MULTIPLE POLYNOMIAL REGRESSION

The data set used is the data set containing the real estate market information. It includes the number of rooms, the age of the building, the number of floors, square meters, and the price information.

In [223...
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score,mean_squared_error
```

In [224...
```
df = pd.read_csv(r'D:\githubProjects\Machine-Learning\Supervised Learning\Model Regu
df.head()
```

Out[224...

	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 [225...
```
df.drop('Unnamed: 0',axis=1,inplace=True)
```

In [226...
```
df.to_csv('real_estate.csv')
```

Hold-Out

In [227...
```
X = df.drop('price',axis=1)
y = df['price']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=4
```

Polynomial Degree Generating Function

In [228...
```
from sklearn.preprocessing import PolynomialFeatures
```

We create the polynomial degrees of the features from the second order to the tenth order and calculate the error score of the model created with each polynomial degree.

Here we use RMSE as the error score.

In [229…

```
rmses = []
degrees = np.arange(1,10)
min_rmse, min_deg = 1e10, 0
```

The R2 measure can be overly sensitive when it comes to polynomial functions. Therefore, RMSE is widely used for error score when Grid Search is done to determine the appropriate polynomial degree.

Train set Features

In [230…

```
for deg in degrees:
    poly_features = PolynomialFeatures(degree=deg,include_bias=False)

    X_poly_train = poly_features.fit_transform(X_train)

    # Linear Reg.
    poly_reg = LinearRegression().fit(X_poly_train,y_train)

    # Comparison with test set
    X_poly_test = poly_features.fit_transform(X_test)
    poly_predict = poly_reg.predict(X_poly_test)
    poly_mse = mean_squared_error(y_test,poly_predict)
    poly_rmse = np.sqrt(mean_squared_error(y_test,poly_predict))
    rmses.append(poly_rmse)

    if min_rmse > poly_rmse:
        min_rmse = poly_rmse
        min_deg = deg
```

In [231…

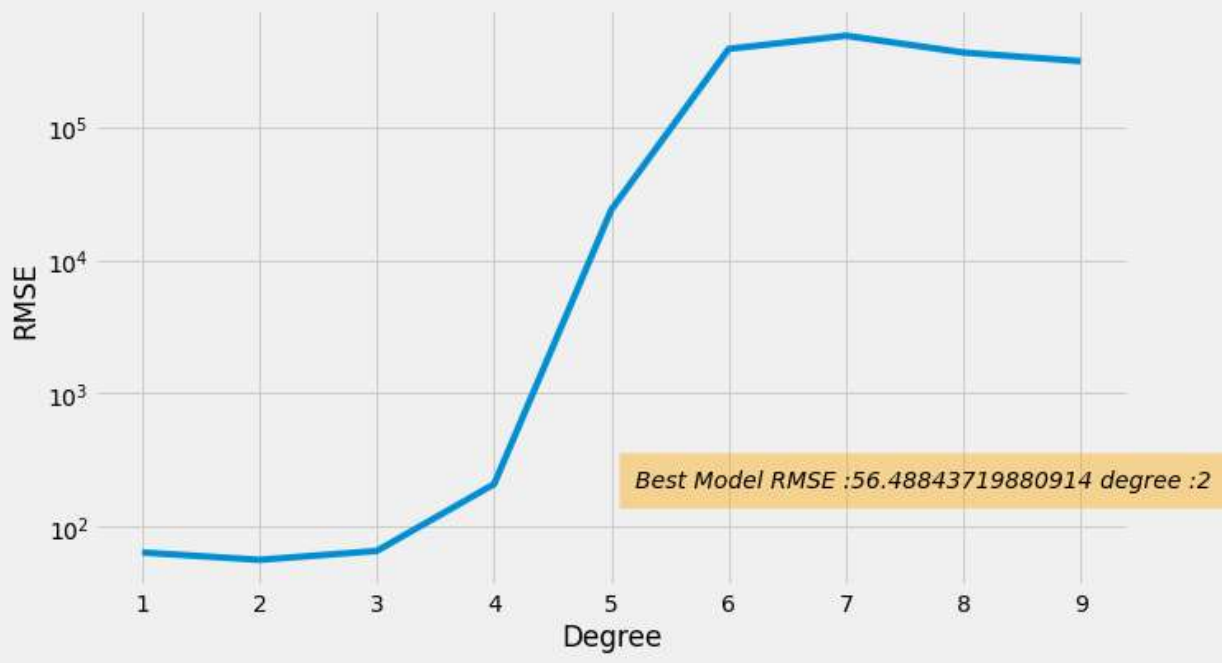
```
print(f'Best Model RMSE :{min_rmse} degree :{min_deg}')
```

```
Best Model RMSE :56.48843719880914 degree :2
```

RMSE scores for all polynomial values on the graph

In [232…

```
plt.style.use('fivethirtyeight')
fig = plt.figure(figsize=(10,6))
ax = fig.add_subplot(111)
ax.plot(degrees,rmses)
ax.set_yscale('log')
ax.set_xlabel('Degree')
ax.set_ylabel('RMSE')
plt.text(5.2,200,(f'Best Model RMSE :{min_rmse} degree :{min_deg}'),style='italic',
         bbox={'facecolor' : 'orange', 'alpha' : 0.4, 'pad' : 10})
plt.show()
```



The lowest error score is obtained when we take the polynomial degrees of all the features in the second degree. Now that we have determined the appropriate polynomial degree as 2, we can now create the model according to this polynomial degree and look at its performance in the test set.

In [233...

```
poly_deg = PolynomialFeatures(degree=2)
X_train_poly = poly_deg.fit_transform(X_train)
X_test_poly = poly_deg.fit_transform(X_test)
```

In [234...

```
poly_reg =LinearRegression().fit(X_train_poly,y_train)

model = LinearRegression().fit(X_train,y_train)
```

Prediction performance of multilinear model for training and test sets

In [235...

```
print(f'Linear Reg Train R2 :{model.score(X_train,y_train)}')
print(f'Linear Reg Test R2 :{model.score(X_test,y_test)}')
```

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

Prediction performance of multiple polynomial model for training and test sets

In [236...

```
print(f'Polynomial Reg Train R2 :{poly_reg.score(X_train_poly,y_train)}')
print(f'Polynomial Reg Test R2 :{poly_reg.score(X_test_poly,y_test)}')
```

```
Polynomial Reg Train R2 :0.8488074651060601
Polynomial Reg Test R2 :0.7663814623333732
```

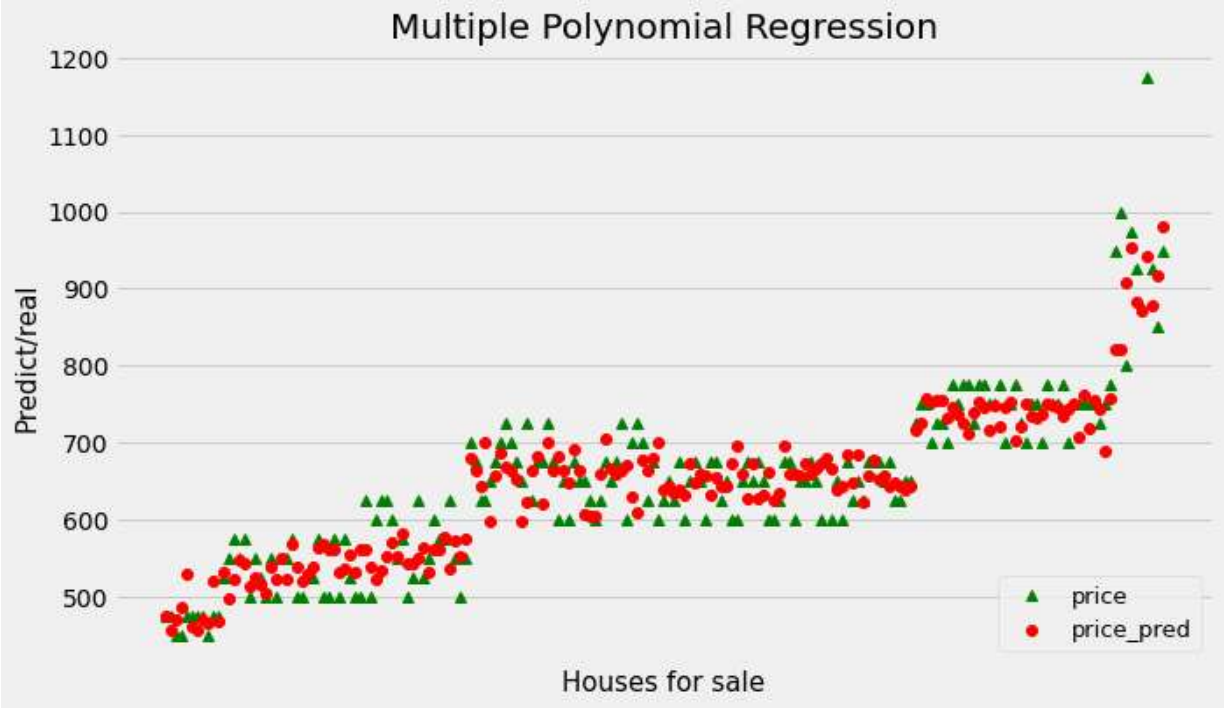
In [237...

```
X_poly = poly_deg.transform(X)
df['price_pred'] = poly_reg.predict(X_poly)
```

Graph

In [238…

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



DETERMINING THE SUITABLE POLYNOMIAL DEGREE

Is there really a non-linear relationship between the prices of the flats for sale and each property? Can't the relation of the target variable with some properties be linear/polynomial with their own self or even if the relation of the target variable with the properties is curvilinear, does it have the same degree of polynomial relation with each of them?

In [239…

```
df = pd.read_csv('real_estate.csv')
df.head()
```

Out[239…

	Unnamed: 0	price	rooms	m2	floor	age
0	0	475	1	40	0	6

	Unnamed: 0	price	rooms	m2	floor	age
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 [240…

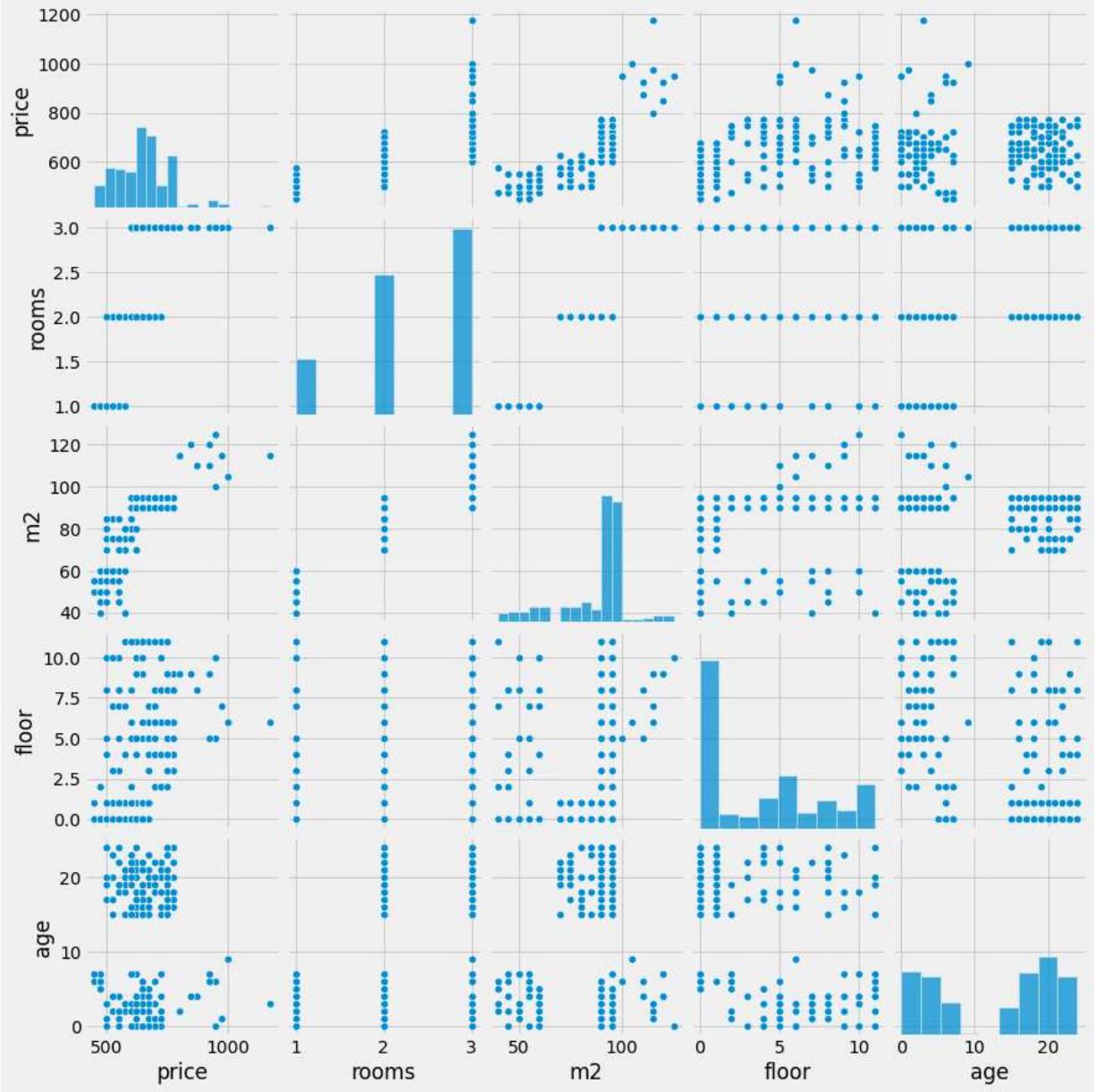
```
df.drop('Unnamed: 0',axis=1,inplace=True)
```

In [241…

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

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=4
```

In [242…

```
g = sns.pairplot(df)
```



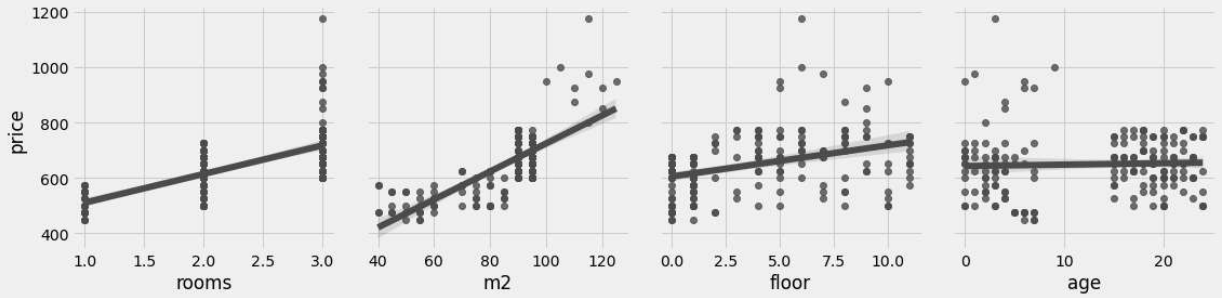
At first glance, a linear relationship between the price and the number of rooms can be clearly seen. We cannot say that there is a linear relationship with the same precision about the data points obtained from the scatter diagrams of the other properties of house prices.

In [243…

```
g = sns.PairGrid(df,y_vars=['price'],x_vars=['rooms','m2','floor','age'],height=4)

g.map(sns.regplot,lowess=False,color='.3')
```

Out[243…

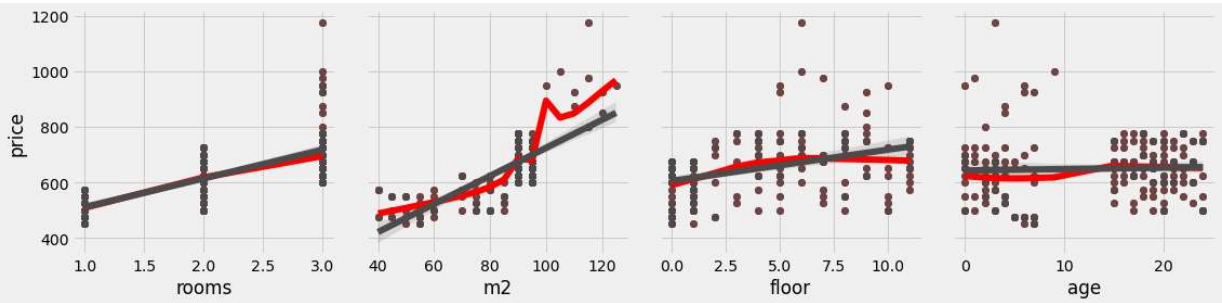
```
<seaborn.axisgrid.PairGrid at 0x13ddf9503d0>
```



When we look at the Local Regression curves, we can observe the trends of the data points. The lowess curve of the price and the number of rooms was not created because these two have a perfectly linear relationship. At first glance, the relationship between house prices and the measurement of the width, the floor and the age of the building does not seem linear. In addition, it can be seen that the polynomial relationship between the face measurement variable and the floor where the house is located and the age of the building is not of the same degree with the target variable.

In [244...
```
g = sns.PairGrid(df,y_vars=['price'],x_vars=['rooms','m2','floor','age'],height=4)

g.map(sns.regplot,lowess=True,color='red')
g.map(sns.regplot,lowess=False,color='.3')
```

Out[244...
```
<seaborn.axisgrid.PairGrid at 0x13ddf950e50>
```

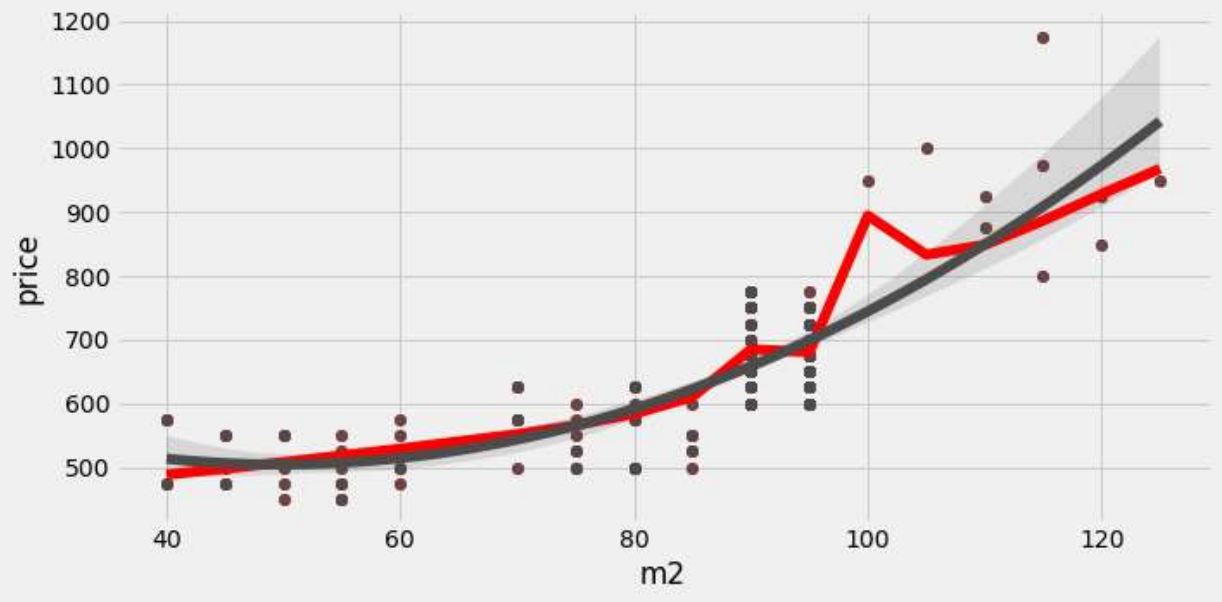


While the relation of the target variable with the number of rooms is linear, its relation with the other properties of the houses is not linear. This is pretty obvious, especially for the m2 and floor variables.

When estimating with polynomial regression, we do not need to take the polynomial degrees of the room number and age variables. We will use them as they are in the model.

In [245...
```
g = sns.PairGrid(df,y_vars=['price'],x_vars=['m2'],height=5,aspect=2)

g.map(sns.regplot,lowess=True,color='red')  # Lowess curve
g.map(sns.regplot,lowess=False,color='.3',order=2)
```

Out[245...
```
<seaborn.axisgrid.PairGrid at 0x13dee311d20>
```



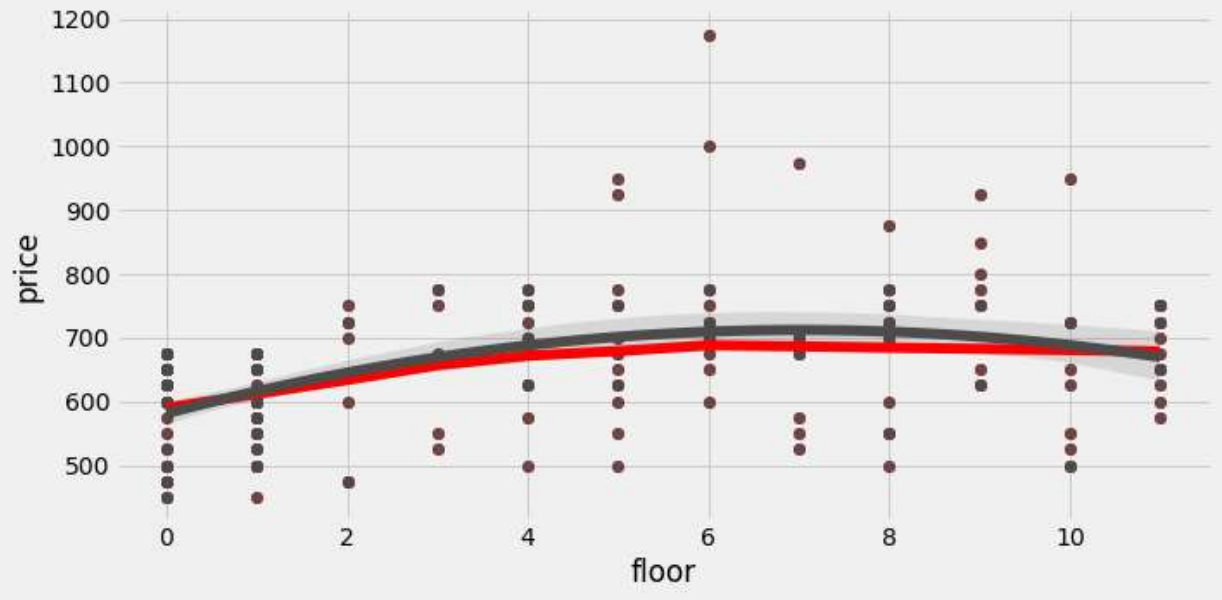
|polynomial degree for price - floor relation

In [246…

```
g = sns.PairGrid(df,y_vars=['price'],x_vars=['floor'],height=5,aspect=2)

g.map(sns.regplot,lowess=True,color='red')  # Lowess curve
g.map(sns.regplot,lowess=False,color='.3',order=2)
```

Out[246…

```
<seaborn.axisgrid.PairGrid at 0x13dee3a32e0>
```



Now, let's set the model by determining the appropriate polynomial degrees for the features that have a polynomial relationship with the target variable.

Train

In [247…

```
df_train = pd.concat([y_train,X_train],axis=1,sort=True)

X1_train = df_train['rooms'].values.reshape(-1,1)
X2_train = df_train['m2'].values.reshape(-1,1)
```

```
X3_train = df_train['floor'].values.reshape(-1,1)
X4_train = df_train['age'].values.reshape(-1,1)
```

Test

In [248…

```
df_test = pd.concat([y_test,X_test],axis=1,sort=True)

X1_test = df_test['rooms'].values.reshape(-1,1)
X2_test = df_test['m2'].values.reshape(-1,1)
X3_test = df_test['floor'].values.reshape(-1,1)
X4_test = df_test['age'].values.reshape(-1,1)
```

RMSE scores for all degrees of Polynomial

1-) rooms

In [249…

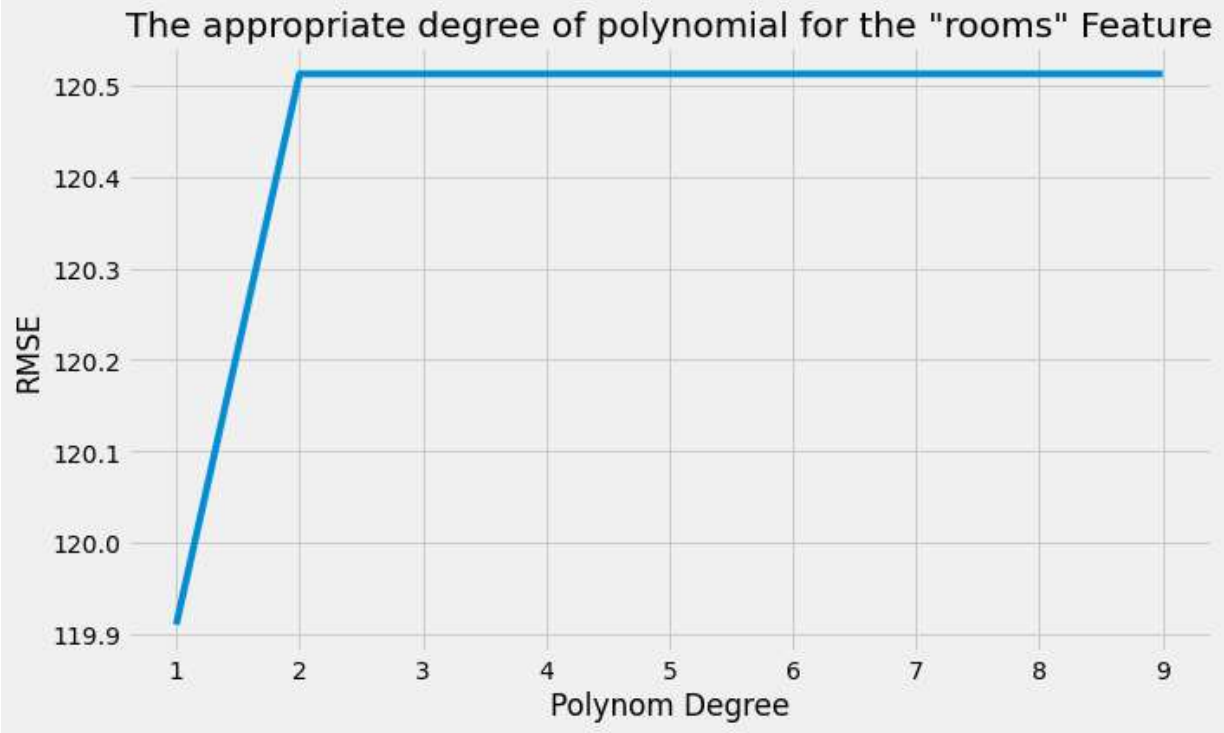
```
rmses = []
degrees = np.arange(1,10)
min_rmse, min_deg = 1e10, 0
```

In [250…

```
for deg in degrees:
    poly_features = PolynomialFeatures(degree=deg,include_bias=False)

    X_poly_train = poly_features.fit_transform(X1_train)

    # Linear Reg.
    poly_reg = LinearRegression().fit(X_poly_train,y_train)

    # Comparison with test set
    X_poly_test = poly_features.fit_transform(X1_test)
    poly_predict = poly_reg.predict(X_poly_test)
    poly_mse = mean_squared_error(y_test,poly_predict)
    poly_rmse = np.sqrt(mean_squared_error(y_test,poly_predict))
    rmses.append(poly_rmse)

    if min_rmse > poly_rmse:
        min_rmse = poly_rmse
        min_deg = deg
```

In [251…

```
plt.figure(figsize=(10,6))
plt.title('The appropriate degree of polynomial for the "rooms" Feature')
sns.lineplot(x=degrees,y=rmses)
plt.xlabel('Polynom Degree')
plt.ylabel('RMSE')
plt.show()
```



2-)m2

In [252…

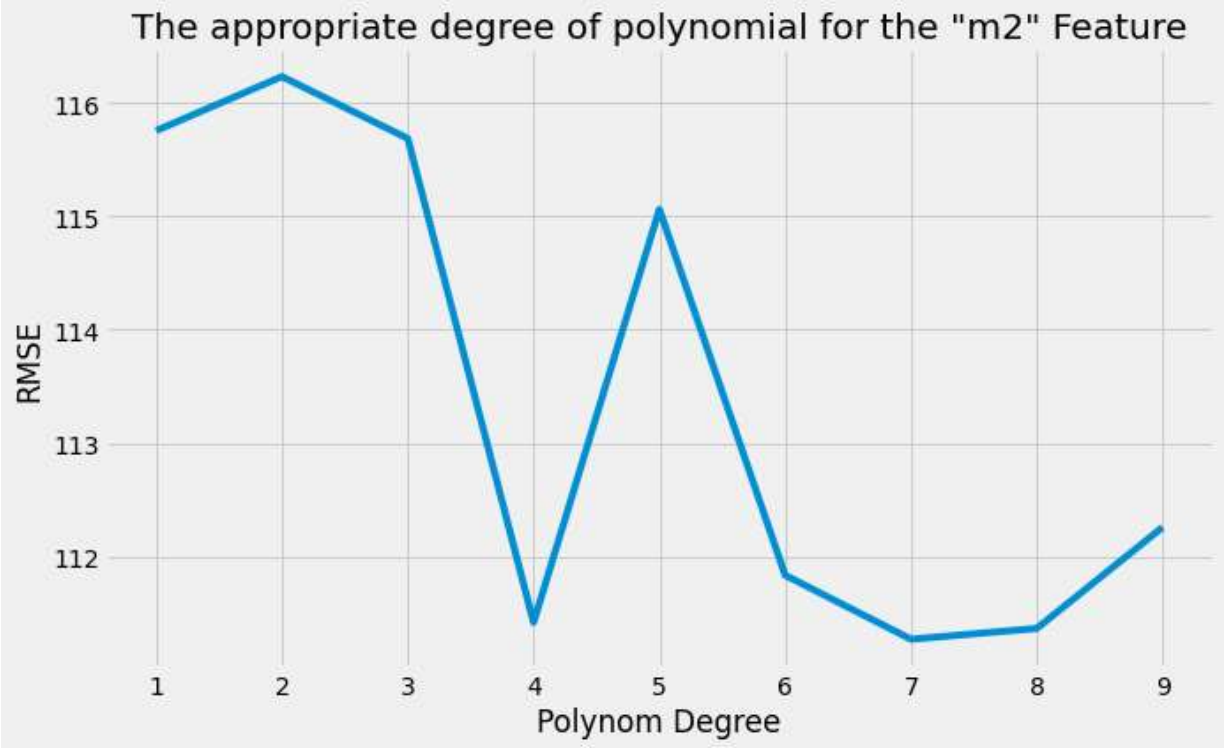
```
rmses = []
degrees = np.arange(1,10)
min_rmse, min_deg = 1e10, 0
```

In [253…

```
for deg in degrees:
    poly_features = PolynomialFeatures(degree=deg,include_bias=False)

    X_poly_train = poly_features.fit_transform(X2_train)

    # Linear Reg.
    poly_reg = LinearRegression().fit(X_poly_train,y_train)

    # Comparison with test set
    X_poly_test = poly_features.fit_transform(X2_test)
    poly_predict = poly_reg.predict(X_poly_test)
    poly_mse = mean_squared_error(y_test,poly_predict)
    poly_rmse = np.sqrt(mean_squared_error(y_test,poly_predict))
    rmses.append(poly_rmse)

    if min_rmse > poly_rmse:
        min_rmse = poly_rmse
        min_deg = deg
```

In [254…

```
plt.figure(figsize=(10,6))
plt.title('The appropriate degree of polynomial for the "m2" Feature')
sns.lineplot(x=degrees,y=rmses)
plt.xlabel('Polynom Degree')
plt.ylabel('RMSE')
plt.show()
```



3-)floor

In [255…

```
rmses = []
degrees = np.arange(1,10)
min_rmse, min_deg = 1e10, 0
```

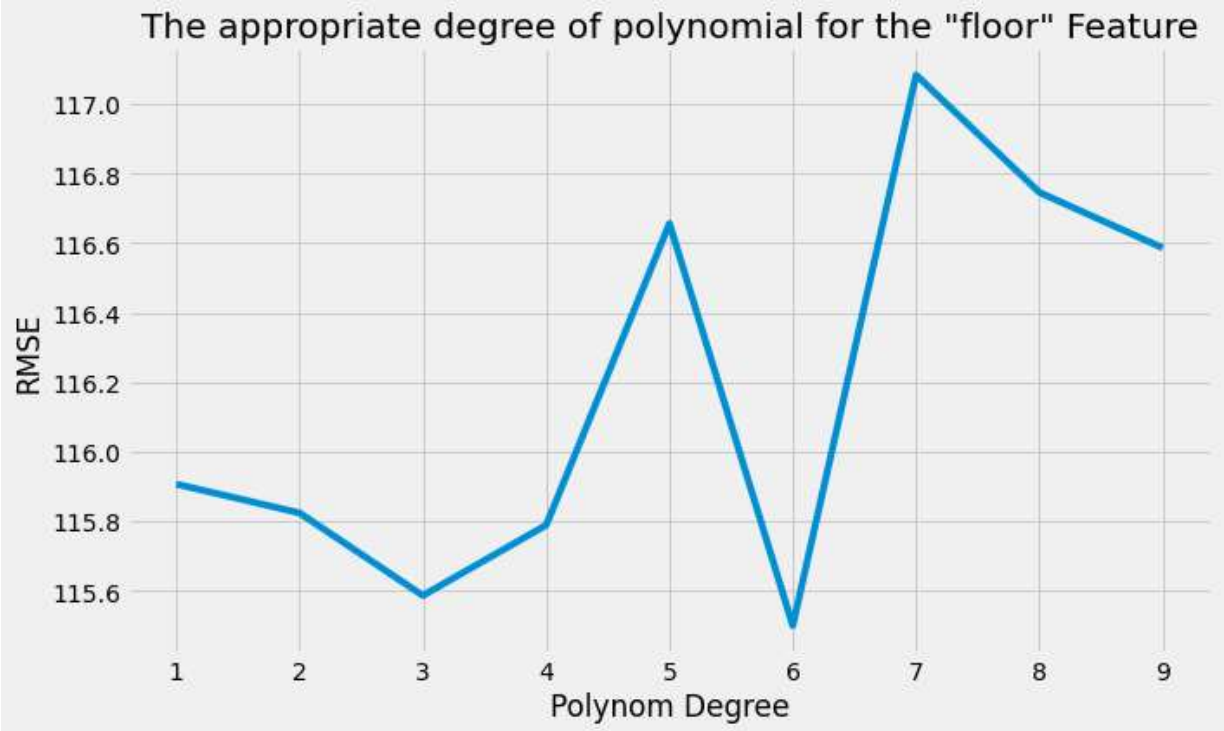
In [256…

```
for deg in degrees:
    poly_features = PolynomialFeatures(degree=deg,include_bias=False)

    X_poly_train = poly_features.fit_transform(X3_train)

    # Linear Reg.
    poly_reg = LinearRegression().fit(X_poly_train,y_train)

    # Comparison with test set
    X_poly_test = poly_features.fit_transform(X3_test)
    poly_predict = poly_reg.predict(X_poly_test)
    poly_mse = mean_squared_error(y_test,poly_predict)
    poly_rmse = np.sqrt(mean_squared_error(y_test,poly_predict))
    rmses.append(poly_rmse)

    if min_rmse > poly_rmse:
        min_rmse = poly_rmse
        min_deg = deg
```

In [257…

```
plt.figure(figsize=(10,6))
plt.title('The appropriate degree of polynomial for the "floor" Feature')
sns.lineplot(x=degrees,y=rmses)
plt.xlabel('Polynom Degree')
```

```
plt.ylabel('RMSE')
plt.show()
```



4-) age

In [258…

```
rmses = []
degrees = np.arange(1,10)
min_rmse, min_deg = 1e10, 0
```

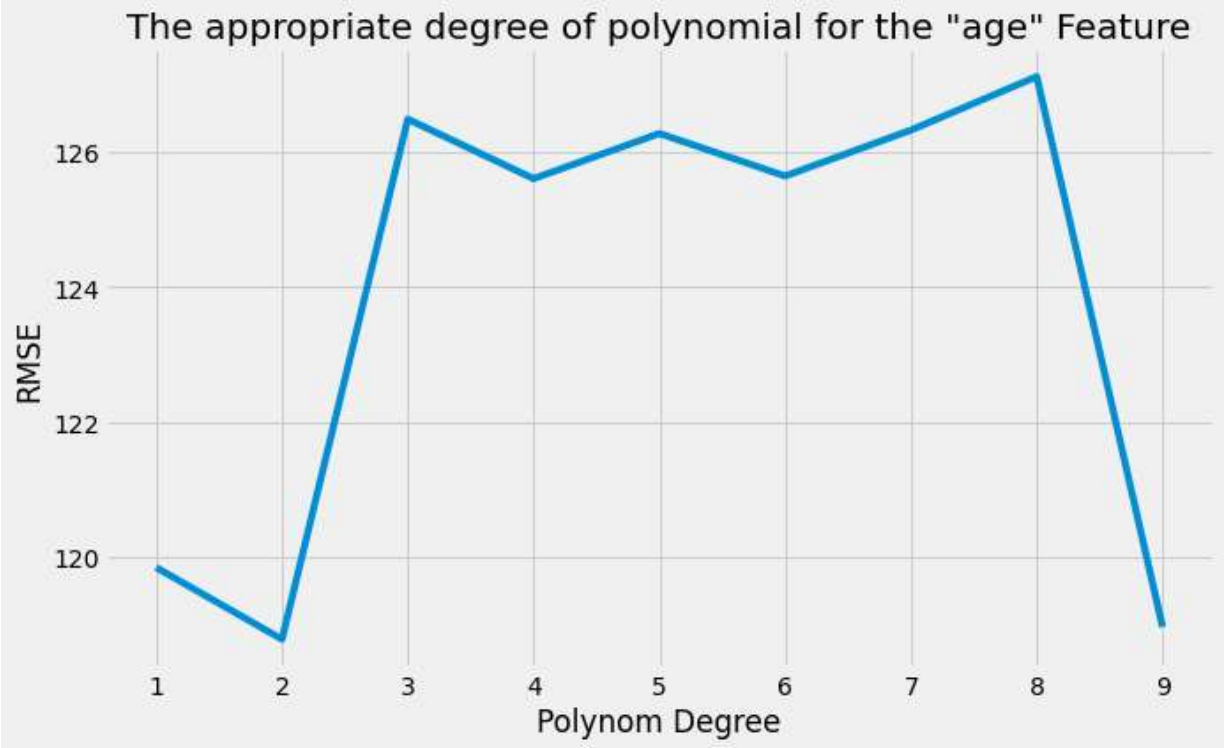
In [259…

```
for deg in degrees:
    poly_features = PolynomialFeatures(degree=deg,include_bias=False)

    X_poly_train = poly_features.fit_transform(X4_train)

    # Linear Reg.
    poly_reg = LinearRegression().fit(X_poly_train,y_train)

    # Comparison with test set
    X_poly_test = poly_features.fit_transform(X4_test)
    poly_predict = poly_reg.predict(X_poly_test)
    poly_mse = mean_squared_error(y_test,poly_predict)
    poly_rmse = np.sqrt(mean_squared_error(y_test,poly_predict))
    rmses.append(poly_rmse)

    if min_rmse > poly_rmse:
        min_rmse = poly_rmse
        min_deg = deg
```

In [260…

```
plt.figure(figsize=(10,6))
plt.title('The appropriate degree of polynomial for the "age" Feature')
sns.lineplot(x=degrees,y=rmses)
```

```
plt.xlabel('Polynom Degree')
plt.ylabel('RMSE')
plt.show()
```



The appropriate polynomial degrees of the features are determined as (1,7,6,2), respectively.

MODEL

In [261…

```
poly_features = PolynomialFeatures(degree=1)
X1_train = poly_features.fit_transform(X1_train)
X1_test = poly_features.fit_transform(X1_test)

poly_features = PolynomialFeatures(degree=7)
X2_train = poly_features.fit_transform(X2_train)
X2_test = poly_features.fit_transform(X2_test)

poly_features = PolynomialFeatures(degree=6)
X3_train = poly_features.fit_transform(X3_train)
X3_test = poly_features.fit_transform(X3_test)

poly_features = PolynomialFeatures(degree=2)
X4_train = poly_features.fit_transform(X4_train)
X4_test = poly_features.fit_transform(X4_test)
```

In [262…

```
X_train1 = np.concatenate((X1_train,X2_train,X3_train,X4_train),axis=1)
X_test1 = np.concatenate((X1_test,X2_test,X3_test,X4_test),axis=1)
```

Let's create and add interaction terms

In [263…
```
X_train['int1'] = X_train['rooms'].mul(X_train['m2'])
X_train['int2'] = X_train['rooms'].mul(X_train['floor'])
X_train['int3'] = X_train['rooms'].mul(X_train['age'])
X_train['int4'] = X_train['m2'].mul(X_train['floor'])
X_train['int5'] = X_train['m2'].mul(X_train['age'])
X_train['int6'] = X_train['floor'].mul(X_train['age'])
```

In [264…
```
X_test['int1'] = X_test['rooms'].mul(X_test['m2'])
X_test['int2'] = X_test['rooms'].mul(X_test['floor'])
X_test['int3'] = X_test['rooms'].mul(X_test['age'])
X_test['int4'] = X_test['m2'].mul(X_test['floor'])
X_test['int5'] = X_test['m2'].mul(X_test['age'])
X_test['int6'] = X_test['floor'].mul(X_test['age'])
```

In [265…
```
X_train2 = np.array(X_train[['int1','int2','int3','int4','int5','int6']])

X_test2 = np.array(X_test[['int1','int2','int3','int4','int5','int6']])
```

In [266…
```
X_train1 = np.concatenate((X_train1, X_train2),axis=1)
X_test1 = np.concatenate((X_test1, X_test2),axis=1)
```

Let's train the polynomial regression algorithm

In [267…
```
poly_reg = LinearRegression().fit(X_train1,y_train)
```

In [269…
```
print(f'Polynomial Reg Train R2 :{poly_reg.score(X_train1,y_train)}')
print(f'Polynomial Reg Test R2 :{poly_reg.score(X_test1,y_test)}')
```

```
Polynomial Reg Train R2 :0.8075964609487128
Polynomial Reg Test R2 :0.7578200785872634
```

Comparing training dataset predictions with actual values

In [271…
```
y_pred_train = poly_reg.predict(X_train1)

for i, prediction in enumerate(y_pred_train):
    print(f'pred_price : {prediction}  real_price : {y[i]}')
```

```
pred_price : 542.1904180595153  real_price : 475
pred_price : 669.020686788061  real_price : 475
pred_price : 532.7479705006466  real_price : 450
pred_price : 689.9639708135431  real_price : 450
pred_price : 685.0236237975081  real_price : 475
pred_price : 650.8048271429283  real_price : 475
pred_price : 558.6660362000924  real_price : 475
pred_price : 658.5126696823539  real_price : 475
pred_price : 545.7848340246027  real_price : 450
```

pred_price : 670.1966108909941 real_price : 475
pred_price : 869.834685123862 real_price : 475
pred_price : 538.6365693256479 real_price : 525
pred_price : 774.823158753746 real_price : 550
pred_price : 517.8264024684107 real_price : 575
pred_price : 565.8317669788222 real_price : 550
pred_price : 678.258792458398 real_price : 575
pred_price : 547.3225670993819 real_price : 500
pred_price : 688.5197131911908 real_price : 550
pred_price : 657.8737710173986 real_price : 525
pred_price : 495.9785430794476 real_price : 500
pred_price : 561.7800971344226 real_price : 550
pred_price : 692.8125776568817 real_price : 500
pred_price : 705.5187370722673 real_price : 550
pred_price : 634.9118028233755 real_price : 550
pred_price : 649.2138942561945 real_price : 575
pred_price : 655.2243963525801 real_price : 500
pred_price : 471.40406161657336 real_price : 500
pred_price : 527.2238570264675 real_price : 525
pred_price : 626.9915924002456 real_price : 525
pred_price : 699.0877017650822 real_price : 575
pred_price : 651.1765038766392 real_price : 500
pred_price : 497.975951842442 real_price : 500
pred_price : 618.5998197934135 real_price : 575
pred_price : 487.46642668613674 real_price : 500
pred_price : 875.3076598570573 real_price : 575
pred_price : 639.8277058702737 real_price : 525
pred_price : 676.3322300877666 real_price : 500
pred_price : 479.05850211407085 real_price : 500
pred_price : 621.8934872401463 real_price : 625
pred_price : 678.1237551707367 real_price : 500
pred_price : 493.7231772177514 real_price : 600
pred_price : 491.61325160819285 real_price : 625
pred_price : 663.3219845953867 real_price : 625
pred_price : 482.6703528520851 real_price : 600
pred_price : 555.5421862903244 real_price : 550
pred_price : 648.8614166971005 real_price : 575
pred_price : 664.1016160973016 real_price : 500
pred_price : 656.4091673735005 real_price : 525
pred_price : 653.9726691455475 real_price : 625
pred_price : 486.19271696053727 real_price : 525
pred_price : 681.6643629420727 real_price : 550
pred_price : 682.2099060522308 real_price : 600
pred_price : 666.8030167778528 real_price : 575
pred_price : 488.80747589960845 real_price : 575
pred_price : 683.9173841027408 real_price : 625
pred_price : 779.5203173888589 real_price : 550
pred_price : 656.4367318212174 real_price : 500
pred_price : 697.0600993786869 real_price : 550
pred_price : 670.2405710862043 real_price : 700
pred_price : 529.0319348439556 real_price : 675
pred_price : 663.5556910619783 real_price : 625
pred_price : 559.8289229418446 real_price : 625
pred_price : 552.2522014337704 real_price : 650
pred_price : 754.8410062768493 real_price : 675
pred_price : 770.4121154852331 real_price : 700
pred_price : 511.00493477904126 real_price : 725
pred_price : 491.5090323159824 real_price : 700
pred_price : 713.3651169533663 real_price : 675
pred_price : 655.9901780311887 real_price : 650

pred_price : 661.4105189020887 real_price : 725
pred_price : 675.7662638213045 real_price : 625
pred_price : 530.341326159982 real_price : 675
pred_price : 651.3230720985706 real_price : 675
pred_price : 652.6170903729547 real_price : 725
pred_price : 642.1808111305621 real_price : 675
pred_price : 866.2846110794056 real_price : 600
pred_price : 762.7045877697204 real_price : 650
pred_price : 515.7195778771642 real_price : 600
pred_price : 673.5895264666837 real_price : 675
pred_price : 766.223063612248 real_price : 650
pred_price : 767.2401740832796 real_price : 650
pred_price : 682.962609245253 real_price : 625
pred_price : 686.7680791878814 real_price : 600
pred_price : 512.1169593928502 real_price : 625
pred_price : 532.9659798699907 real_price : 675
pred_price : 790.2937046971932 real_price : 650
pred_price : 762.0469357826408 real_price : 675
pred_price : 682.6370858203157 real_price : 725
pred_price : 838.2802626164051 real_price : 600
pred_price : 630.2438058757474 real_price : 700
pred_price : 490.8581396462108 real_price : 725
pred_price : 648.7874628288231 real_price : 700
pred_price : 519.8461655907489 real_price : 625
pred_price : 716.8394508580644 real_price : 675
pred_price : 700.7186333156573 real_price : 600
pred_price : 750.6314421022984 real_price : 625
pred_price : 561.8369334409002 real_price : 650
pred_price : 662.0185323658046 real_price : 625
pred_price : 546.3203837078112 real_price : 675
pred_price : 495.5629624387151 real_price : 600
pred_price : 666.6509055657665 real_price : 650
pred_price : 686.6718878456126 real_price : 675
pred_price : 659.2752818577504 real_price : 650
pred_price : 733.2598742632621 real_price : 600
pred_price : 751.5465972134774 real_price : 675
pred_price : 635.2316794201364 real_price : 675
pred_price : 485.39248051784705 real_price : 625
pred_price : 510.3157442236155 real_price : 650
pred_price : 682.6265414858138 real_price : 600
pred_price : 689.4310298145905 real_price : 600
pred_price : 668.8195794568459 real_price : 650
pred_price : 500.50108416044566 real_price : 675
pred_price : 683.0206388807568 real_price : 650
pred_price : 765.5570741961532 real_price : 675
pred_price : 678.2384122132247 real_price : 650
pred_price : 688.1493787902294 real_price : 600
pred_price : 745.3910217669912 real_price : 600
pred_price : 692.5951590268672 real_price : 625
pred_price : 595.0208534937414 real_price : 675
pred_price : 678.3718202491949 real_price : 675
pred_price : 595.1911637805682 real_price : 600
pred_price : 891.7468049579624 real_price : 650
pred_price : 689.2761532874177 real_price : 650
pred_price : 531.4271214929653 real_price : 675
pred_price : 674.5221569882818 real_price : 650
pred_price : 537.832011886618 real_price : 600
pred_price : 546.3308455981972 real_price : 675
pred_price : 721.9156313710959 real_price : 600
pred_price : 918.6039273672508 real_price : 650

```
pred_price : 664.5696302356158  real_price : 600
pred_price : 574.9707423610311  real_price : 675
pred_price : 713.3617318591063  real_price : 625
pred_price : 751.5580382462543  real_price : 650
pred_price : 537.5923130939631  real_price : 625
pred_price : 577.7193127464528  real_price : 675
pred_price : 728.0879430193239  real_price : 675
pred_price : 721.3855989315856  real_price : 675
pred_price : 690.8178054187523  real_price : 650
pred_price : 660.6172374803394  real_price : 675
pred_price : 651.0146479201943  real_price : 625
pred_price : 643.4409863641926  real_price : 625
pred_price : 678.5254072585489  real_price : 650
pred_price : 671.5464383332259  real_price : 650
pred_price : 688.2933895058162  real_price : 725
pred_price : 943.0466733676843  real_price : 750
pred_price : 518.4877205582819  real_price : 750
pred_price : 683.2244219222124  real_price : 700
pred_price : 685.3457652320579  real_price : 725
pred_price : 535.3300842598914  real_price : 725
pred_price : 691.041031686815  real_price : 700
pred_price : 690.5075873821555  real_price : 775
pred_price : 651.8585630356905  real_price : 750
```

Comparing test dataset predictions with actual values

In [272…

```
y_pred_test = poly_reg.predict(X_test1)

for i, prediction in enumerate(y_pred_test):
    print(f'pred_price : {prediction}  real_price : {y[i]}')
```

```
pred_price : 696.6052770299833  real_price : 475
pred_price : 667.1789126195239  real_price : 475
pred_price : 769.9294410409906  real_price : 450
pred_price : 663.8415811153594  real_price : 450
pred_price : 684.6118655105341  real_price : 475
pred_price : 509.2362062612027  real_price : 475
pred_price : 503.0927074493228  real_price : 475
pred_price : 505.26052719254074  real_price : 475
pred_price : 631.3544574664178  real_price : 450
pred_price : 668.2969459042655  real_price : 475
pred_price : 654.2992520374831  real_price : 475
pred_price : 724.9418296581356  real_price : 525
pred_price : 510.5270188799699  real_price : 550
pred_price : 733.3407037463406  real_price : 575
pred_price : 695.685994103082  real_price : 550
pred_price : 817.2189116386729  real_price : 575
pred_price : 722.2182955982283  real_price : 500
pred_price : 673.2606390391014  real_price : 550
pred_price : 736.8336332692492  real_price : 525
pred_price : 504.37431306589895  real_price : 500
pred_price : 535.7511287022538  real_price : 550
pred_price : 520.7657244554288  real_price : 500
pred_price : 660.4806369533547  real_price : 550
pred_price : 649.2119200917517  real_price : 550
pred_price : 665.7009528443235  real_price : 575
pred_price : 642.3159711420161  real_price : 500
pred_price : 702.5071558789281  real_price : 500
pred_price : 509.1227290151877  real_price : 525
```

```
pred_price : 545.5641888121285  real_price : 525
pred_price : 675.9499587016984  real_price : 575
pred_price : 762.5441739987294  real_price : 500
pred_price : 676.8938535545316  real_price : 500
pred_price : 517.5502211620372  real_price : 575
pred_price : 716.8130265471267  real_price : 500
pred_price : 690.7369819542513  real_price : 575
pred_price : 751.8350455546318  real_price : 525
pred_price : 706.6289132877358  real_price : 500
pred_price : 849.6962857022376  real_price : 500
pred_price : 886.981408671047  real_price : 625
```

In [274…

```
pred_price = np.append(y_pred_train,y_pred_test)
df['pred_price'] = pred_price
```

Graph

In [276…

```
plt.figure(figsize=(10,6))
plt.style.use('fivethirtyeight')
plt.title('Multiple Polynomial 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()
```