Analysis on Predicting House Prices in New Taipei

1 Introduction

Housing prices has been a concern for a long time due to many reasons, such as house affordability and profitting from house se lling. There are a huge variety of factors contributing to the change of housing prices. In common sense, houses that are closer to downtown and have convenient transportation nearby usually have higher prices. Moreover, the depreciation of house and the housing market also have a significant influence on the housing prices.

In this analysis, we first aim to build a prediction model to estimate the housing prices. Then, we study the impact of the certain features of the real estates on the prices of housing. This analysis is intended to support a lawyer from New Taipei City for a housing fraud case by comparing the estimated housing prices to the actual prices. Our client also concerns about how the actual and expected prices deviate. By investigating this regression problem, we found the location of houses matters heavily on the housing price. In addition, the housing market also played an important role in housing prices and as the prices increase as time passes.

2 Methods

There are two main goals of our analysis. One is to predict the house price of the unit area in New Taipei and the client will be able to compare the estimated prices to the actual ones they have. The prediction model should be easy to understand for our clients who do not have a statistical background. Another is to figure out how different factors such as the house age and the number of convenience stores nearby can impact the housing price using the graphics. To address our goals, we conducted Generalized Additive Models (GAMs) with smoothing splines for this prediction (regression) task. This aligned with the data and goals posed.

The real estate valuation data were collected from Sindian District, New Taipei City. The dependent variable is the house price of unite area that is continuous and estimating real estate valuation is a regression problem. Using GAMs allows us not to assume a priori any specific form of the relationship between the dependent variable and covariates. More importantly, GAMs are particularly useful to reveal and estimate non-linear effects of covariates on the dependent variable.

Before going deeper into the GAMs, we'd like to get an overview of generalized linear models (GLMs). The response is a weighted sum of covariates in GLMs which allows us to build a linear relationship between even though their underlying relationship is not linear. Thus, GAMs serve as an extension of GLMs by relaxing the linearity restriction and instead assume that the response can be modelled by a sum of arbitrary functions of each covariate. This makes GAMs easy to intepret as the regression model is additive and we are able to

interpret the marginal impact of a single variable which does not depend on other variables in the model. This is to say we can tell the effect of a certain feature of real estates on the housing prices.

In detail, we use a flexible function called a spline in GAMs and the sum of many splines forms a GAM. The GAMs can be highly flexible as the result depends on the choice of splines, while they still have some explainability of a linear regression. There are many choices of splines and in our modelling we chose the smoothing splines. The smoothing splines allow us to control overfitting by regularization. Regularization means that we use a penalty term to control how wiggle the spline is. It is worthwhile to care about the overfitting problem as overfitting could make our model useless when new data comes in. The penalty term also trades off bias-variance in the smoothing spline GAM. If we have a large penalty term, the GAM will be less wiggle and there will be lower variance and higher bias. When the penalty is small, the GAM can be wiggle so the variance is high but bias is low. Tuning the number of splines and penalty term increases the complexity of GAMs. So a general rule of thrumb is to use a high number of splines and use cross-validation of penalty terms to find the model that generalizes best.

In general, the GAM we used takes the form of

$$g(E[Y \mid X = x]) = \sum_{j=1}^{p} f_j\left(x_j\right), \tag{1}$$

where a bunch of splines f_j was chosen and we added them together. f_j is optimized with a penalty

$$\hat{f}_1, \ldots, \hat{f}_p = \arg\min_{f_1,\ldots,f_p} \ell\left(y; f_1, \ldots, f_p\right) + \sum_{j=1}^{p} \lambda_j \int \left[f_j''(t)\right]^2 dt, \tag{2}$$

where λ_j is the penalty term and ℓ is the log-likelihood.

In order to evaluate our model's performance, we used the mean squared error (MSE) metric. MSE is calculated as the avergae of the sum of squares between the predicted value on and real value on our test data set. A small value of MSE indicates a better model for our prediction task. However, this metric is sensitive to outliers and it punishes larger error more.

3 Results

3.1 Overview of the Data

The data set contains 414 rows and 7 variables where each row represents one real estate and each column represents features of a certain real estate. The *house price of unit area* is the dependent variable in our model. The *transaction date* was presented with a real number. For example, transaction date June 2012 was presented as 2012.5 and December 2012 was presented as 2013.0. Other covariates were house age (in year unit), the number of convenience stores in the living circle on foot where the distance is smaller than 500m, the latitude and longtitude of houses.

Data Pre-processing For the purpose of exploratory data analysis, I formatted the transaction date into explicit columns of year and month. For our prediction task, I split 2/3 of the data set as the training set and left as the test set.

Baseline Table Below is the baseline table that displays the median and inner quartile range (pr percentage of categorical variables) for the variables of interest.

Variable	Median (IQR) or Percentage
transaction date	Feb 2013(Nov 2012, May 2013)
house age (in year unit)	16.10(9.02, 28.15)
distance to the nearest MRT station (in meter unit)	492.23(289.32, 1454.28)
number of convenience stores	4(1, 6)
latitude (in degree unit)	24.97(24.96, 24.98)
longitude (in degree unit)	121.54(121.53, 121.54)
house price of unit area (10000 New Taiwan Dollar/Ping)	38.45(27.70, 46.60)

Table 1: Baseline Table for variables of interest

3.2 Results and Analysis

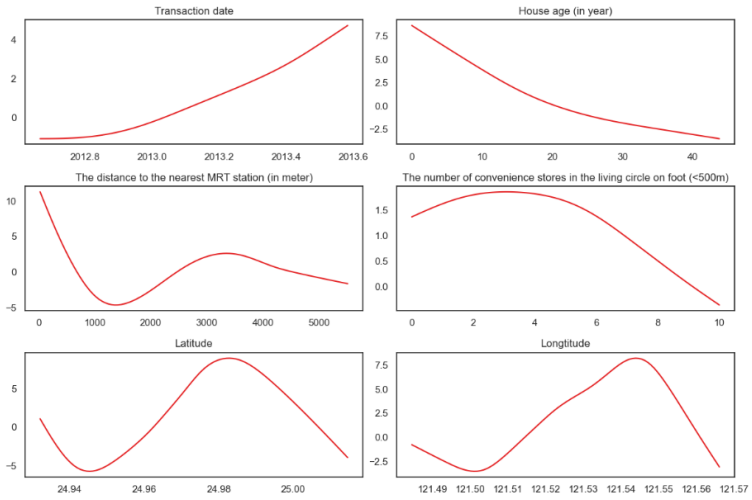


Figure 1: How each feature impacts the overall prediction of housing prices

Firstly, we split the dataset where 2/3 of it is used as a training set and 1/3 as a test set. GAM was applied for this regression problem and we implemented it with **pygam** package in python. Specifically, we used linear GAM for regression. The outcome is the prediction of the housing price of unit area. Observed from the EDA (see appendix), there could be some interaction between the latitude and longitude. Hence, we would include this interaction in our linear GAM and compared its performance to the one without interaction.
Both GAMs were given a list of penalty terms (lambdas) and we performed a grid-search over multiple lambdas to see if we could improve our model. We also chose the number of splines (20) for our GAMs. **pygam** automatically tuned the model if lambdas are given and we obtained the model with lowest generalized cross-validation score. Those two GAMs obtained similar MSEs, 71.51 for one without interaction and 71.90 for one with interaction. Hence, we focused on the GAM without interaction to determine how different factors could influence housing prices.

Figure 1 above illustrates the contribution of each feature to the overall prediction of housing prices. We can find that the housing prices decrease as longer the house age is while the prices increase as time passes. Both covariates shows less non-linear effect on the housing prices. The spline curve for the distance to the MRT station varies a lot. The housing price of unit area increases by approximately 7 units at 3.2k meters away from the nearest MRT station compared to the price of house at 1.3k meters away from the MRT station. This finding is counterintuitive as a convenient transportation nearby can result in a growth in housing prices. However, the decreasing in prices could account for the noise caused by the transportation nearby.
Moreover, the housing prices only increase by 0.5 unit if there are 4 convenience stores nearby compared to the one with no nearby stores. It is surprising to find as the number of convenience store gets even more the housing prices will decrease. From last two pairs of plots in Figure 1, we can find that the latitude and longitude have similar non-linear trend of influence on the housing prices. Both latitude and longitude obtained a peak for the increase in housing prices. For instance, the price of house expected increases by 10 units at 121.55 degrees compared to the one at 121.49 degrees.

4 Conclusion

In our analysis, we utilized the smoothing spline GAMs to predict the housing prices from New Taipei City and analyzed how each covariate can influence the prices of houses from the spline curves. The performance of our prediction model was measured by MSE. From the spline curves, we can find the transaction date and house age have a more linear impact on housing prices where houses transacted earlier with less age will have a higher price per unit. Moreover, the location of house influenced the housing prices heavily. The closer to certain area (such as the downtown) will definitely increase the housing prices.
The MSEs of our GAMs might not be small enough which means the prediction performance of our model might be the best. If time permitted, I would tune more hyperparameters to get a more fitted model. Also, I did not tune the number of splines in our model but used 20 as a good starting point.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
```

1 Load Data

In [2]:
```
dat = pd.read_excel('Real estate valuation data set.xlsx')
print(dat.shape)
dat.head()
```

```
(414, 8)
```

Out[2]:

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
0	1	2012.916667	32.0	84.87882	10	24.98298	121.54024	37.9
1	2	2012.916667	19.5	306.59470	9	24.98034	121.53951	42.2
2	3	2013.583333	13.3	561.98450	5	24.98746	121.54391	47.3
3	4	2013.500000	13.3	561.98450	5	24.98746	121.54391	54.8
4	5	2012.833333	5.0	390.56840	5	24.97937	121.54245	43.1

In [3]:
```
print(dat.dtypes)
```

```
No                                      int64
X1 transaction date                   float64
X2 house age                          float64
X3 distance to the nearest MRT station    float64
X4 number of convenience stores         int64
X5 latitude                           float64
X6 longitude                          float64
Y house price of unit area            float64
dtype: object
```

In [4]:
```
dat.iloc[:,1].unique()
# data were collected between June 2021 and May 2013 according to the paper
# June = 6/12, July = 7/12, Dec = 0/12
```

Out[4]:
```
array([2012.9166667, 2013.5833333, 2013.5      , 2012.8333333,
       2012.6666667, 2013.4166667, 2013.0833333, 2013.3333333,
       2013.25     , 2012.75     , 2013.       , 2013.1666667])
```

In [5]:
```
# format transaction date
def transaction_date_to_year_month(date):
    """
    Input: a pd.Series of scaled transaction dates
    Output: a dataframe of year and month
    """
    # separate year and month
    ym = [math.modf(date[i]) for i in range(len(date))]
    l = list(zip(*ym))
    year = [int(_) for _ in l[1]]
    month = [int(_*12) for _ in l[0]]
    new_date = pd.DataFrame(list(zip(year, month)), columns=['year', 'month'])

    # convert month number to month name
    new_date['year'] = np.where(new_date['month']==0, 2012, 2013) # 2013.0 = Dec 2
    look_up = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'June',
               7: 'July', 8: 'Aug', 9: 'Sept', 10: 'Oct', 11: 'Nov', 0: 'Dec'}
    new_date['month'] = new_date['month'].apply(lambda x: look_up[x])
    return new_date
```

In [6]:
```
year_and_month = transaction_date_to_year_month(dat.iloc[:,1])
new_dat = pd.concat([year_and_month, dat], axis=1)
new_dat.tail()
```

Out[6]:

	year	month	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
409	2012	Dec	410	2013.000000	13.7	4082.01500	0	24.94155	121.50381	15.4
410	2013	Aug	411	2012.666667	5.6	90.45606	9	24.97433	121.54310	50.0
411	2013	Mar	412	2013.250000	18.8	390.96960	7	24.97923	121.53986	40.6
412	2012	Dec	413	2013.000000	8.1	104.81010	5	24.96674	121.54067	52.5
413	2013	June	414	2013.500000	6.5	90.45606	9	24.97433	121.54310	63.9

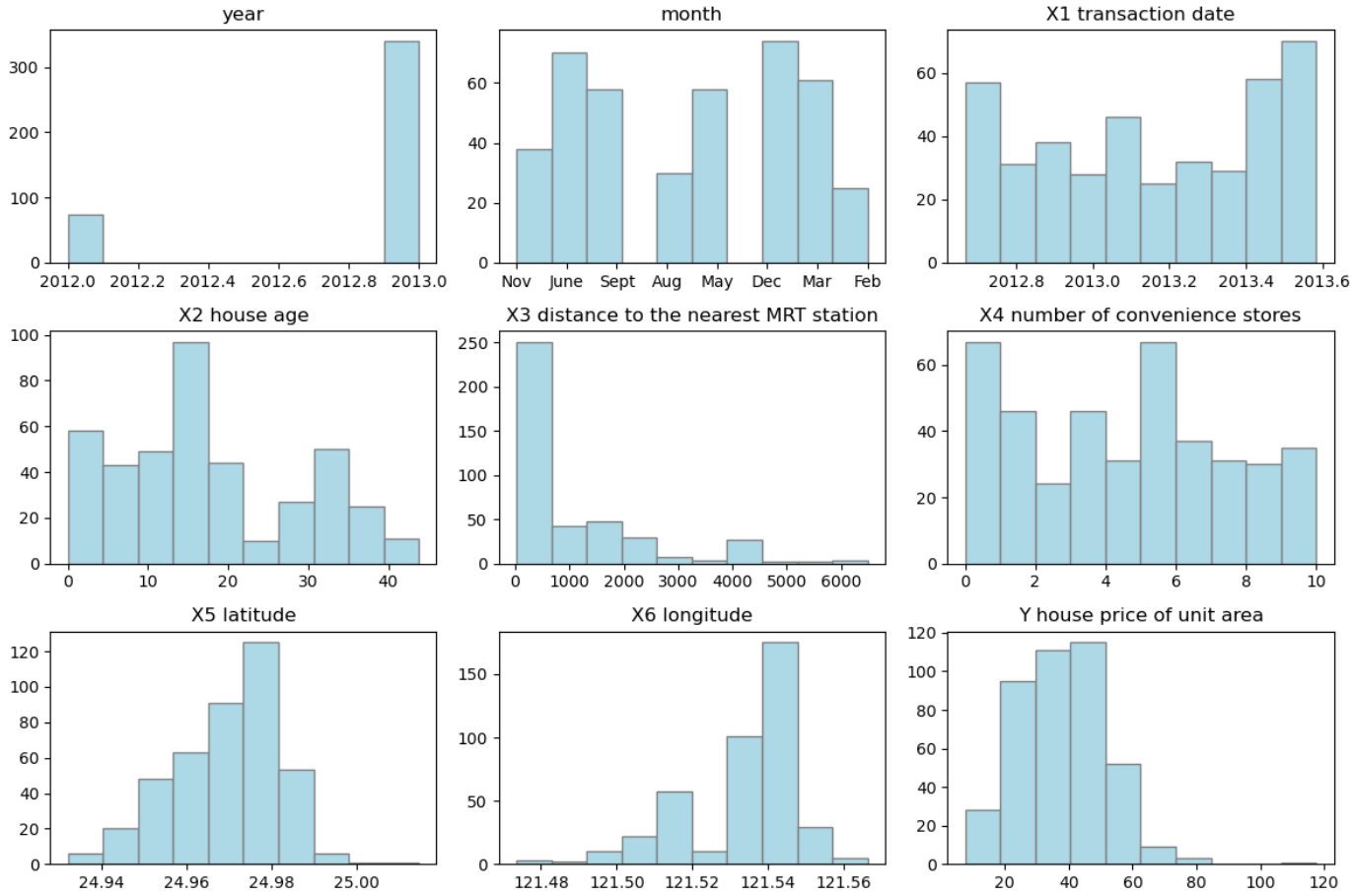
In [7]:
```
new_dat.isna().sum() # no missing value
```

Out[7]:
```
year                                    0
month                                   0
No                                      0
X1 transaction date                     0
X2 house age                            0
X3 distance to the nearest MRT station  0
X4 number of convenience stores         0
X5 latitude                             0
X6 longitude                            0
Y house price of unit area              0
dtype: int64
```

2 EDA

2.1 Variable Distributions

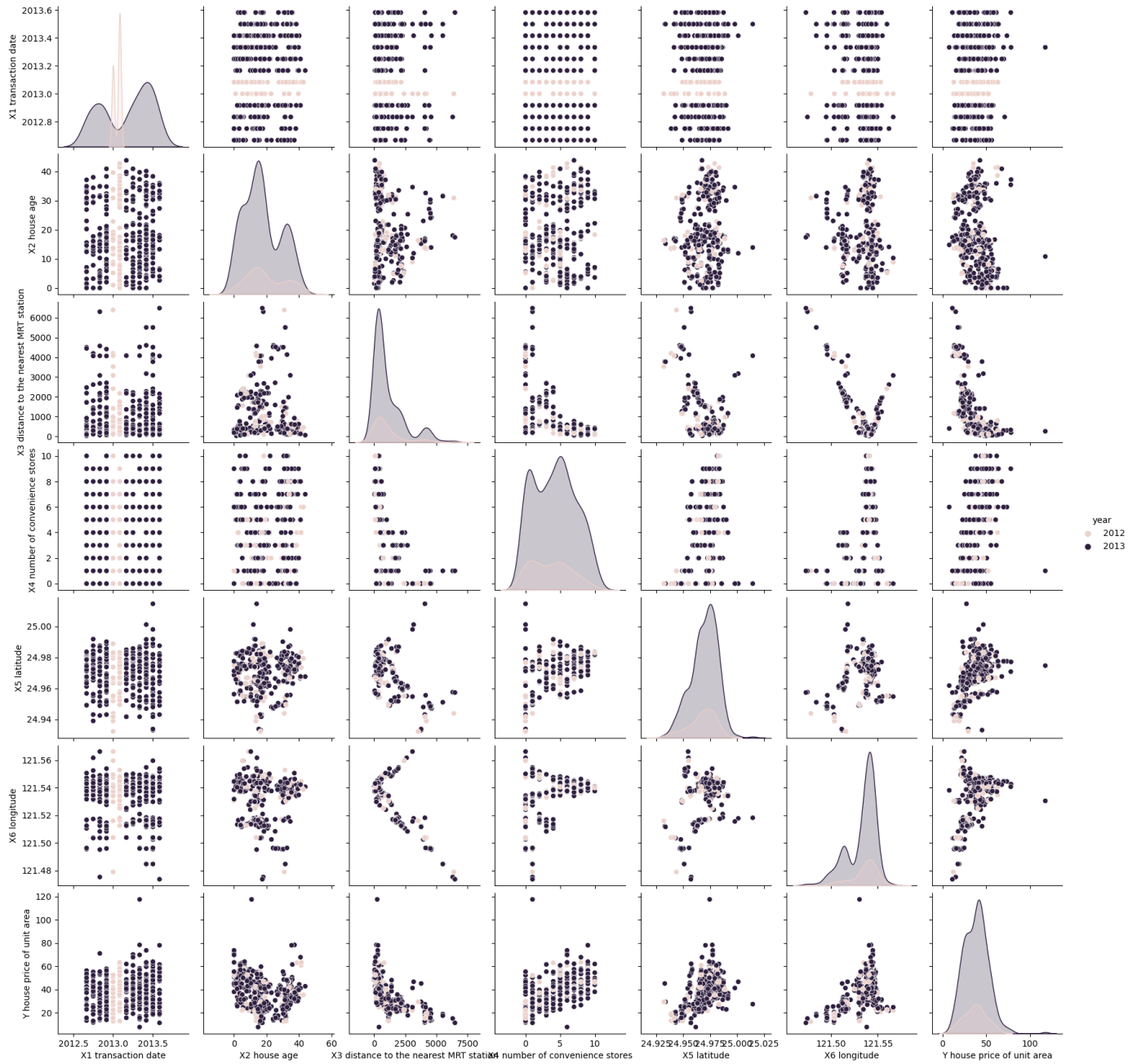
In [8]:
```
plt.rcParams['figure.figsize'] = [12, 8]
def draw_histogram(df, variables, n_rows, n_cols):
    fig = plt.figure()
    for i, var_name in enumerate(variables):
        ax = fig.add_subplot(n_rows,n_cols,i+1)
        df[var_name].hist(ax=ax, color='lightblue', ec='grey')
        ax.set_title(var_name)
        ax.grid(False)
    fig.tight_layout()
    plt.show()

df_eda = new_dat.drop(['No'], axis=1)
draw_histogram(df_eda, df_eda.columns, 3, 3)
```



2.2 Pair Plots

In [9]:
```
sns.pairplot(df_eda, hue='year')
```

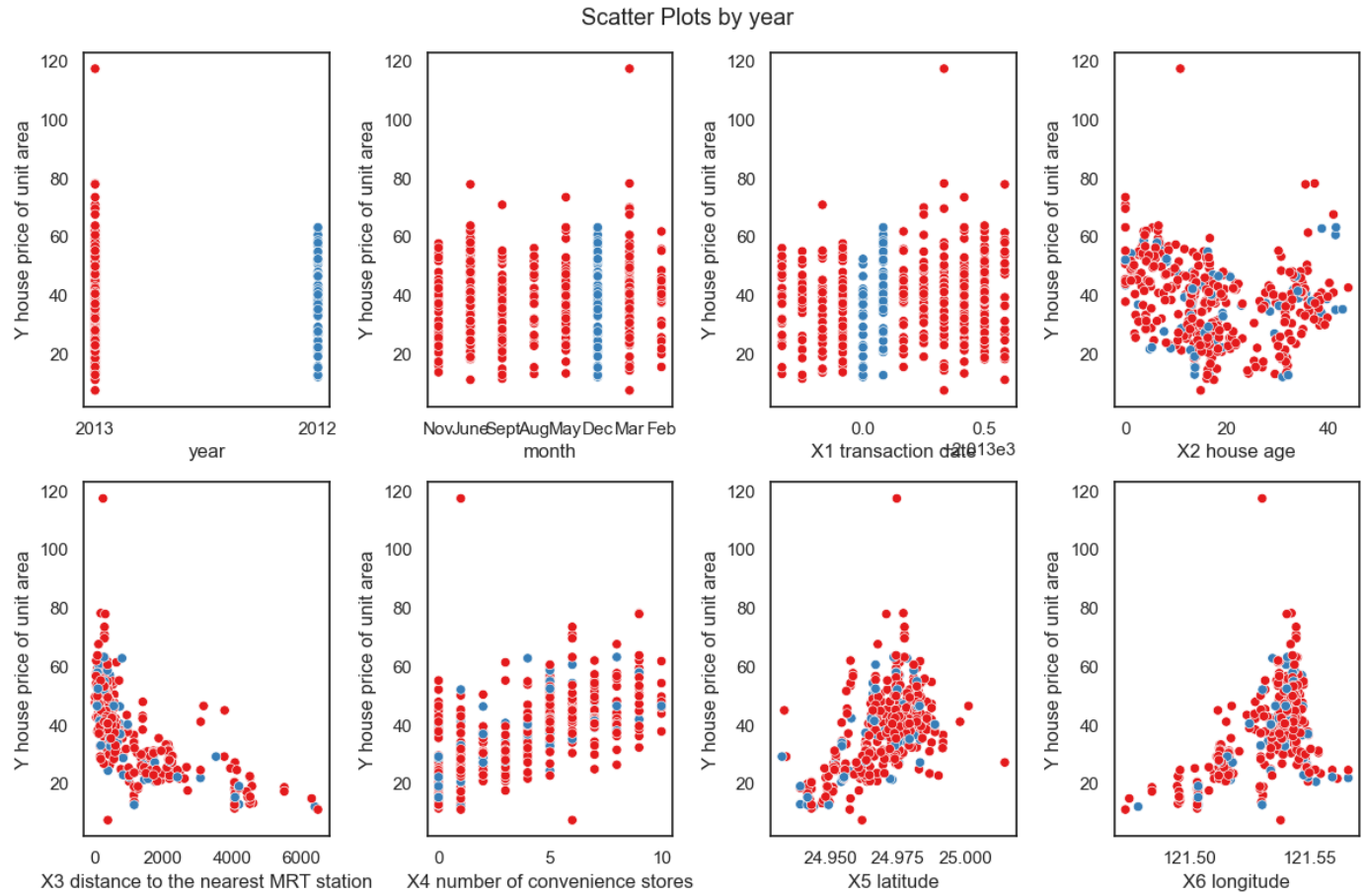
Out[9]: <seaborn.axisgrid.PairGrid at 0x1691c55a0>



2.3 Scatter plots

We want to see how each variable associates with the price.

In [45]:
```
def draw_scatters_bygroup(df, outcome, n_rows, n_cols, group):
    fig = plt.figure()
    variables = df.columns.drop(outcome)
    sns.set(style='white', palette='Set1')
    for i, var_name in enumerate(variables):
        ax = fig.add_subplot(n_rows, n_cols, i+1)
        sns.scatterplot(x=var_name, y=outcome, hue=group, data=df, legend=False, a
    fig.suptitle('Scatter Plots by year')
    fig.tight_layout()
    plt.show()

draw_scatters_bygroup(df_eda, 'Y house price of unit area', 2, 4, 'year')
```



2.4 Pairwise Correlation

In [11]:
```
dat.drop(['No'], axis=1).corr()
```

Out[11]:

	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
X1 transaction date	1.000000	0.017542	0.060880	0.009544	0.035016	-0.041065	0.087529
X2 house age	0.017542	1.000000	0.025622	0.049593	0.054420	-0.048520	-0.210567
X3 distance to the nearest MRT station	0.060880	0.025622	1.000000	-0.602519	-0.591067	-0.806317	-0.673613
X4 number of convenience stores	0.009544	0.049593	-0.602519	1.000000	0.444143	0.449099	0.571005
X5 latitude	0.035016	0.054420	-0.591067	0.444143	1.000000	0.412924	0.546307
X6 longitude	-0.041065	-0.048520	-0.806317	0.449099	0.412924	1.000000	0.523287
Y house price of unit area	0.087529	-0.210567	-0.673613	0.571005	0.546307	0.523287	1.000000

2.5 Baseline Table

In [12]:
```
df_eda['year'] = df_eda['year'].astype(str)
df_eda.describe().round(2).transpose()
```

Out[12]:

	count	mean	std	min	25%	50%	75%	max
X1 transaction date	414.0	2013.15	0.28	2012.67	2012.92	2013.17	2013.42	2013.58
X2 house age	414.0	17.71	11.39	0.00	9.02	16.10	28.15	43.80
X3 distance to the nearest MRT station	414.0	1083.89	1262.11	23.38	289.32	492.23	1454.28	6488.02
X4 number of convenience stores	414.0	4.09	2.95	0.00	1.00	4.00	6.00	10.00
X5 latitude	414.0	24.97	0.01	24.93	24.96	24.97	24.98	25.01
X6 longitude	414.0	121.53	0.02	121.47	121.53	121.54	121.54	121.57
Y house price of unit area	414.0	37.98	13.61	7.60	27.70	38.45	46.60	117.50

In [13]:
```
def get_categorical_percentages(df):
    df_cat = df.select_dtypes(exclude=np.number)
    for var in df_cat.columns:
        perc = df[var].value_counts() / df[var].count()
        print(var)
        print(perc)

get_categorical_percentages(df_eda)
```

```
year
2013    0.821256
2012    0.178744
Name: year, dtype: float64
month
Dec     0.178744
June    0.169082
Mar     0.147343
Sept    0.140097
May     0.140097
Nov     0.091787
Aug     0.072464
Feb     0.060386
Name: month, dtype: float64
```

3 Modelling

3.1 Linear Regression

In [14]:
```
import statsmodels.formula.api as smf
df = dat.drop(['No'], axis=1)
df.columns = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6' ,'y']
linear_mod = smf.ols(formula='y ~ x1 + x2 + x3 + x4 + x5 + x6', data=df).fit()
print(f'AIC: {round(linear_mod.aic, 2)}')
linear_mod.summary()
```

```
AIC: 2987.91
```

Out[14]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.582
Model:	OLS	**Adj. R-squared:**	0.576
Method:	Least Squares	**F-statistic:**	94.60
Date:	Sun, 30 Oct 2022	**Prob (F-statistic):**	4.83e-74
Time:	12:18:17	**Log-Likelihood:**	-1487.0
No. Observations:	414	**AIC:**	2988.
Df Residuals:	407	**BIC:**	3016.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.444e+04	6775.386	-2.132	0.034	-2.78e+04	-1122.863
x1	5.1490	1.557	3.307	0.001	2.088	8.210
x2	-0.2697	0.039	-7.000	0.000	-0.345	-0.194
x3	-0.0045	0.001	-6.250	0.000	-0.006	-0.003
x4	1.1333	0.188	6.023	0.000	0.763	1.503
x5	225.4701	44.566	5.059	0.000	137.862	313.078
x6	-12.4291	48.581	-0.256	0.798	-107.930	83.072

Omnibus:	231.615	**Durbin-Watson:**	2.153
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3598.692
Skew:	2.026	**Prob(JB):**	0.00
Kurtosis:	16.864	**Cond. No.**	3.72e+07

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.72e+07. This might indicate that there are
strong multicollinearity or other numerical problems.

In [15]:
```
# with interaction
linear_mod1 = smf.ols(formula='y ~ x1 + x2 + x3 + x4 + x5*x6', data=df).fit()
print(f'AIC: {round(linear_mod1.aic, 2)}')
linear_mod1.summary()
```

```
AIC: 2955.89
```

Out[15]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.615
Model:	OLS	**Adj. R-squared:**	0.609
Method:	Least Squares	**F-statistic:**	92.78
Date:	Sun, 30 Oct 2022	**Prob (F-statistic):**	3.20e-80
Time:	12:18:17	**Log-Likelihood:**	-1469.9
No. Observations:	414	**AIC:**	2956.
Df Residuals:	406	**BIC:**	2988.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	4.527e+07	7.68e+06	5.895	0.000	3.02e+07	6.04e+07
x1	5.4125	1.497	3.616	0.000	2.470	8.355
x2	-0.2861	0.037	-7.704	0.000	-0.359	-0.213
x3	-0.0055	0.001	-7.775	0.000	-0.007	-0.004
x4	0.7441	0.192	3.866	0.000	0.366	1.123
x5	-1.814e+06	3.08e+05	-5.896	0.000	-2.42e+06	-1.21e+06
x6	-3.727e+05	6.32e+04	-5.897	0.000	-4.97e+05	-2.48e+05
x5:x6	1.493e+04	2531.694	5.897	0.000	9951.710	1.99e+04

Omnibus:	256.717	**Durbin-Watson:**	2.224
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	4745.784
Skew:	2.276	**Prob(JB):**	0.00
Kurtosis:	18.950	**Cond. No.**	7.02e+10

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.23e-12. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Linear regression models could be too simple and result in underfitting. We can also find from the EDA there could be nonlinear trend. So we look at the GAMs.

3.2 GAMs

3.2.1 Linear GAM

pygam (https://pygam.readthedocs.io/en/latest/notebooks/tour_of_pygam.html#Introduction)

- pyGAM can also fit interactions using tensor products via te()
- pygam is a smoothing spline approach (uses regularization)

In [16]:
```
# preprocess data
dat_x = new_dat.drop(['No', 'year', 'month', 'Y house price of unit area'], axis=1
≈ = new_dat['Y house price of unit area']

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(dat_x, dat_y,
                                    random_state=504,
                                    test_size=0.33,
                                    shuffle=True)
```

In [17]:

```
from pygam import GAM, LinearGAM, s, f, te
n_features=6
lams = [np.logspace(-3,3,4)] * n_features
gam = LinearGAM(s(0) + s(1) + s(2) + s(3) + s(4) + s(5)).gridsearch(x_train.values
                                                                    y_train.values
gam.summary()
```

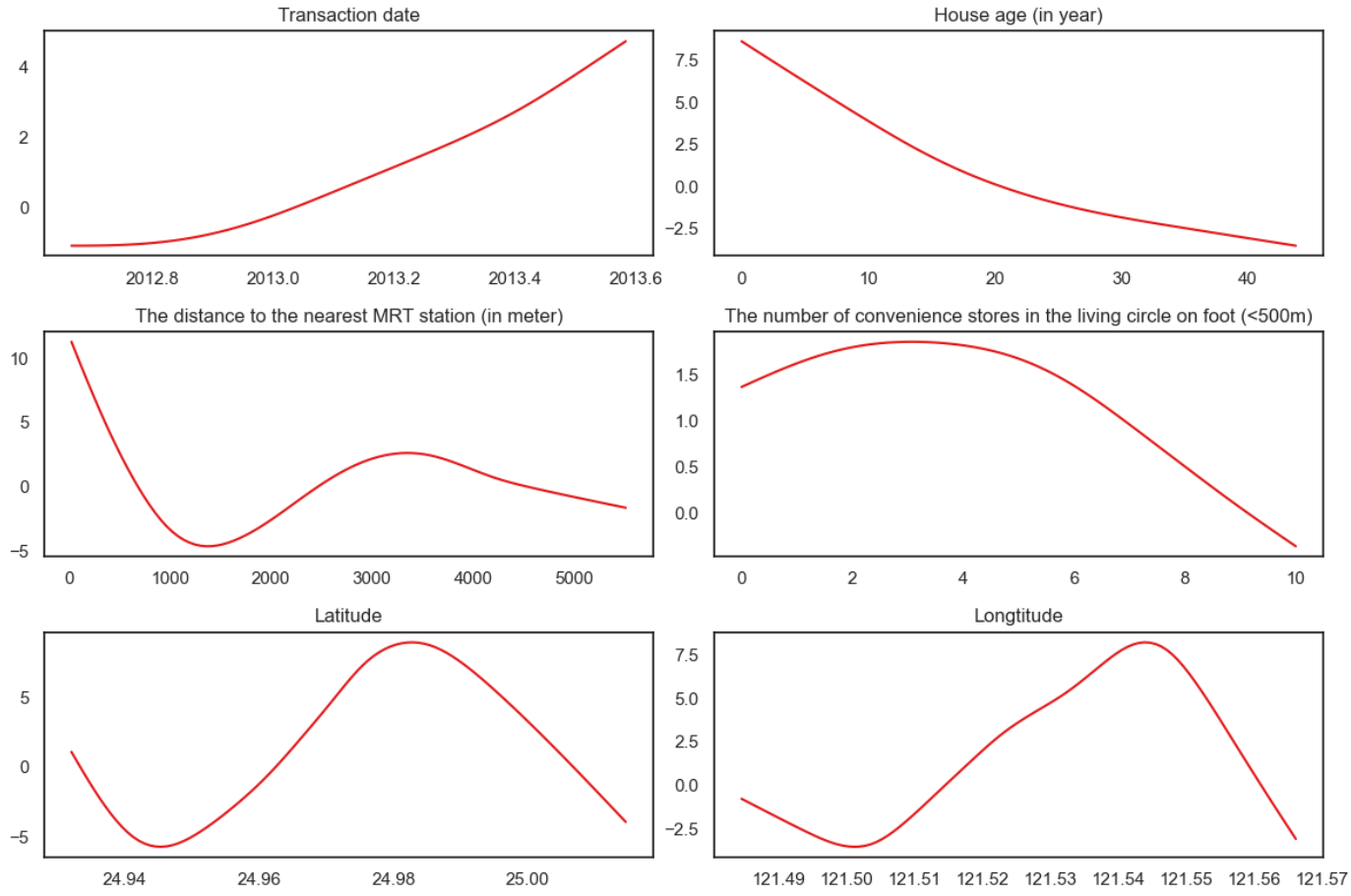
```
100% (4096 of 4096) |####################| Elapsed Time: 0:05:02 Time:  0:05:02
/var/folders/qx/wlpqnbz178962z833f_yq5gh0000gn/T/ipykernel_69703/4129619790.py:6:
UserWarning: KNOWN BUG: p-values computed in this summary are likely much smaller
than they should be.

Please do not make inferences based on these values!

Collaborate on a solution, and stay up to date at:
github.com/dswah/pyGAM/issues/163

  gam.summary()
```

```
LinearGAM
============================================== ================================
==========================
Distribution:                        NormalDist Effective DoF:
20.5235
Link Function:                     IdentityLink Log Likelihood:
-1519.4728
Number of Samples:                          310 AIC:
3081.9928
                                                AICc:
3085.3655
                                                GCV:
60.331
                                                Scale:
53.1867
                                                Pseudo R-Squared:
0.7198
=========================================================================================
==========================
Feature Function                  Lambda               Rank         EDoF
P > x        Sig. Code
================================= ==================== ============ ============
============ ============
s(0)                              [1000.]              20           4.9
7.26e-03     **
s(1)                              [1000.]              20           3.5
1.14e-08     ***
s(2)                              [10.]                20           6.2
1.81e-07     ***
s(3)                              [1000.]              20           2.4
8.89e-01
s(4)                              [10.]                20           2.5
9.86e-10     ***
s(5)                              [10.]                20           1.1
3.41e-02     *
intercept                                              1            0.0
1.11e-16     ***
=========================================================================================
==========================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
WARNING: Fitting splines and a linear function to a feature introduces a model i
dentifiability problem
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized mo
dels or models with
         known smoothing parameters, but when smoothing parameters have been est
imated, the p-values
         are typically lower than they should be, meaning that the tests reject
the null too readily.
```

In [39]:

```
fig = plt.figure()
n_rows, n_cols = 3, 2
titles = ['Transaction date', 'House age (in year)', 'The distance to the nearest
          'The number of convenience stores in the living circle on foot (<500m)',
for i, ax in enumerate(titles):
    XX = gam.generate_X_grid(term=i)
    ax = fig.add_subplot(n_rows, n_cols, i+1)
    ax.plot(XX[:,i], gam.partial_dependence(term=i, X=XX))
    ax.set_title(titles[i])
fig.tight_layout()
```



In [22]:

```
from sklearn.metrics import mean_squared_error
pred1 = gam.predict(x_test.values)
mean_squared_error(y_test, pred1)
```

Out[22]:

```
71.51048318354172
```

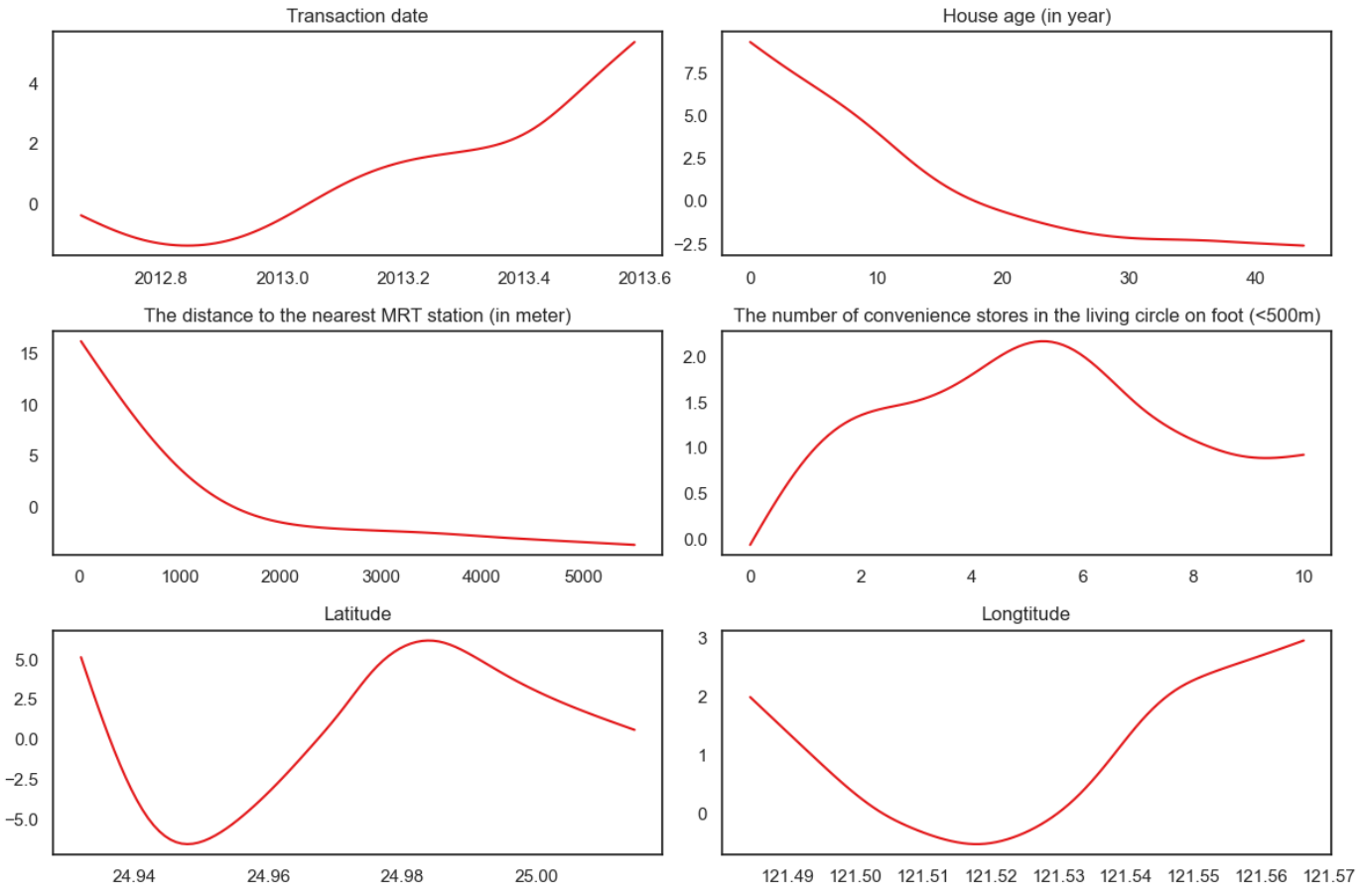
In [30]:
```
lams1= [np.array([0.6, 1, 5, 10, 100])] * n_features
gam1 = LinearGAM(s(0) + s(1) + s(2) + s(3) + s(4) + s(5)).gridsearch(x_train.value
                                                                     y_train.values
gam1.summary()
```

```
LinearGAM
============================================== ================================
========================
Distribution:                          NormalDist Effective DoF:
21.5635
Link Function:                       IdentityLink Log Likelihood:
-1525.7588
Number of Samples:                            310 AIC:
3096.6447
                                                  AICc:
3100.3571
                                                  GCV:
61.9986
                                                  Scale:
54.2875
                                                  Pseudo R-Squared:
0.715
============================================================================================
========================
Feature Function                  Lambda               Rank         EDoF         P
> x        Sig. Code
================================= ==================== ============ ============ =
========== ============
s(0)                              [100.]               20           6.5
8.47e-03     **
s(1)                              [100.]               20           4.6
2.28e-10     ***
s(2)                              [100.]               20           4.7
1.01e-08     ***
s(3)                              [100.]               20           3.1
8.61e-01
s(4)                              [10.]                20           2.2
9.40e-10     ***
s(5)                              [100.]               20           0.5
1.45e-01
intercept                                              1            0.0
1.11e-16     ***
============================================================================================
========================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

WARNING: Fitting splines and a linear function to a feature introduces a model ide
ntifiability problem
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized mode
ls or models with
         known smoothing parameters, but when smoothing parameters have been estim
ated, the p-values
         are typically lower than they should be, meaning that the tests reject th
e null too readily.
```

```
/var/folders/qx/wlpqnbz178962z833f_yq5gh0000gn/T/ipykernel_69703/1772604297.py:1:
```

```
UserWarning: KNOWN BUG: p-values computed in this summary are likely much smaller
than they should be.

Please do not make inferences based on these values!

Collaborate on a solution, and stay up to date at:
github.com/dswah/pyGAM/issues/163

  gam1.summary()
```

In [40]:
```
fig = plt.figure()
n_rows, n_cols = 3, 2
titles = ['Transaction date', 'House age (in year)', 'The distance to the nearest
          'The number of convenience stores in the living circle on foot (<500m)',
for i, ax in enumerate(titles):
    XX = gam1.generate_X_grid(term=i)
    ax = fig.add_subplot(n_rows, n_cols, i+1)
    ax.plot(XX[:,i], gam1.partial_dependence(term=i, X=XX))
    ax.set_title(titles[i])
fig.tight_layout()
```



In [38]:
```
pred2 = gam1.predict(x_test.values)
mean_squared_error(y_test, pred2)
```

Out[38]:
```
73.89985077696912
```

3.2.2 Linear GAM with interaction

In [41]:
```
gam_inte = LinearGAM(s(0) + s(1) + s(2) + s(3) + te(4,5)).gridsearch(x_train.value
                                                                     y_train.value
gam_inte.summary()
```

```
100% (4096 of 4096) |####################| Elapsed Time: 0:10:10 Time:  0:10:10

LinearGAM
============================================= ==================================
========================
Distribution:                        NormalDist Effective DoF:
18.7265
Link Function:                     IdentityLink Log Likelihood:
-1524.2378
Number of Samples:                          310 AIC:
3087.9286
                                                AICc:
3090.7554
                                                GCV:
60.5625
                                                Scale:
54.0149
                                                Pseudo R-Squared:
0.7136
=========================================================================================
========================
Feature Function                  Lambda               Rank         EDoF         P
> x        Sig. Code
================================= ==================== ============ ============ =
========== ============
s(0)                              [1000.]              20           4.8
6.19e-03     **
s(1)                              [1000.]              20           3.2
1.53e-08     ***
s(2)                              [10.]                20           5.7
1.20e-10     ***
s(3)                              [1000.]              20           2.1
8.83e-01
te(4, 5)                          [1.e-01 1.e+03]      100          2.9
3.68e-10     ***
intercept                                              1            0.0
1.11e-16     ***
=========================================================================================
========================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

WARNING: Fitting splines and a linear function to a feature introduces a model ide
ntifiability problem
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized mode
ls or models with
         known smoothing parameters, but when smoothing parameters have been estim
ated, the p-values
         are typically lower than they should be, meaning that the tests reject th
e null too readily.
```

```
/var/folders/qx/wlpqnbz178962z833f_yq5gh0000gn/T/ipykernel_69703/3270001194.py:
3: UserWarning: KNOWN BUG: p-values computed in this summary are likely much sma
```

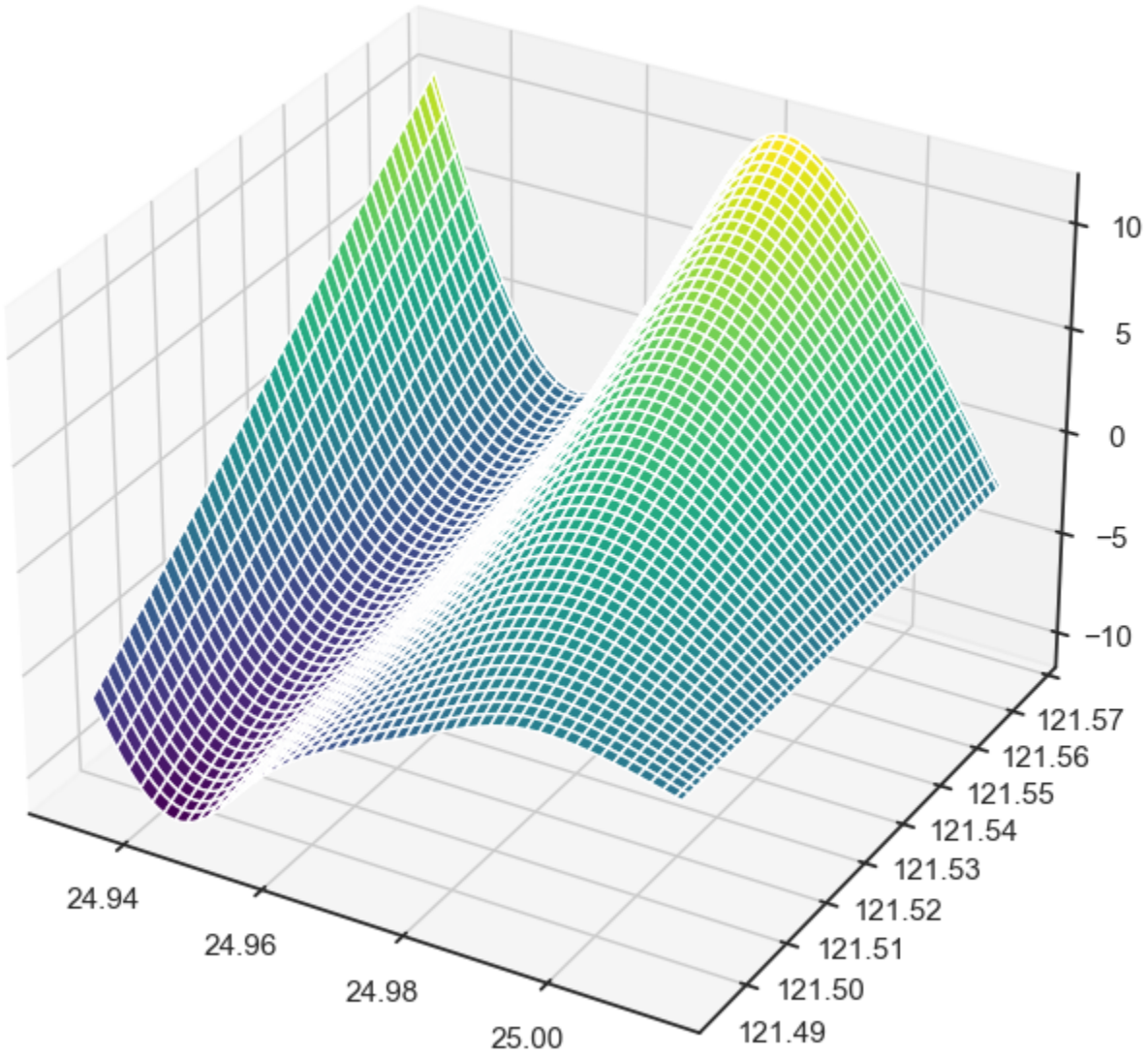
```
ller than they should be.

Please do not make inferences based on these values!

Collaborate on a solution, and stay up to date at:
github.com/dswah/pyGAM/issues/163

  gam_inte.summary()
```

In [43]:
```
pred3 = gam_inte.predict(x_test.values)
mean_squared_error(y_test, pred3)
```

Out[43]:
```
71.9007519937477
```

In [44]:
```
from mpl_toolkits import mplot3d

plt.ion()
plt.rcParams['figure.figsize'] = (12, 8)
XX = gam_inte.generate_X_grid(term=4, meshgrid=True)
Z = gam_inte.partial_dependence(term=4, X=XX, meshgrid=True)
ax = plt.axes(projection='3d')
ax.plot_surface(XX[0], XX[1], Z, cmap='viridis')
```

Out[44]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x16ccdf8e0>



In [ ]: