Boston house price prediction

Importing libraries:

In [31]:

```
# Import

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
```

Boston dataset from sklearn

In [3]:

```
from sklearn.datasets import load_boston

boston = load_boston()
```

In [4]:

```
data = pd.DataFrame(boston.data)

print(data)
```

```
           0     1      2    3      4      5      6       7     8      9  \
0    0.00632  18.0   2.31  0.0  0.538  6.575   65.2  4.0900   1.0  296.0
1    0.02731   0.0   7.07  0.0  0.469  6.421   78.9  4.9671   2.0  242.0
2    0.02729   0.0   7.07  0.0  0.469  7.185   61.1  4.9671   2.0  242.0
3    0.03237   0.0   2.18  0.0  0.458  6.998   45.8  6.0622   3.0  222.0
4    0.06905   0.0   2.18  0.0  0.458  7.147   54.2  6.0622   3.0  222.0
5    0.02985   0.0   2.18  0.0  0.458  6.430   58.7  6.0622   3.0  222.0
6    0.08829  12.5   7.87  0.0  0.524  6.012   66.6  5.5605   5.0  311.0
7    0.14455  12.5   7.87  0.0  0.524  6.172   96.1  5.9505   5.0  311.0
8    0.21124  12.5   7.87  0.0  0.524  5.631  100.0  6.0821   5.0  311.0
9    0.17004  12.5   7.87  0.0  0.524  6.004   85.9  6.5921   5.0  311.0
10   0.22489  12.5   7.87  0.0  0.524  6.377   94.3  6.3467   5.0  311.0
11   0.11747  12.5   7.87  0.0  0.524  6.009   82.9  6.2267   5.0  311.0
12   0.09378  12.5   7.87  0.0  0.524  5.889   39.0  5.4509   5.0  311.0
13   0.62976   0.0   8.14  0.0  0.538  5.949   61.8  4.7075   4.0  307.0
14   0.63796   0.0   8.14  0.0  0.538  6.096   84.5  4.4619   4.0  307.0
15   0.62739   0.0   8.14  0.0  0.538  5.834   56.5  4.4986   4.0  307.0
16   1.05393   0.0   8.14  0.0  0.538  5.935   29.3  4.4986   4.0  307.0
17   0.78420   0.0   8.14  0.0  0.538  5.990   81.7  4.2579   4.0  307.0
18   0.80271   0.0   8.14  0.0  0.538  5.456   36.6  3.7965   4.0  307.0
19   0.72580   0.0   8.14  0.0  0.538  5.727   69.5  3.7965   4.0  307.0
20   1.25179   0.0   8.14  0.0  0.538  5.570   98.1  3.7979   4.0  307.0
21   0.85204   0.0   8.14  0.0  0.538  5.965   89.2  4.0123   4.0  307.0
22   1.23247   0.0   8.14  0.0  0.538  6.142   91.7  3.9769   4.0  307.0
23   0.98843   0.0   8.14  0.0  0.538  5.813  100.0  4.0952   4.0  307.0
24   0.75026   0.0   8.14  0.0  0.538  5.924   94.1  4.3996   4.0  307.0
25   0.84054   0.0   8.14  0.0  0.538  5.599   85.7  4.4546   4.0  307.0
26   0.67191   0.0   8.14  0.0  0.538  5.813   90.3  4.6820   4.0  307.0
27   0.95577   0.0   8.14  0.0  0.538  6.047   88.8  4.4534   4.0  307.0
28   0.77299   0.0   8.14  0.0  0.538  6.495   94.4  4.4547   4.0  307.0
29   1.00245   0.0   8.14  0.0  0.538  6.674   87.3  4.2390   4.0  307.0
..       ...   ...    ...  ...    ...    ...    ...     ...   ...    ...
476  4.87141   0.0  18.10  0.0  0.614  6.484   93.6  2.3053  24.0  666.0
```

477	15.02340	0.0	18.10	0.0	0.614	5.304	97.3	2.1007	24.0	666.0
478	10.23300	0.0	18.10	0.0	0.614	6.185	96.7	2.1705	24.0	666.0
479	14.33370	0.0	18.10	0.0	0.614	6.229	88.0	1.9512	24.0	666.0
480	5.82401	0.0	18.10	0.0	0.532	6.242	64.7	3.4242	24.0	666.0
481	5.70818	0.0	18.10	0.0	0.532	6.750	74.9	3.3317	24.0	666.0
482	5.73116	0.0	18.10	0.0	0.532	7.061	77.0	3.4106	24.0	666.0
483	2.81838	0.0	18.10	0.0	0.532	5.762	40.3	4.0983	24.0	666.0
484	2.37857	0.0	18.10	0.0	0.583	5.871	41.9	3.7240	24.0	666.0
485	3.67367	0.0	18.10	0.0	0.583	6.312	51.9	3.9917	24.0	666.0
486	5.69175	0.0	18.10	0.0	0.583	6.114	79.8	3.5459	24.0	666.0
487	4.83567	0.0	18.10	0.0	0.583	5.905	53.2	3.1523	24.0	666.0
488	0.15086	0.0	27.74	0.0	0.609	5.454	92.7	1.8209	4.0	711.0
489	0.18337	0.0	27.74	0.0	0.609	5.414	98.3	1.7554	4.0	711.0
490	0.20746	0.0	27.74	0.0	0.609	5.093	98.0	1.8226	4.0	711.0
491	0.10574	0.0	27.74	0.0	0.609	5.983	98.8	1.8681	4.0	711.0
492	0.11132	0.0	27.74	0.0	0.609	5.983	83.5	2.1099	4.0	711.0
493	0.17331	0.0	9.69	0.0	0.585	5.707	54.0	2.3817	6.0	391.0
494	0.27957	0.0	9.69	0.0	0.585	5.926	42.6	2.3817	6.0	391.0
495	0.17899	0.0	9.69	0.0	0.585	5.670	28.8	2.7986	6.0	391.0
496	0.28960	0.0	9.69	0.0	0.585	5.390	72.9	2.7986	6.0	391.0
497	0.26838	0.0	9.69	0.0	0.585	5.794	70.6	2.8927	6.0	391.0
498	0.23912	0.0	9.69	0.0	0.585	6.019	65.3	2.4091	6.0	391.0
499	0.17783	0.0	9.69	0.0	0.585	5.569	73.5	2.3999	6.0	391.0
500	0.22438	0.0	9.69	0.0	0.585	6.027	79.7	2.4982	6.0	391.0
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0

	10	11	12
0	15.3	396.90	4.98
1	17.8	396.90	9.14
2	17.8	392.83	4.03
3	18.7	394.63	2.94
4	18.7	396.90	5.33
5	18.7	394.12	5.21
6	15.2	395.60	12.43
7	15.2	396.90	19.15
8	15.2	386.63	29.93
9	15.2	386.71	17.10
10	15.2	392.52	20.45
11	15.2	396.90	13.27
12	15.2	390.50	15.71
13	21.0	396.90	8.26
14	21.0	380.02	10.26
15	21.0	395.62	8.47
16	21.0	386.85	6.58
17	21.0	386.75	14.67
18	21.0	288.99	11.69
19	21.0	390.95	11.28
20	21.0	376.57	21.02
21	21.0	392.53	13.83
22	21.0	396.90	18.72
23	21.0	394.54	19.88
24	21.0	394.33	16.30
25	21.0	303.42	16.51
26	21.0	376.88	14.81
27	21.0	306.38	17.28
28	21.0	387.94	12.80
29	21.0	380.23	11.98
..	...	...	...
476	20.2	396.21	18.68
477	20.2	349.48	24.91
478	20.2	379.70	18.03
479	20.2	383.32	13.11
480	20.2	396.90	10.74
481	20.2	393.07	7.74
482	20.2	395.28	7.01
483	20.2	392.92	10.42
484	20.2	370.73	13.34
485	20.2	388.62	10.58

```
486  20.2  392.68  14.98
487  20.2  388.22  11.45
488  20.1  395.09  18.06
489  20.1  344.05  23.97
490  20.1  318.43  29.68
491  20.1  390.11  18.07
492  20.1  396.90  13.35
493  19.2  396.90  12.01
494  19.2  396.90  13.59
495  19.2  393.29  17.60
496  19.2  396.90  21.14
497  19.2  396.90  14.10
498  19.2  396.90  12.92
499  19.2  395.77  15.10
500  19.2  396.90  14.33
501  21.0  391.99   9.67
502  21.0  396.90   9.08
503  21.0  396.90   5.64
504  21.0  393.45   6.48
505  21.0  396.90   7.88

[506 rows x 13 columns]
```

Notations (Those notations are copied from the source):

1- CRIM per capita crime rate by town 2- ZN proportion of residential land zoned for lots over 25,000 sq.ft. 3- INDUS proportion of non-retail business acres per town 4- CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise) 5- NOX nitric oxides concentration (parts per 10 million) 6- RM average number of rooms per dwelling 7- AGE proportion of owner-occupied units built prior to 1940 8- DIS weighted distances to five Boston employment centres 9- RAD index of accessibility to radial highways 10- TAX full-value property-tax rate per 10,000usd 11- PTRATIO pupil-teacher ratio by town 12- B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town 13- LSTAT % lower status of the population

In [5]:

```
# now, we need to add what each column corresponds to:

data.columns = boston.feature_names
data.head()
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [6]:

```
# specifing our target

data["Price"] = boston.target
```

In [7]:

```
# Properties:

print(data.shape)

print(data.columns)

print(data.dtypes)
```

```
print(data.nunique())

print(data.isnull().sum())

print(data.describe())
```

```
(506, 14)
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'Price'],
      dtype='object')
CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD        float64
TAX        float64
PTRATIO    float64
B          float64
LSTAT      float64
Price      float64
dtype: object
CRIM       504
ZN          26
INDUS       76
CHAS         2
NOX         81
RM         446
AGE        356
DIS        412
RAD          9
TAX         66
PTRATIO     46
B          357
LSTAT      455
Price      229
dtype: int64
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
Price      0
dtype: int64
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.593761   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.596783   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.647423   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
```

```
50%      77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%      94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max     100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

             LSTAT       Price
count   506.000000  506.000000
mean     12.653063   22.532806
std       7.141062    9.197104
min       1.730000    5.000000
25%       6.950000   17.025000
50%      11.360000   21.200000
75%      16.955000   25.000000
max      37.970000   50.000000
```

In [8]:

```
# Creating correlation between the features


correlation = data.corr()

correlation.shape
```

Out[8]:

```
(14, 14)
```

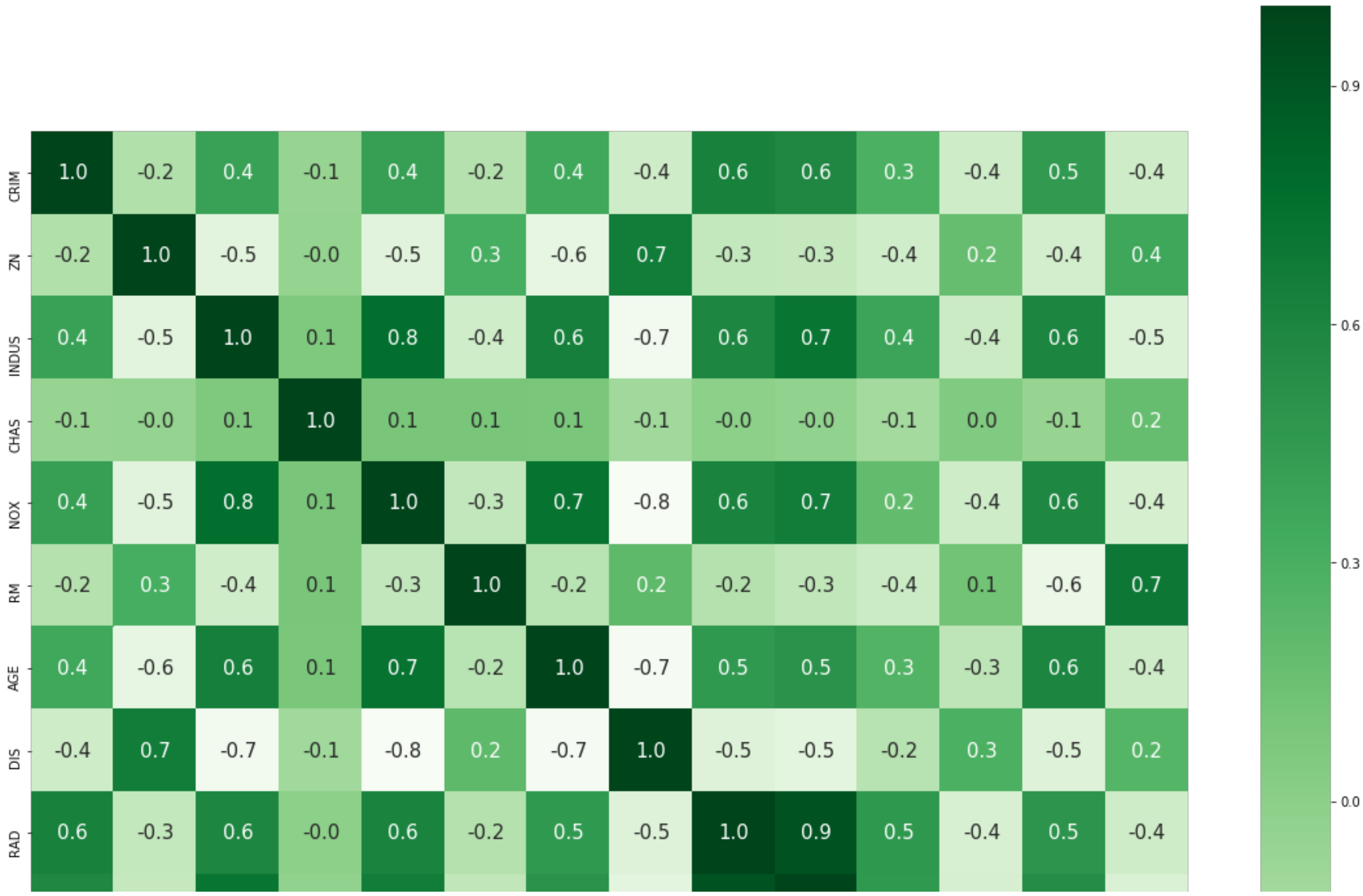
In [9]:

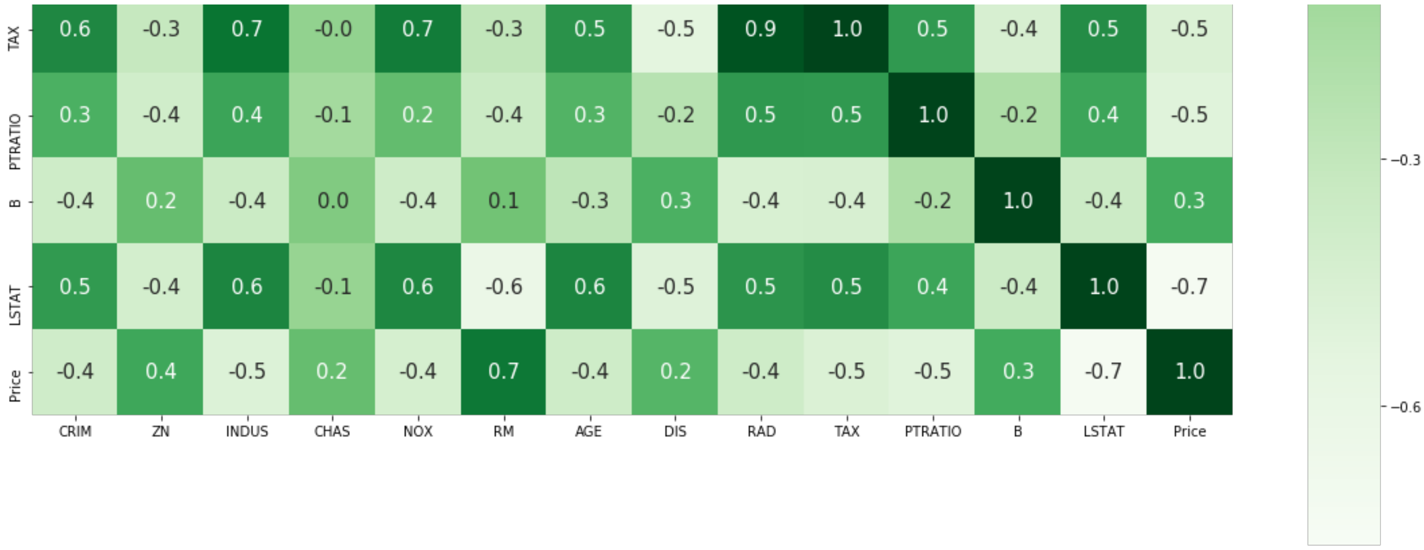
```
# Plotting heat map of the correlated features:

plt.figure(figsize = (20,20))
print("\n This is the Heatmap: \n")
sns.heatmap(correlation, cbar=True, square= True, fmt='.1f', annot=True, annot_kws={'siz
e':15}, cmap='Greens')
```

```
 This is the Heatmap:
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x14e1852d5f8>
```





In [10]:

```
# specifying target variable and independent variable:

X = data.drop(['Price'], axis = 1)
y = data['Price']
```

In [11]:

```
# specifying training and testing data

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state =
4)
```

Now, we are going to create the ML path, using Linear Regression and Random Forest Model:

Linear Regression:

Training data:

In [12]:

```
# creating our model

model = LinearRegression()

# using the model on the training data

model.fit(X_train, y_train)
```

Out[12]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [14]:

```
# the intercept

model.intercept_
```

Out[14]:

```
36.357041376595035
```

In [18]:

```
# coefficients:
```

```
coeff = pd.DataFrame([X_train.columns,model.coef_]).T
coeff = coeff.rename(columns ={0:"Attributes",1:"Coefficients"})

coeff
```

Out[18]:

	Attributes	Coefficients
0	CRIM	-0.12257
1	ZN	0.0556777
2	INDUS	-0.00883428
3	CHAS	4.69345
4	NOX	-14.4358
5	RM	3.28008
6	AGE	-0.00344778
7	DIS	-1.55214
8	RAD	0.32625
9	TAX	-0.0140666
10	PTRATIO	-0.803275
11	B	0.00935369
12	LSTAT	-0.523478

In [24]:

```
# Evaluating the Model

y_pred = model.predict(X_train)

pd.DataFrame(y_pred)
```

Out[24]:

	0
0	24.522480
1	15.197510
2	25.577206
3	13.939400
4	39.466513
5	17.459599
6	39.710299
7	16.517481
8	20.197333
9	40.797755
10	33.572450
11	14.504206
12	11.445145
13	23.065640
14	24.397344
15	25.010961
16	14.361165
17	28.283415

	0
18	25.049315
19	22.428252
20	21.815885
21	18.852087
22	13.356212
23	13.657927
24	23.647660
25	18.068763
26	16.129572
27	41.124149
28	19.433918
29	13.179809
...	...
324	15.155495
325	17.994417
326	30.761722
327	29.543075
328	6.259368
329	27.179352
330	14.805956
331	23.594438
332	22.668606
333	16.020712
334	24.058107
335	20.661335
336	25.379353
337	27.553691
338	26.950710
339	26.755668
340	19.869935
341	19.690256
342	24.332599
343	21.924869
344	20.354469
345	35.338450
346	13.007641
347	25.813350
348	22.959968
349	8.608369
350	31.511078
351	13.647191
352	26.501062
353	20.540965

354 rows × 1 columns

Metrics:

In [25]:

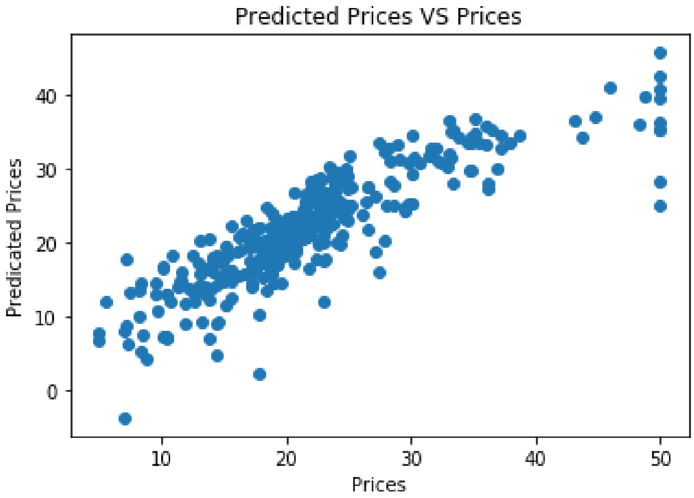
```
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_
train)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.7465991966746854
Adjusted R^2: 0.736910342429894
MAE: 3.0898610949711287
MSE: 19.073688703469028
RMSE: 4.367343437774161
```

Plotting:

In [26]:

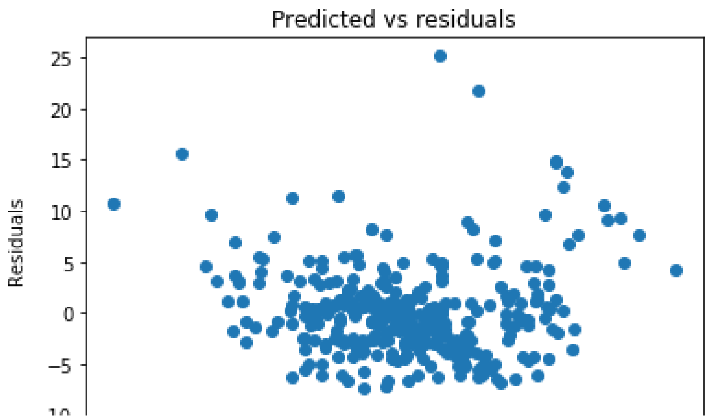
```
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicated Prices")
plt.title("Predicted Prices VS Prices")
plt.show()
```

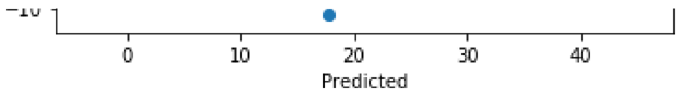


Testing for residuals:

In [27]:

```
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```





Now, we need to use test data to validate our LR model:

In [29]:

```
y_test_pred = model.predict(X_test)
```

In [30]:

```
# Evaluating our model

acc_linreg = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_linreg)
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-1)/(len
(y_test)-X_test.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.7120461624218655
Adjusted R^2: 0.6849200762732006
MAE: 3.867069394655806
MSE: 30.06816053374662
RMSE: 5.483444221814117
```

- **The results are similar to the of the train data. As such, we are not over fitting or under fitting our model.**

Random Forest Model (RFM)

In [32]:

```
# initiating our RFM

rfm = RandomForestRegressor()

# Training our data

rfm.fit(X_train, y_train)
```

Out[32]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [33]:

```
# Evaluating our model

y_pred = rfm.predict(X_train)
```

In [34]:

```
# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_
train)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.9729121848992065
```

```
Adjusted R^2: 0.9718764743218232
MAE: 0.8334180790960453
MSE: 2.038922316384181
RMSE: 1.4279083711443745
```

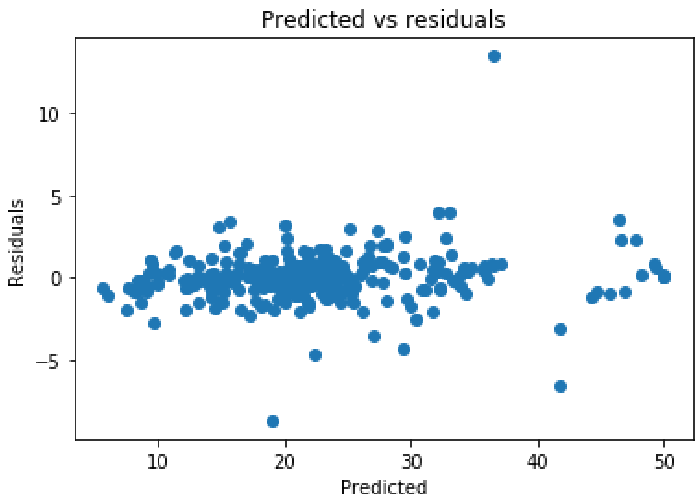
Plotting:

In [35]:

```
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

In [36]:

```
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



Testing our the validity of our model:

In [37]:

```
y_pred_test = rfm.predict(X_test)
```

In [38]:

```
# Model Evaluation
```

```
acc_rf = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_rf)
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-1)/(len
(y_test)-X_test.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.7120461624218655
Adjusted R^2: 0.6849200762732006
MAE: 3.867069394655806
MSE: 30.06816053374662
RMSE: 5.483444221814117
```