STOCHASTIC GRADIENT DESCENT (SGD)

$$\theta^{j+1} := \theta^j - \eta(\epsilon_j)x_j$$

θ^{j+1} : Next location θ^j : Current location η : Learning coefficient ϵ_j : Error of J data point x_j : The value of the J th feature

In [36]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split,GridSearchCV,cross_val_score,KF
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.linear_model import SGDRegressor,LinearRegression

from yellowbrick.model_selection import CVScores
```

In [8]:

```
df = pd.read_csv(r'D:\githubProjects\Machine-Learning\Supervised Learning\Model Regu
df.head()
```

Out[8]:

	Unnamed: 0	price	rooms	m2	floor	age
0	0	475	1	40	0	6
1	1	475	1	55	0	5
2	2	450	1	50	0	7
3	3	450	1	55	1	6
4	4	475	1	45	2	7

Out[8]:

	Unnamed: 0	price	rooms	m2	floor	age
0	0	475	1	40	0	6
1	1	475	1	55	0	5
2	2	450	1	50	0	7
3	3	450	1	55	1	6
4	4	475	1	45	2	7

In [9]:

```
X = df.drop(['Unnamed: 0','price'],axis=1)
y = df['price']
```

SGD is sensitive to the scale of the properties and target variables, so we apply standard scaling to the values of all variables.

In [10]:
```
from sklearn.preprocessing import MinMaxScaler

X_scaler = MinMaxScaler()
y_scaler = MinMaxScaler()

X = X_scaler.fit_transform(X)

y = y_scaler.fit_transform(y.values.reshape(-1,1))
```

HOLD-OUT

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=42,test_size=0.
```

In [12]:
```
sgd_reg = SGDRegressor(random_state=42,penalty='elasticnet')
```

For the penalty rule in SGD optimization, we used the ElasticNet regression, which evaluates the L1 and L2 rules together. In this case, we need to determine how much weight we will make for L1 and L2.

Hyperparameter Optimization

In [ ]:
```
params = {'alpha' : [0.0001,0.001,0.01],
          'eta0' : [0.0001,0.001,0.01,0.1],
          'l1_ratio' : [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1],
          'learning_rate' : ['constant','optimal','inscaling','adaptive'],
          'max_iter' : [50,100,500,1000]}

sgd_Gs = GridSearchCV(sgd_reg,params,n_jobs=-1,scoring='r2',cv=5).fit(X_train,y_trai
```

In [14]:
```
sgd_Gs.best_params_
```

Out[14]:
```
{'alpha': 0.001,
 'eta0': 0.1,
 'l1_ratio': 0.7,
 'learning_rate': 'adaptive',
 'max_iter': 50}
```

If l1_ratio = 1, the SGD model would be built according to Lasso(L1), if l1_ratio = 0, it would be built according to Ridge(L2). The l1_ratio being 0.7 means that the linear regression model will be evaluated according to the ElasticNet regression.

In [15]:
```
sgd_Gs.best_estimator_
```

Out[15]:
```
SGDRegressor(alpha=0.001, eta0=0.1, l1_ratio=0.7, learning_rate='adaptive',
             max_iter=50, penalty='elasticnet', random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

R2 of the optimum model

In [16]:
```
sgd_Gs.best_score_
```

Out[16]:
```
0.7083899182622208
```

Final Model

In [ ]:
```
sgd_final = sgd_reg.set_params(**sgd_Gs.best_params_).fit(X_train,y_train)
```

Cross-Validate

In [ ]:
```
scores = cross_val_score(sgd_final,X_train,y_train,cv=5)
```

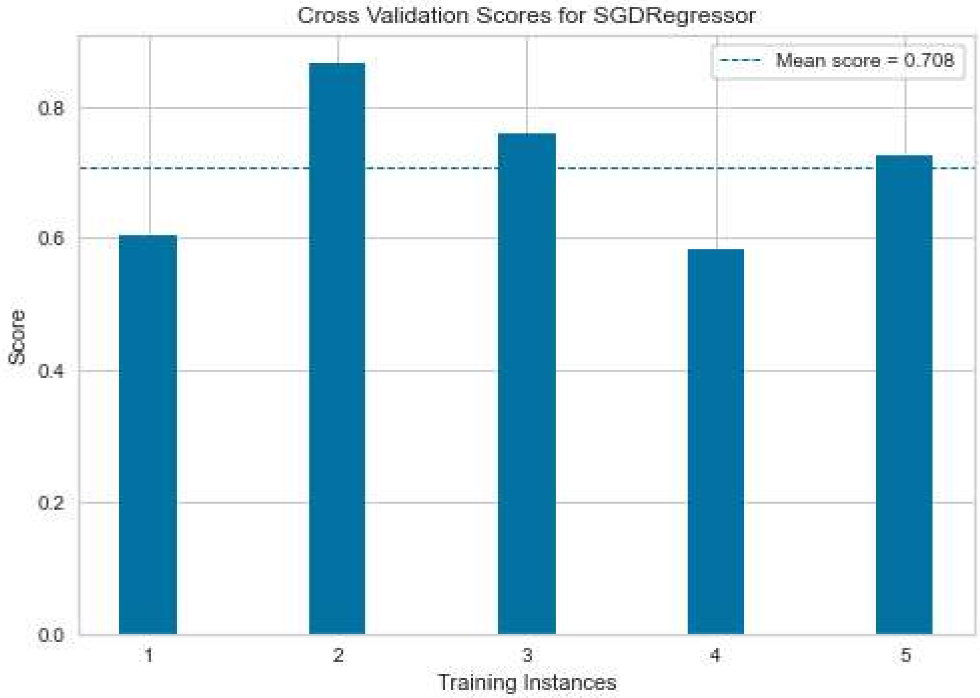
In [19]:
```
scores
```

Out[19]:
```
array([0.60528962, 0.86596214, 0.76049417, 0.58436574, 0.72583792])
```

visualization of cross-validation results

In [21]:
```
cv= KFold(n_splits=5)
model = sgd_final
visualizer = CVScores(model,cv=cv,scoring='r2').fit(X_train,y_train)
visualizer.show()
```

```
C:\ProgramData\Anaconda3\envs\batuhan\lib\site-packages\sklearn\utils\validation.py:
1111: DataConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\envs\batuhan\lib\site-packages\sklearn\utils\validation.py:
1111: DataConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\envs\batuhan\lib\site-packages\sklearn\utils\validation.py:
1111: DataConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\envs\batuhan\lib\site-packages\sklearn\utils\validation.py:
1111: DataConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\envs\batuhan\lib\site-packages\sklearn\utils\validation.py:
1111: DataConversionWarning: A column-vector y was passed when a 1d array was expect
ed. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```



Out[21]: <AxesSubplot:title={'center':'Cross Validation Scores for SGDRegressor'}, xlabel='Training Instances', ylabel='Score'>

Estimation parameters coefficients

In [22]:
```
sgd_final.intercept_
```

Out[22]: array([0.01692942])

In [23]:
```
sgd_final.coef_
```

Out[23]: array([0.29368591, 0.16057681, 0.11152484, -0.12767369])

- $\hat{y} = 0.016 + 0.293 * rooms + 0.16 * m_2 + 0.111 * floor - 0.128 * age$

In [25]:
```
print(f'Train R2 :',{sgd_final.score(X_train,y_train)})
print(f'Test R2 :',{sgd_final.score(X_test,y_test)})
```

```
Train R2 : {0.7321784405612313}
Test R2 : {0.6901787126678489}
```

In [29]:
```
df['pred_price'] = sgd_final.predict(X)

df['pred_price'] = y_scaler.inverse_transform(df['pred_price'].values.reshape(-1,1))

df[['price','pred_price']].head()
```

Out[29]:

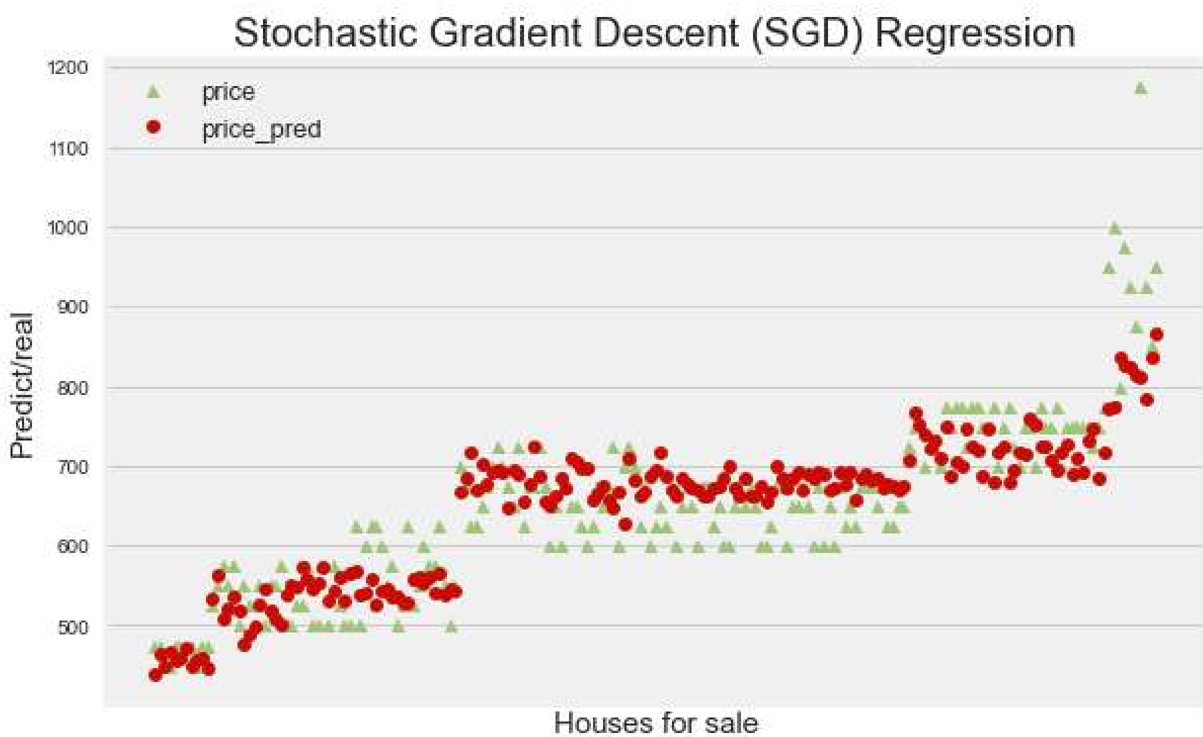
	price	pred_price
0	475	439.132969

	price	pred_price
1	475	463.534165
2	450	448.972417
3	450	467.027856
4	475	456.825290

Graph

In [30]:

```
plt.figure(figsize=(10,6))
plt.style.use('fivethirtyeight')
plt.title('Stochastic Gradient Descent (SGD) Regression')
plt.xticks(df['price'],df.index.values)
plt.plot(df['price'],'g^',label='price')
plt.xticks(df['pred_price'],df.index.values)
plt.plot(df['pred_price'],'ro',label='price_pred')
plt.xlabel('Houses for sale',fontsize=15)
plt.ylabel('Predict/real',fontsize=15)
plt.legend(fontsize=13,loc='upper left')
plt.show()
```



SGD optimization and Linear Regression Performance Comparison

In [38]:

```
MSE_train = mean_squared_error(y_train,sgd_final.predict(X_train))
print('Train MSE:',y_scaler.inverse_transform(MSE_train.reshape(-1,1))[0,0])

MSE_test = mean_squared_error(y_test,sgd_final.predict(X_test))
print('Test MSE: ',y_scaler.inverse_transform(MSE_test.reshape(-1,1))[0,0])
```

```
Train MSE: 454.3634585018962
```

```
Test MSE:  455.8369378079987
```

Linear Regression

In [41]:
```
X = df[['rooms','m2','floor','age']]
y = df['price']

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

In [43]:
```
lr =LinearRegression().fit(X_train,y_train)

y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)
```

In [44]:
```
print('Linear Reg Train MSE :',mean_squared_error(y_train,y_train_pred))
print('Linear Reg Test MSE :',mean_squared_error(y_test,y_test_pred))
```

```
Linear Reg Train MSE : 3145.0627316271507
Linear Reg Test MSE : 4115.003886902745
```

In [45]:
```
print('Linear Reg Train R2 :',lr.score(X_train,y_train))
print('Linear Reg Test R2 :',lr.score(X_test,y_test))
```

```
Linear Reg Train R2 : 0.733739961656849
Linear Reg Test R2 : 0.6987282352837991
```

The error of the optimum model determined as a result of SGD optimization (MSE was used as a criterion) was found to be 454 for the training data and 455 for the test set. The error values of the classical linear regression model were found to be approximately 3145 for the training data and 4115 for the test data. Thus, it was revealed that the classical multiple linear regression model was not the model with the lowest possible error.