In [3]:
```
#coding:utf-8

import pandas as pd
import numpy as np
from geopy.distance import geodesic
from tqdm import tqdm
import json
import os
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("darkgrid")  # 图表风格
plt.rcParams['font.sans-serif']=['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False  # 用来正常显示负号
```

In [4]:
```
import warnings
warnings.filterwarnings('ignore')
```

数据读取预览

In [5]:
```
ori_df = pd.read_excel("../data/lianjia.xls", index_col='ID')
```

In [6]:
```
# 数据一览
ori_df.head(10)
```

Out[6]:

ID	name	address	model	area	direct	perfect	floor	year	type	total	unit	traffic
1	厦罐宿舍	禾祥西路	1室1厅	46.89平米	南	简装	高楼层(共7层)	1998	板楼	216.0	单价46066元/平米	近1号线文灶站
2	万寿北路	文园路	2室1厅	48.67平米	南	简装	低楼层(共7层)	1991	板楼	310.0	单价63695元/平米	近1号线将军祠站
3	菁英公馆	翔安新城	NaN	NaN	NaN	NaN	NaN	NaN	NaN	45.0	单价12163元/平米	NaN
4	南湖中祥大厦	槟榔	1室1厅	45.08平米	东南 南	精装	低楼层(共31层)	2007	板塔结合	308.0	单价68323元/平米	近2号线体育中心站
5	王子广场	海沧生活区	1室1厅	34.2平米	南	简装	中楼层(共25层)	2011	板塔结合	99.4	单价29065元/平米	近2号线海沧行政中心站
6	文化大厦	莲坂	1室0厅	44.45平米	南	简装	中楼层(共31层)	2005	板楼	230.0	单价51744元/平米	NaN
7	天伦花园	莲花	1室1厅	47.9平米	北	简装	中楼层(共10层)	2000	塔楼	238.0	单价49687元/平米	近2号线江头站
8	槟榔东里单号区	槟榔	1室1厅	44.72平米	南 西 北	简装	高楼层(共6层)	1989	板楼	240.0	单价53668元/平米	近2号线育秀东路站
9	未来橙堡	海沧体育中心	1室1厅	30.7平米	南	简装	中楼层(共5层)	2005	板楼	93.0	单价30294元/平米	NaN
10	信洲国际	翔安新城	1室1厅	40平米	南	精装	中楼层(共25层)	2016	板楼	46.0	单价11500元/平米	NaN

缺失数据预处理

In [7]:
```
# 发现存在缺失信息较多的房源，接下来进行筛选后删除
# 删除总价及单价均为空的数据
ori_df.drop(index=ori_df[(ori_df['total'].isnull()) & (ori_df['unit'].isnull())].index, inplace=True)
# 筛选并保留缺失比例小于50
tmp_df = ori_df[['name', 'address', 'model', 'area', 'direct', 'perfect', 'floor','year','type', 'traffic']]
rows_null = tmp_df.isnull().sum(axis=1) / len(tmp_df.columns)
ori_df = ori_df.loc[rows_null < 0.5, :]
```

调用百度地理-逆地理api，使用小区名进行经纬度转换

In [8]:
```
from urllib.request import urlopen,quote
```

In [9]:
```
# 转换函数
def getlnglat(adress):
    url = 'http://api.map.baidu.com/geocoder/v2/?address='
    output = 'json'
    ak = 'NncC2ROHKWGoZ158tfqdziUprQW81ins'
    add = quote(adress)#使用quote进行编码 为了防止中文乱码
    url2 = url + add + '&output=' + output + '&ak=' + ak
    req = urlopen(url2)
    res = req.read().decode()
    temp = json.loads(res)
    return temp
```

In [10]:
```
# 地址前加上厦门进行范围限制，防止跨省同名小区覆盖
ori_df['name'] = '厦门' + ori_df['name']
# 开辟新列表用来存储转换结果
geo_list = np.zeros((ori_df.shape[0], 2))
```

In [11]:
```
if not os.path.exists("../data/geo_trans.csv"):
    dim = ori_df.shape
    [row,col] = dim  #获取行列数
    cnt = 0
    for i in tqdm(ori_df.values):
        try:
            b = i[0]  #首列的小区名
            geo_list[cnt][0] = getlnglat(b)['result']['location']['lng']#获取经度并写入
            geo_list[cnt][1] = getlnglat(b)['result']['location']['lat']#获取纬度并写入
        except:
            print("第{}条数据转换出错 ".format(cnt))
            geo_list[cnt][0] = 0
            geo_list[cnt][1] = 0
        cnt += 1
    # api转换错误的少数小区手动转化
    geo_list[605][0], geo_list[605][1] = 118.098148, 24.446382
    geo_list[5737][0], geo_list[5737][1] = 118.24771,24.601405
    geo_list[6555][0], geo_list[6555][1] = 118.087447,24.501763
    geo_list[7790][0], geo_list[7790][1] = 118.087447,24.501763
    geo_list[8311][0], geo_list[8311][1] = 118.098148,24.446382
    ori_df['经度'] = geo_list[:, 0]
    ori_df['纬度'] = geo_list[:, 1]
    ori_df.to_csv("../data/geo_trans.csv")

else:
    ori_df = pd.read_csv("../data/geo_trans.csv")
```

正式数据预处理

正则提取、分割字符串中信息

In [12]:
```
# 室、厅数量提取
ori_df[['室数量', '厅数量']] = ori_df['model'].str.split('室',expand=True)
ori_df['厅数量'] = ori_df['厅数量'].str.extract("(\d+)", expand=True)
# 0室0厅处理为1室
ori_df.iloc[ori_df[(ori_df['室数量']==0) & (ori_df['厅数量']==0)].index]['室数量'] = 1
ori_df['室数量'] = ori_df['室数量'].astype('int')
ori_df['厅数量'] = ori_df['厅数量'].astype('int')
```

In [13]:
```
# 面积提取
ori_df['面积'] = ori_df['area'].str.extract("(\d+\.?\d*\.\d+)", expand=True)
```

In [14]:
```
# 单价提取
ori_df['单价'] = ori_df['unit'].str.extract("(\d+)", expand=True)
ori_df['单价'] = ori_df['单价'].astype('float')
ori_df['单价'] = ori_df['单价'] / 10000
```

In [15]:
```
# 楼层等级及总楼层提取
ori_df['总楼层'] = ori_df['floor'].str.extract("(\d+)", expand=True)
ori_df['楼层等级'] = ori_df['floor'].str.split('(',expand=True)[0]
ori_df['总楼层'] = ori_df['总楼层'].astype('int')
```

In [16]:
```
# 朝向提取，多朝向直接提取第一个朝向
ori_df['朝向'] = ori_df['direct'].str.split(' ',expand=True)[0]
```

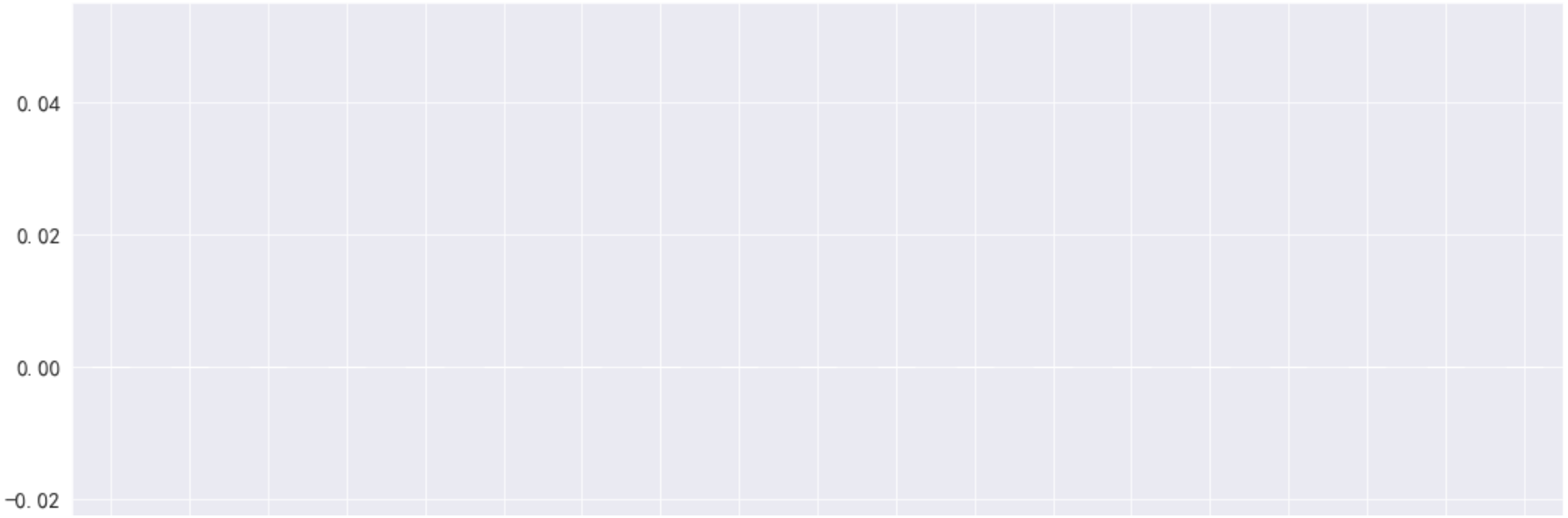
In [17]:
```
# 近地铁若为空值填为空值，否则为1
ori_df['traffic'].fillna(0, inplace=True)
ori_df.loc[ori_df['traffic']!=0, 'traffic'] = 1
ori_df['traffic'] = ori_df['traffic'].astype('int')
```

离散变量处理

空值处理

In [18]:
```
discrete_feat = [feat for feat in ori_df.columns if feat not in ['year', '面积', 'total']]
ori_df[discrete_feat].isna().sum().plot(kind='bar', figsize=(18, 9), rot=45, fontsize=15)
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1b5f16e3710>



离散变量不存在空值

In [19]:
```
# 存在层级关系的离散变量，使用label encode进行编码
# 装修程度
ori_df.loc[ori_df['perfect'] == '其他', 'perfect'] = 0
ori_df.loc[ori_df['perfect'] == '毛坯', 'perfect'] = 1
ori_df.loc[ori_df['perfect'] == '简装', 'perfect'] = 2
ori_df.loc[ori_df['perfect'] == '精装', 'perfect'] = 3
ori_df['perfect'] = ori_df['perfect'].astype('int')
# 楼层等级
ori_df.loc[ori_df['楼层等级'] == '上叠', '楼层等级'] = '中楼层'
ori_df.loc[ori_df['楼层等级'] == '下叠', '楼层等级'] = '低楼层'
ori_df.loc[ori_df['楼层等级'] == '地下室', '楼层等级'] = '低楼层'
ori_df.loc[~ori_df['楼层等级'].isin(['高楼层', '中楼层', '低楼层']), '楼层等级'] = '低楼层'
ori_df.loc[ori_df['楼层等级'] == '低楼层', '楼层等级'] = 1
ori_df.loc[ori_df['楼层等级'] == '中楼层', '楼层等级'] = 2
ori_df.loc[ori_df['楼层等级'] == '高楼层', '楼层等级'] = 3
ori_df['楼层等级'] = ori_df['楼层等级'].astype('int')
```

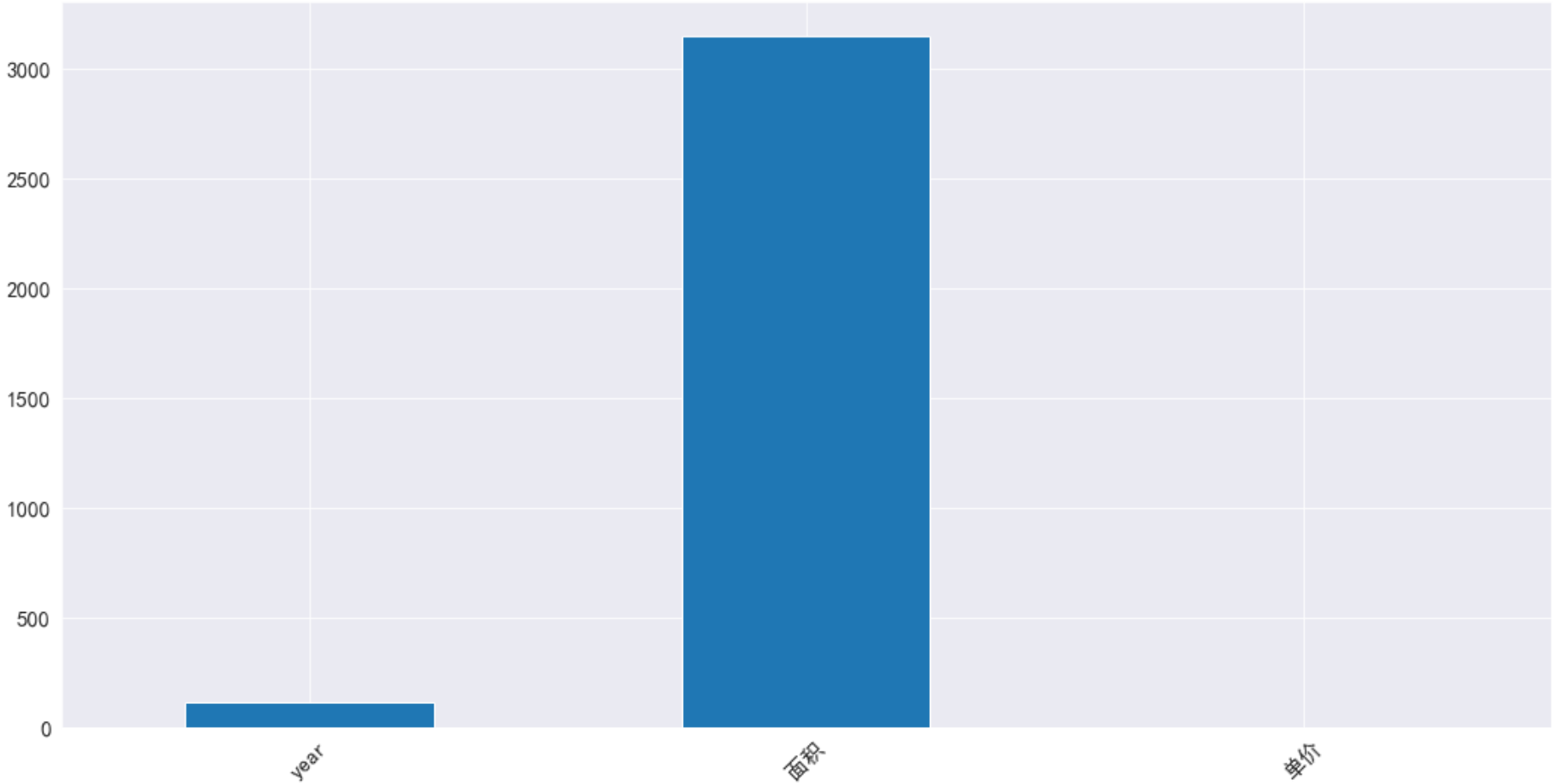
In [20]:
```
ori_df.drop(['name', 'address', 'model', 'area', 'direct', 'floor', 'unit', 'total'], axis=1, inplace=True)
```

In [21]:
```
# 不存在层级关系的离散变量，使用onehot encode进行编码
# 包含朝向、建筑类型
ori_df = pd.get_dummies(ori_df, columns=['type', '朝向'])
```

连续变量处理

In [22]:
```
numeric_feat = [feat for feat in ori_df.columns if feat in ['year', '面积', '单价']]
ori_df[numeric_feat].isna().sum().plot(kind='bar', figsize=(18, 9), rot=45, fontsize=15)
```

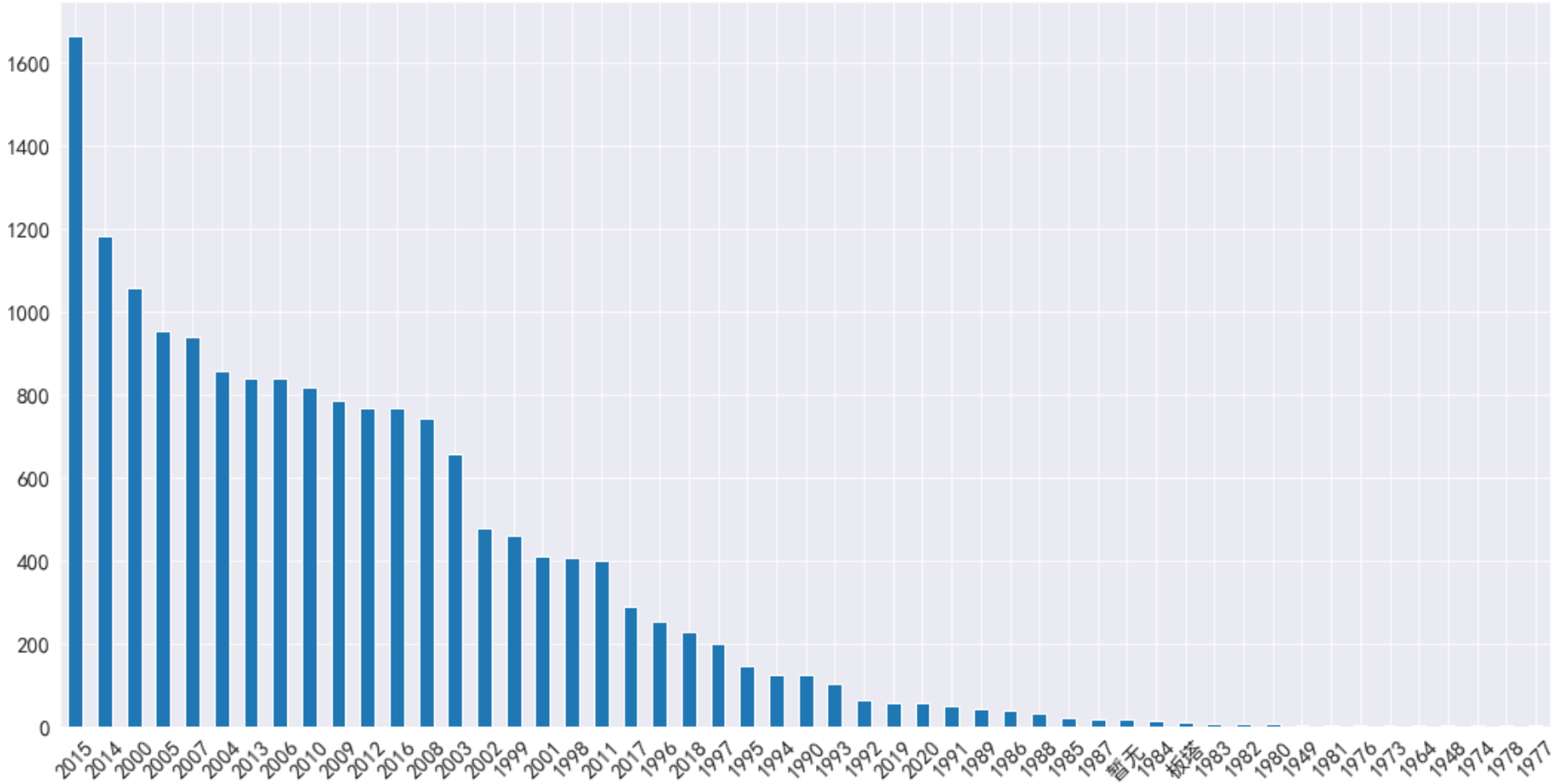
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1b5f0bf9fd0>



年份和面积存在空值

In [23]:
```
# 查看年份分布
ori_df['year'].value_counts().plot(kind='bar', figsize=(18, 9), rot=45, fontsize=15)
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1b5f149d128>



In [24]:
```
# 通过分布图发现年份存在异常值，将异常值填补为众数后，把年份处理成最新建筑的年份偏移
year_mode = ori_df[~ori_df['year'].isin(['板塔', '暂无'])]['year'].mode()[0]
ori_df.loc[ori_df['year'].isin(['板塔', '暂无']), 'year'] = year_mode
ori_df.loc[ori_df['year'].isna(), 'year'] = year_mode
ori_df['year'] = ori_df['year'].astype('int')
ori_df['year'] = abs(ori_df['year'] - ori_df['year'].max())
```

In [25]:
```
# 面积使用均值填补
ori_df['面积'] = ori_df['面积'].astype("float")
ori_df.loc[ori_df['面积'].isna(), '面积'] = ori_df[~ori_df['面积'].isna()]['year'].mean()
```

分别求每个房源指定范围内学校、医院、商场的数量

使用apply函数对以上三种场所进行距离计算，并取5km范围内数量

经纬度距离计算转化为km：使用geopy进行距离计算

使用经纬度进行特征扩充，计算指定距离范围(单位:km)内学校、医院、商场数量

In [26]:
```
# 重点学校、医院、商场地理坐标，通过常量存储
kp_schcool = {'幼儿园':{'厦门毅康幼儿园':(24.510597, 118.11754),
                        '厦门市海城幼儿园':(24.474125, 118.099648),
                        '厦门市湖里区南泰苑特房艺术幼儿园':(24.508546, 118.1114),
                        '厦门市金鸡亭幼儿园':(24.480669, 118.151603),
                        '厦门市第九幼儿园':(24.481007, 118.114035),
                        '厦门市鼓浪屿日光幼儿园':(24.449937, 118.073416),
                        '厦门市振兴幼儿园':(24.485166, 118.099807),
                        '厦门市蓝天幼儿园':(24.500507, 118.151464),
                        '厦门市仙岳幼儿园':(24.497445, 118.10177),
                        '厦门莲龙幼儿园':(24.494872, 118.145078)},
              '小学':{'厦门市集美小学':(24.575549, 118.105817),
                        '厦门市滨东小学':(24.482332, 118.115181,),
                        '厦门外国语学校附属小学':(24.479366, 118.108217),
                        '厦门同安第一实验小学':(24.737489, 118.163726),
                        '厦门第二实验小学':(24.488178, 118.13328),
                        '厦门市槟榔小学':(24.486965, 118.117595),
                        '厦门市民立小学':(24.460889, 118.082973),
                        '厦门市实验小学':(24.461625, 118.095239),
                        '厦门市演武小学':(24.444945, 118.09853)},
              '中学':{'厦门市双十中学':(24.522979, 118.161578),
                        '厦门市外国语学校':(24.483883, 118.09878),
                        '厦门市第一中学':(24.465068, 118.105468),
                        '厦门市第十一中学':(24.472203, 118.091047),
                        '厦门市松柏中学':(24.496271, 118.124842),
                        '厦门市第六中学':(24.466131, 118.088328),
                        '厦门市同安一中':(24.744241, 118.163146),
                        '厦门市莲花中学':(24.489817, 118.135714),
                        '厦门市集美中学':(24.601198, 118.121767),
                        }}
kp_hospital = {'厦门市眼科中心':(24.514054, 118.197927),
               '厦门市仙岳医院':(24.501499, 118.117878),
               '厦门大学附属厦门眼科中心':(24.466303, 118.087465),
               '解放军第一七四医院':(24.465084, 118.101848),
               '厦门大学附属中山医院':(24.477694, 118.104437),
               '厦门大学附属翔安医院':(24.593282, 118.27267),
               '厦门长庚医院':(24.540208, 118.015807),
               '厦门市精神卫生中心':(24.508779, 118.1141),
               '厦门市第三医院':(24.711822, 118.153719),
               '厦门市第二医院':(24.590336, 118.11087),
               }
```

```
kp_mall = {'SM城市广场':(24.50701, 118.13372),
           '万象城MIXC':(24.478183, 118.117889),
           '宝龙一城':(24.492034, 118.178751),
           '湖里万达广场':(24.510421, 118.183849),
           '罗宾森广场':(24.474916, 118.119239),
           '建发湾悦城':(24.521607, 118.167925),
           '世茂Emall':(24.442476, 118.094721),
           '磐基名品中心':(24.488179, 118.126967),
           '老虎城欢乐购物中心':(24.459434, 118.087391)
           }
```

In [27]:
```
def get_thre_num(house_geo, geo_list, threshold=5):
    num_cnt = 0
    for geo_value in geo_list.values():
        if geodesic(house_geo, geo_value).km < threshold:
            num_cnt += 1
    return num_cnt
```

In [28]:
```
ori_df['geo'] = tuple(zip(ori_df['纬度'], ori_df['经度']))
```

In [29]:
```
# 学校
ori_df['5公里内重点幼儿园数量'] = ori_df['geo'].apply(get_thre_num, args=(kp_schcool['幼儿园'], 5))
ori_df['5公里内重点小学数量'] = ori_df['geo'].apply(get_thre_num, args=(kp_schcool['小学'], 5))
ori_df['5公里内重点中学数量'] = ori_df['geo'].apply(get_thre_num, args=(kp_schcool['中学'], 5))
# 医院
ori_df['5公里内重点医院数量'] = ori_df['geo'].apply(get_thre_num, args=(kp_hospital, 5))
# 商场
ori_df['5公里内重点商场数量'] = ori_df['geo'].apply(get_thre_num, args=(kp_mall, 5))
```

开始训练模型

In [30]:
```
from sklearn.model_selection import KFold
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error
import lightgbm as lgb
# 创建Kfold划分器,进行10折划分
folds = 10
kf = KFold(n_splits=folds, random_state=2020, shuffle=True)
```

```
c:\program files\python36\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests is a
n internal NumPy module and should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

In [31]:
```
# 重新命名列
ori_df.columns = ['ID', '装修类型', '年份', '是否近地铁', '经度', '纬度','室数量', '厅数量', '面积', '单价', '总楼层', '楼层等级',
                  '双拼别墅', '叠拼别墅', '塔楼', '平房', '暂无数据', '板塔结合', '板楼', '独栋别墅', '联排别墅', '朝向东', '朝向东北
                  '朝向东南', '朝向北', '朝向南', '朝向西', '朝向西北', '朝向_西南', '坐标', '5公里内重点幼儿园数量', '5公里内重点小学
                  '5公里内重点中学数量', '5公里内重点医院数量', '5公里内重点商场数量', ]
```

In [32]:
```
feat_names = [feat for feat in ori_df.columns if feat not in ['ID', '单价', '坐标']]
```

In [33]:
```
# 划分90%作为训练集，10%作为测试集
# 其中训练集分10折进行验证
ori_df.reset_index(drop=True)
ori_df.drop(['ID'], axis=1, inplace=True)
df_train = ori_df.iloc[:int(len(ori_df)*0.9), :]
df_test = ori_df.iloc[int(len(ori_df)*0.9):, :]
```

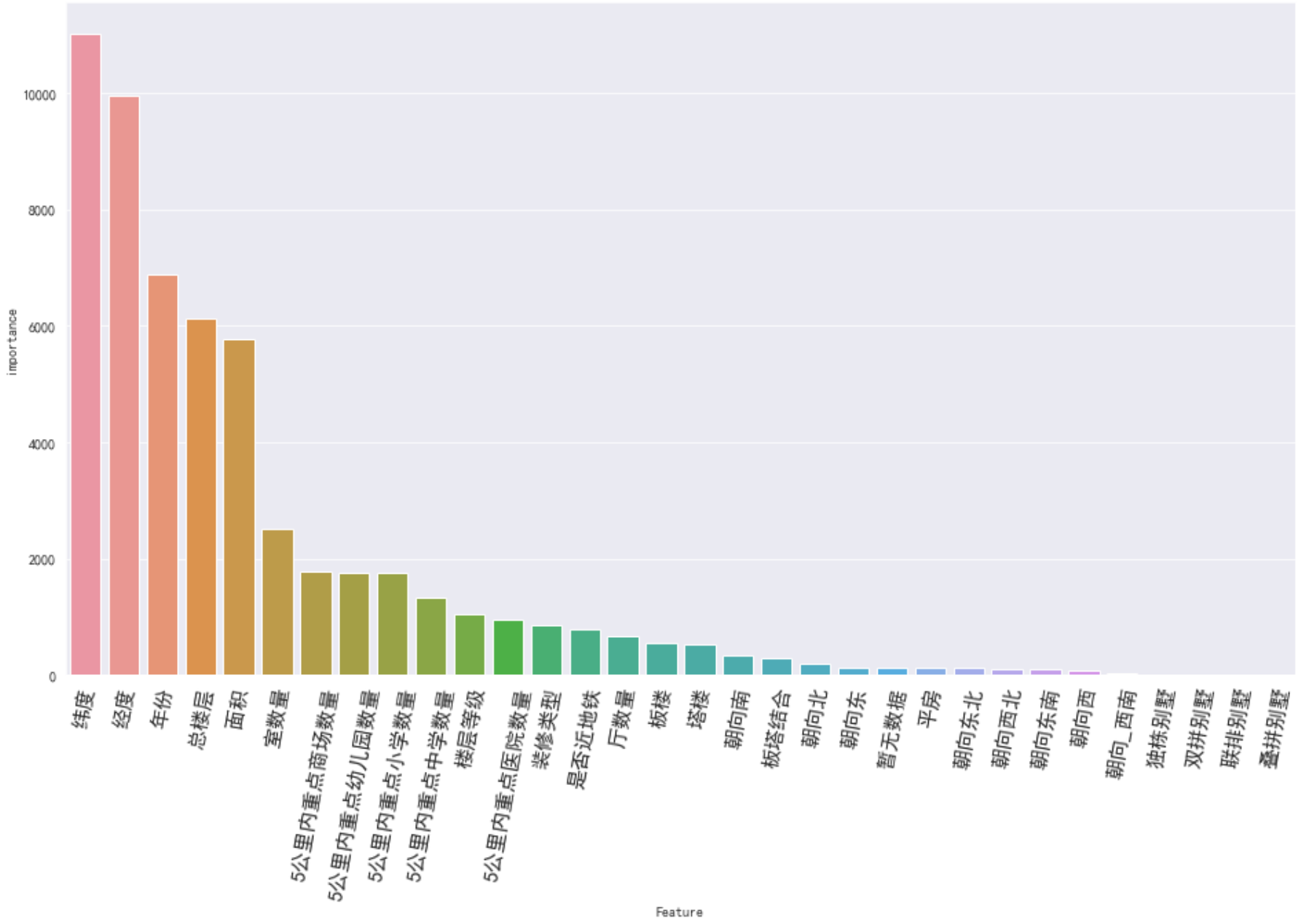
In [37]:
```
param = {"num_leaves": 40,
         "min_data_in_leaf": 30,
         "objective":'regression',
         "max_depth": 5,
         "n_estimators":3000,
         "learning_rate": 0.01,
         "min_child_samples": 20,
         "boosting": "gbdt",
         "feature_fraction": 0.9,
         "bagging_freq": 1,
         "alpha":0.1,
         "bagging_fraction": 0.9 ,
         "bagging_seed": 11,
         "metric": 'mae',
         "lambda_l1": 0.1,
         "verbosity": -1,
         "random_state": 2020
        }

oof = np.zeros(len(df_train))
predictions = np.zeros(len(df_test))
feature_importance_df = pd.DataFrame()

for fold_, (trn_idx, val_idx) in enumerate(kf.split(df_train)):
    print("fold {}".format(fold_))
    # 训练集
    train_data = df_train[feat_names].iloc[trn_idx,:]
    train_label = df_train["单价"].iloc[trn_idx]
    # 验证集
    valid_data = df_train[feat_names].iloc[val_idx,:]
    valid_label = df_train["单价"].iloc[val_idx]

    trn_data = lgb.Dataset(train_data, label=train_label)
    val_data = lgb.Dataset(valid_data, label=valid_label)

    num_round = 10000
    model = lgb.train(param, trn_data, num_round, valid_sets = [trn_data, val_data], verbose_eval=200, early_stopping_rounds = 100)
    oof[val_idx] = model.predict(df_train.iloc[val_idx][feat_names], num_iteration=model.best_iteration)

    fold_importance_df = pd.DataFrame()
    fold_importance_df["Feature"] = feat_names
```

```
    fold_importance_df["importance"] = model.feature_importance()
    fold_importance_df["fold"] = fold_ + 1
    feature_importance_df = pd.concat([feature_importance_df, fold_importance_df], axis=0)

    predictions += model.predict(df_test[feat_names], num_iteration=model.best_iteration) / kf.n_splits
```

```
fold 0
Training until validation scores don't improve for 100 rounds.
[200]   training's l1: 0.617341 valid_1's l1: 0.608818
[400]   training's l1: 0.528138 valid_1's l1: 0.533629
[600]   training's l1: 0.491705 valid_1's l1: 0.502966
[800]   training's l1: 0.464181 valid_1's l1: 0.480478
[1000]  training's l1: 0.442741 valid_1's l1: 0.462631
[1200]  training's l1: 0.425081 valid_1's l1: 0.447615
[1400]  training's l1: 0.410439 valid_1's l1: 0.435198
[1600]  training's l1: 0.397304 valid_1's l1: 0.424389
[1800]  training's l1: 0.387324 valid_1's l1: 0.416704
[2000]  training's l1: 0.378012 valid_1's l1: 0.409447
[2200]  training's l1: 0.370036 valid_1's l1: 0.403469
[2400]  training's l1: 0.362922 valid_1's l1: 0.398274
[2600]  training's l1: 0.356228 valid_1's l1: 0.393719
[2800]  training's l1: 0.350242 valid_1's l1: 0.389513
[3000]  training's l1: 0.345384 valid_1's l1: 0.386249
Did not meet early stopping. Best iteration is:
[3000]  training's l1: 0.345384 valid_1's l1: 0.386249
```

In [38]:
```
feature_importance_df = feature_importance_df.groupby(by=['Feature'], as_index=False)['importance'].agg('mean').reset_index(drop=Tru
feature_importance_df.sort_values(by='importance', ascending=False, inplace=True)
```

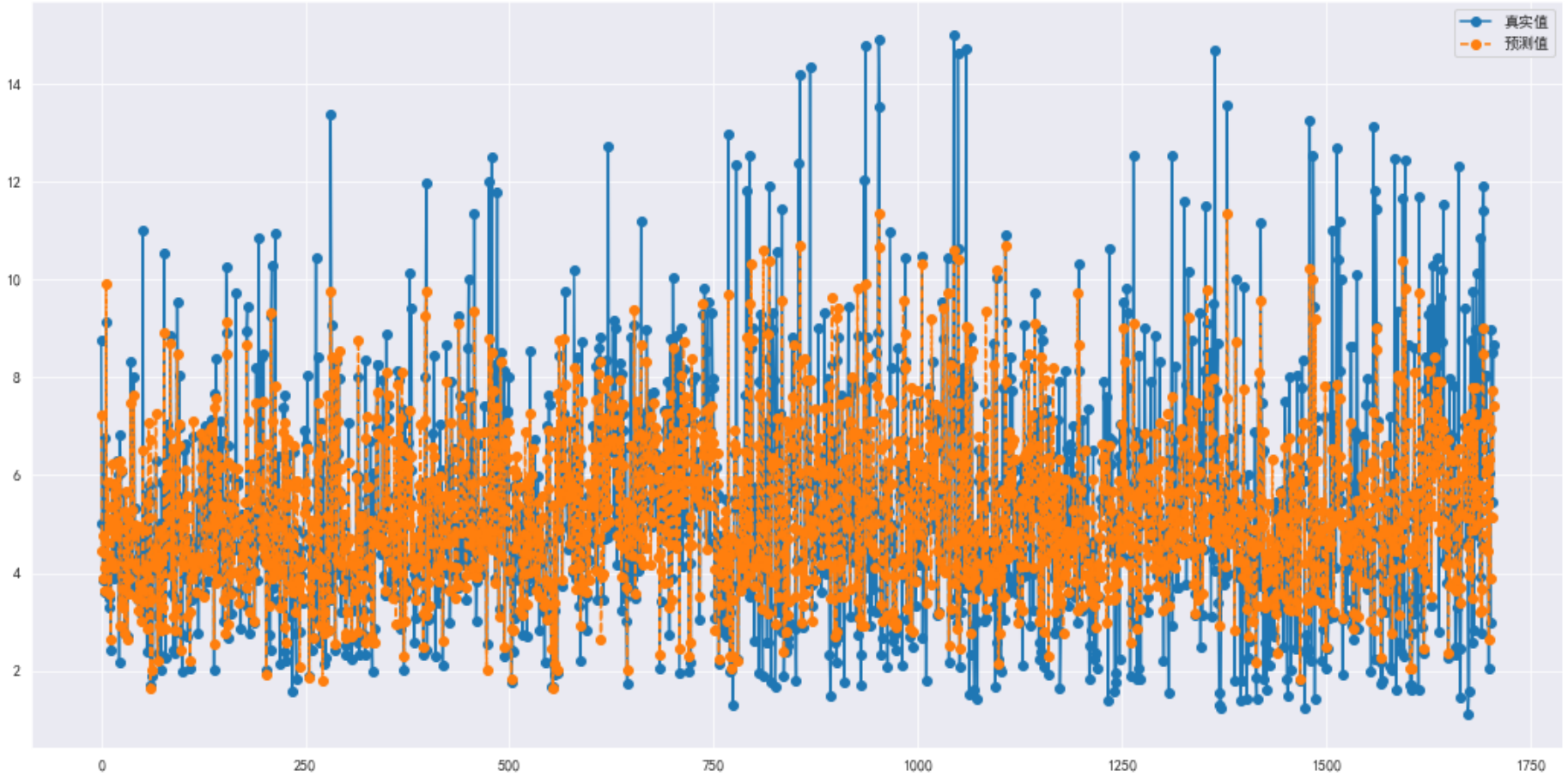
In [39]:
```
plt.figure(figsize=(16, 9))
fig= sns.barplot(x='Feature', y='importance', data=feature_importance_df)
fig.set_xticklabels(labels=feature_importance_df['Feature'].values, rotation=80, size=15)
plt.show()
```


importance
10000
8000
6000
4000
2000
0
纬度
经度
年份
总楼层
面积
室数量
5公里内重点商场数量
5公里内重点幼儿园数量
5公里内重点小学数量
5公里内重点中学数量
楼层等级
5公里内重点医院数量
装修类型
是否近地铁
厅数量
板楼
塔楼
朝向南
板塔结合
朝向北
朝向东
暂无数据
平房
朝向东北
朝向西北
朝向东南
朝向西
朝向_西南
独栋别墅
双拼别墅
联排别墅
叠拼别墅
Feature

In [40]:
```
# 模型在测试集上的表现
results = model.predict(df_test[feat_names])
mean_absolute_error(df_test['单价'], predictions)
```

Out[40]: 0.9724408866700103

In [41]:
```
plt.figure(figsize=(18, 9))
plt.plot(list(range(len(df_test))), df_test['单价'], "-o", label='真实值')
plt.plot(list(range(len(df_test))), results, "--o", label='预测值')
plt.legend()
# 展现画布
plt.show()
```



In [46]:
```
from sklearn.metrics import r2_score
```

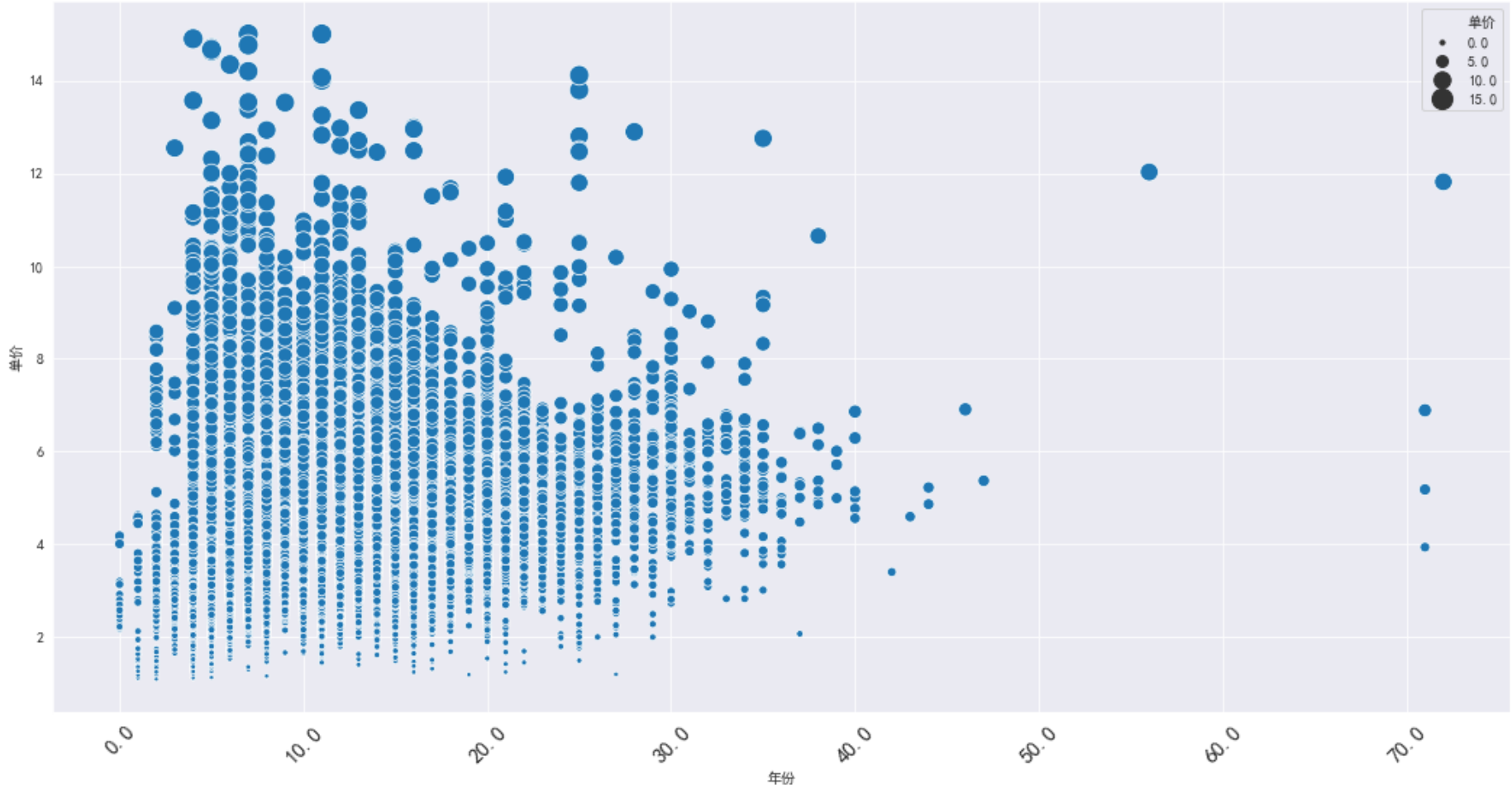
因子分析

In [54]:
```
# 经纬度
import folium
from folium.plugins import HeatMap

geo_center = [24.5580803, 118.0747703]
geo_map = folium.Map(location=geo_center, zoom_start=13.5)

heatdata = ori_df[['纬度', '经度', '单价']].values.tolist()

HeatMap(heatdata, radius=7).add_to(geo_map) # gradient={.4:'blue',.65:'yellow',1:'red'}

geo_map
```

Out[54]: Make this Notebook Trusted to load map: File -> Trust Notebook

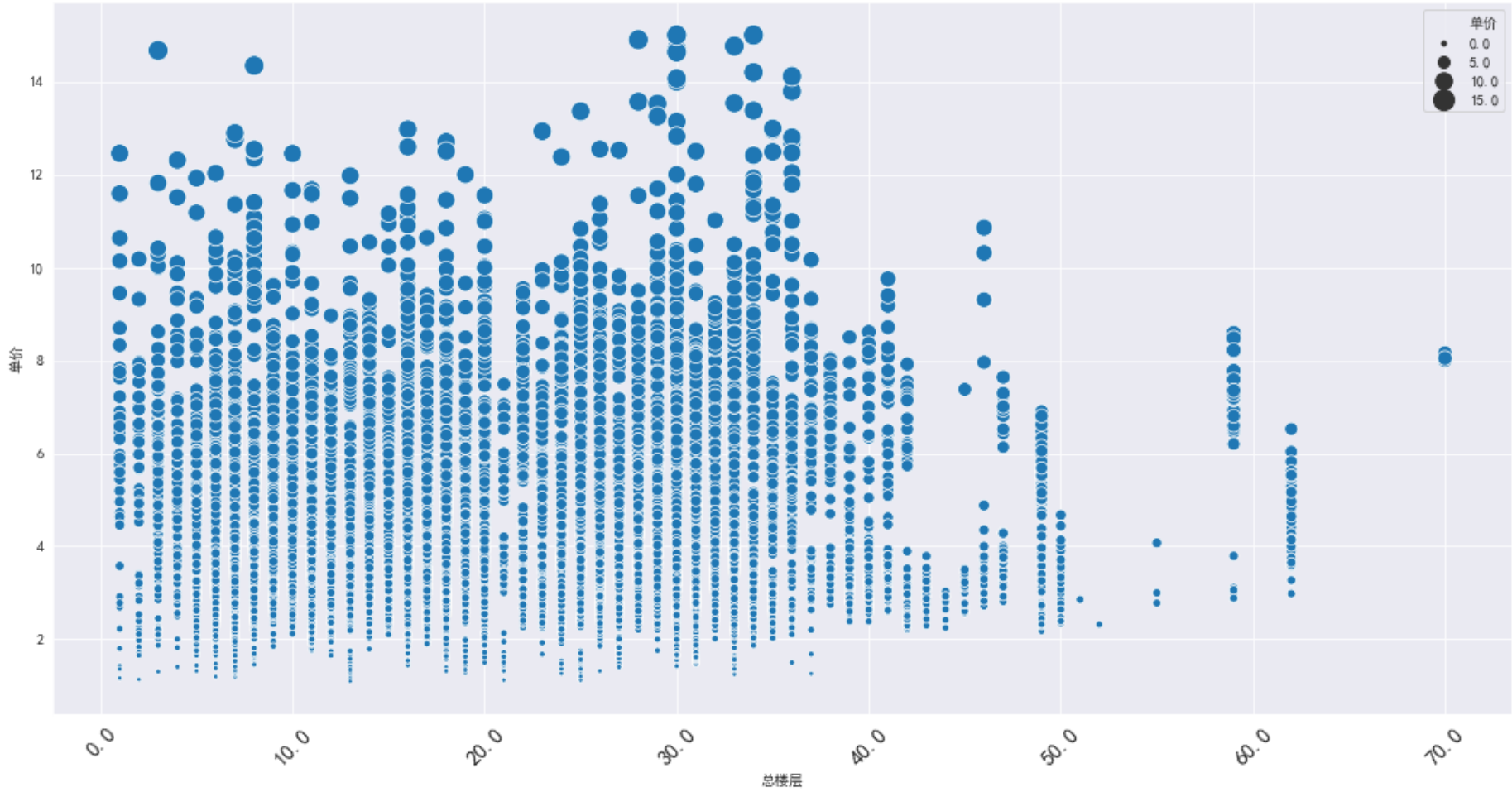
从重要性分析可看出，经纬度分布是影响房价的最主要因素，即“地段”为影响房价的强因子
通过房价地理热图，可发现：高房价房源主要分布在思明区及湖里区，并且处在厦门市中心区域，其中房价最高的区域落在湖里区“厦门市软件园”地带
另外，集美区及海沧区高房价区有着沿海、靠近市中心等特点

In [76]:
```
# 年份与房价分布关系
ori_df.sort_values(by='年份', ascending=True, inplace=True)
plt.figure(figsize=(18, 9))
cmap = sns.cubehelix_palette(rot=-.2, as_cmap=True)
fig = sns.scatterplot(x="年份", y="单价",size="单价",
                      palette=cmap, sizes=(10, 200),
                      data=ori_df)
fig.set_xticklabels(labels=fig.get_xticks(), rotation=45, size=15)
plt.show()
```



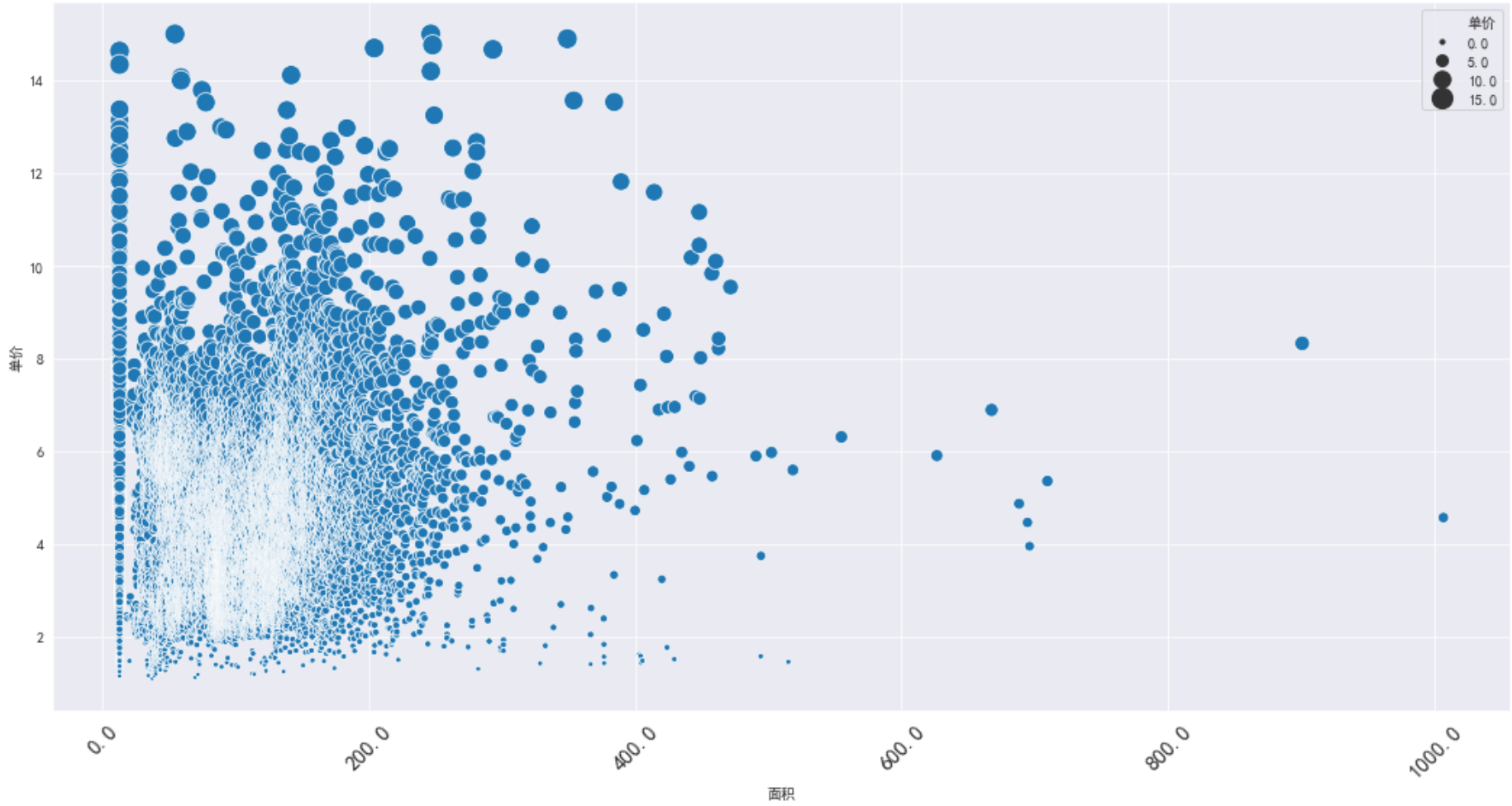
可看出，整体来看越新的房越受欢迎，但并不是最新的房房价最高，建成3-10年的房最受欢迎，且随房子建成年限越长，受欢迎程序越低

In [75]:
```
# 年份与总楼层分布关系
ori_df.sort_values(by='总楼层', ascending=True, inplace=True)
plt.figure(figsize=(18, 9))
cmap = sns.cubehelix_palette(rot=-.2, as_cmap=True)
fig = sns.scatterplot(x="总楼层", y="单价",size="单价",
                      palette=cmap, sizes=(10, 200),
                      data=ori_df)
fig.get_xlabel
fig.set_xticklabels(labels=fig.get_xticks(), rotation=45, size=15)
plt.show()
```



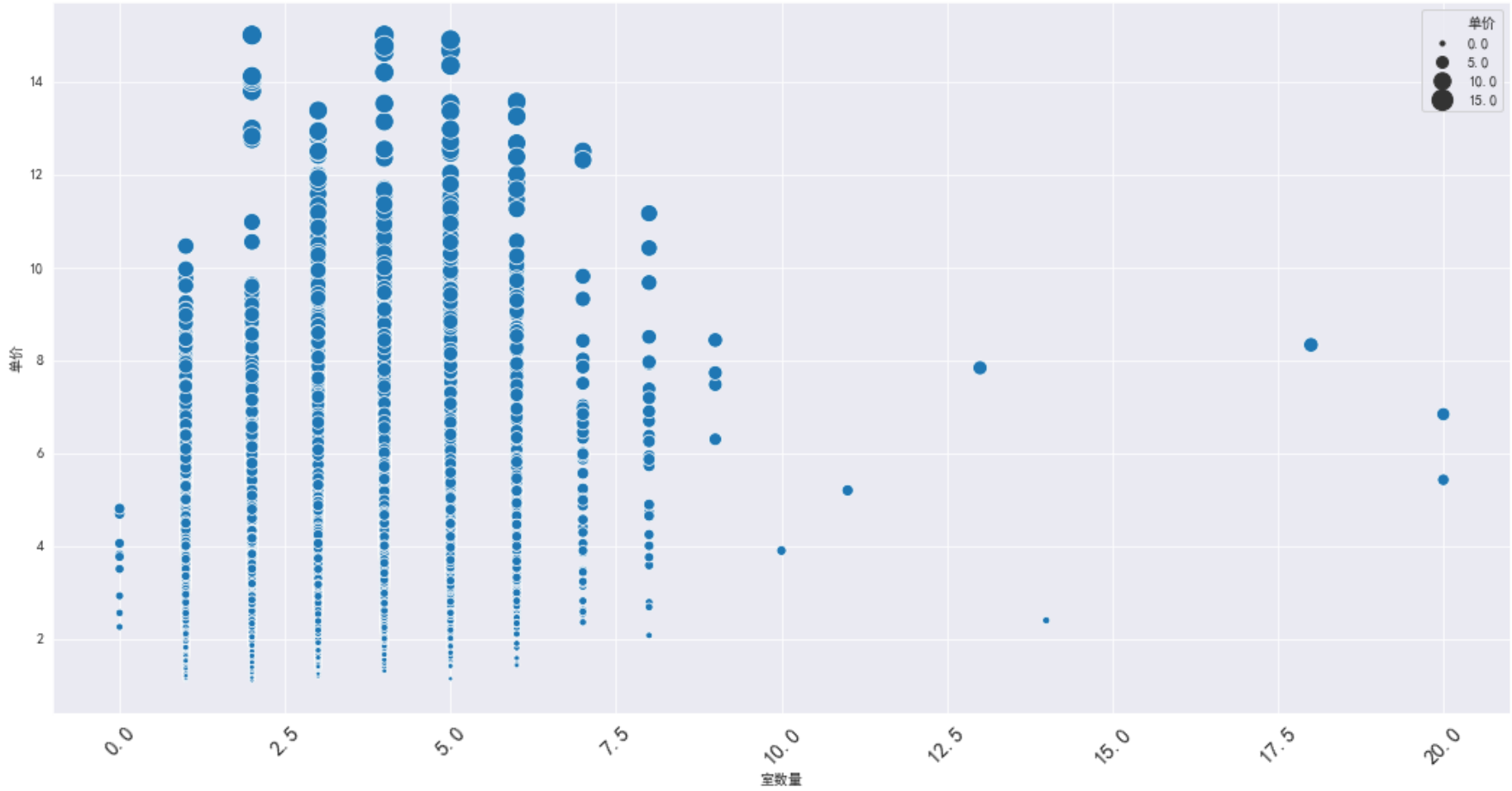
建筑总楼层在10层以下的房源能卖出高达14W的单价，中高层则在25-35之间的房源最受欢迎，其高层拥有较高的视野，而40层以上房源受欢迎程度明显降低，其高楼层存在较大的生活隐患（停电、应急逃生等）

In [78]:
```
# 年份与面积分布关系
ori_df.sort_values(by='面积', ascending=True, inplace=True)
plt.figure(figsize=(18, 9))
cmap = sns.cubehelix_palette(rot=-.2, as_cmap=True)
fig = sns.scatterplot(x="面积", y="单价",size="单价",
                      palette=cmap, sizes=(10, 200),
                      data=ori_df)
fig.get_xlabel
fig.set_xticklabels(labels=fig.get_xticks(), rotation=45, size=15)
plt.show()
```



与我们常规认知有区别的是，小面积房源（公寓）能卖出比大面积房源更高的价钱，其主要归咎于公寓主要为城市中心地带及大开发商开发，一般拥有极佳的地理优势，而200至400平的房源一般为别墅或高层，也能卖出较高价钱

In [80]:
```
# 年份与室数量分布关系
ori_df.sort_values(by='室数量', ascending=True, inplace=True)
plt.figure(figsize=(18, 9))
cmap = sns.cubehelix_palette(rot=-.2, as_cmap=True)
fig = sns.scatterplot(x="室数量", y="单价",size="单价",
                      palette=cmap, sizes=(10, 200),
                      data=ori_df)
fig.get_xlabel
fig.set_xticklabels(labels=fig.get_xticks(), rotation=45, size=15)
plt.show()
```



人们对于房源室数量需求很明确，越多室数量越好，并在5室达到极值，5室以上一般已超过生活需求，所以单价反而有所下降

In [ ]:

In [ ]: