In [1]:
```
from sklearn.linear_model import LinearRegression
import numpy as np
import pandas as pd
from tqdm import tqdm
import tensorflow.keras.utils as image
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt
import cv2
from sklearn.svm import SVR
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
import tensorflow as tf
import numpy as np
import pandas as pd
from tqdm import tqdm
import tensorflow.keras.utils as image
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import EarlyStopping
import colorsys
from keras.models import Sequential
from keras.layers import Dense
from keras import backend as K
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
```

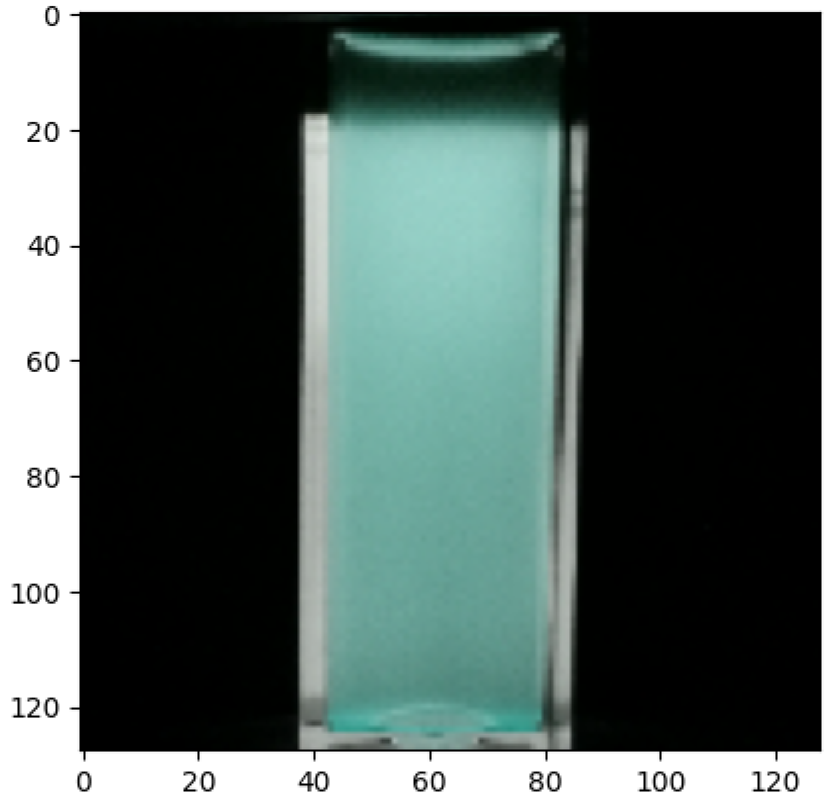
In [2]:
```
# function to create a list of numbers as x for graph plotting
def createList(r1, r2):
    # Testing if range r1 and r2
    # are equal
    if (r1 == r2):
        return r1
    else:
        # Create empty list
        res = []
        # loop to append successors to
        # list until r2 is reached.
        while(r1 < r2+1 ):
            res.append(r1)
            r1 += 1
    return res

# function to determine R-squred for keras models
def r_square(y_true, y_pred):
    SS_res =  K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )
```

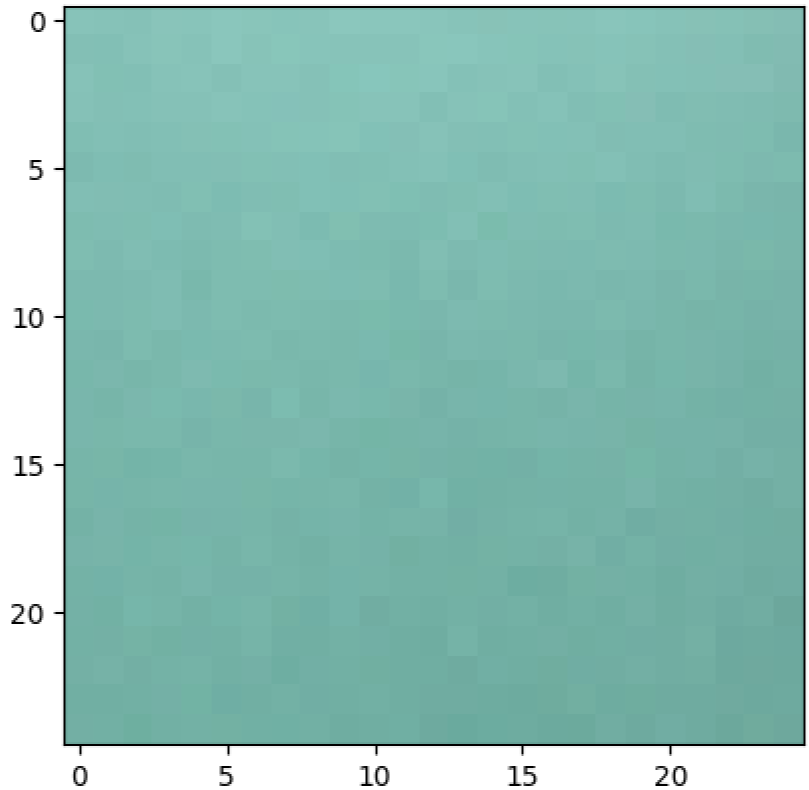
In [3]:
```
# Load the data
csv_file = r'C:\\Users\\User\\Documents\\Year 4\\ml_mep2\\image\\image_dataset\\ima
df = pd.read_csv(csv_file)
```

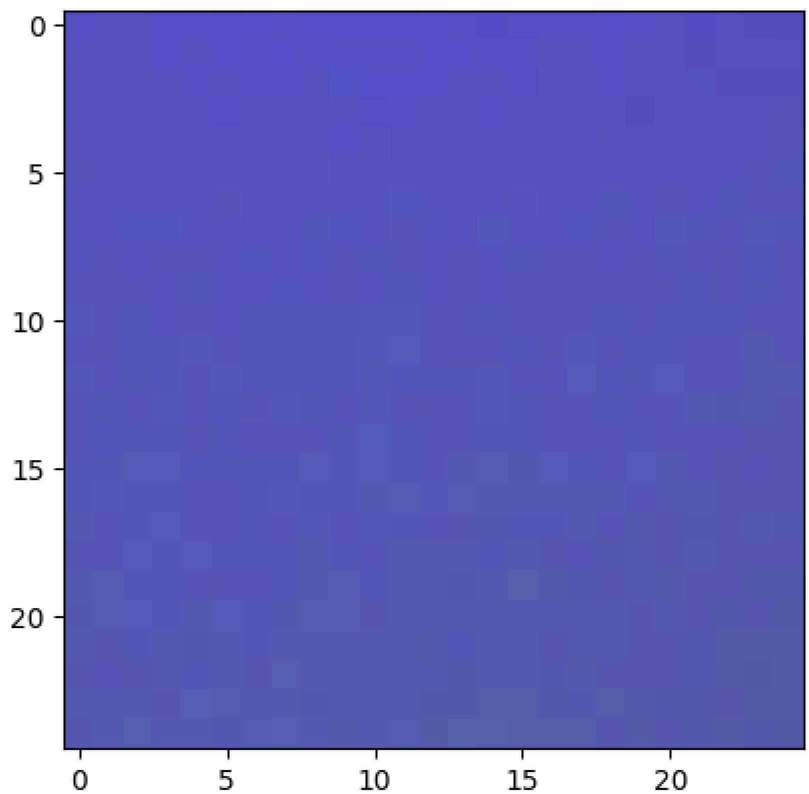
```
df_RGBimage = []
df_HSVimage = []
df_RGB = []
df_HSV = []

image_width = 128
image_height = 128

for i in tqdm(range(df.shape[0])):
    # Load images according to the csv file
    img = cv2.imread(r'C:\\Users\\User\\Documents\\Year 4\\ml_mep2\\image\\image_da
    img = cv2.resize(img, (image_width, image_height))

    # Crop 20% of the image from the image midpoint
    left = int(image_width*0.4)
    top = int(image_height*0.4)
    right = int(image_width*0.6)
    bottom = int(image_height*0.6)
    img_crop = img[left:right, top:bottom]

    # Image to array
    img_RGB = image.img_to_array(img_crop)

     # Convert image into HSV image
    img_HSV1 = cv2.cvtColor(img_crop, cv2.COLOR_RGB2HSV)
    img_HSV = image.img_to_array(img_HSV1)

    # Obtain average RGB value of the image
    average_RGB = img_crop.mean(axis=(0, 1))

    # Convert the average RGB value to HSV
    average_HSV = colorsys.rgb_to_hsv(*average_RGB)

    # Append Data into df
    df_RGBimage.append(img_RGB)
    df_HSVimage.append(img_HSV)
    df_RGB.append(average_RGB)
    df_HSV.append(average_HSV)

# Prepare X and Y for the models

num_channel = 3 #RGB and HSV both = 3
image_width = int(right-left)
image_height = int(bottom-top)
X_col = len(df.index)

X_RGBimage = np.array(df_RGBimage)
X_RGBimage = X_RGBimage.reshape(X_col, image_width, image_height, num_channel)
X_RGBimage /= 255

X_HSVimage = np.array(df_HSVimage)
X_HSVimage = X_HSVimage.reshape(X_col, image_width, image_height, num_channel)
X_HSVimage /= 360

X_RGB = np.array(df_RGB)
```

```
X_RGB /= 255

X_HSV = np.array(df_HSV)
X_HSV /= 360

y1 = np.array(df['absorbance'])

# Plot and show example result
plt.figure()
plt.imshow(img)
plt.show()
plt.figure()
plt.imshow(img_crop)
plt.show()
plt.figure()
plt.imshow(img_HSV1)
plt.show()
print(average_RGB)
print(average_HSV)
```

```
100%|██████████████████████████████████████████
███| 226/226 [00:18<00:00, 12.28it/s]
```







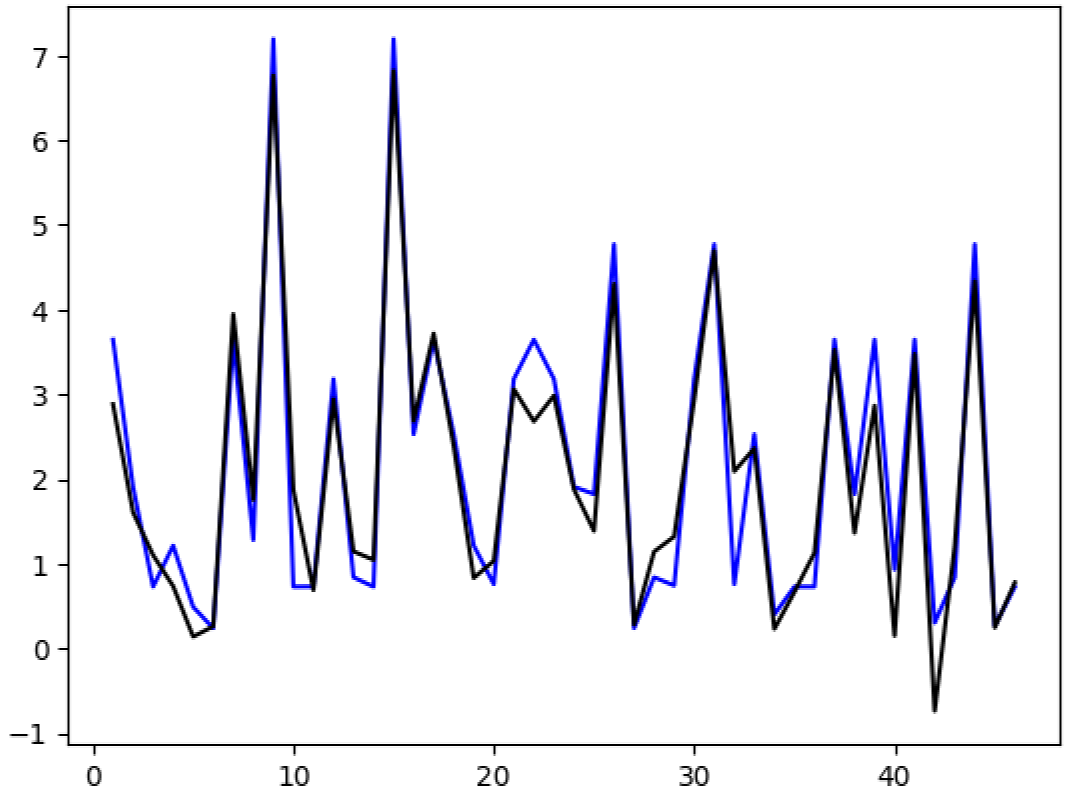

```
[121.6944 183.0256 171.9264]
(0.4698380813245678, 0.33509629254049705, 183.0256)
```

In [4]:
```
# Split the data into training and validation sets
X_RGBimage_train, X_RGBimage_test, y1_RGBimage_train, y1_RGBimage_test = train_test
    X_RGBimage,
    y1,
```

```
    test_size=0.2,
    random_state=1
)

X_HSVimage_train, X_HSVimage_test, y1_HSVimage_train, y1_HSVimage_test = train_test
    X_HSVimage,
    y1,
    test_size=0.2,
    random_state=1
)

X_RGB_train, X_RGB_test, y1_RGB_train, y1_RGB_test = train_test_split(
    X_RGB,
    y1,
    test_size=0.2,
    random_state=1
)

X_HSV_train, X_HSV_test, y1_HSV_train, y1_HSV_test = train_test_split(
    X_HSV,
    y1,
    test_size=0.2,
    random_state=1
)
```

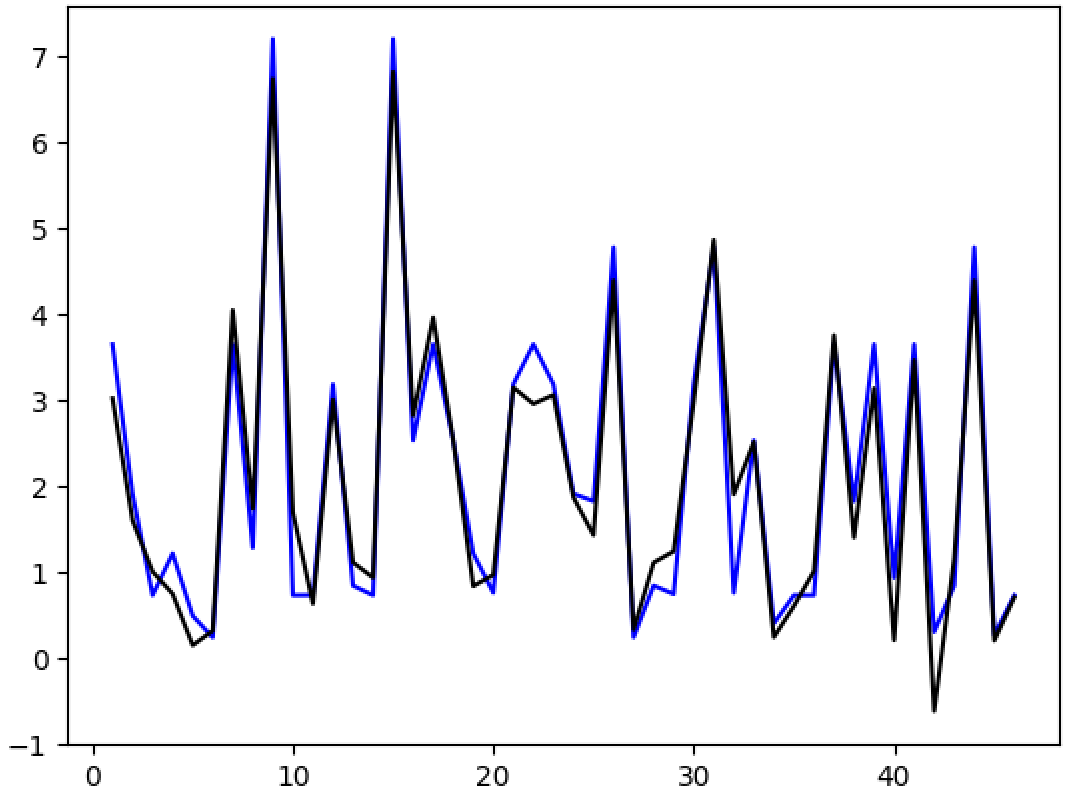
In [5]:
```
# Linear Regression Model - RGB
model = LinearRegression()
model.fit(X_RGB_train, y1_RGB_train)

# Evaluate the model on the validation set
score_LR_RGB = model.score(X_RGB_test, y1_RGB_test)
print('LR_RGB - R^2:', score_LR_RGB)

y_pred = model.predict(X_RGB_test)

# Calculate the MSE and MAE
mse_LR_RGB = mean_squared_error(y1_RGB_test, y_pred)
mae_LR_RGB = mean_absolute_error(y1_RGB_test, y_pred)

# Print the results
print("LR_RGB - MSE:", mse_LR_RGB)
print("LR_RGB - MAE:", mae_LR_RGB)

#plot
x_plt = createList(1, len(y1_RGB_test))

plt.plot(x_plt, y1_RGB_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

```
LR_RGB - R^2: 0.9263658123802966
LR_RGB - MSE: 0.22623414986670956
LR_RGB - MAE: 0.3643029555517797
```



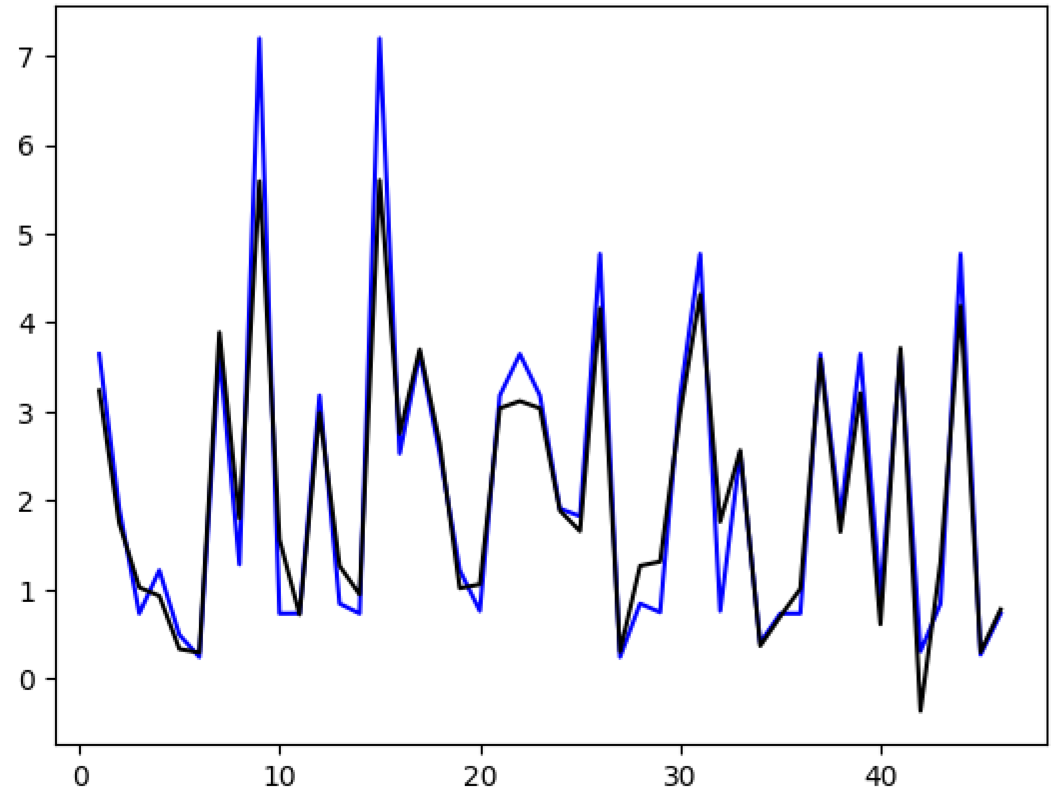
In [7]:
```
# Linear Regression Model - HSV
model = LinearRegression()
model.fit(X_HSV_train, y1_HSV_train)

# Evaluate the model on the validation set
score_LR_HSV = model.score(X_HSV_test, y1_HSV_test)
print('LR_HSV - R^2:', score_LR_HSV)

y_pred = model.predict(X_HSV_test)

# Calculate the MSE and MAE
mse_LR_HSV = mean_squared_error(y1_HSV_test, y_pred)
mae_LR_HSV = mean_absolute_error(y1_HSV_test, y_pred)

# Print the results
print("LR_HSV - MSE:", mse_LR_HSV)
print("LR_HSV - MAE:", mae_LR_HSV)

#plot
x_plt = createList(1, len(y1_HSV_test))

plt.plot(x_plt, y1_HSV_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

```
LR_HSV - R^2: 0.9447663596758173
LR_HSV - MSE: 0.16970019045122714
LR_HSV - MAE: 0.3242236316288698
```



In [8]:
```
# Support Vector Regression - RGB
model = SVR(kernel='linear')

# Train the model
model.fit(X_RGB_train, y1_RGB_train)

# Evaluate the model on the validation set
score_SVR_RGB = model.score(X_RGB_test, y1_RGB_test)
print('SVR_RGB - R^2:', score_SVR_RGB)

# Make predictions on the test set
y_pred = model.predict(X_RGB_test)

# Evaluate the model
mse_SVR_RGB = mean_squared_error(y1_RGB_test, y_pred)
mae_SVR_RGB = mean_absolute_error(y1_RGB_test, y_pred)

# Print the results
print("SVR_RGB - MSE:", mse_SVR_RGB)
print("SVR_RGB - MAE:", mae_SVR_RGB)

#plot
x_plt = createList(1, len(y1_RGB_test))

plt.plot(x_plt, y1_RGB_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

```
SVR_RGB - R^2: 0.9228615244648459
SVR_RGB - MSE: 0.23700074651247788
SVR_RGB - MAE: 0.33646104500648566
```

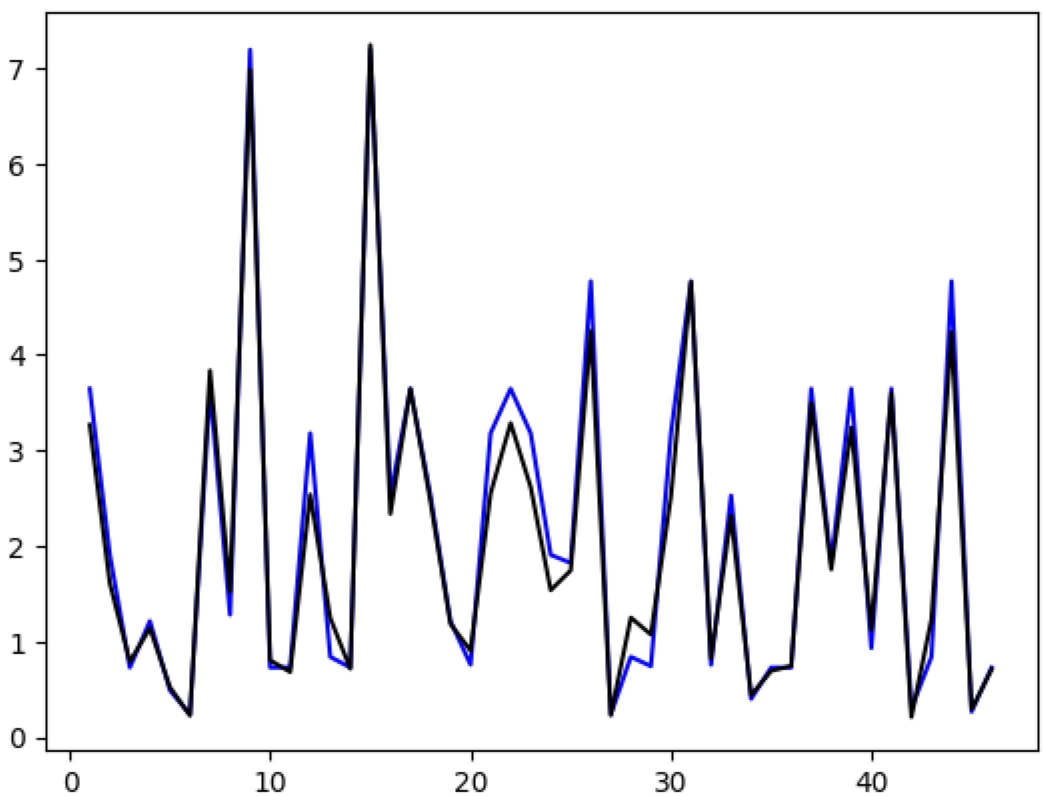


In [9]:

```
# Support Vector Regression - RGB (optimized via grid search method)

# Set the parameters for the grid search
parameters = {
    'kernel': ['linear', 'rbf'],
    'C': [1, 10, 100],
    'epsilon': [0.1, 0.01]
}

# Create the grid search object
grid_search = GridSearchCV(SVR(), parameters, cv=5, verbose=1)

# Fit the grid search object to the training data
grid_search.fit(X_RGB_train, y1_RGB_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print(best_params)

# Create an SVM model with the best parameters
model = SVR(**best_params)

# Train the model
model.fit(X_RGB_train, y1_RGB_train)

# Evaluate the model on the validation set
print('SVR_RGB - R^2            :', score_SVR_RGB)
score_SVR_RGB = model.score(X_RGB_test, y1_RGB_test)
```

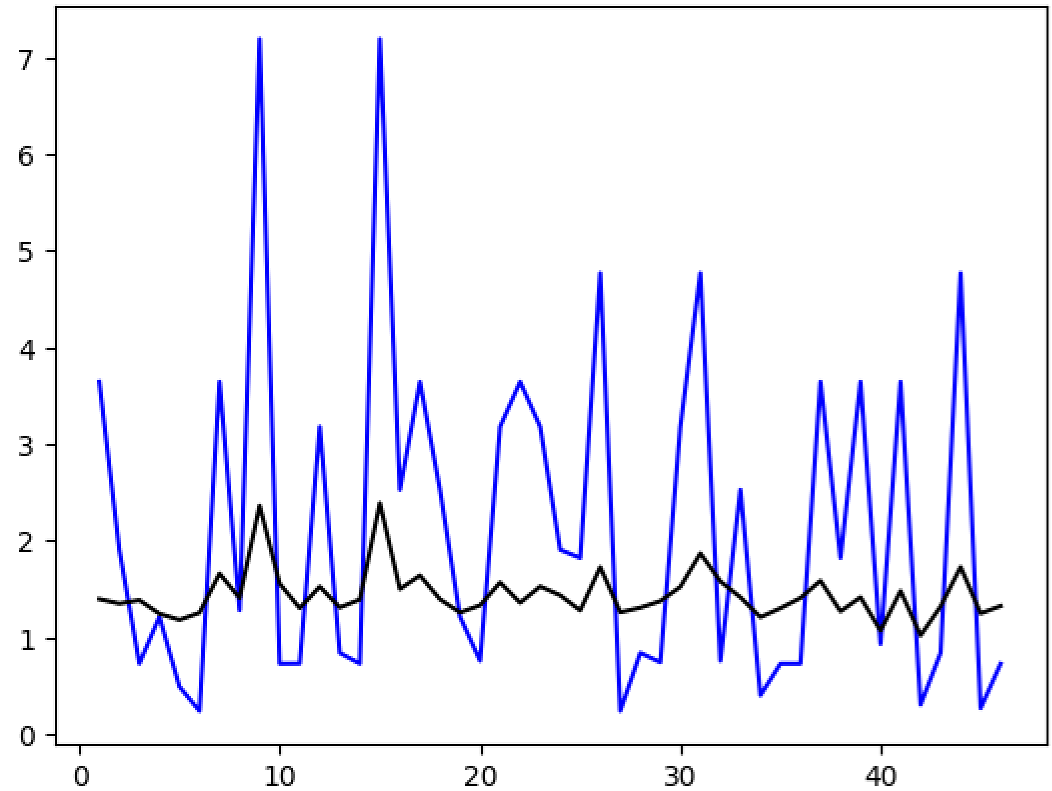
```
print('SVR_RGB - R^2 (optimized):', score_SVR_RGB)

# Make predictions on the test set
y_pred = model.predict(X_RGB_test)

# Evaluate the model
print("SVR_RGB - MSE            :", mse_SVR_RGB)
mse_SVR_RGB = mean_squared_error(y1_RGB_test, y_pred)
print("SVR_RGB - MSE (optimized):", mse_SVR_RGB)

print("SVR_RGB - MAE            :", mae_SVR_RGB)
mae_SVR_RGB = mean_absolute_error(y1_RGB_test, y_pred)
print("SVR_RGB - MAE (optimized):", mae_SVR_RGB)

#plot
x_plt = createList(1, len(y1_RGB_test))

plt.plot(x_plt, y1_RGB_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
{'C': 100, 'epsilon': 0.01, 'kernel': 'rbf'}
SVR_RGB - R^2            : 0.9228615244648459
SVR_RGB - R^2 (optimized): 0.9735840886952669
SVR_RGB - MSE            : 0.23700074651247788
SVR_RGB - MSE (optimized): 0.08116041515723273
SVR_RGB - MAE            : 0.33646104500648566
SVR_RGB - MAE (optimized): 0.20426112807795507
```



In [10]:
```
# Support Vector Regression - HSV
model = SVR(kernel='linear')

# Train the model
model.fit(X_HSV_train, y1_HSV_train)

# Evaluate the model on the validation set
score_SVR_HSV = model.score(X_HSV_test, y1_HSV_test)
print('SVR_HSV - R^2:', score_SVR_HSV)

# Make predictions on the test set
y_pred = model.predict(X_HSV_test)

# Evaluate the model
mse_SVR_HSV = mean_squared_error(y1_HSV_test, y_pred)
mae_SVR_HSV = mean_absolute_error(y1_HSV_test, y_pred)

# Print the results
print("SVR_HSV - MSE:", mse_SVR_HSV)
print("SVR_HSV - MAE:", mae_SVR_HSV)

#plot
x_plt = createList(1, len(y1_HSV_test))

plt.plot(x_plt, y1_HSV_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

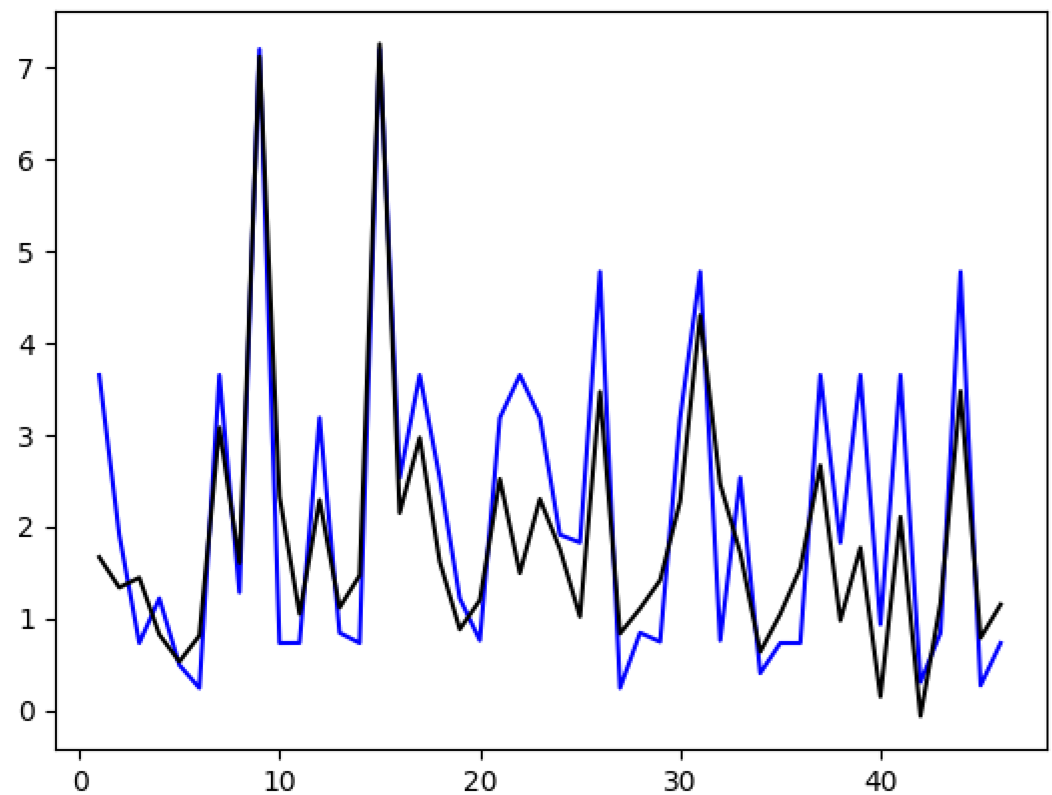
```
SVR_HSV - R^2: 0.0745572292107286
SVR_HSV - MSE: 2.8433362989093327
SVR_HSV - MAE: 1.2860904552372276
```



In [11]:
```
# Support Vector Regression - HSV (optimized via grid search method)

# Set the parameters for the grid search
parameters = {
    'kernel': ['linear', 'rbf'],
    'C': [1, 10, 100],
    'epsilon': [0.1, 0.01]
}

# Create the grid search object
grid_search = GridSearchCV(SVR(), parameters, cv=5, verbose=1)

# Fit the grid search object to the training data
grid_search.fit(X_HSV_train, y1_HSV_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print(best_params)

# Create an SVM model with the best parameters
model = SVR(**best_params)

# Train the model
model.fit(X_HSV_train, y1_HSV_train)

# Evaluate the model on the validation set
print('SVR_HSV - R^2            :', score_SVR_HSV)
score_SVR_HSV = model.score(X_HSV_test, y1_HSV_test)
print('SVR_HSV - R^2 (optimized):', score_SVR_HSV)

# Make predictions on the test set
```

```
y_pred = model.predict(X_HSV_test)

# Evaluate the model
print("SVR_HSV - MSE            :", mse_SVR_HSV)
mse_SVR_HSV = mean_squared_error(y1_HSV_test, y_pred)
print("SVR_HSV - MSE (optimized):", mse_SVR_HSV)

print("SVR_HSV - MAE            :", mae_SVR_HSV)
mae_SVR_HSV = mean_absolute_error(y1_HSV_test, y_pred)
print("SVR_HSV - MAE (optimized):", mae_SVR_HSV)

#plot
x_plt = createList(1, len(y1_HSV_test))

plt.plot(x_plt, y1_HSV_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

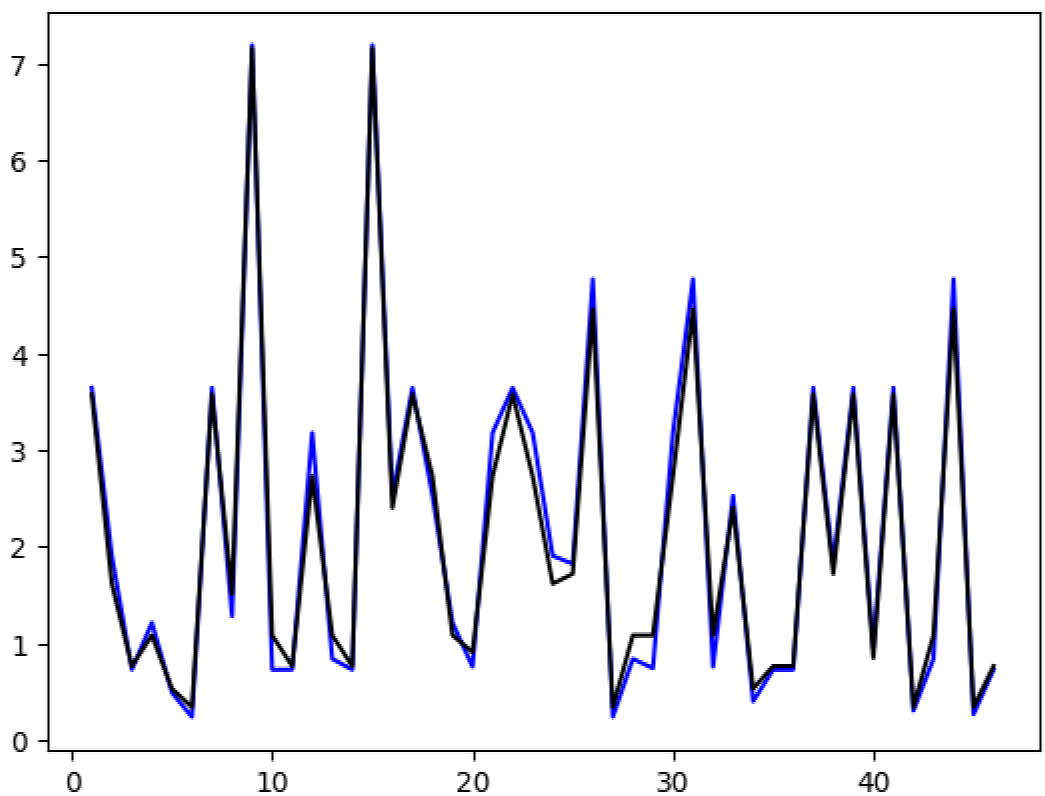
```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
{'C': 100, 'epsilon': 0.1, 'kernel': 'rbf'}
SVR_HSV - R^2            : 0.0745572292107286
SVR_HSV - R^2 (optimized): 0.7404713090458613
SVR_HSV - MSE            : 2.8433362989093327
SVR_HSV - MSE (optimized): 0.7973776130629638
SVR_HSV - MAE            : 1.2860904552372276
SVR_HSV - MAE (optimized): 0.7309039806203493
```



In [18]:
```
# Gradient Boosting Regressor - RGB

# Create the model
model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=1,
```

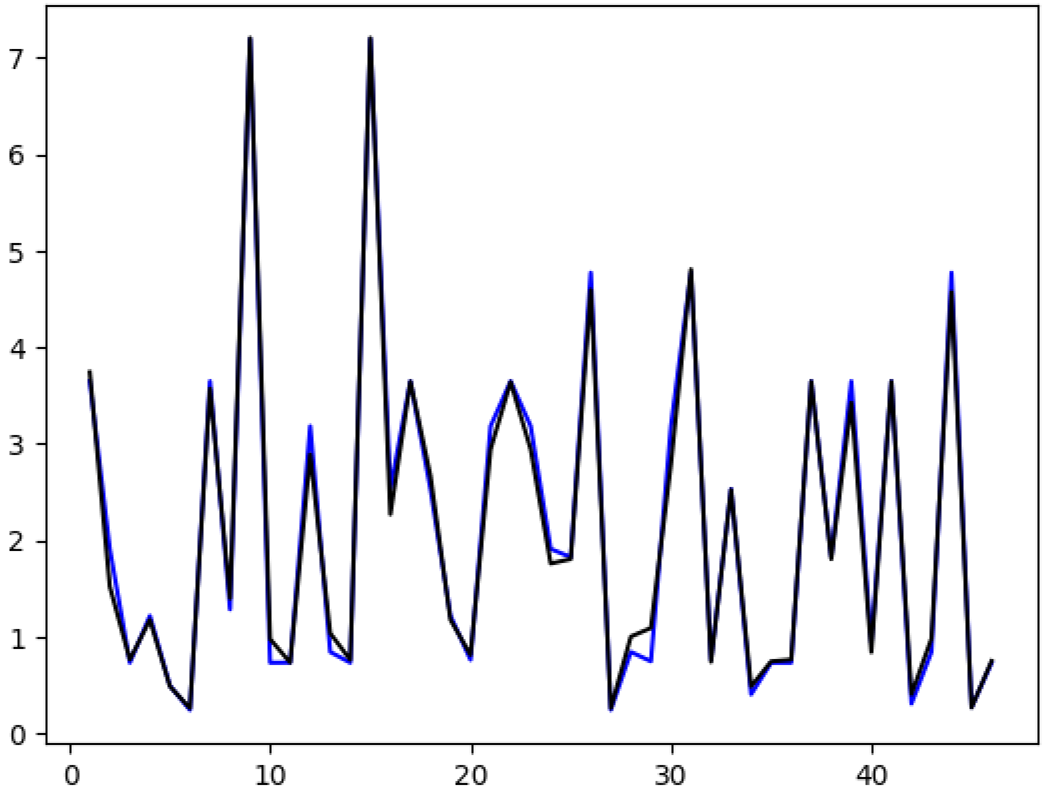
```
# Fit the model to the training data
model.fit(X_RGB_train, y1_RGB_train)

# Evaluate the model on the validation set
score_GBR_RGB = model.score(X_RGB_test, y1_RGB_test)
print('GBR_RGB - R^2:', score_GBR_RGB)

# Make predictions on the test set
y_pred = model.predict(X_RGB_test)

# Evaluate the model
mse_GBR_RGB = mean_squared_error(y1_RGB_test, y_pred)
mae_GBR_RGB = mean_absolute_error(y1_RGB_test, y_pred)

# Print the results
print("GBR_RGB - MSE:", mse_GBR_RGB)
print("GBR_RGB - MAE:", mae_GBR_RGB)

#plot
x_plt = createList(1, len(y1_RGB_test))

plt.plot(x_plt, y1_RGB_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

```
GBR_RGB - R^2: 0.9851552229136885
GBR_RGB - MSE: 0.04560918824048817
GBR_RGB - MAE: 0.16764285881086632
```



In [19]:
```
# Gradient Boosting Regressor - RGB (optimized)

# Set the parameters for the grid search
parameters = {
    'n_estimators': [100, 200],
    'max_depth': [1, 2, 3, 4],
    'learning_rate': [0.1, 0.05],
    'loss': ['squared_error', 'absolute_error', 'huber']
}

# Create the grid search object
grid_search = GridSearchCV(GradientBoostingRegressor(random_state=0), parameters, c

# Fit the grid search object to the training data
grid_search.fit(X_RGB_train, y1_RGB_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print(best_params)

# Create model with the best parameters
model = GradientBoostingRegressor(**best_params)

# Train the model
model.fit(X_RGB_train, y1_RGB_train)

# Evaluate the model on the validation set
print('GBR_RGB - R^2            :', score_GBR_RGB)
score_GBR_RGB = model.score(X_RGB_test, y1_RGB_test)
print('GBR_RGB - R^2 (optimized):', score_GBR_RGB)

# Make predictions on the test set
y_pred = model.predict(X_RGB_test)

# Evaluate the model
print("GBR_RGB - MSE            :", mse_GBR_RGB)
mse_GBR_RGB = mean_squared_error(y1_RGB_test, y_pred)
print("GBR_RGB - MSE (optimized):", mse_GBR_RGB)

print("GBR_RGB - MAE            :", mae_GBR_RGB)
mae_GBR_RGB = mean_absolute_error(y1_RGB_test, y_pred)
print("GBR_RGB - MAE (optimized):", mae_GBR_RGB)

#plot
x_plt = createList(1, len(y1_RGB_test))

plt.plot(x_plt, y1_RGB_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
{'learning_rate': 0.1, 'loss': 'huber', 'max_depth': 3, 'n_estimators': 100}
GBR_RGB - R^2            : 0.9851552229136885
GBR_RGB - R^2 (optimized): 0.9924469047619349
GBR_RGB - MSE            : 0.04560918824048817
GBR_RGB - MSE (optimized): 0.02320617820720988
GBR_RGB - MAE            : 0.16764285881086632
GBR_RGB - MAE (optimized): 0.10419132544457241
```

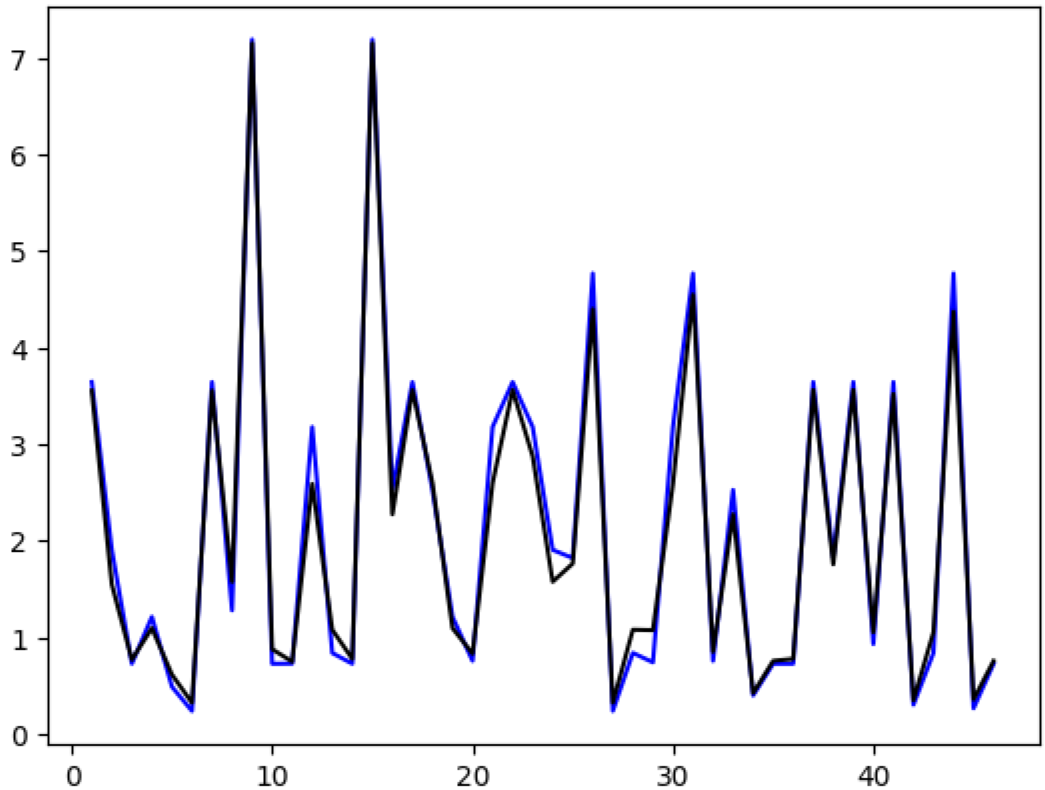


In [20]:

```python
# Gradient Boosting Regressor - HSV

# Create the model
model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=1,

# Fit the model to the training data
model.fit(X_HSV_train, y1_HSV_train)

# Evaluate the model on the validation set
score_GBR_HSV = model.score(X_HSV_test, y1_HSV_test)
print('GBR_HSV - R^2:', score_GBR_HSV)

# Make predictions on the test set
y_pred = model.predict(X_HSV_test)

# Evaluate the model
mse_GBR_HSV = mean_squared_error(y1_HSV_test, y_pred)
mae_GBR_HSV = mean_absolute_error(y1_HSV_test, y_pred)

# Print the results
print("GBR_HSV - MSE:", mse_GBR_HSV)
print("GBR_HSV - MAE:", mae_GBR_HSV)
```

```
#plot
x_plt = createList(1, len(y1_HSV_test))

plt.plot(x_plt, y1_HSV_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

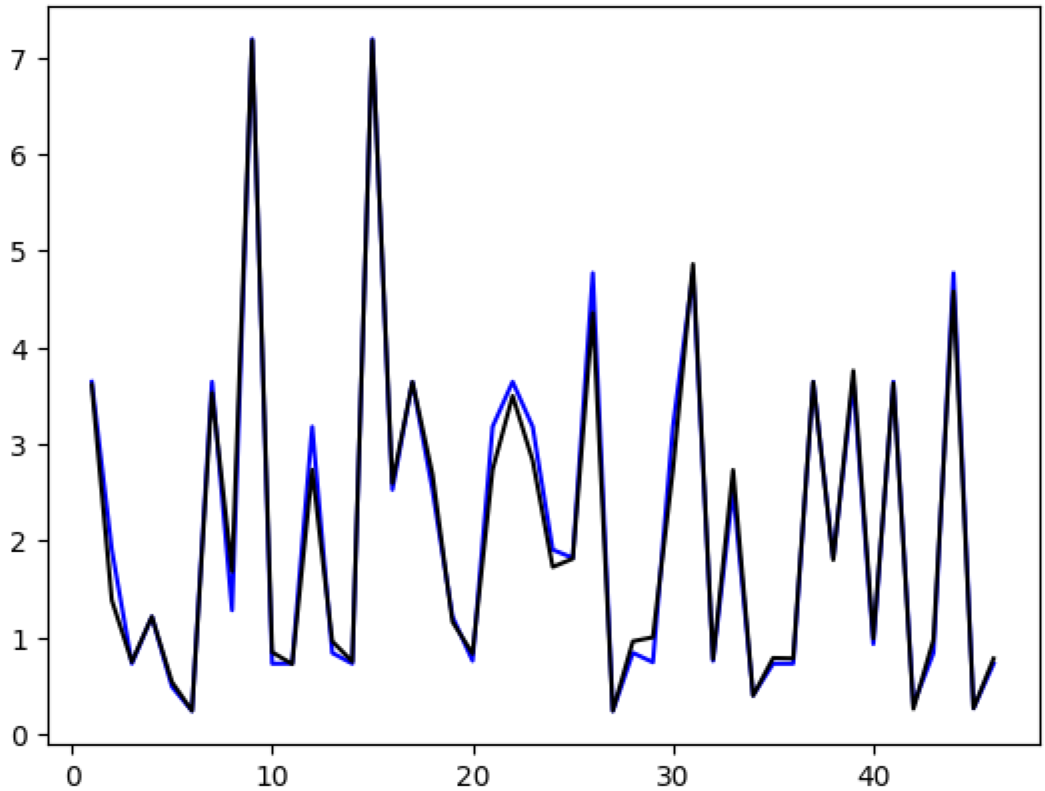
```
GBR_HSV - R^2: 0.9830324719420408
GBR_HSV - MSE: 0.05213114192767685
GBR_HSV - MAE: 0.1702481025469859
```



In [21]:
```
# Gradient Boosting Regressor - HSV (optimized)

# Set the parameters for the grid search
parameters = {
    'n_estimators': [100, 200, 300, 400],
    'max_depth': [1, 2, 3],
    'learning_rate': [0.1, 0.05],
    'loss': ['squared_error', 'absolute_error', 'huber']
}

# Create the grid search object
grid_search = GridSearchCV(GradientBoostingRegressor(random_state=0), parameters, c

# Fit the grid search object to the training data
grid_search.fit(X_HSV_train, y1_HSV_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print(best_params)
```

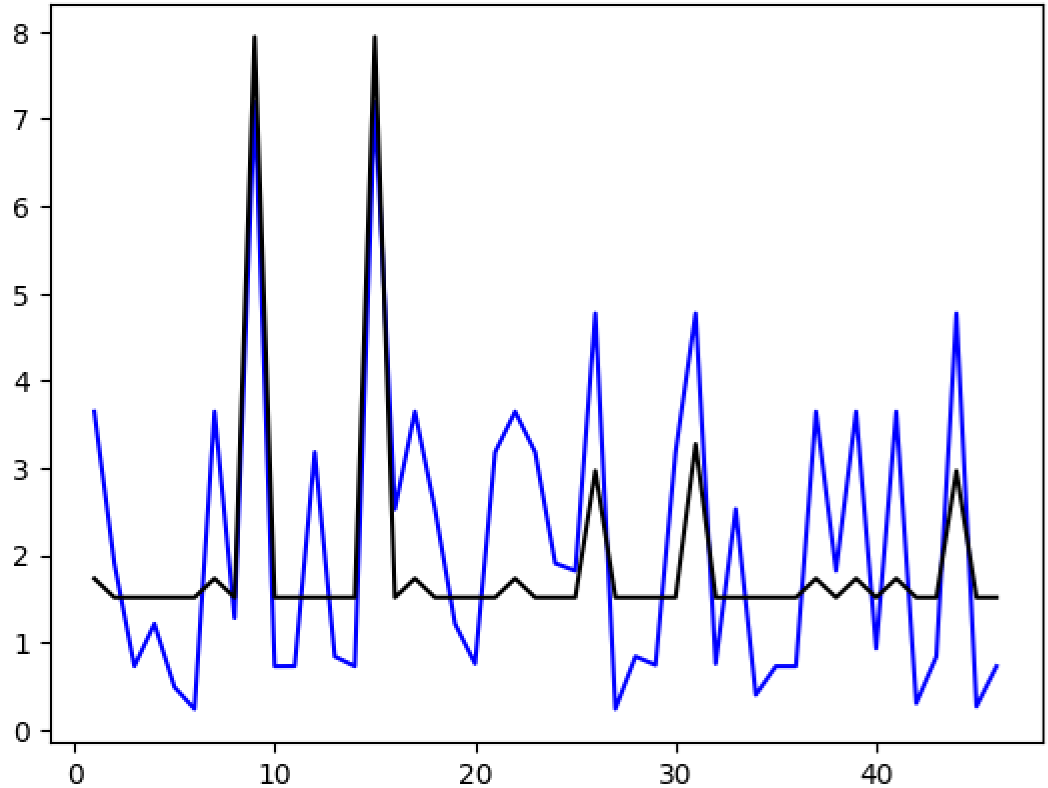
```
# Create model with the best parameters
model = GradientBoostingRegressor(**best_params)

# Train the model
model.fit(X_HSV_train, y1_HSV_train)

# Evaluate the model on the validation set
print('GBR_HSV - R^2            :', score_GBR_HSV)
score_GBR_HSV = model.score(X_HSV_test, y1_HSV_test)
print('GBR_HSV - R^2 (optimized):', score_GBR_HSV)

# Make predictions on the test set
y_pred = model.predict(X_HSV_test)

# Evaluate the model
print("GBR_HSV - MSE            :", mse_GBR_HSV)
mse_GBR_HSV = mean_squared_error(y1_HSV_test, y_pred)
print("GBR_HSV - MSE (optimized):", mse_GBR_HSV)

print("GBR_HSV - MAE            :", mae_GBR_HSV)
mae_GBR_HSV = mean_absolute_error(y1_HSV_test, y_pred)
print("GBR_HSV - MAE (optimized):", mae_GBR_HSV)

#plot
x_plt = createList(1, len(y1_HSV_test))

plt.plot(x_plt, y1_HSV_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
{'learning_rate': 0.05, 'loss': 'huber', 'max_depth': 3, 'n_estimators': 200}
GBR_HSV - R^2            : 0.9830324719420408
GBR_HSV - R^2 (optimized): 0.9881539047585942
GBR_HSV - MSE            : 0.05213114192767685
GBR_HSV - MSE (optimized): 0.03639601892562181
GBR_HSV - MAE            : 0.1702481025469859
GBR_HSV - MAE (optimized): 0.12317983151752247
```



In [22]:
```
# Random Forest Regressor - RGB

# Create the model
model = RandomForestRegressor(n_estimators=100, max_depth=1, random_state=0)

# Fit the model to the training data
model.fit(X_RGB_train, y1_RGB_train)

# Evaluate the model on the validation set
score_RFR_RGB = model.score(X_RGB_test, y1_RGB_test)
print('RFR_RGB - R^2:', score_RFR_RGB)

# Make predictions on the test set
y_pred = model.predict(X_RGB_test)

# Evaluate the model
mse_RFR_RGB = mean_squared_error(y1_RGB_test, y_pred)
mae_RFR_RGB = mean_absolute_error(y1_RGB_test, y_pred)

# Print the results
print("RFR_RGB - MSE:", mse_RFR_RGB)
print("RFR_RGB - MAE:", mae_RFR_RGB)

#plot
x_plt = createList(1, len(y1_RGB_test))

plt.plot(x_plt, y1_RGB_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

```
RFR_RGB - R^2: 0.5278806918044185
RFR_RGB - MSE: 1.450542387687126
RFR_RGB - MAE: 1.0756192169485517
```

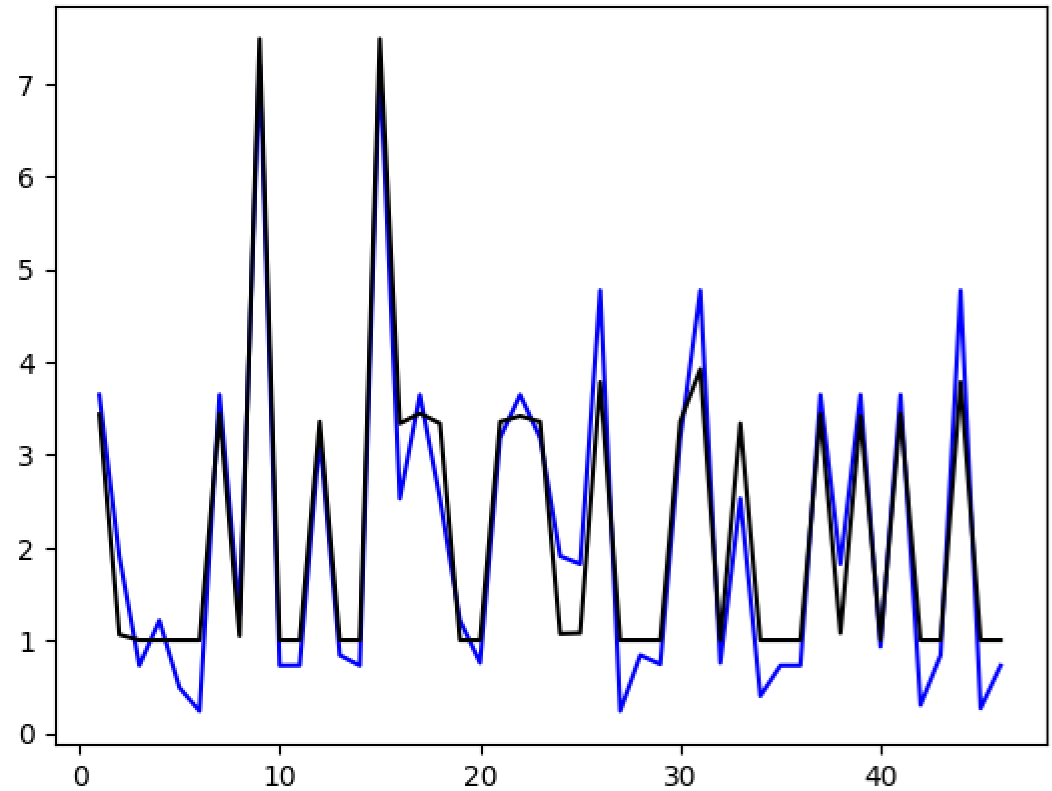


In [23]:

```
# Random Forest Regressor - RGB (optimized)

# Set the parameters for the grid search
parameters = {
    'n_estimators': [100, 200, 300, 400],
    'max_depth': [1, 2, 3, 4, 5],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 3]
}

# Create the grid search object
grid_search = GridSearchCV(GradientBoostingRegressor(random_state=0), parameters, c

# Fit the grid search object to the training data
grid_search.fit(X_RGB_train, y1_RGB_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print(best_params)

# Create model with the best parameters
model = RandomForestRegressor(**best_params)

# Train the model
model.fit(X_RGB_train, y1_RGB_train)

# Evaluate the model on the validation set
print('RFR_RGB - R^2            :', score_RFR_RGB)
```

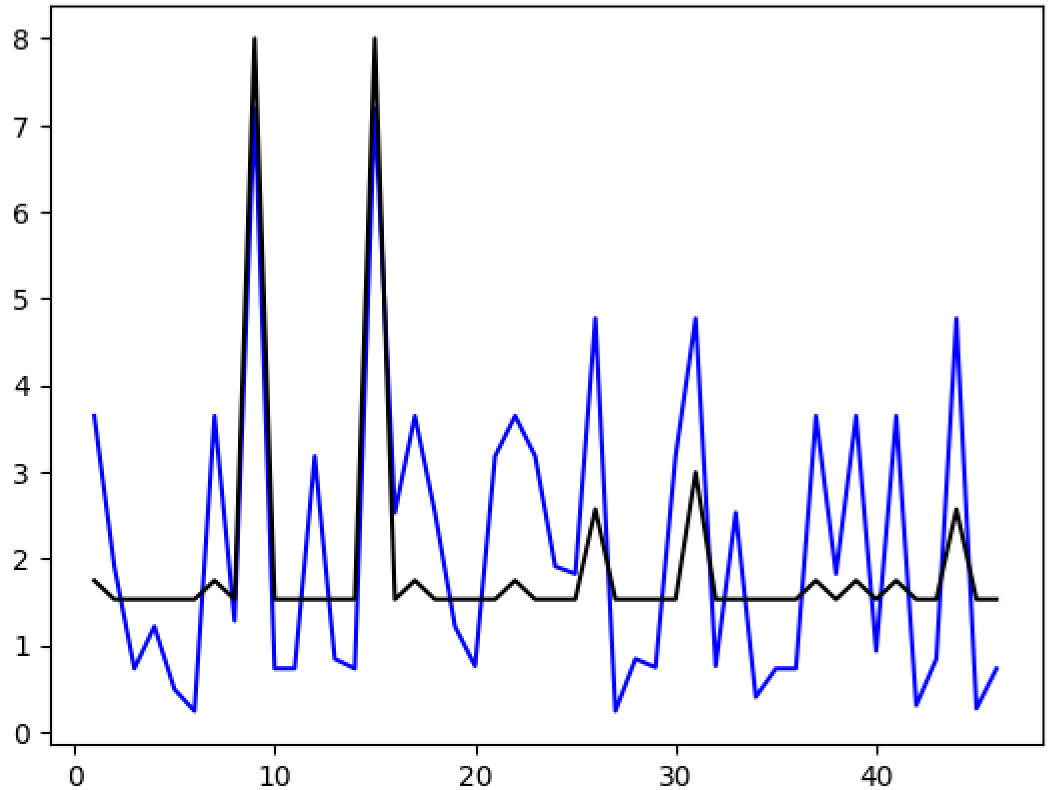
```
score_RFR_RGB = model.score(X_RGB_test, y1_RGB_test)
print('RFR_RGB - R^2 (optimized):', score_RFR_RGB)

# Make predictions on the test set
y_pred = model.predict(X_RGB_test)

# Evaluate the model
print("RFR_RGB - MSE            :", mse_RFR_RGB)
mse_RFR_RGB = mean_squared_error(y1_RGB_test, y_pred)
print("RFR_RGB - MSE (optimized):", mse_RFR_RGB)

print("RFR_RGB - MAE            :", mae_RFR_RGB)
mae_RFR_RGB = mean_absolute_error(y1_RGB_test, y_pred)
print("RFR_RGB - MAE (optimized):", mae_RFR_RGB)

#plot
x_plt = createList(1, len(y1_RGB_test))

plt.plot(x_plt, y1_RGB_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

```
Fitting 5 folds for each of 180 candidates, totalling 900 fits
{'max_depth': 2, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 40
0}
RFR_RGB - R^2            : 0.5278806918044185
RFR_RGB - R^2 (optimized): 0.9185336200759778
RFR_RGB - MSE            : 1.450542387687126
RFR_RGB - MSE (optimized): 0.2502978276886401
RFR_RGB - MAE            : 1.0756192169485517
RFR_RGB - MAE (optimized): 0.4158609182236845
```



In [24]:
```
# Random Forest Regressor - HSV

# Create the model
model = RandomForestRegressor(n_estimators=100, max_depth=1, random_state=0)

# Fit the model to the training data
model.fit(X_HSV_train, y1_HSV_train)

# Evaluate the model on the validation set
score_RFR_HSV = model.score(X_HSV_test, y1_HSV_test)
print('RFR_HSV - R^2:', score_RFR_HSV)

# Make predictions on the test set
y_pred = model.predict(X_HSV_test)

# Evaluate the model
mse_RFR_HSV = mean_squared_error(y1_HSV_test, y_pred)
mae_RFR_HSV = mean_absolute_error(y1_HSV_test, y_pred)

# Print the results
print("RFR_HSV - MSE:", mse_RFR_HSV)
print("RFR_HSV - MAE:", mae_RFR_HSV)

#plot
x_plt = createList(1, len(y1_HSV_test))

plt.plot(x_plt, y1_HSV_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

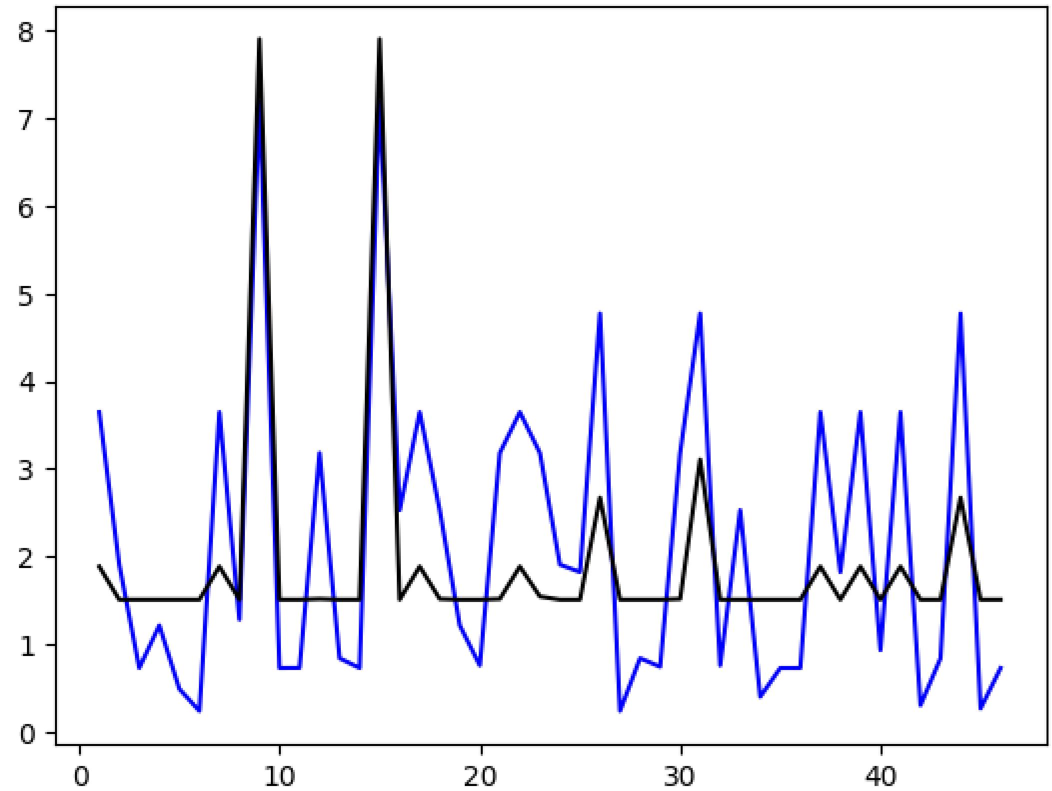
```
RFR_HSV - R^2: 0.49837507872254505
RFR_HSV - MSE: 1.541195622382249
RFR_HSV - MAE: 1.1028965720838613
```



In [25]:
```
# Random Forest Regressor - HSV (optimized)

# Set the parameters for the grid search
parameters = {
    'n_estimators': [100, 200, 300, 400],
    'max_depth': [1, 2],
    'min_samples_split': [2, 4],
    'min_samples_leaf': [1, 2]
}

# Create the grid search object
grid_search = GridSearchCV(GradientBoostingRegressor(random_state=0), parameters, c

# Fit the grid search object to the training data
grid_search.fit(X_HSV_train, y1_HSV_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print(best_params)

# Create model with the best parameters
model = RandomForestRegressor(**best_params)

# Train the model
model.fit(X_HSV_train, y1_HSV_train)

# Evaluate the model on the validation set
print('RFR_HSV - R^2            :', score_RFR_HSV)
```

```
score_RFR_HSV = model.score(X_HSV_test, y1_HSV_test)
print('RFR_HSV - R^2 (optimized):', score_RFR_HSV)

# Make predictions on the test set
y_pred = model.predict(X_HSV_test)

# Evaluate the model
print("RFR_HSV - MSE            :", mse_RFR_HSV)
mse_RFR_HSV = mean_squared_error(y1_HSV_test, y_pred)
print("RFR_HSV - MSE (optimized):", mse_RFR_HSV)

print("RFR_HSV - MAE            :", mae_RFR_HSV)
mae_RFR_HSV = mean_absolute_error(y1_HSV_test, y_pred)
print("RFR_HSV - MAE (optimized):", mae_RFR_HSV)

#plot
x_plt = createList(1, len(y1_HSV_test))

plt.plot(x_plt, y1_RGB_test, color ='b')
plt.plot(x_plt, y_pred, color ='k')
#plt.xlim([50, 75])
plt.show()
```

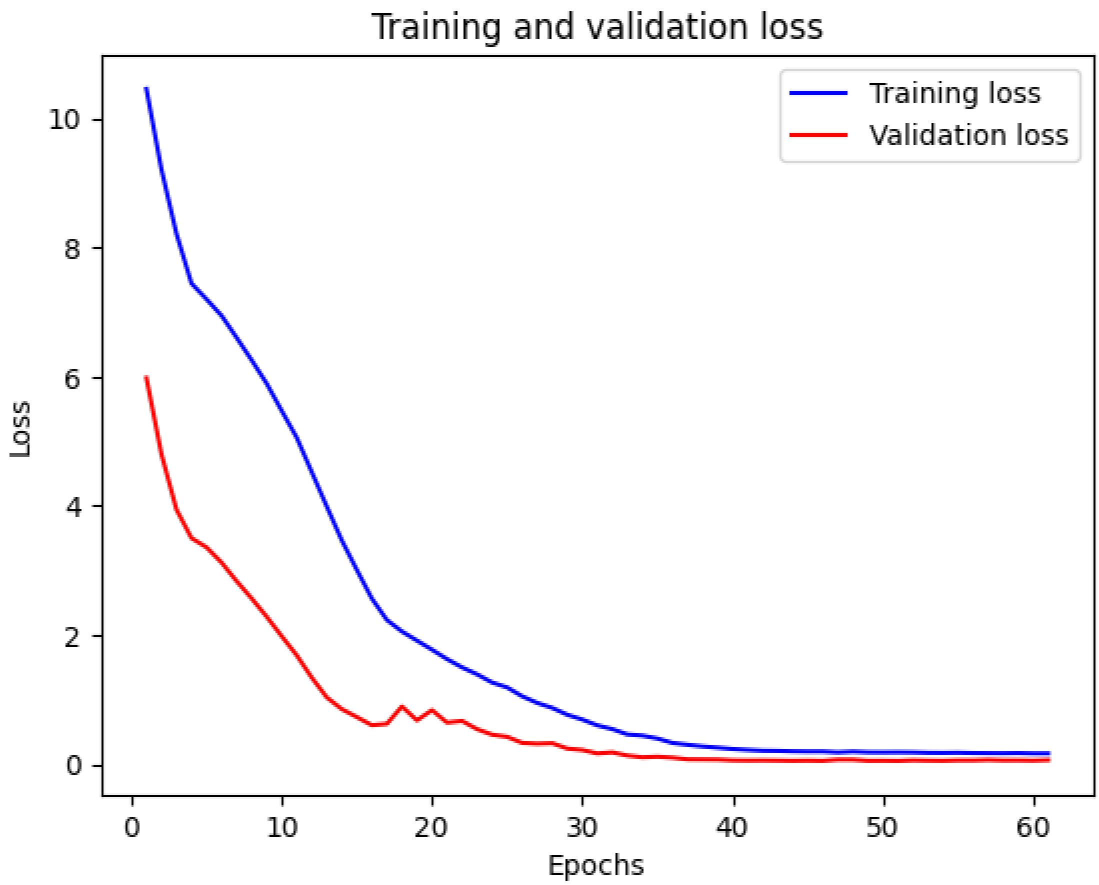
```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
{'max_depth': 1, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 40
0}
RFR_HSV - R^2            : 0.49837507872254505
RFR_HSV - R^2 (optimized): 0.53869782980641
RFR_HSV - MSE            : 1.541195622382249
RFR_HSV - MSE (optimized): 1.4173077435771042
RFR_HSV - MAE            : 1.1028965720838613
RFR_HSV - MAE (optimized): 1.063551133656494
```



In [26]:
```
# Dense Neural Network (DNN) - RGB

# Create the model
model = Sequential()
model.add(Dense(32, input_dim=X_RGB_train.shape[1], activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(1))

# Compile the model
model.compile(
    loss='mean_squared_error',
    optimizer='adam',
    metrics=['mae', 'mse', r_square]
)

# Fit the model to the training data
es = EarlyStopping(
    monitor='val_loss',
    mode='min',
    verbose=1,
    patience=10
)

history = model.fit(
    X_RGB_train,
    y1_RGB_train,
    epochs=1000,
    batch_size=32,
    validation_data=(X_RGB_test, y1_RGB_test),
    callbacks=[es],
```

```
    verbose=0
)

# Evaluate the model on the test data
score = model.evaluate(X_RGB_test, y1_RGB_test)
mse_DNN_RGB = score[1]
mae_DNN_RGB = score[2]
score_DNN_RGB = score[3]

# Print the results
print("DNN_RGB - R^2:", score_DNN_RGB)
print("DNN_RGB - MSE:", mse_DNN_RGB)
print("DNN_RGB - MAE", mae_DNN_RGB)

# Get the training and validation loss values
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Get the number of epochs
epochs = range(1, len(train_loss) + 1)

# Plot the training and validation loss
plt.plot(epochs, train_loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
Epoch 61: early stopping
2/2 [==============================] - 0s 3ms/step - loss: 0.0632 - mae: 0.1975 -
mse: 0.0632 - r_square: 0.9815
DNN_RGB - R^2: 0.9814810752868652
DNN_RGB - MSE: 0.19751287996768951
DNN_RGB - MAE 0.06321351230144501
```

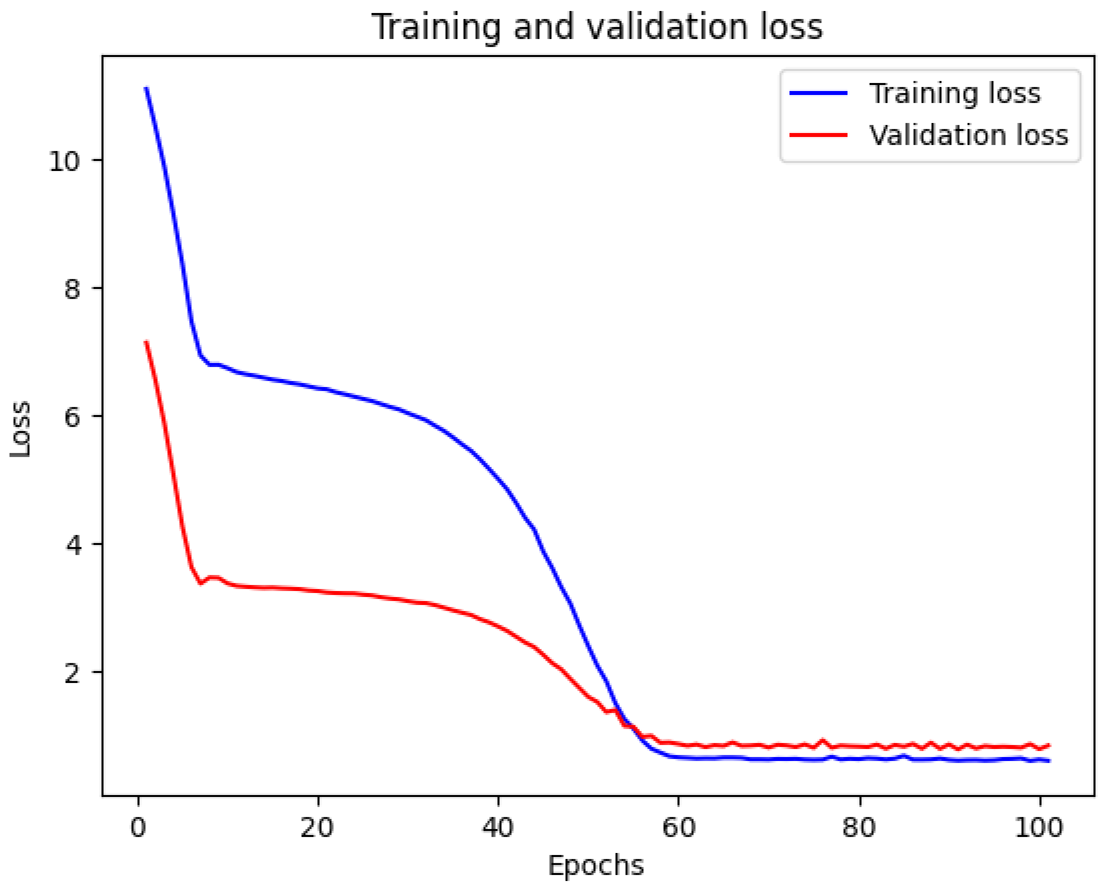


In [27]:

```
# Dense Neural Network (DNN) - HSV

# Create the model
model = Sequential()
model.add(Dense(32, input_dim=X_HSV_train.shape[1], activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(1))

# Compile the model
model.compile(
    loss='mean_squared_error',
    optimizer='adam',
    metrics=['mae', 'mse', r_square]
)

# Fit the model to the training data
es = EarlyStopping(
    monitor='val_loss',
    mode='min',
    verbose=1,
    patience=10
)

history = model.fit(
    X_HSV_train,
    y1_HSV_train,
    epochs=1000,
```

```
    batch_size=32,
    validation_data=(X_HSV_test, y1_HSV_test),
    callbacks=[es],
    verbose=0
)

# Evaluate the model on the test data
score = model.evaluate(X_HSV_test, y1_HSV_test)
mse_DNN_HSV = score[1]
mae_DNN_HSV = score[2]
score_DNN_HSV = score[3]

# Print the results
print("DNN_HSV - R^2:", score_DNN_HSV)
print("DNN_HSV - MSE:", mse_DNN_HSV)
print("DNN_HSV - MAE", mae_DNN_HSV)

# Get the training and validation loss values
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Get the number of epochs
epochs = range(1, len(train_loss) + 1)

# Plot the training and validation loss
plt.plot(epochs, train_loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

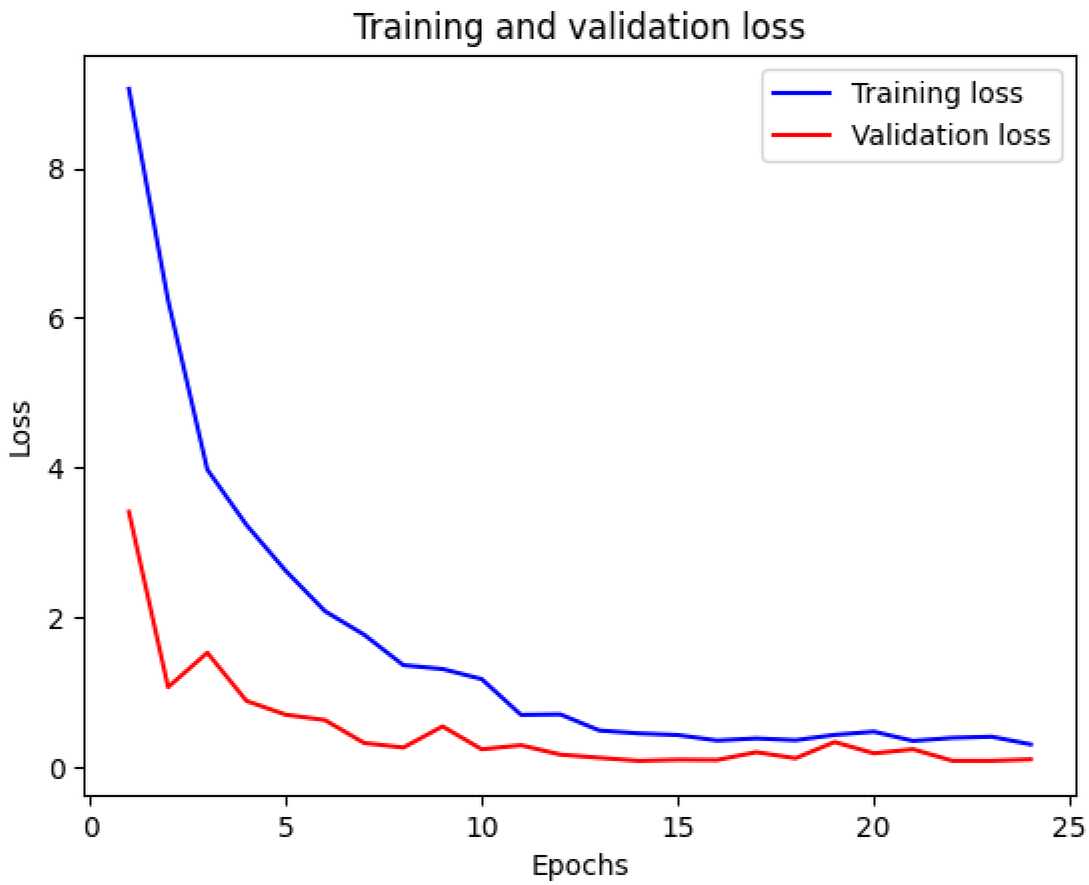
```
Epoch 101: early stopping
2/2 [==============================] - 0s 2ms/step - loss: 0.8329 - mae: 0.7608 -
mse: 0.8329 - r_square: 0.6851
DNN_HSV - R^2: 0.6851169466972351
DNN_HSV - MSE: 0.7607755064964294
DNN_HSV - MAE 0.8329450488090515
```



In [28]:
```
# Convolutionary Neural Network (CNN) - RGB

# Define model architecture
model = tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, padding='same', activat
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same', activat
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=128, activation='relu'))
model.add(tf.keras.layers.Dense(units=1, activation='linear'))

# Compile model
model.compile(
    loss='mean_squared_error',
    optimizer='adam',
    metrics=['mae', 'mse', r_square]
)

# Train model
es = EarlyStopping(
    monitor='val_loss',
    mode='min',
    verbose=1,
    patience=10
)

history = model.fit(
```

```
    X_RGBimage_train,
    y1_RGBimage_train,
    epochs=1000,
    batch_size=32, validation_data=(X_RGBimage_test, y1_RGBimage_test),
    callbacks=[es],
    verbose=0
)

# Evaluate the model on the test data
score = model.evaluate(X_RGBimage_test, y1_RGBimage_test)
mse_CNN_RGB = score[1]
mae_CNN_RGB = score[2]
score_CNN_RGB = score[3]

# Get the training and validation loss values
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Get the number of epochs
epochs = range(1, len(train_loss) + 1)

# Plot the training and validation loss
plt.plot(epochs, train_loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

# Print the results
print("CNN_RGB - R^2:", score_CNN_RGB)
print('CNN_RGB - MSE:', mse_CNN_RGB)
print('CNN_RGB - MAE:', mae_CNN_RGB)
```

```
Epoch 24: early stopping
2/2 [==============================] - 0s 5ms/step - loss: 0.1033 - mae: 0.2348 -
mse: 0.1033 - r_square: 0.9681
```

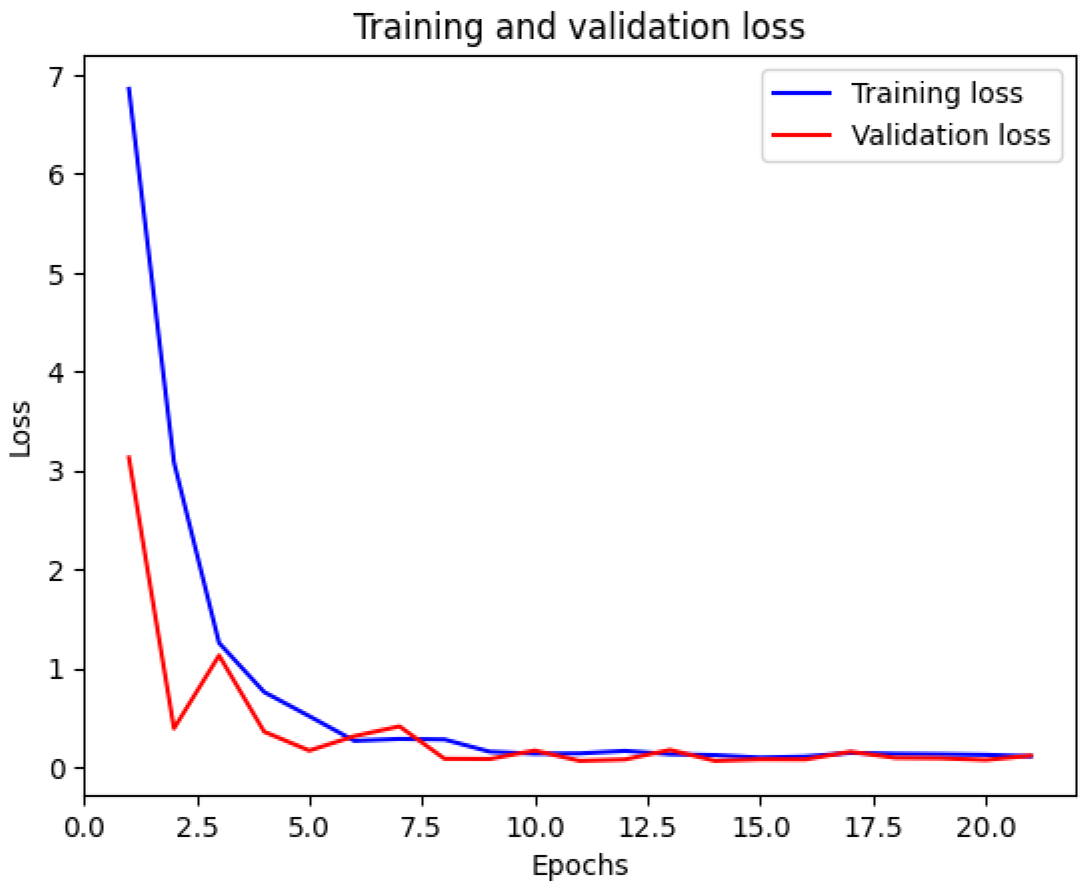



```
CNN_RGB - R^2: 0.9681246876716614
CNN_RGB - MSE: 0.2347865253686905
CNN_RGB - MAE: 0.10327482968568802
```

In [29]:
```
# Convolutionary Neural Network (CNN) - HSV

# Define model architecture
model = tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, padding='same', activat
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same', activat
model.add(tf.keras.layers.MaxPooling2D(pool_size=2))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=128, activation='relu'))
model.add(tf.keras.layers.Dense(units=1, activation='linear'))

# Compile model
model.compile(
    loss='mean_squared_error',
    optimizer='adam',
    metrics=['mae', 'mse', r_square]
)

# Train model
es = EarlyStopping(
    monitor='val_loss',
    mode='min',
    verbose=1,
    patience=10
```

```
)

history = model.fit(
    X_HSVimage_train,
    y1_HSVimage_train,
    epochs=1000,
    batch_size=32,
    validation_data=(X_HSVimage_test, y1_HSVimage_test),
    callbacks=[es],
    verbose=0
)

# Evaluate the model on the test data
score = model.evaluate(X_HSVimage_test, y1_HSVimage_test)
mse_CNN_HSV = score[1]
mae_CNN_HSV = score[2]
score_CNN_HSV = score[3]

# Get the training and validation loss values
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Get the number of epochs
epochs = range(1, len(train_loss) + 1)

# Plot the training and validation loss
plt.plot(epochs, train_loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

# Print the results
print("CNN_HSV - R^2:", score_CNN_HSV)
print('CNN_HSV - MSE:', mse_CNN_HSV)
print('CNN_HSV - MAE:', mae_CNN_HSV)
```

```
Epoch 21: early stopping
2/2 [==============================] - 0s 4ms/step - loss: 0.1178 - mae: 0.2579 - mse: 0.1178 - r_square: 0.9641
```




```
CNN_HSV - R^2: 0.9640809297561646
CNN_HSV - MSE: 0.25790923833847046
CNN_HSV - MAE: 0.11778333783149719
```

In [31]:

```
# Summary

# Summary for RGB
print("\x1B[4m" + 'RGB' + "\x1B[0m")
print("\x1B[4m" + 'R-square, Mean-squared Error, Mean Absolute Error' + "\x1B[0m")
print('LR : "%.4f", "%.4f", "%.4f"' % (score_LR_RGB, mse_LR_RGB, mae_LR_RGB))
print('SVR: "%.4f", "%.4f", "%.4f"' % (score_SVR_RGB, mse_SVR_RGB, mae_SVR_RGB))
print('GBR: "%.4f", "%.4f", "%.4f"' % (score_GBR_RGB, mse_GBR_RGB, mae_GBR_RGB))
print('RFR: "%.4f", "%.4f", "%.4f"' % (score_RFR_RGB, mse_RFR_RGB, mae_RFR_RGB))
print('DNN: "%.4f", "%.4f", "%.4f"' % (score_DNN_RGB, mse_DNN_RGB, mae_DNN_RGB))
print('CNN: "%.4f", "%.4f", "%.4f"' % (score_CNN_RGB, mse_CNN_RGB, mae_CNN_RGB))

# Summary for HSV
print("\x1B[4m" + '\nHSV' + "\x1B[0m")
print("\x1B[4m" + 'R-square, Mean-squared Error, Mean Absolute Error' + "\x1B[0m")
print('LR : "%.4f", "%.4f", "%.4f"' % (score_LR_HSV, mse_LR_HSV, mae_LR_HSV))
print('SVR: "%.4f", "%.4f", "%.4f"' % (score_SVR_HSV, mse_SVR_HSV, mae_SVR_HSV))
print('GBR: "%.4f", "%.4f", "%.4f"' % (score_GBR_HSV, mse_GBR_HSV, mae_GBR_HSV))
print('RFR: "%.4f", "%.4f", "%.4f"' % (score_RFR_HSV, mse_RFR_HSV, mae_RFR_HSV))
print('DNN: "%.4f", "%.4f", "%.4f"' % (score_DNN_HSV, mse_DNN_HSV, mae_DNN_HSV))
print('CNN: "%.4f", "%.4f", "%.4f"' % (score_CNN_HSV, mse_CNN_HSV, mae_CNN_HSV))
```

```
RGB
R-square, Mean-squared Error, Mean Absolute Error
LR : "0.9264", "0.2262", "0.3643"
SVR: "0.9736", "0.0812", "0.2043"
GBR: "0.9924", "0.0232", "0.1042"
RFR: "0.9185", "0.2503", "0.4159"
DNN: "0.9815", "0.1975", "0.0632"
CNN: "0.9681", "0.2348", "0.1033"

HSV
R-square, Mean-squared Error, Mean Absolute Error
LR : "0.9448", "0.1697", "0.3242"
SVR: "0.7405", "0.7974", "0.7309"
GBR: "0.9882", "0.0364", "0.1232"
RFR: "0.5387", "1.4173", "1.0636"
DNN: "0.6851", "0.7608", "0.8329"
CNN: "0.9641", "0.2579", "0.1178"
```

In [ ]: