REGULARIZATION

In Machine Learning, we train our data and let it learn from it. When fitting our data points, our goal is to find the best fit when it can find all necessary patterns in our data and avoid random data points and unnecessary patterns called Noise. While training a machine learning model, the model can easily be overfitted or under fitted. To avoid this, we use regularization in machine learning to properly fit a model onto our test set.

Regularization: This is the technique that helps in avoiding overfitting and also it increasese model interpretability. This is a form of regression, that constrains / regularizes or shrinks the coefficient estimates towards zero. In addition, this technique discourages learning a more complex or flexible model, to avoid the risk of overfitting. Further, Regularization refers to techniques that are used to calibrate machine learning models in order to minimize the adjusted loss function and prevent overfitting or underfitting.

Overfitted Model: The model is not able to predict the output or target column for the unseen data by introducing noise in the output. When a Machine learning model tries to learn from the details along with the noise in the data and tries to fit each data point on the curve.

Underfitting model: When a Machine learning model can neither learn the relationship between variables in the testing data nor predict or classify a new data point.

"Noise" - this means those data points in the dataset which don't really represent the true properties of your data, but only due to a random chance.

Types of Regularization Techniques

1. Ridge Regression

Ridge Regression is also known as L-2 norm. It is one of the Regularization technique that is used to avoid overfitting or underfitting models by adding the penalty equivalent to the sum of the squares of the magnitude of coefficients. By changing the values of the penalty function, we are controlling the penalty term. The higher the penalty, it reduces the magnitude of coefficients. This technique shrinks the estimated coefficients towards zero. It seeks coefficient estimates that fit the data well, by making the RSS small.

1. Lasso Regression

Application on Dataset

We will use dataset that describes First year students GPA. The data set contains information about 219 randomly selected first-year students at a midwestern college. This dataset contains variables such as GPA(first year GPA), HSGPA(high school GPA), SATV, SATM, Male(an indicator variable for Male vs Female), HU, SS, White(an indicator variable for white student vs non white), FirstGen(an indicator variable for first-generation students), and CollegeBound. We will be using GPA as our response veariable and other variables as predictors. Our main goals are;

1. Pull off some of the data to use as training data and some to use as test data. Set up a grid for λ (but here we will use alpha instead).
2. Fit a ridge regression to the training data, then use cross-validation to choose the optimal value of λ. For this value of λ, calculate the test MSE on our test (validation) set. Record this test MSE, the value of λ, and the coefficients from the fitted ridge regression. Are any of the estimated coefficients 0? Is this what you would expect?
3. Repeat the previous bullet point for lasso regression.

Note: Lower RMSE and higher R-Squared values are indicative of a good model.

In [89]:
```
# importing libraries
%matplotlib inline

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt


from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
```

In [90]:
```
# Loading dataset
GPA = pd.read_csv('FirstYearGPA.csv')
```

In [91]:
```
# Looking at the first 5 column of the FirstYearGPA dataset
GPA.head()
```

Out[91]:

	GPA	HSGPA	SATV	SATM	Male	HU	SS	FirstGen	White	CollegeBound
0	3.06	3.83	680	770	1	3.0	9.0	1	1	1
1	4.15	4.00	740	720	0	9.0	3.0	0	1	1
2	3.41	3.70	640	570	0	16.0	13.0	0	0	1
3	3.21	3.51	740	700	0	22.0	0.0	0	1	1
4	3.48	3.83	610	610	0	30.5	1.5	0	1	1

In [92]:
```
# Preparing dataset
target_column = ['GPA']
predictors = list(set(list(GPA.columns))-set(target_column))

# Normalizing by scaling the predictors between 0 and 1. This is done when the
units of the varibalrs differ significantly and may influence the modeling pro
cess.
#GPA[predictors] = GPA[predictors]/GPA[predictors].max()
```

We will build our model on the training set and evaluate its performance on the test set. This is called the holdout-validation approach for evaluating model performance.

X : independent variables (Other variables except GPA)

y : dependent variables (GPA)

The output shows that the shape of the training set has 153 observation of 9 variables and test set has 66 observation of 9 variables.

In [93]:
```
# Creating the training and Test datasets
X = GPA[predictors].values
y = GPA[target_column].values

# or could use
# X = GPA.iloc[:, :-1].values
# y = GPA.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, rand
om_state=40)

print(X_train.shape)
print(X_test.shape)

def MAPE(Y_actual,Y_Predicted):
    mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
    return mape
```

```
(153, 9)
(66, 9)
```

Linear Regression

The simplest form of regression is linear regression, which assumes that the predictors have a linear relationship with the target variable. The input variables are assumed to have a Gaussian distribution and are not correlated with each other (a problem called multi-collinearity).

The linear regression equation can be expressed in the following form: y = a_1x_1 + a_2x_2 + a_3x_3 + ... + a_x_n + b, where y is the target variable, x1, x2, x3, ... xn are the features, a1, a2, a3, ... an are the coefficients, and b is the parameter of the model.

The parameters a and b in the model are selected through the ordinary least squares (OLS) method. This method works by minimizing the sum of squares of residuals (actual value - predicted value).

Linear regression algorithm works by selecting coefficients for each independent variable that minimizes a loss function. However, if the coefficients are large, they can lead to over-fitting on the training dataset, and such a model will not generalize well on the unseen test data. To overcome this shortcoming, we will do regularization as shown below, which penalizes large coefficients.

In [94]:
```
# Multiple Linear regression model
lm = LinearRegression()
lm.fit(X_train, y_train)
```

Out[94]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
```

In [95]:
```
# Genereating prediction on test set
lm_train_pred = lm.predict(X_train)
```

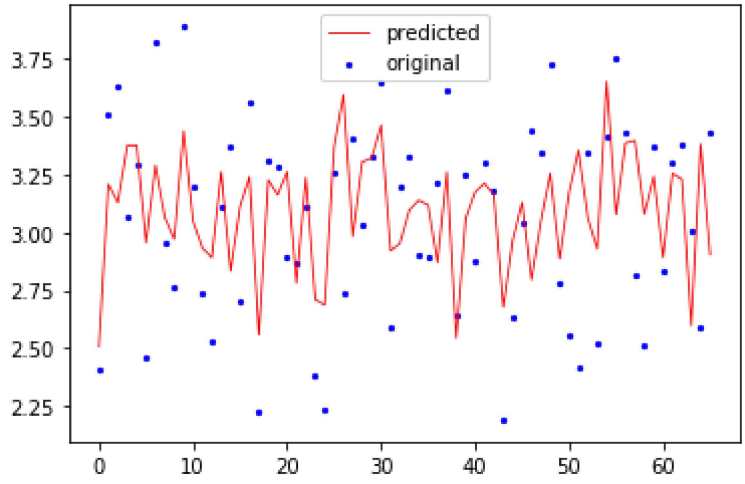
In [96]:
```
# prints the evaluation metrics (RMSE and R-squared) on the train set
print('RMSE of train set:            ',np.sqrt(mean_squared_error(y_train,lm_t
rain_pred)))
print('R-Squared of train set:      ',r2_score(y_train, lm_train_pred)*100,'%'
)

# prints the evaluation metrics (RMSE and R-squared) on the test set
lm_test_pred = lm.predict(X_test)
print('RMSE of est set:              ',np.sqrt(mean_squared_error(y_test,lm_te
st_pred)))
print('R-Squared of test set:       ',r2_score(y_test, lm_test_pred)*100,"%")

Lm_MAPE = MAPE(y_test,lm_test_pred)
print("MAPE value:                  ",Lm_MAPE)
Accuracy = 100 - Lm_MAPE
print('Accuracy of Linear Regression: {:0.2f}%.'.format(Accuracy))
```

```
RMSE of train set:             0.3796918126336791
R-Squared of train set:       37.486548794143516 %
RMSE of est set:               0.37363947388891877
R-Squared of test set:        21.748129673159333 %
MAPE value:                   10.601484148864845
Accuracy of Linear Regression: 89.40%.
```

In [97]:
```
# retrieving the intercept and the coefficients
print(lm.intercept_,lm.coef_)
```

```
[0.52401237] [[ 5.43897320e-05  1.10147628e-02 -8.42162608e-02  5.08814019e-0
1
   1.53005147e-02 -4.95044729e-03 -7.74565719e-02  7.82661088e-04
   1.61712780e-01]]
```

In [98]:
```
x_ax = range(len(X_test))
plt.scatter(x_ax, y_test, s=5, color="blue", label="original")
plt.plot(x_ax, lm_test_pred, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



RIDGE REGRESSION

Ridge Regression is a model that analyses any data that seems to have multicollinearity. In ridge regression, we have a term lambda. This term is denoted by an alpha parameter in the ridge function. When the values of alpha changes then the penalty term is controlled. By using FirstYearGPA dataset, our y value will be the GPA of the student (response variable) and x will be the predictor variables. We expect coefficients that do not contribute to the model to be shrinked by letting them have smaller value.

The main goal is to obtain the mean squared error and observe if the estimated coefficients are 0. The mean Square error will show us the model error; thus, we want our MSE to be as small as possible. Also,we will calculate R-squared which measures the goodness of fit. R-squared measures the strength of the relationship between Ridge Regression model and the predictor variables.

R-squared of 0% represents a model that does not explain any of the variation in the response variable around its mean. The mean of the dependent variable predicts the dependent variable as well as the regression model. While, R-squared of 100% represents a model that explains all the variation in the response variable around its mean. For R-square we would want our result to have higher value.

Here we used different alpha values to check and see which model would give us a smaller RMSE and higher R-squared. Also, our models were normalized to have mean zero to avoid regularization biases of the model away from the data. In addition, we checked for the accuracy of the model.

Ridge Regression Model of Alpha Value 0.01 which is not Normalized.

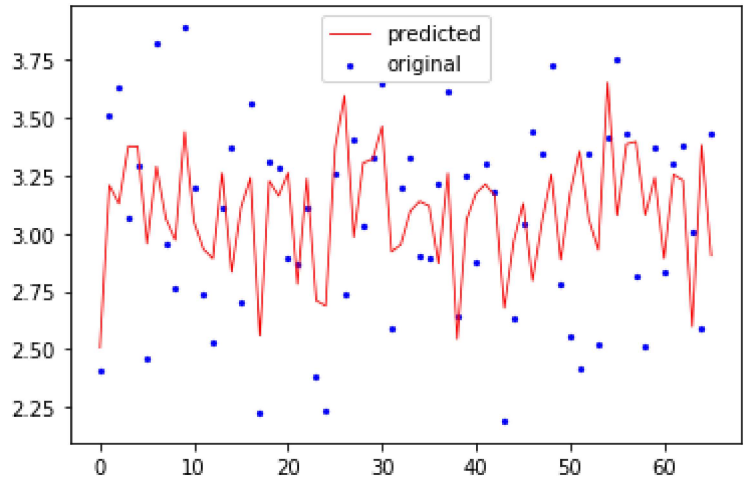
In [99]:
```
# Ridge regression model with an alpha value of 0.01
rr1 = Ridge(alpha=0.01)

# fitting the model to the training data
rr1.fit(X_train, y_train)

# predicting
pred_train_rr1= rr1.predict(X_train)

# prints the evaluation metrics (RMSE and R-squared) on the train set
print('RMSE of train set:       ', np.sqrt(mean_squared_error(y_train,pred_tra
in_rr1)))
print('R-Squared of train set: ',r2_score(y_train, pred_train_rr1)*100,'%')

# prints the evaluation metrics (RMSE and R-squared) on the test set
pred_test_rr1= rr1.predict(X_test)
print('RMSE of test set:        ',np.sqrt(mean_squared_error(y_test,pred_test_
rr1)))
print('R-Squared of test set:  ',r2_score(y_test, pred_test_rr1)*100,'%')

Ridge_MAPE = MAPE(y_test,pred_test_rr1)
print("MAPE value:             ",Ridge_MAPE)
Accuracy = 100 - Ridge_MAPE
print('Accuracy of Ridge Regression: {:0.2f}%.'.format(Accuracy))
```

```
RMSE of train set:        0.37969182630313725
R-Squared of train set:  37.48654429299291 %
RMSE of test set:         0.37363514830697914
R-Squared of test set:   21.749941488780454 %
MAPE value:              10.6015151669463
Accuracy of Ridge Regression: 89.40%.
```

In [100]:
```
# retrieving the intercept and the coefficients
print(rr1.intercept_,rr1.coef_)
```

```
[0.52455127] [[ 5.46101724e-05  1.10174104e-02 -8.42127159e-02  5.08545821e-0
1
   1.53048554e-02 -4.98757225e-03 -7.73468585e-02  7.83065411e-04
   1.61626624e-01]]
```

In [101]:
```
x_ax = range(len(X_test))
plt.scatter(x_ax, y_test, s=5, color="blue", label="original")
plt.plot(x_ax, pred_test_rr1, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



Normalized Ridge Regression Model of Alpha Value 0.01.

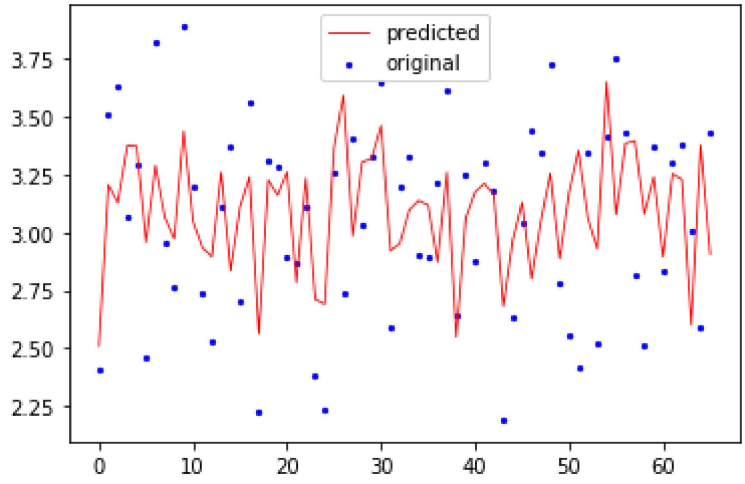
In [102]:
```
# Ridge regression model with an alpha value of 0.01
rr2 = Ridge(alpha=0.01, normalize = True)

# fitting the model to the training data
rr2.fit(X_train, y_train)

# predicting
pred_train_rr2= rr2.predict(X_train)

# prints the evaluation metrics (RMSE and R-squared) on the train set
print('RMSE of train set:          ',np.sqrt(mean_squared_error(y_train,pred_t
rain_rr2)))
print('R-Squared of train set:    ',r2_score(y_train, pred_train_rr2)*100,'%')

# prints the evaluation metrics (RMSE and R-squared) on the test set
pred_test_rr2= rr2.predict(X_test)
print('RMSE of test set:           ',np.sqrt(mean_squared_error(y_test,pred_te
st_rr2)))
print('R-Squared of test set:     ',r2_score(y_test, pred_test_rr2)*100,'%')

Ridge_MAPE = MAPE(y_test,pred_test_rr2)
print("MAPE value:                ",Ridge_MAPE)
Accuracy = 100 - Ridge_MAPE
print('Accuracy of Ridge Regression: {:0.2f}%.'.format(Accuracy))
```

```
RMSE of train set:           0.3796990576307267
R-Squared of train set:     37.4841631009292 %
RMSE of test set:            0.3734184974914102
R-Squared of test set:      21.840661133467687 %
MAPE value:                 10.604035171972166
Accuracy of Ridge Regression: 89.40%.
```

In [103]:
```
# retrieving the intercept and the coefficients
print(rr2.intercept_,rr2.coef_)
```

```
[0.54197673] [[ 6.14694178e-05  1.08384177e-02 -8.44872478e-02  5.04046384e-0
1
   1.51695717e-02 -5.48528571e-03 -7.70159415e-02  7.79675807e-04
   1.60891570e-01]]
```

In [104]:
```
x_ax = range(len(X_test))
plt.scatter(x_ax, y_test, s=5, color="blue", label="original")
plt.plot(x_ax, pred_test_rr2, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



Ridge Regression Model of Alpha Value 1 which is not Normalized

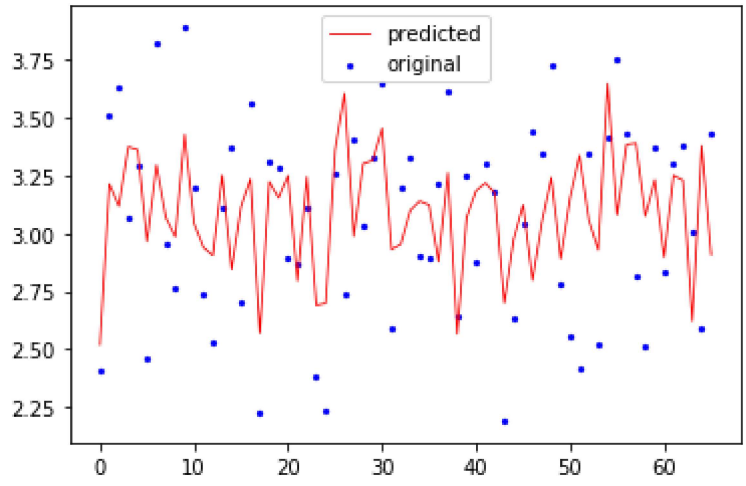
In [105]:
```
# Ridge regression model with an alpha value of 1
rr3 = Ridge(alpha=1)

# fitting the model to the training data
rr3.fit(X_train, y_train)

# predicting
pred_train_rr3= rr3.predict(X_train)

# prints the evaluation metrics (RMSE and R-squared) on the train set
print('RMSE of train set:          ',np.sqrt(mean_squared_error(y_train,pred_t
rain_rr3)))
print('R-Squared of train set:    ',r2_score(y_train, pred_train_rr3)*100,'%')

# prints the evaluation metrics (RMSE and R-squared) on the test set
pred_test_rr3= rr3.predict(X_test)
print('RMSE of test set:           ',np.sqrt(mean_squared_error(y_test,pred_te
st_rr3)))
print('R-Squared of test set:     ',r2_score(y_test, pred_test_rr3)*100,'%')

Ridge_MAPE = MAPE(y_test,pred_test_rr3)
print("MAPE value:                ",Ridge_MAPE)
Accuracy = 100 - Ridge_MAPE
print('Accuracy of Ridge Regression: {:0.2f}%.'.format(Accuracy))
```

```
RMSE of train set:           0.37981403906006417
R-Squared of train set:     37.446294960322355 %
RMSE of test set:            0.3733251036097011
R-Squared of test set:      21.879752347552618 %
MAPE value:                 10.603653039588385
Accuracy of Ridge Regression: 89.40%.
```

In [106]:
```
# retrieving the intercept and the coefficients
print(rr3.intercept_,rr3.coef_)
```

```
[0.57520146] [[ 7.49157196e-05  1.12628953e-02 -8.36278449e-02  4.83453682e-0
1
   1.57071505e-02 -8.37361322e-03 -6.74230654e-02  8.20613339e-04
   1.53539780e-01]]
```

In [107]:
```
x_ax = range(len(X_test))
plt.scatter(x_ax, y_test, s=5, color="blue", label="original")
plt.plot(x_ax, pred_test_rr3, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



Normalize Ridge Regression Model of Alpha Value 1

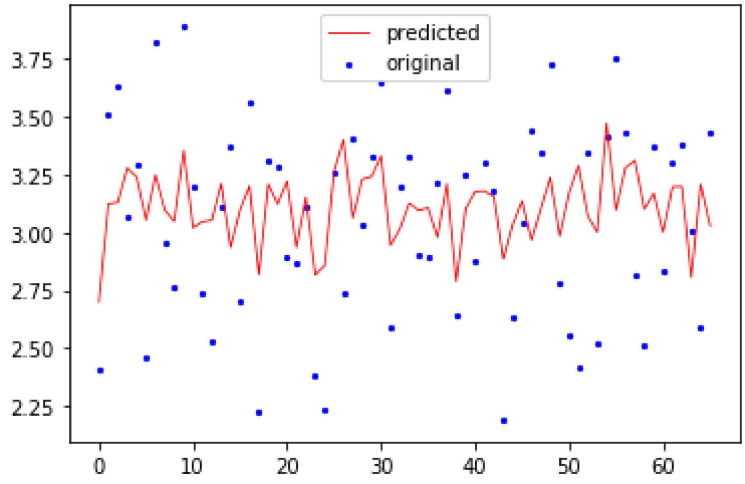
In [108]:
```
# Ridge regression model with an alpha value of 1
rr4 = Ridge(alpha=1, normalize = True)

# fitting the model to the training data
rr4.fit(X_train, y_train)

# predicting
pred_train_rr4= rr4.predict(X_train)

# prints the evaluation metrics (RMSE and R-squared) on the train set
print('RMSE of train set:          ',np.sqrt(mean_squared_error(y_train,pred_t
rain_rr4)))
print('R-Squared of train set:    ',r2_score(y_train, pred_train_rr4)*100,'%')

# prints the evaluation metrics (RMSE and R-squared) on the test set
pred_test_rr4= rr4.predict(X_test)
print('RMSE of test set:           ',np.sqrt(mean_squared_error(y_test,pred_te
st_rr4)))
print('R-Squared of test set:     ',r2_score(y_test, pred_test_rr4)*100,'%')

Ridge_MAPE = MAPE(y_test,pred_test_rr4)
print("MAPE value:                ",Ridge_MAPE)
Accuracy = 100 - Ridge_MAPE
print('Accuracy of Ridge Regression: {:0.2f}%.'.format(Accuracy))
```

```
RMSE of train set:           0.39930755764999093
R-Squared of train set:     30.86052600825987 %
RMSE of test set:            0.37499802465314597
R-Squared of test set:      21.17804853690568 %
MAPE value:                 11.028851789833146
Accuracy of Ridge Regression: 88.97%.
```

In [109]:
```
# retrieving the intercept and the coefficients
print(rr4.intercept_,rr4.coef_)
```

```
[1.55389443] [[ 2.23215997e-04  3.63230351e-03 -6.77800319e-02  2.69336935e-0
1
   8.49064540e-03 -1.24488321e-02 -5.11830432e-02  5.57552118e-04
   1.09144569e-01]]
```

In [110]:
```
x_ax = range(len(X_test))
plt.scatter(x_ax, y_test, s=5, color="blue", label="original")
plt.plot(x_ax, pred_test_rr4, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



Looking at the reults of estimated coefficients, we can see that, as we increase the value of alpha without normalizing, the magnitude of the coefficients decreases, where the values reaches to zero but not absolute zero. On the other hand, if we look at the results of the R-Squared, we see that the value of R-Squared is at higher value when we normalize and increase alpha to 1. By changing the values of alpha, we are basically controlling the penalty term. The higher the values of alpha, bigger is the penalty and therefore the magnitude of coefficients are reduced. Thus, Ridge Regression model with alpha 1 is the best model. Also, we saw that the accuracy is about 89.40%. However, this doesn't mean that it is the best model overall. We will look into Lasso Regression to see if it performs better than Ridge Regression by sending estimated coefficients to zero.

LASSO REGRESSION

Lasso stands for Least Absolute and Selection Operator which is know as the L-1 norm. Lasso regression is used to reduce the complexity of the model. Lasso regression is similar to Ridge regression except that the penalty term incudes the absolute weights instead of a square weights. In this technique, the L1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero which means there is a complete removal of some of the features for model evaluation when the tuning parameter λ is sufficiently large. Therefore, the lasso method also performs Feature selection and is said to yield sparse models.

In Lasso Regression, we will perfom similar analysis as Ridge Regression. Here our main goal is to see if Lasso Regression out perfoms Ridge Regression.

Lasso Regression Model of Alpha Value 0.01

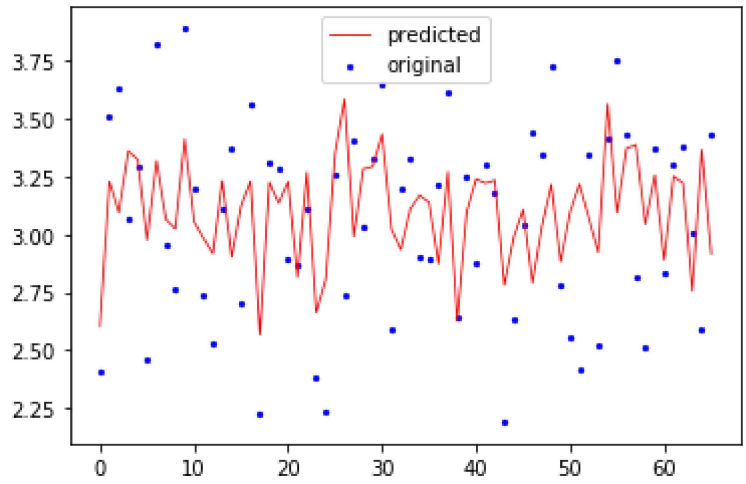
In [111]:
```
# Lasso Regression Model with an alpha value of 0.01
model_lasso1 = Lasso(alpha=0.01)

# fitting the model to the training data
model_lasso1.fit(X_train, y_train)

# predicting
pred_train_lasso1= model_lasso1.predict(X_train)

# prints the evaluation metrics (RMSE and R-squared) on the train set
print('RMSE of train set:           ',np.sqrt(mean_squared_error(y_train,pred_
train_lasso1)))
print('R-Squared of train set:     ',r2_score(y_train, pred_train_lasso1)*100,
'%')

# prints the evaluation metrics (RMSE and R-squared) on the test set
pred_test_lasso1= model_lasso1.predict(X_test)
print('RMSE of test set:            ',np.sqrt(mean_squared_error(y_test,pred_t
est_lasso1)))
print('R-Squared of test set:      ',r2_score(y_test, pred_test_lasso1)*100,
"%")

Lasso_MAPE = MAPE(y_test,pred_test_lasso1)
print("MAPE value:                 ",Lasso_MAPE)
Accuracy = 100 - Lasso_MAPE
print('Accuracy of Lasso Regression: {:0.2f}%.'.format(Accuracy))
```

```
RMSE of train set:            0.38299609568292753
R-Squared of train set:      36.393762814253115 %
RMSE of test set:             0.3700547692811224
R-Squared of test set:       23.242426989805555 %
MAPE value:                  13.560080032088507
Accuracy of Lasso Regression: 86.44%.
```

In [112]:
```
# retrieving the intercept and the coefficients
print(model_lasso1.intercept_,model_lasso1.coef_)
```

```
[0.57499914] [ 8.11128793e-05  1.18657207e-02 -0.00000000e+00  4.36250795e-01
  1.70581867e-02 -0.00000000e+00 -0.00000000e+00  9.77781015e-04
  9.68056334e-02]
```

In [113]:
```
x_ax = range(len(X_test))
plt.scatter(x_ax, y_test, s=5, color="blue", label="original")
plt.plot(x_ax, pred_test_lasso1, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



Normalize Lasso Regression Model of Alpha Value 0.01

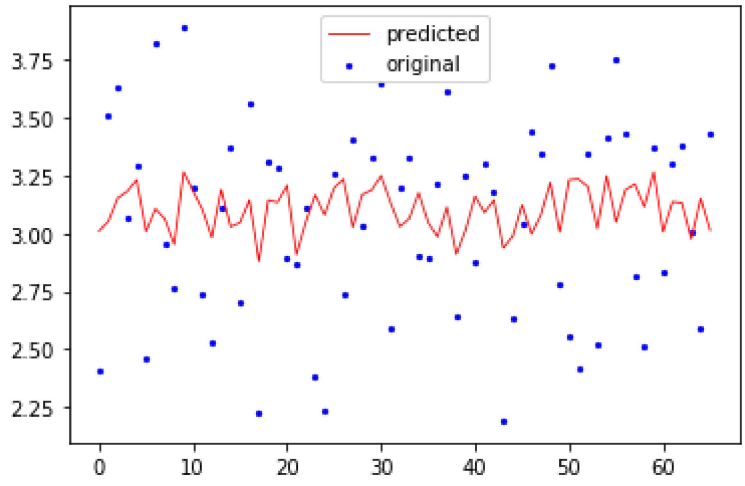
In [114]:
```
# Lasso Regression Model with an alpha value of 1
model_lasso2 = Lasso(alpha=0.01, normalize = True)

# fitting the model to the training data
model_lasso2.fit(X_train, y_train)

# predicting
pred_train_lasso2= model_lasso2.predict(X_train)

# prints the evaluation metrics (RMSE and R-squared) on the train set
print('RMSE of train set:           ',np.sqrt(mean_squared_error(y_train,pred_
train_lasso2)))
print('R-Squared of train set:     ',r2_score(y_train, pred_train_lasso2)*100,
"%")

# prints the evaluation metrics (RMSE and R-squared) on the test set
pred_test_lasso2= model_lasso2.predict(X_test)
print('RMSE of test set:            ',np.sqrt(mean_squared_error(y_test,pred_t
est_lasso2)))
print('R-Squared of test set:       ',r2_score(y_test, pred_test_lasso2)*100,
"%")

Lasso_MAPE = MAPE(y_test,pred_test_lasso2)
print("MAPE value:                 ",Lasso_MAPE)
Accuracy = 100 - Lasso_MAPE
print('Accuracy of Lasso Regression: {:0.2f}%.'.format(Accuracy))
```

```
RMSE of train set:            0.4324456084360554
R-Squared of train set:      18.908750344447668 %
RMSE of test set:             0.40271612478685065
R-Squared of test set:        9.09510676617763 %
MAPE value:                  12.670433780859417
Accuracy of Lasso Regression: 87.33%.
```

In [115]:
```
# retrieving the intercept and the coefficients
print(model_lasso2.intercept_,model_lasso2.coef_)
```

```
[2.1003695] [ 0.00000000e+00  0.00000000e+00 -0.00000000e+00  2.84245925e-01
  1.85076277e-03  0.00000000e+00 -0.00000000e+00  7.79765908e-06
  0.00000000e+00]
```

In [116]:
```
x_ax = range(len(X_test))
plt.scatter(x_ax, y_test, s=5, color="blue", label="original")
plt.plot(x_ax, pred_test_lasso2, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



Lasso Regression Model of Alpha Value 1

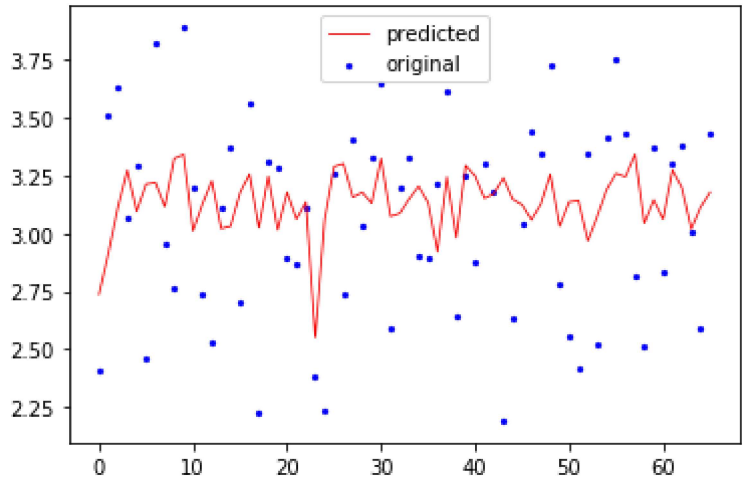
In [117]:
```
# Lasso Regression Model with an alpha value of 1
model_lasso3 = Lasso(alpha=1)

# fitting the model to the training data
model_lasso3.fit(X_train, y_train)

# predicting
pred_train_lasso3= model_lasso3.predict(X_train)

# prints the evaluation metrics (RMSE and R-squared) on the train set
print('RMSE of train set:           ',np.sqrt(mean_squared_error(y_train,pred_
train_lasso3)))
print('R-Squared of train set:      ',r2_score(y_train, pred_train_lasso3)*100
,"%")

# prints the evaluation metrics (RMSE and R-squared) on the test set
pred_test_lasso3= model_lasso3.predict(X_test)
print('RMSE of test set:            ',np.sqrt(mean_squared_error(y_test,pred_t
est_lasso3)))
print('R-Squared of test set:      ',r2_score(y_test, pred_test_lasso3)*100,
"%")

Lasso_MAPE = MAPE(y_test,pred_test_lasso3)
print("MAPE value:                 ",Lasso_MAPE)
Accuracy = 100 - Lasso_MAPE
print('Accuracy of Lasso Regression: {:0.2f}%.'.format(Accuracy))
```

```
RMSE of train set:            0.4552687777818668
R-Squared of train set:       10.12337700784528 %
RMSE of test set:             0.40808326361806885
R-Squared of test set:       6.655917700583602 %
MAPE value:                  12.991568593652921
Accuracy of Lasso Regression: 87.01%.
```

In [118]:
```
# retrieving the intercept and the coefficients
print(model_lasso3.intercept_,model_lasso3.coef_)
```

```
[2.12500489] [ 0.          0.         -0.          0.          0.00071261 -0.
 -0.          0.0016298   0.        ]
```

In [119]:
```
x_ax = range(len(X_test))
plt.scatter(x_ax, y_test, s=5, color="blue", label="original")
plt.plot(x_ax, pred_test_lasso3, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



Normalize Lasso Regression Model of Alpha Value 1

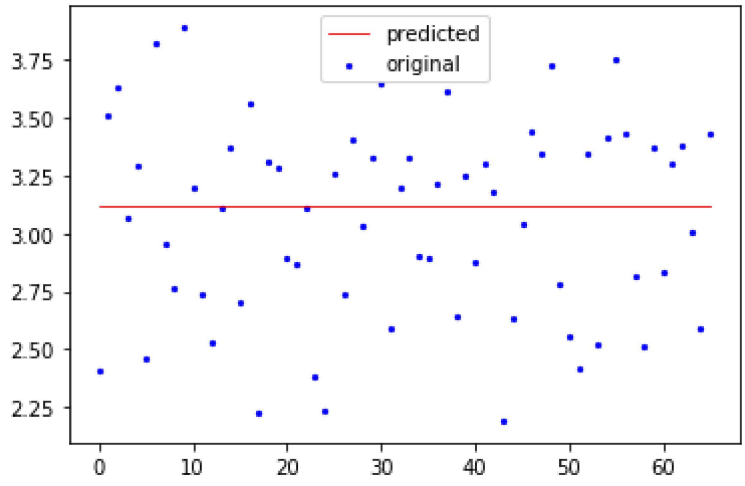
In [120]:
```
# Lasso Regression Model with an alpha value of 1
model_lasso4 = Lasso(alpha=1, normalize = True)

# fitting the model to the training data
model_lasso4.fit(X_train, y_train)

# predicting
pred_train_lasso4= model_lasso4.predict(X_train)

# prints the evaluation metrics (RMSE and R-squared) on the train set
print('RMSE of train set:           ',np.sqrt(mean_squared_error(y_train,pred_
train_lasso4)))
print('R-Squared of train set:      ',r2_score(y_train, pred_train_lasso4)*100
,"%")

# prints the evaluation metrics (RMSE and R-squared) on the test set
pred_test_lasso4= model_lasso4.predict(X_test)
print('RMSE of test set:            ',np.sqrt(mean_squared_error(y_test,pred_t
est_lasso4)))
print('R-Squared of test set:      ',r2_score(y_test, pred_test_lasso4)*100,
"%")

Lasso_MAPE = MAPE(y_test,pred_test_lasso4)
print("MAPE value:                 ",Lasso_MAPE)
Accuracy = 100 - Lasso_MAPE
print('Accuracy of Lasso Regression: {:0.2f}%.'.format(Accuracy))
```

```
RMSE of train set:            0.4802247007058386
R-Squared of train set:       0.0 %
RMSE of test set:             0.4268392591735874
R-Squared of test set:       -2.121676937881878 %
MAPE value:                  12.550515815039695
Accuracy of Lasso Regression: 87.45%.
```

In [121]:
```
# retrieving the intercept and the coefficients
print(model_lasso4.intercept_,model_lasso4.coef_)
```

```
[3.11470588] [ 0.  0. -0.  0.  0. -0. -0.  0.  0.]
```

In [122]:
```
x_ax = range(len(X_test))
plt.scatter(x_ax, y_test, s=5, color="blue", label="original")
plt.plot(x_ax, pred_test_lasso4, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



We can see that in Ridge Regression as we increased the value of alpha, coefficients were approaching towards zero, but in the case of Lasso Regression, even at smaller alpha's, our coefficients are reducing to absolute zeroes. Therefore, lasso selects some features while reduces the coefficients of others to zero. This property is known as feature selection and which is absent in Ridge Regression.

On the other hand, we can observe that our R-Squared are smaller and RSME are larger than in Ridge Regression. Overall, I observe that Normalized Lasso Regression of alpha 1 has the best estimates for the coefficients where they are reduced to zero. Since this is what Lasso Regression does; thus, it is the best model.

Based on RMSE and R-Squared Ridge Regression of alpha 1 is the best model since RSME is small enough and R-Squared is higher.

Based on estimated coefficients, Normalized Lasso Regression of alpha 1 is the best model since all it's estimated coefficients were reduced to zero.