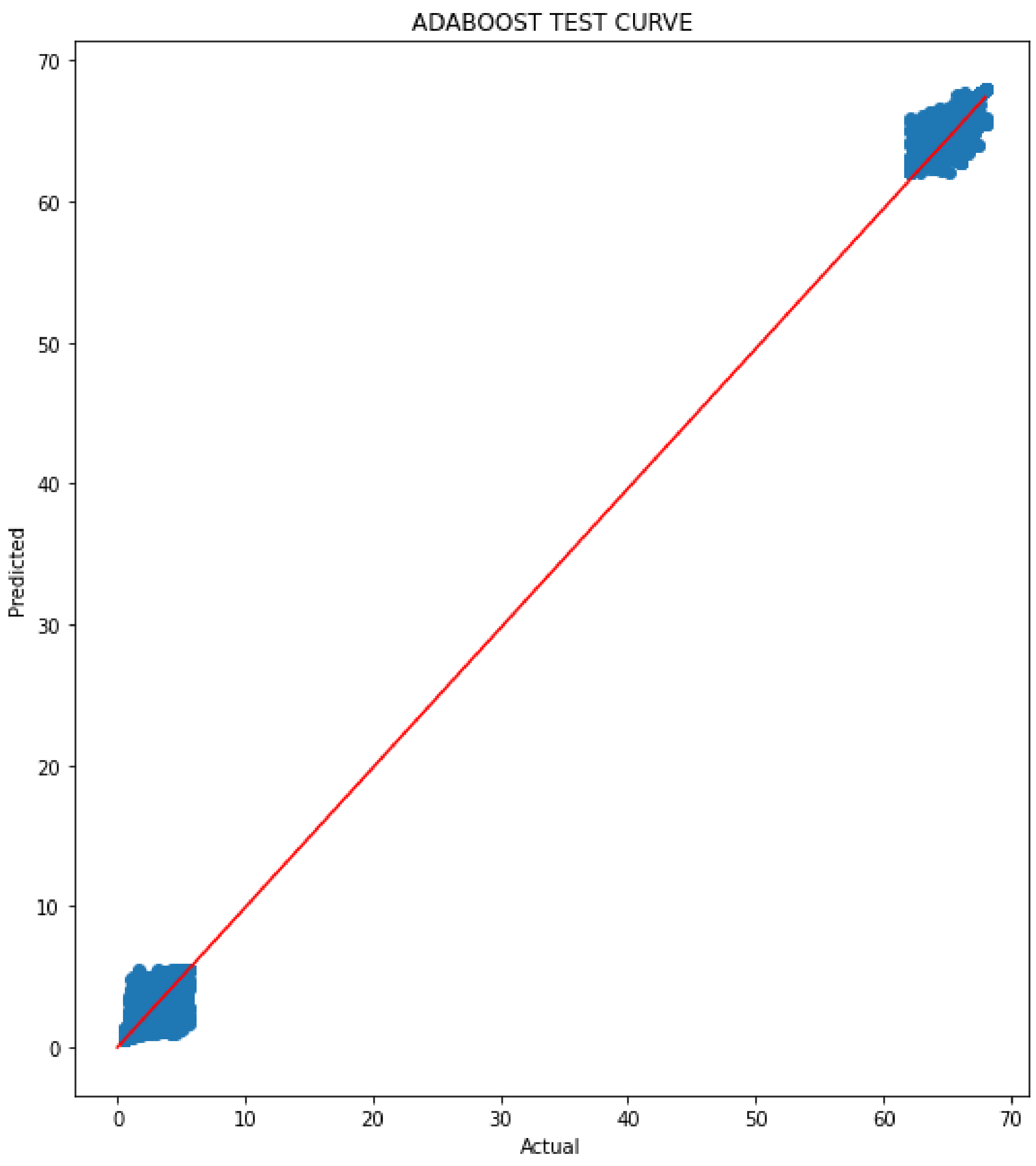
[0]:
```
(0.9991389949078793,
 0.9745304869923623,
 0.9733092536247556,
 0.0010902161013306814,
 0.03139462458532873,
 0.03293804176581637)
```

22.9 PLOTS

[0]:
```
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111)
ax.set(title="ADABOOST TRAINING CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_train, y_pred_train)
ax.plot([0,max(y_train['% Iron Concentrate'])], [0,max(y_pred_train[0])],␣
 ↪color='r')
fig.show()
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(121)
ax.set(title="ADABOOST VALIDATION CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_cv, y_pred)
ax.plot([0,max(y_cv['% Iron Concentrate'])], [0,max(y_pred[0])], color='r')
fig.show()
fig = plt.figure(figsize=(30, 10))
ax = fig.add_subplot(131)
ax.set(title="ADABOOST TEST CURVE", xlabel="Actual", ylabel="Predicted")
ax.scatter(y_test, y_pred_test)
ax.plot([0,max(y_test['% Iron Concentrate'])], [0,max(y_pred_test[0])],␣
 ↪color='r')
fig.show()
```




ADABOOST VALIDATION CURVE
Predicted
Actual
0
10
20
30
40
50
60
70



22.10 REPORT

[5]:
```
#https://pypi.org/project/PrettyTable/-- for representation of data
#https://stackoverflow.com/questions/36423259/
 ↪how-to-use-pretty-table-in-python-to-print-out-data-from-multiple-lists-- to␣
 ↪add the rows in table
from prettytable import PrettyTable
x = PrettyTable(border=True, header=True, padding_width=15)
```

```
x.field_names = ["MODEL1",␣
↪"R2_TRAIN","R2_CV","R2_TEST","MSE_TRAIN","MSE_CV","MSE_TEST"]
x.add_row(["ADABOOST MTR",0.9989177977576809,
 0.9750738456318357,
 0.9745449086918032, 0.0013715157924193923,
 0.03139462458532873,
 0.03203487819099303


])
x.add_row(["ADABOOST RIGRESSION CHAIN",0.9991389949078793,
 0.9745304869923623,
 0.9733092536247556,
 0.0010902161013306814,
 0.03139462458532873,
 0.03293804176581637])

print(x)
```

```
+-------------------------------------------------------+----------------------
-------------------------+-------------------------------------------------+-----
--------------------------------------------+-----------------------------------
---------------+-------------------------------------------------+--------------
-----------------------------------+
|                          MODEL1                       |
R2_TRAIN                    |                     R2_CV                       |
R2_TEST                     |                     MSE_TRAIN
|                       MSE_CV                      |                     MSE_TEST
|
+-------------------------------------------------------+----------------------
-------------------------+-------------------------------------------------+-----
--------------------------------------------+-----------------------------------
---------------+-------------------------------------------------+--------------
-----------------------------------+
|                       ADABOOST MTR                    |
0.9989177977576809               |               0.9750738456318357
|               0.9745449086918032               |
0.0013715157924193923               |               0.03139462458532873
|               0.03203487819099303               |
|               ADABOOST RIGRESSION CHAIN               |
0.9991389949078793               |               0.9745304869923623
|               0.9733092536247556               |
0.0010902161013306814               |               0.03139462458532873
|               0.03293804176581637               |
+-------------------------------------------------------+----------------------
-------------------------+-------------------------------------------------+-----
--------------------------------------------+-----------------------------------
```

```
---------------+-------------------------------------------------+-------------
-----------------------------------+
```

22.11 METRIC ANALYSIS

23 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html

24 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_square

R2 SCORE

Acoording to literature, the r2 score is good when it is closer to 1 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

TRAIN R2 is so closer to 0 and TEST r2 is also to 0 and hence inorder to get better result , we must try other models

MSE VALUE

A non-negative floating point value (the best value is 0.0), or an array of floating point values, one for each individual target.

The MSE value and R2 score value is better than ridge and MSE value of MTR is same as REGRESSION CHAIN model.

24.1 PLOTS ANALYSIS

1. MTR MODEL

In train curve, the points tend to overlay on the line and in both iron and silica the points are overfalling on regression line

In test and validation curve, the models are able to predict iron concetrate and are able to predict the silica better in the same way for iron concentrate

2. Regressor chain

In train curve, the points tend to overlay on the line and in both iron and silica the points are overfalling on regression line

In test curve, the models are able to predict iron concetrate and are able to predict the silica the same way of iron concentrate.

24.2 CONCLUSION

AdaBoost model can be used for quality prediction. It shows the scatter plot of the model that predicts two target variables: silica and iron concentrates by AdaBoost regressor.

ADABOOST>RANDOMFOREST>DECISION TREE>XGBOOST>RIDGE