In [2]:

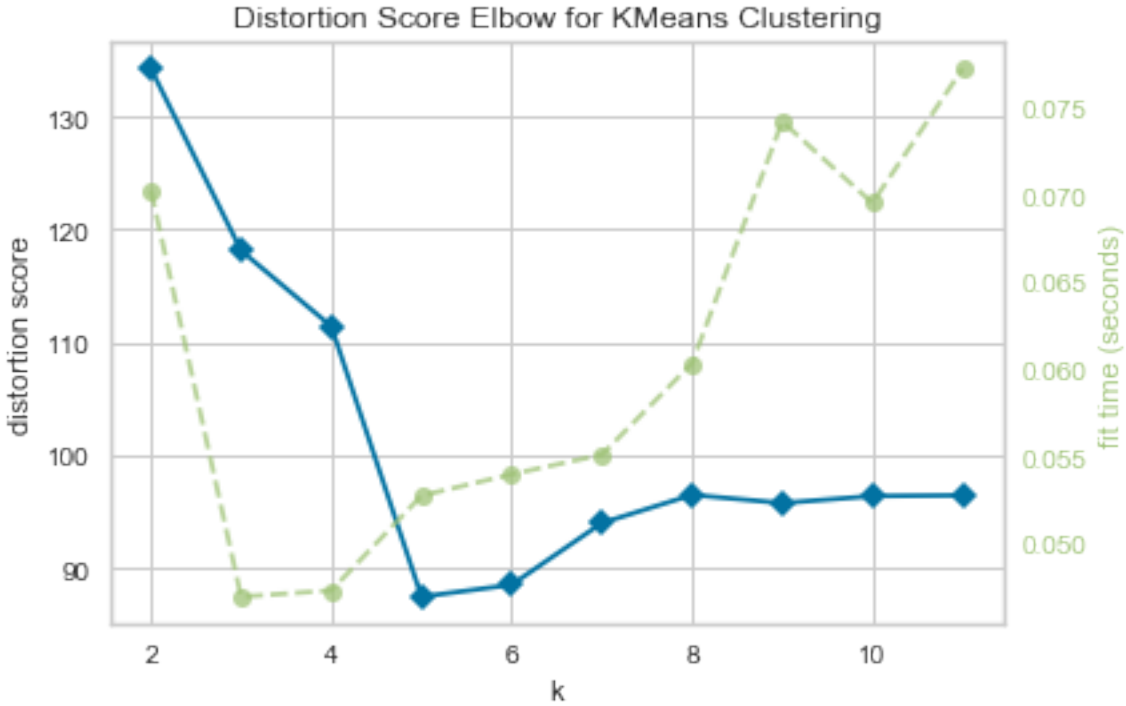
```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn import preprocessing
from yellowbrick.cluster import KElbowVisualizer
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


pd.set_option('display.max_columns', 500)
boston_data = load_boston()
df = pd.DataFrame(boston_data.data,columns=boston_data.feature_names)
df['target'] = pd.Series(boston_data.target)
```

In [3]:

```
#plot the prediction to a cross validation set evaluate the difference in pred
iction and reality add the average to the model.
```

In [20]:

```
L = df[['LSTAT','target']]
# L = preprocessing.scale(L)
mets = ['silhouette', 'calinski_harabaz', 'distortion']
km = KMeans()
visualize = KElbowVisualizer(km, k = (2,12),metric=mets[2])

visualize.fit(L)
visualize.poof()
```



In [21]:

```
def centroid_sort(centroids):
    new_centroids = []
    c0s = np.sort(centroids[:,0])

    for j in c0s:
        for i in range(len(centroids)):
            if j == centroids[i][0]:
                new_centroids.append([j, centroids[i][1]])
            else:
                continue

    new_centroids = np.array(new_centroids)
    return new_centroids

def generate_points(poly,start,stop,number_of_points=100):
    y_points=[]
    dummys = np.linspace(start,stop,number_of_points)
    for i in range(len(dummys)):
        y_points.append(poly(dummys[i]))
    return dummys, y_points

def get_ave_coefficients(dimension_of_eq, scribbling):
    ave_coefficients = [0 for k in range(dimension_of_eq)]
    for j in range(dimension_of_eq):
        for i in range(len(scribbling)):
            ave_coefficients[j] += scribbling[f'run_{i}'][j]

        ave_coefficients[j] = ave_coefficients[j]/len(scribbling)
    return ave_coefficients
```

In [6]:

```
# # print(centroids)

# colors = ['azure', 'darkolivegreen', 'darkblue', 'm', 'y', 'green', 'r', 'pa
payawhip' ]

# labels = km.labels_

# plt.scatter(L[:,0], L[:,1])
# plt.scatter(centroids[:, 0], centroids[:, 1], marker = 'x',c='m', s=75, line
widths = 5, zorder=10)



# # thats the line of best fit
# x, y = generate_points(p, -1, 2.5, 100)

# plt.plot(x, y,'go-')
```

In [22]:

```
model_testing = {}
```

```
# feeding through KMeans on sample, generating the centroids again and again with the best fit line
# Average over the result and return
# Assumptions data is in format already prepared for KMeans ie.: preprocess, operations, aggregations and so on.

def rigorous_test(Var_col, no_of_clusters, leading_order=3, iterations=23, preproc = False):

    X = df[[Var_col]]
    y = df[['target']]
    ave_eq = []
    ave_centroids = []

    if preproc:
        X = df[[Var_col]]
        X = preprocessing.scale(X)

    for i in range(iterations):

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.35)
        data = np.column_stack((X_train, y_train))

        km = KMeans(n_clusters = no_of_clusters)
        km.fit(data)
        centroids = km.cluster_centers_
        centroids = centroid_sort(centroids) #sorting centroids in order of x coord

        # fiting current centroids

        cx = centroids[:,0]
        cy = centroids[:,1]

        ploy = np.polyfit(cx, cy, leading_order)

        ave_eq.append(ploy)
        ave_centroids.append(centroids)


    ave_coefficients = np.mean(ave_eq, axis = 0)

    average_line = np.poly1d(ave_coefficients)


    ave_centroids = np.mean(ave_centroids, axis = 0)


    return average_line, ave_centroids
```

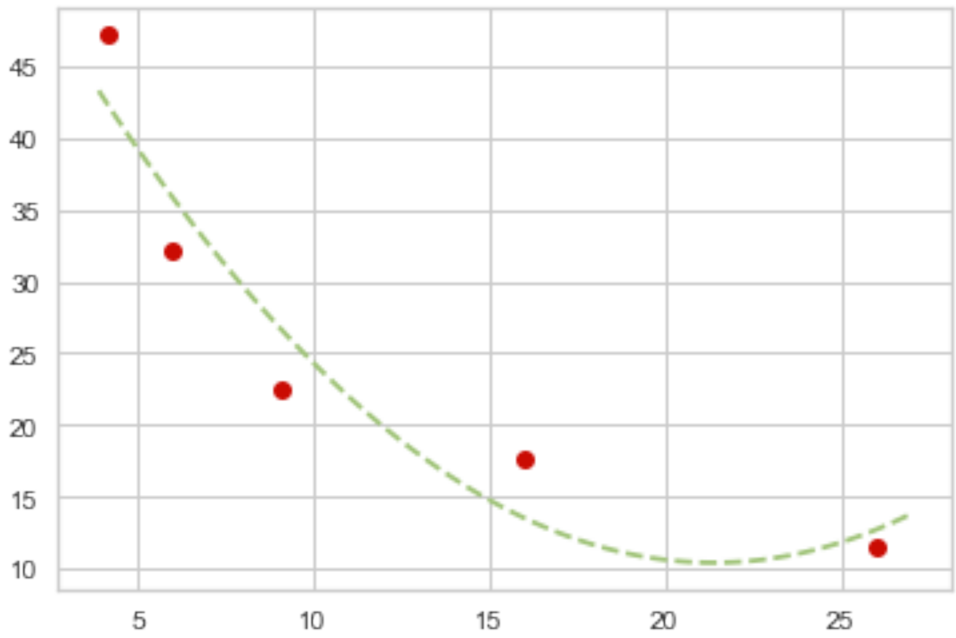
In [24]:

```
line, centroids = rigorous_test('LSTAT', no_of_clusters=5, leading_order=2, pr
eproc=False )
print('the line is: \n ', line)
print('\n \n' , 'Centroids are:\n',centroids)

plt.plot(centroids[:,0], centroids[:,1], 'ro')
xz, yz = generate_points(line ,3.9, 27, number_of_points=100)
plt.plot(xz, yz, 'g--')
```

```
the line is:
          2
0.1076 x - 4.603 x + 59.6


 Centroids are:
 [[ 4.16940125 47.18531813]
 [ 5.98285427 32.09676768]
 [ 9.06745139 22.53030121]
 [16.01402642 17.5566861 ]
 [26.00057608 11.58485563]]
```

Out[24]:

```
[<matplotlib.lines.Line2D at 0x12c689cf8>]
```



In [25]:

```
line.c
```

Out[25]:

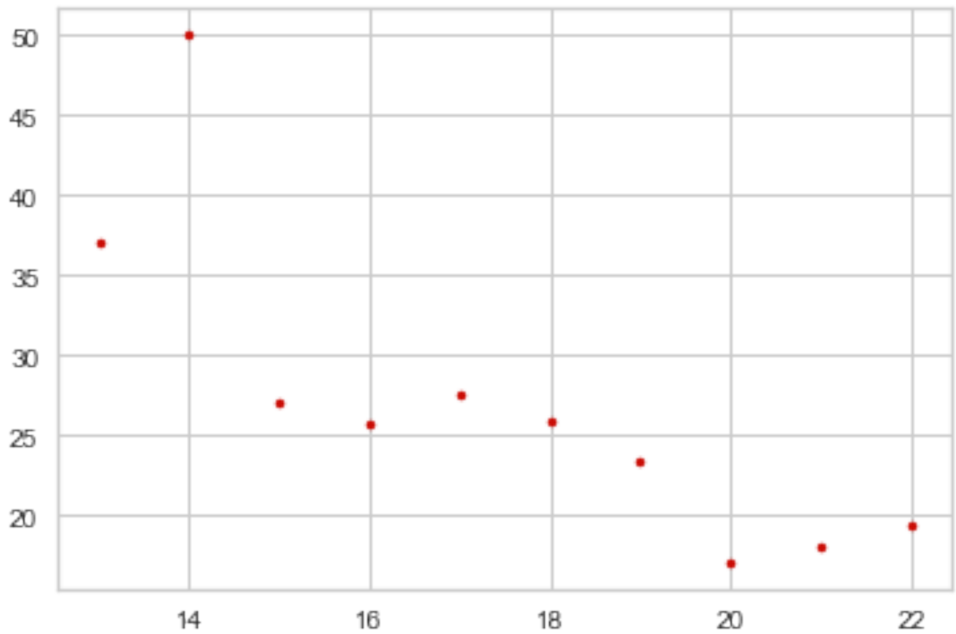
```
array([ 0.10764865, -4.60304943, 59.60211847])
```

In [26]:

```
# I forgot to mention that PTRATIO was calculated with averaging each rounded instance of student ratio
# per housing value to deliver a distribution that is not a "radiator" distribution
# it follows the pattern of regressing to the mean like all other categories.

roundy = round(df['PTRATIO'])
uniques = list(set(roundy))
yz = df['target']
dfesse = pd.concat([roundy, yz], axis=1)
uniques
cases = {}
for u in uniques:
    asdf = dfesse[dfesse['PTRATIO']==u]
    altering = asdf['target'].mean()
    cases[u] = altering

pt_x = cases.keys()
pt_y = cases.values()
plt.plot(pt_x,pt_y, 'r.')
```

Out[26]:

```
[<matplotlib.lines.Line2D at 0x12c6f4080>]
```



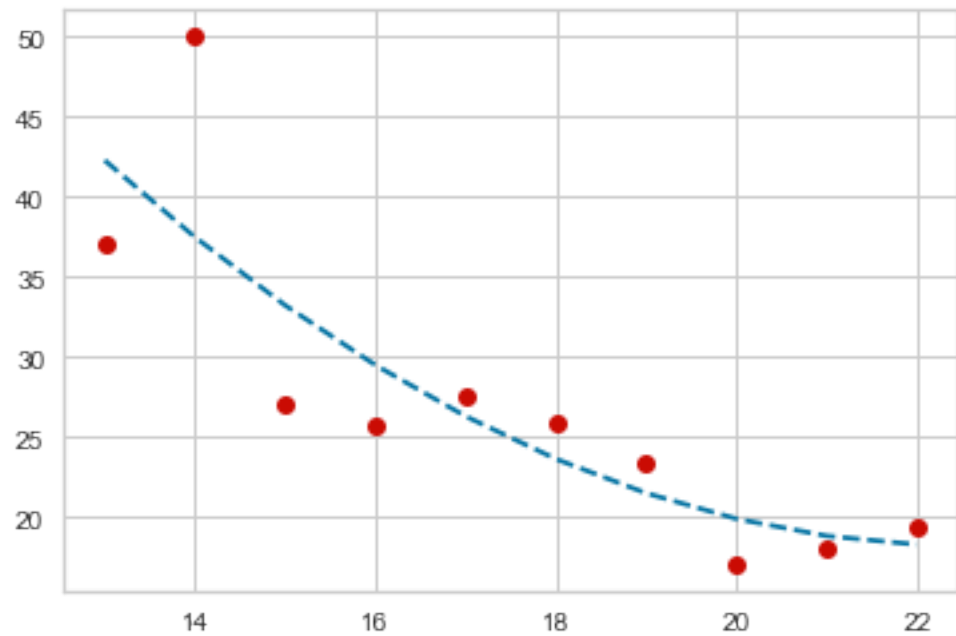
In [11]:

```
#PLotted PTRATIO
pt_x = list(pt_x)
pt_y = list(pt_y)
pt_fit = np.polyfit(pt_x, pt_y, 2)
pt_eq = np.poly1d(pt_fit)
print(pt_eq.c)
pt_fit_y = pt_eq(pt_x)
plt.plot(pt_x,pt_y, 'ro')
plt.plot(pt_x,pt_fit_y, 'b--')
```

```
[   0.26609764 -11.97890043 153.01549492]
```

Out[11]:

```
[<matplotlib.lines.Line2D at 0x12be450b8>]
```



In [ ]:

In [ ]:

In [ ]: