In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score,r2_score

import ipywidgets as widgets

%matplotlib inline
```

data preparation

In [2]:
```
# reading data file in to dataframe
data = pd.read_excel('Data.xlsx')
data.shape
```

Out[2]: (2360, 11)

In [3]:
```
# changing categorical features to dummy variables
categoricalColumns = ['Planform shape', 'Behavior with time', 'Crest orientatio on transverse',

dataDummies = pd.get_dummies(data,dtype = int,columns= categoricalColumns)
dataDummies.to_excel('Data2.xlsx', index=False)
dataDummies.shape
```

Out[3]: (2360, 17)

In [4]:
```
# seperating 51 training bedforms and storing in file
bedforms51 = dataDummies[ (dataDummies["BedformNo"] != 3) & (dataDummies["BedformNo"] != 5) & (da
bedforms51.to_excel('bedforms51.xlsx', index=False)
bedforms51.shape
```

Out[4]: (2040, 17)

In [5]:
```
# seperating 8 test bedforms and storing in file
bedforms8 = dataDummies[ (dataDummies["BedformNo"] == 3) | (dataDummies["BedformNo"] == 5) | (dat
bedforms8.to_excel('bedforms8.xlsx', index=False)
bedforms8.shape
```

Out[5]: (320, 17)

In [6]:
```
# reading in data from files and dropping excess columns
bed51 = pd.read_excel("bedforms51.xlsx")
bed8 = pd.read_excel("bedforms8.xlsx")

bed51 = bed51.drop(labels=['BedformNo','Global Entropy'],axis=1)
bed8 = bed8.drop(labels=['BedformNo','Global Entropy'],axis=1)

print(bed51.shape, bed8.shape)
```

(2040, 15) (320, 15)

In [7]:

```
# seperating bedform data into x (input) and y (output)
# only continuous features are used
dataset = bed51.values
x = dataset[:,1:4] # 'Delta'-'Entropic Scale'
y = dataset[:,0] # 'Sat'

dataset2 = bed8.values
x2 = dataset2[:,1:4] # 'Delta'-'Entropic Scale'
y2 = dataset2[:,0] # 'Sat'
```

scaling functions

In [8]:

```
# function for scaling data with StandardScaler()
def standardScaler(x,y,x2,y2,categorical):
    y=np.reshape(y, (-1,1))
    y2=np.reshape(y2, (-1,1))

    scaler_x = StandardScaler()
    scaler_y = StandardScaler()

    scaler_x.fit(x)
    xscale =scaler_x.transform(x)
    scaler_y.fit(y)
    yscale =scaler_y.transform(y)

    scaler_x.fit(x2)
    x2scale =scaler_x.transform(x2)
    scaler_y.fit(y2)
    y2scale =scaler_y.transform(y2)

    # adding categorical variables to scaled data
    if (categorical == True):
        xTemp = dataset[:,4:15] # 'Planform shape_2D' - 'Longitudinal_Yes'
        xscale = np.concatenate((xscale, xTemp), axis=1)

        xTemp = dataset2[:,4:15] # 'Planform shape_2D' - 'Longitudinal_Yes'
        x2scale = np.concatenate((x2scale, xTemp), axis=1)

    return (xscale,yscale, x2scale,y2scale)
```

In [9]:

```
# function for scaling data with MinMaxScaler()
def minMaxScaler(x,y,x2,y2, categorical):
    y=np.reshape(y, (-1,1))
    y2=np.reshape(y2, (-1,1))

    scaler_x = MinMaxScaler()
    scaler_y = MinMaxScaler()

    scaler_x.fit(x)
    xscale=scaler_x.transform(x)
    scaler_y.fit(y)
    yscale=scaler_y.transform(y)

    scaler_x.fit(x2)
    x2scale=scaler_x.transform(x2)
    scaler_y.fit(y2)
    y2scale=scaler_y.transform(y2)

    # adding categorical variables to scaled data
    if (categorical == True):
        xTemp = dataset[:,4:15] # 'Planform shape_2D' - 'Longitudinal_Yes'
        xscale = np.concatenate((xscale, xTemp), axis=1)

        xTemp = dataset2[:,4:15] # 'Planform shape_2D' - 'Longitudinal_Yes'
        x2scale = np.concatenate((x2scale, xTemp), axis=1)

    return (xscale,yscale, x2scale,y2scale)
```

bedform plotting function and widgets

In [10]:

```
#function to plot accuracy of model with labeled bedforms
def scatterBedforms(ytest, ymodel, lineMin, lineMax):
    plt.figure(figsize=(15,10))

    count=1
    for i in range(len(ytest)):
        if count <= 40:
            if count == 40: plt.scatter(ytest[i],ymodel[i],color='red',label='3')
            else: plt.scatter(ytest[i],ymodel[i],color='red')

        elif count <=80:
            if count == 80: plt.scatter(ytest[i],ymodel[i],color='purple',label='5')
            else:  plt.scatter(ytest[i],ymodel[i],color='purple')

        elif count <=120:
            if count == 120: plt.scatter(ytest[i],ymodel[i],color='hotpink',label='13')
            else: plt.scatter(ytest[i],ymodel[i],color='hotpink')

        elif count <=160:
            if count == 160: plt.scatter(ytest[i],ymodel[i],color='green',label='19')
            else: plt.scatter(ytest[i],ymodel[i],color='green')

        elif count <=200:
            if count == 200: plt.scatter(ytest[i],ymodel[i],color='coral',label='27')
            else: plt.scatter(ytest[i],ymodel[i],color='coral')

        elif count <=240:
            if count == 240: plt.scatter(ytest[i],ymodel[i],color='DeepSkyBlue',label='36')
            else: plt.scatter(ytest[i],ymodel[i],color='DeepSkyBlue')

        elif count <=280:
            if count == 280: plt.scatter(ytest[i],ymodel[i],color='lawngreen',label='42a')
            else: plt.scatter(ytest[i],ymodel[i],color='lawngreen')

        elif count <=320:
            if count == 320: plt.scatter(ytest[i],ymodel[i],color='slategray',label='63')
            else: plt.scatter(ytest[i],ymodel[i],color='slategray')

        else:
            plt.scatter(ytest[i],ymodel[i],color='blue',legend='error')
        count+=1

    #plot properties
    plt.legend(loc='best', title='Bedform No.', title_fontsize = 15, fontsize = 12);
    plt.xlabel('Actual', fontsize= 25)
    plt.ylabel('Predicted', fontsize= 25);
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)

    #plotting diagonal line
    xrange, yrange = np.linspace(lineMin,lineMax,3),np.linspace(lineMin,lineMax,3)
    plt.plot(xrange,yrange,color='black');
```

In [11]:

```
# scaling options widget
scaleWidg = widgets.RadioButtons(
    options=['No Scaling', 'Standard Scaler', 'Min Max Scaler'],
    value='Standard Scaler',
    description='Scaling:',
    disabled= False
)
scaleWidg.value
```

Out[11]: 'Standard Scaler'

machine learning models with interactive widgets

Linear Regression

In [12]:

```
def linearReg(categorical, scaling):
    global x,y,x2,y2
    xa = x
    ya = y
    x2a = x2
    y2a = y2

    # scaling and assigning training/testing data
    if (scaling == 'Standard Scaler'):
        xscale,yscale, x2scale,y2scale = standardScaler(xa,ya,x2a,y2a, categorical)
        Xtrain, ytrain = xscale,yscale
        Xtest, ytest = x2scale,y2scale
        lineMin, lineMax = -1,1.5

    elif (scaling == 'Min Max Scaler'):
        xscale,yscale, x2scale,y2scale = minMaxScaler(xa,ya,x2a,y2a, categorical)
        Xtrain, ytrain = xscale,yscale
        Xtest, ytest = x2scale,y2scale
        lineMin, lineMax = 0,1

    elif (scaling == 'No Scaling'):
        if (categorical == True):
            # adding categorical variables to data
            xTemp = dataset[:,4:15] # 'Planform shape_2D' - 'Longitudinal_Yes'
            xa = np.concatenate((xa, xTemp), axis=1)
            xTemp = dataset2[:,4:15] # 'Planform shape_2D' - 'Longitudinal_Yes'
            x2a = np.concatenate((x2a, xTemp), axis=1)

        lineMin, lineMax = 0,70
        Xtrain, ytrain = xa,ya
        Xtest, ytest = x2a,y2a

    # model training
    modelLR = LinearRegression()
    modelLR.fit(Xtrain, ytrain)
    yLinearModel = modelLR.predict(Xtest)

    print("R2 Score: ", r2_score(ytest,yLinearModel))

    # cross validation
    scores = cross_val_score(modelLR, Xtrain, ytrain, cv=5)
    print("\nCV Scores (training):")
    print("Mean: %0.5f \nStandard Deviation: %0.5f"% (scores.mean(),scores.std()))


    scatterBedforms(ytest, yLinearModel, lineMin, lineMax)
```

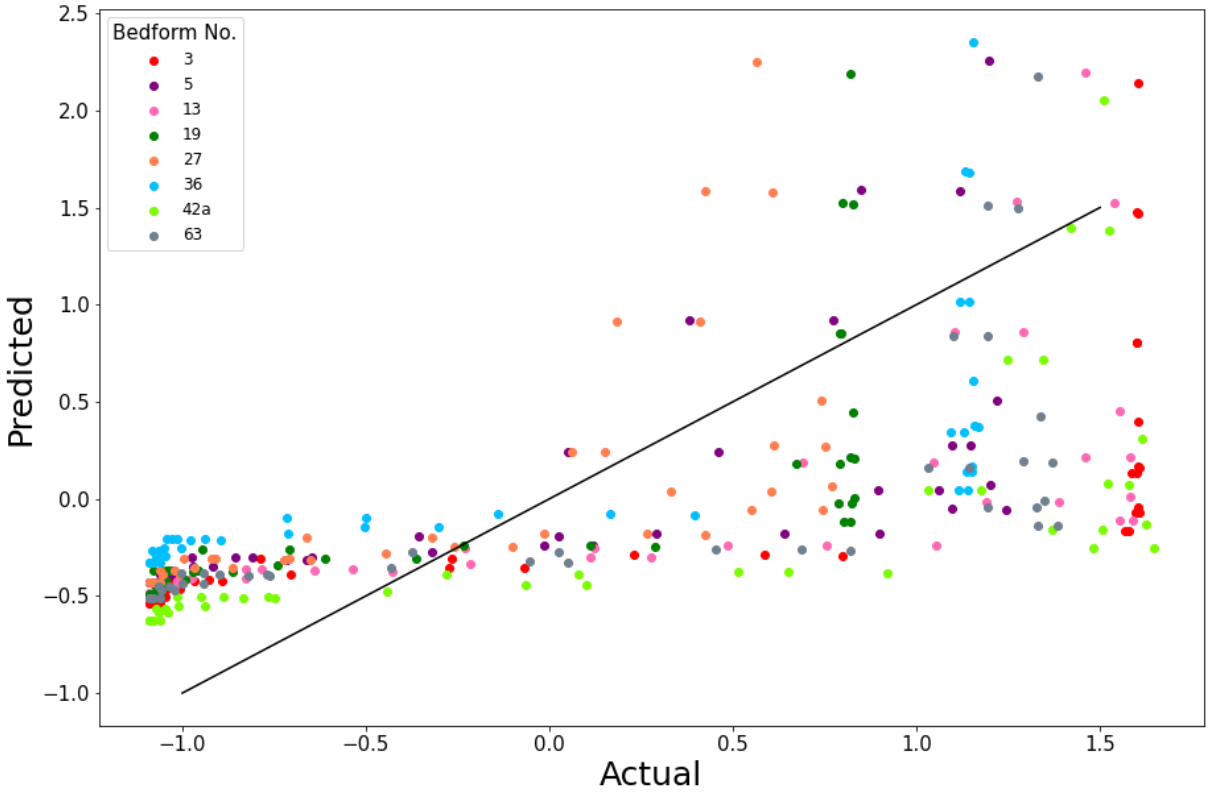
In [13]:
```
1 widgets.interact(linearReg, scaling=scaleWidg, categorical=True);
```

☑ categorical

Scaling: ○ No Scaling
◉ Standard Scaler
○ Min Max Scaler

```
R2 Score:  0.4164173504839257

CV Scores (training):
Mean: 0.29285
Standard Deviation: 0.06361
```



K-Nearest Neighbors

In [14]:

```
def KNN(categorical, scaling, neighbors=1):
    global x,y,x2,y2
    xa = x
    ya = y
    x2a = x2
    y2a = y2

    # scaling and assigning training/testing data
    if (scaling == 'Standard Scaler'):
        xscale,yscale, x2scale,y2scale = standardScaler(xa,ya,x2a,y2a, categorical)
        Xtrain, ytrain = xscale,yscale
        Xtest, ytest = x2scale,y2scale
        lineMin, lineMax = -1,1.5

    elif (scaling == 'Min Max Scaler'):
        xscale,yscale, x2scale,y2scale = minMaxScaler(xa,ya,x2a,y2a, categorical)
        Xtrain, ytrain = xscale,yscale
        Xtest, ytest = x2scale,y2scale
        lineMin, lineMax = 0,1

    elif (scaling == 'No Scaling'):
        if (categorical == True):
            # adding categorical variables to data
            xTemp = dataset[:,4:15] # 'Planform shape_2D' - 'Longitudinal_Yes'
            xa = np.concatenate((xa, xTemp), axis=1)
            xTemp = dataset2[:,4:15] # 'Planform shape_2D' - 'Longitudinal_Yes'
            x2a = np.concatenate((x2a, xTemp), axis=1)

        lineMin, lineMax = 0,70
        Xtrain, ytrain = xa,ya
        Xtest, ytest = x2a,y2a

    # model training
    knnModel = KNeighborsRegressor(n_neighbors = neighbors)
    knnModel.fit(Xtrain,ytrain)
    yModel = knnModel.predict(Xtest)

    print("R2 Score: ", r2_score(ytest,yModel))

    # cross validation
    scores = cross_val_score(knnModel, Xtrain, ytrain, cv=5)
    print("\nCV Scores (training):")
    print("Mean: %0.5f \nStandard Deviation: %0.5f"% (scores.mean(),scores.std()))

    scatterBedforms(ytest, yModel, lineMin, lineMax)
```

In [15]:

```
1 widgets.interact(KNN, neighbors=(1,30,1), scaling=scaleWidg, categorical=True);
```

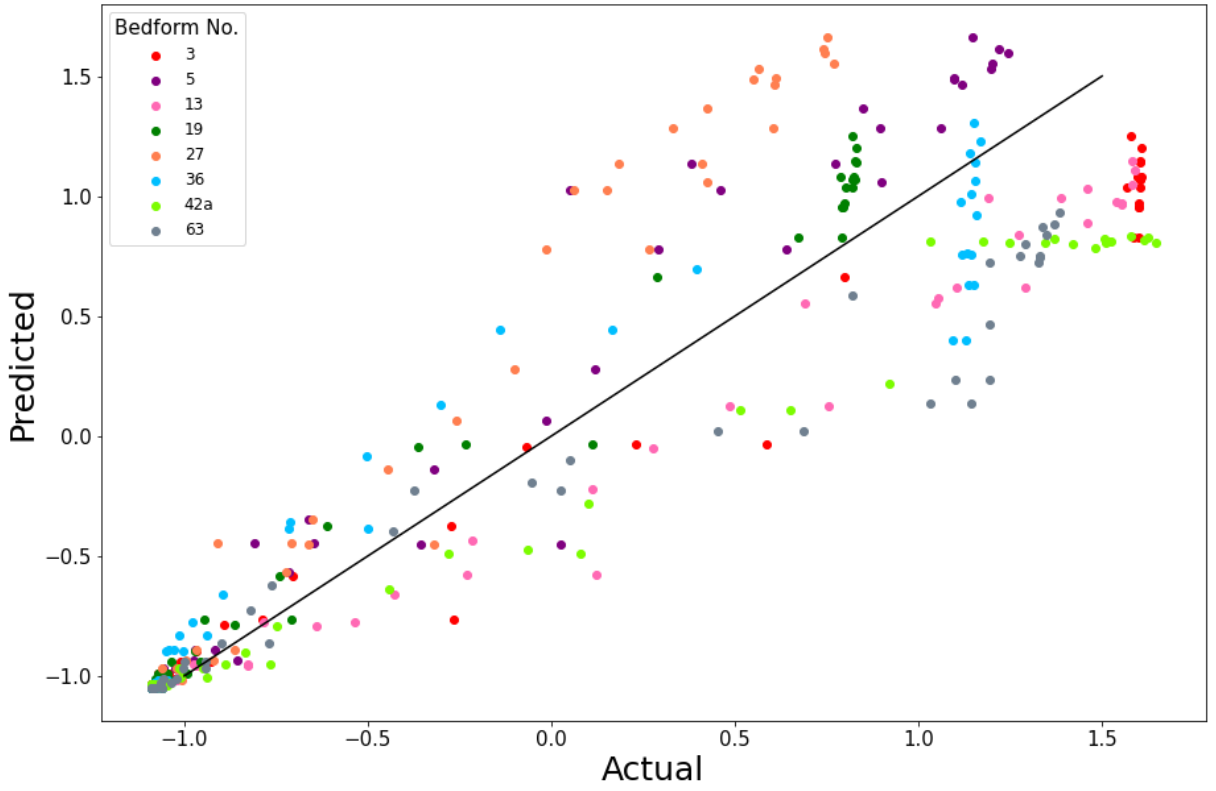
categorical

Scaling:
- No Scaling
- Standard Scaler
- Min Max Scaler

neighbors 1

```
R2 Score:  0.8441340147180192

CV Scores (training):
Mean: 0.57490
Standard Deviation: 0.15003
```



Ramdom Forest

In [16]:

```
def RandomForest( categorical, estimators=150, depth=12):
    global x,y,x2,y2
    xa = x
    ya = y
    x2a = x2
    y2a = y2

    # adding categorical features
    if (categorical == True):
        # adding categorical variables to data
        xTemp = dataset[:,4:15] # 'Planform shape_2D' - 'Longitudinal_Yes'
        xa = np.concatenate((xa, xTemp), axis=1)
        xTemp = dataset2[:,4:15] # 'Planform shape_2D' - 'Longitudinal_Yes'
        x2a = np.concatenate((x2a, xTemp), axis=1)

    # assigning training/testing data
    lineMin, lineMax = 0,70
    Xtrain, ytrain = xa,ya
    Xtest, ytest = x2a,y2a

    # model training
    forestModel = RandomForestRegressor(n_estimators= estimators, max_depth = depth,  max_feature
    forestModel.fit(Xtrain, ytrain)
    yforestModel = forestModel.predict(Xtest)

    print("R2 Score: ", r2_score(ytest,yforestModel))

    # cross validation
    scores = cross_val_score(forestModel, Xtrain, ytrain, cv=5)
    print("\nCV Scores (training):")
    print("Mean: %0.5f \nStandard Deviation: %0.5f"% (scores.mean(),scores.std()))

    scatterBedforms(ytest, yforestModel, lineMin, lineMax)
```

In [17]:

```
1 widgets.interact(RandomForest, categorical=True, estimators=(1,150,1), depth=(1,50,1));
```

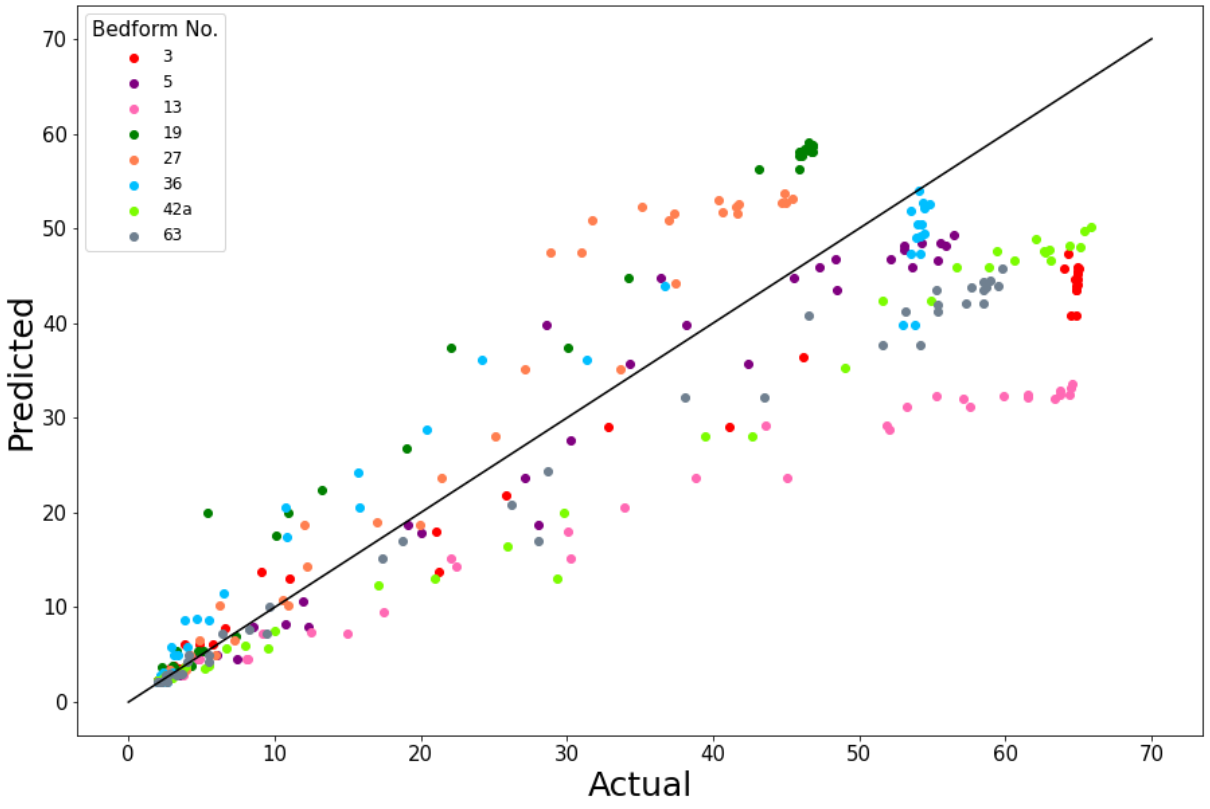
categorical

estimators 150

depth 12

```
R2 Score:  0.7963485107078327

CV Scores (training):
Mean: 0.64634
Standard Deviation: 0.16185
```



Gradient Boosted Regression Tree

In [18]:

```
def gradientBoosted(categorical, learningRate=.18, estimators=150, depth=3):
    global x,y,x2,y2
    xa = x
    ya = y
    x2a = x2
    y2a = y2

    # adding categorical variables to data
    if (categorical == True):
        xTemp = dataset[:,4:15] # 'Planform shape_2D' - 'Longitudinal_Yes'
        xa = np.concatenate((xa, xTemp), axis=1)

        xTemp = dataset2[:,4:15] # 'Planform shape_2D' - 'Longitudinal_Yes'
        x2a = np.concatenate((x2a, xTemp), axis=1)

    lineMin, lineMax = 0,70
    Xtrain, ytrain = xa,ya
    Xtest, ytest = x2a,y2a
    cols = xa.shape[1]

    # model training
    modelBoost = GradientBoostingRegressor(learning_rate = .1, n_estimators= estimators, max_dept
    modelBoost.fit(Xtrain, ytrain)
    ymodelBoost = modelBoost.predict(Xtest)

    print("R2 Score: ", r2_score(ytest,ymodelBoost))

    # cross validation
    scores = cross_val_score(modelBoost, Xtrain, ytrain, cv=5)
    print("\nCV Scores (training):")
    print("Mean: %0.5f \nStandard Deviation: %0.5f"% (scores.mean(),scores.std()))

    scatterBedforms(ytest, ymodelBoost, lineMin, lineMax)
    plt.show()

    #plotting feature importance
    plt.barh(np.arange(cols), modelBoost.feature_importances_, align='center')
    plt.yticks(np.arange(cols), bed8.columns[1:(cols+1)], fontsize = 10)
    plt.xlabel("Feature Importance", fontsize = 15)
    plt.ylabel("Feature", fontsize = 15)
    plt.ylim(-1, cols)
    plt.xlim(0,1)
    plt.show();

```

In [19]:

```
1 widgets.interact(gradientBoosted, categorical=True, learningRate=(.01,1,.01), estimators=(1,150,1
```

categorical

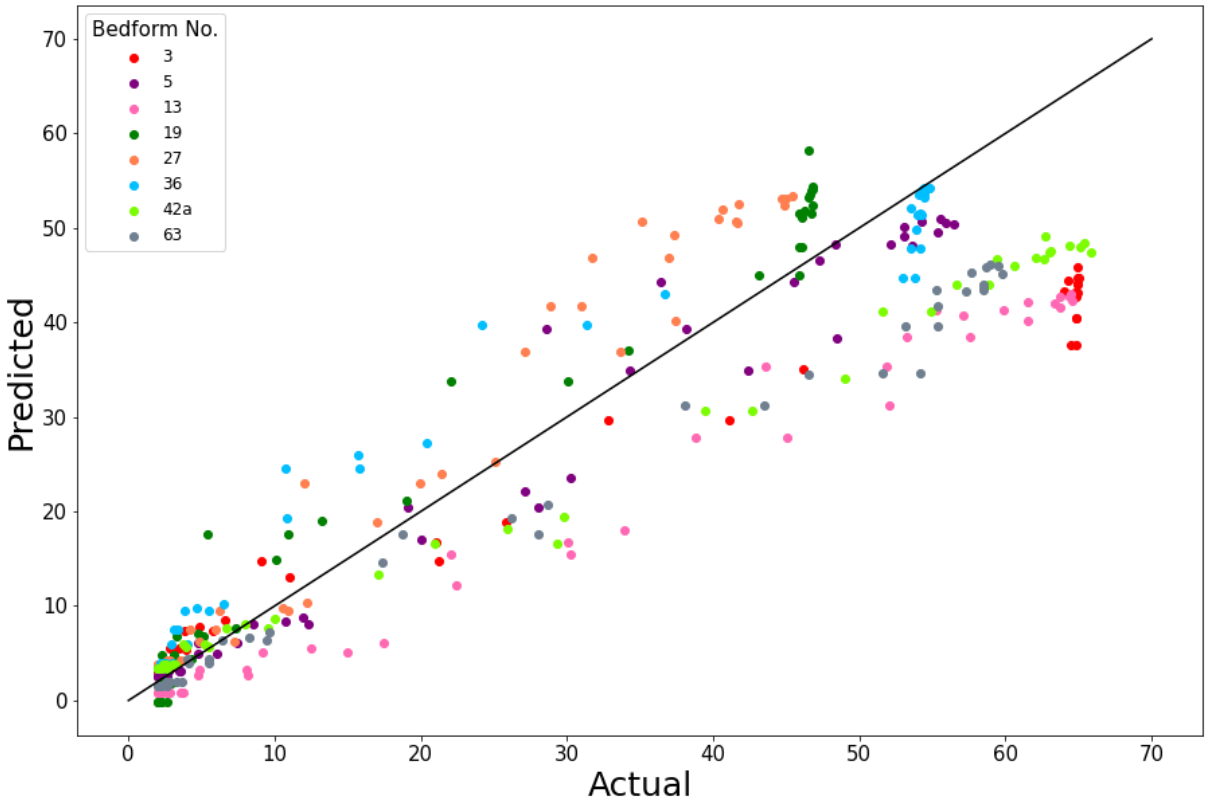
learningRate 0.18

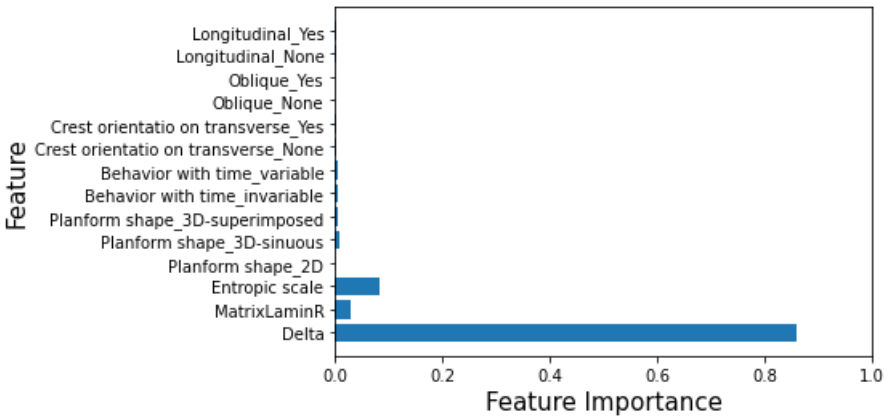
estimators 150

depth 3

```
R2 Score:  0.8374980871644221

CV Scores (training):
Mean: 0.68304
Standard Deviation: 0.13593
```





In [20]:

```
1 %reset
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```