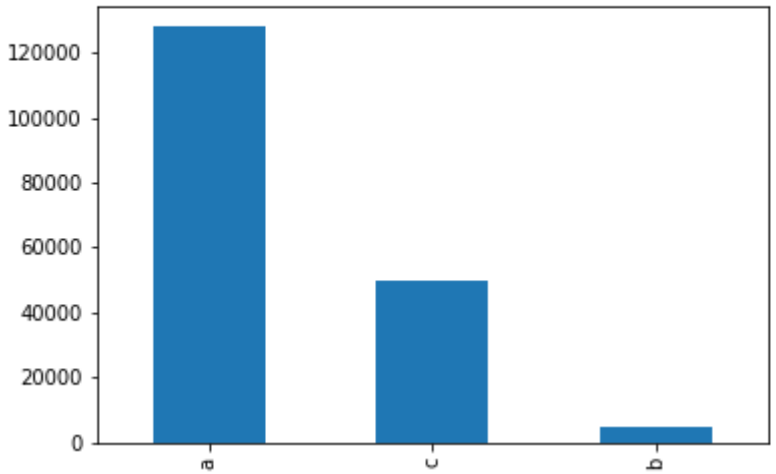
Task-1 Minnibatch kmeans算法聚类过程

```
#%% 导入数据
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
os.chdir(r'C:\Users\REGGIE\Desktop\Machine Learning Model Group')

df = pd.read_csv('site.csv')

#%% 数据的简单观察
print(df.info(),df.describe())

#%% 查看缺失值
df[df.var_value.isna()]
#10510/172191 缺失值只占了总数据的6%,使用均值填补
df.var_value = df.var_value.fillna(np.mean(df.var_value))

#%% 查看异常值
a = df[df.var_value>1000]
#df = df.drop(df[df.var_value>1000].index)
#%%
b = df[df.var_value<1]
#df = df.drop(df[df.var_value<1].index)
```

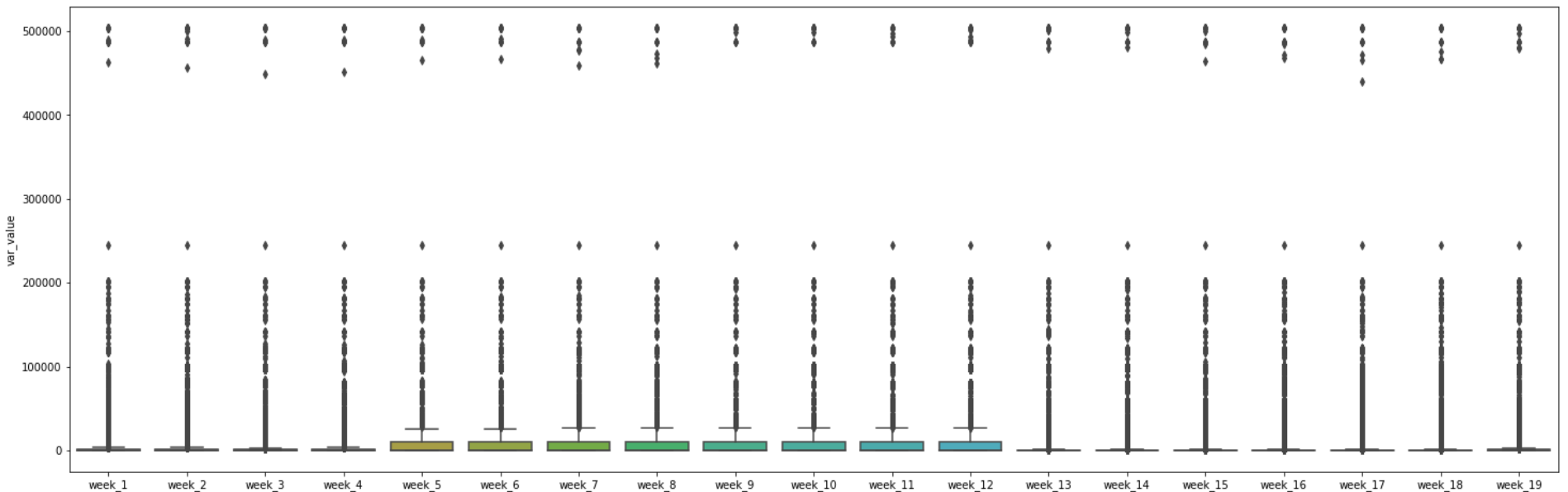
读取数据并对数据进行简单的描述性统计观察，首先存在缺失值，使用均值填补法进行填补，同时存在很多的极大值，和极小值。

```
In [6]: df.site_type.value_counts().plot(kind= 'bar')
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x2061df88808>
```

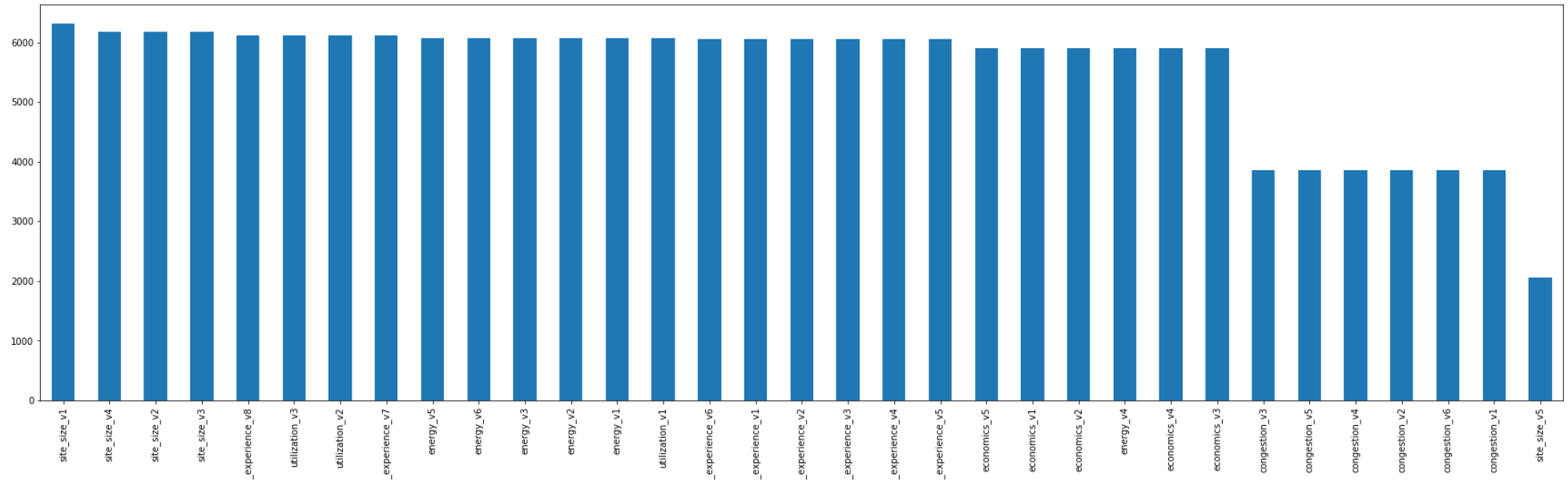


通过画图可以看出基本上的充电桩都是处于A部分，C部分也有少量的充电桩，B部分基本可以忽略不记。
同时，对不同周的充电量进行箱线图分析，发现5到12周的充电量比其他周有明显的上升趋势，并且，每周都存在之前所说的极大值。

```
In [8]: plt.figure(figsize=(25,8))
   ...: sns.boxplot(x = 'week',y = 'var_value',data = df)
   ...: plt.show()
```



```
In [10]: plt.figure(figsize=(30,8))
    ...: df.var_name.value_counts().plot(kind='bar')
    ...: plt.show()
```




```
#%%数据可视化
#大量版本的充电桩都处于低范围的值，有几个版本有很多处于高范围的值，
#site——sizev2,和v3,utilization_v3,energy版本的值普遍较高，更为突出的是V6
plt.figure(figsize=(60,15))
sns.boxplot(x = 'var_name',y = 'var_value',data = df)
plt.show()
```

通过可视化发现大量版本的充电桩都大约在6000这个数值，但有少数是4000，3000，推测可能是新版本充电桩。
同时通过第二个可视化，发现在某些版本的充电值普遍很高，推测可能是高级充电值。

```
#下面进行聚类模型构建前的准备
#%%-- 删除无用变量id
data = df.iloc[:,1:]

#%% 生成哑变量
data = pd.concat([data,pd.get_dummies(df['site_type']),pd.get_dummies(df['week']),pd.get_du

#%%
#一般会保留K-1个哑变量
del data['site_type']
del data['week']
del data['var_name']
del data['c']
del data['week_19']
del data['utilization_v2']
```

这里因为存在分类变量，所以对分类变量进行one-hot编码处理，同时生成K-1虚拟变量。然后对连续变量进行标准化。

```
#%% 进行算法之前要进行标准化！
data.info()
from sklearn import preprocessing
data.var_value = preprocessing.scale(data.var_value)
```

```
#%% 因为数据处于非常大的样本，所以需要进行Minnibatch kmeans算法处理
#这里采用了手肘法选取最佳K值
#分群效果可以从图中看出
from sklearn.cluster import  MiniBatchKMeans, KMeans
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']

SSE = []  # 存放每次结果的误差平方和
for k in range(1,50):
    km = MiniBatchKMeans(init='k-means++', n_clusters=k, batch_size=batch_size, random_state=50)
    km.fit(data)
    SSE.append(km.inertia_)

X = range(1,50)
# 绘制K的个数与SSE的关系
plt.figure(figsize=(30,10))
plt.plot(X,SSE,'o-')
plt.xlabel('聚类个数')
plt.ylabel('簇内离差平方和')
plt.title('选择最优的聚类个数')
plt.show()
```

Code analysis

'sklearn.cluster.KMeans' imported but unused

因为数据过大，所以使用了Minnibatch Kmeans算法处理这个数据，并采用了手肘法选取最佳K值，发现大约在K等于10-20左右，SSE的下降趋势变缓，推测可能的分类在10-20之间。

```
#%%
#对于上面的图中显示，K大约在10到20之间，簇的方差开始缓慢减少。所以K值取在这里比较合适
km = MiniBatchKMeans(init='k-means++', n_clusters=15, batch_size=batch_size, random_state=100)
result = km.fit(data)

#%% 把聚类结果标记在原来的数据集上
data_1=df.join(pd.DataFrame(result.labels_))
data_1=data_1.rename(columns={0: "cluster"})
data_1.head()

data_1.cluster.value_counts().plot(kind = 'bar')

#%% 把数据帧输出为csv文件
data_1.to_csv("task1.csv",index=False)
```

使用模型设置K为15，然后把分类的标签合并在原来的数据集中，并且把最后结果输出为CSV文件保存。

KPrototypes算法进行聚类

```
#%% 这里存在分类变量，所以如果使用简单kmeans分类效果可能不会太理想，把分类变量进行处理
#然后标准化连续变量，使用KPrototypes算法进行聚类
from kmodes.kprototypes import KPrototypes
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
cat = data[['site_type','week','var_name']]
data_cat = cat.apply(le.fit_transform)

data_num = data[['var_value']]
data_num.var_value = preprocessing.scale(data_num.var_value)

data_cust = pd.concat([data_cat, data_num], ignore_index=True,axis = 1)
```

因为存在了分类变量，前面也进行了聚类，但可能聚类效果不会太理想，因为one-hot编码导致了丢失了大量的信息。这里采用Kprototypes，把分类变量标记处理之后，直接和连续变量一起放入这个模型当中。

```
#%% 把数据变为矩阵
data_cust_matrix = data_cust.as_matrix()
#%% 选择最合适的K,根据使用MiniBatchKMeans的效果，猜测可能K存在不会很大
#但由于计算机问题，没有运行出结果
cost = []
for num_clusters in list(range(1,20)):
    kproto = KPrototypes(n_clusters=num_clusters, init='Cao')
    kproto.fit_predict(data, categorical=[0,1,2])
    cost.append(kproto.cost_)

plt.plot(cost)

#%% 分类结果结合到数据集上
kproto = KPrototypes(n_clusters=k, init='Cao')
clusters = kproto.fit_predict(data_cust_matrix, categorical=[0,1,2])
data['cluster'] = clusters

#%% 分类结果进行可视化
datacluster = pd.DataFrame(data['cluster'].value_counts())
sns.barplot(x=datacluster.index, y=datacluster['cluster'])

#%% 把数据帧输出为csv文件
data.to_csv("result1.csv",index=False)
```

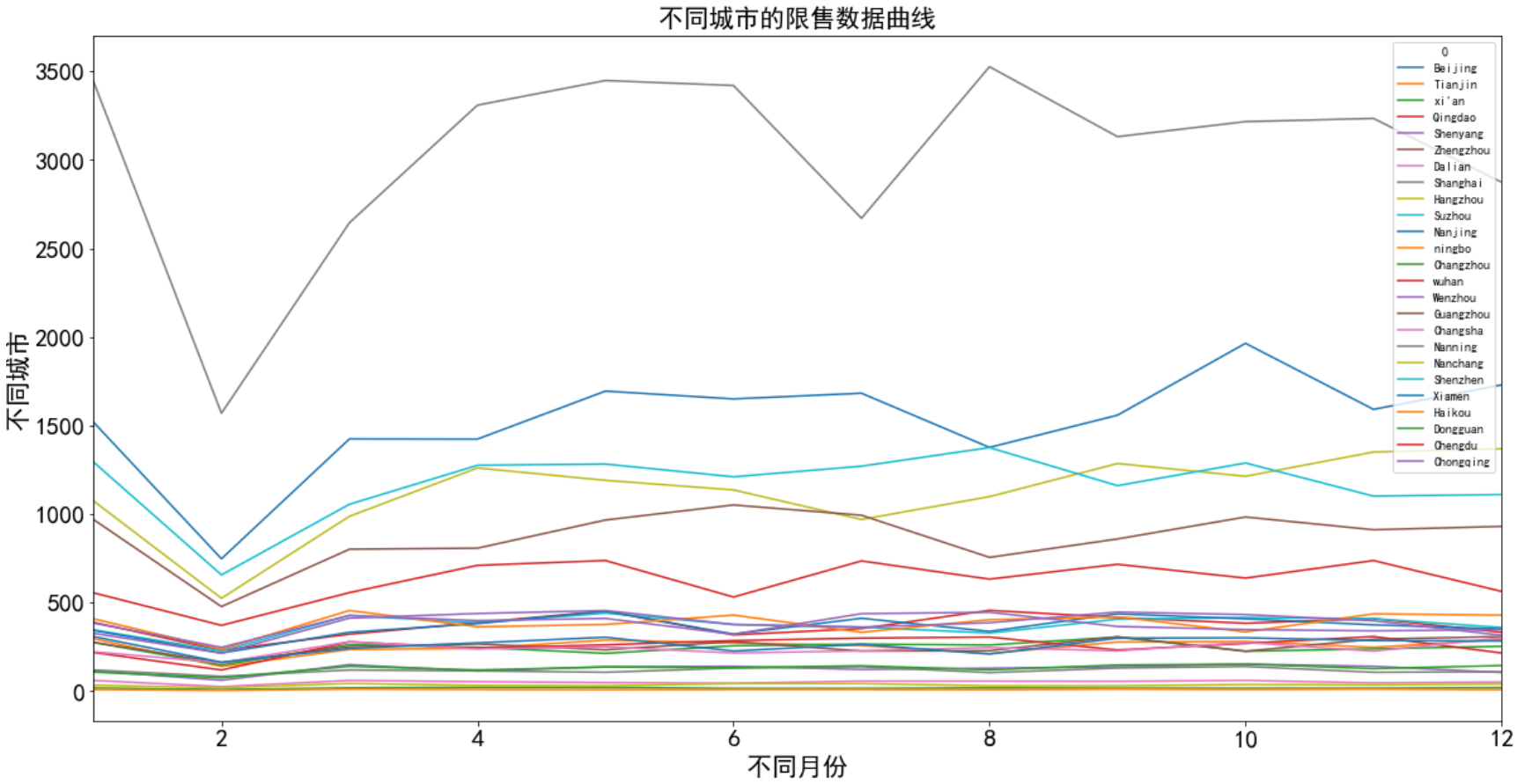
根据上面的结果，所以推测K可能不会很大，但由于计算机问题，没有运算出结果。

Task-2 时间序列预测

```python
#%%$
import numpy as np
import pandas as pd
import seaborn as sns
import os

os.chdir(r'C:\Users\REGGIE\Desktop\Machine Learning Model Group')
df = pd.read_csv('sales.csv',header=None)
pd.options.display.max_columns = None
pd.options.display.max_rows = None

#%% 通过观察数据可以得出这是一个周期性的时间序列数据
#city = df[0]
#columns = city.drop(index =0)

#%% 对数据进行变动
df= df.drop(index=0)
df = df.T
df.columns = df.iloc[0,:]
df = df.drop(index=0)
data = df
data.iloc[:,0:25].astype(int)
#%% 观察了一下这些城市的数据大小情况
print(data.iloc[:,0:25].agg(['min','mean','median','max','std']))
```

```
0   Chongqing
1         385
2         248
3         428
4         399
5         411
6         322
7         438
8         446
9         366
10        348
11        341
12        348

In [3]: print(data.iloc[:,0:25].agg(['min','mean','median','max','std']))
            Beijing     Tianjin       xi'an     Qingdao    Shenyang  \
min      748.000000  140.000000  148.000000  231.000000   62.000000
mean    1530.750000  253.916667  246.500000  371.250000  125.916667
median  1575.500000  265.500000  255.500000  386.000000  134.000000
max     1965.000000  297.000000  306.000000  457.000000  153.000000
std      294.876284   44.087637   38.937012   63.953286   24.916071
```

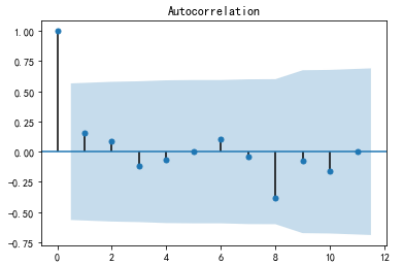
首先导入数据，发现这个数据应该是不同城市的周期性时间序列数据，所以对数据进行调整，然后简单查看每个城市的描述性统计情况。

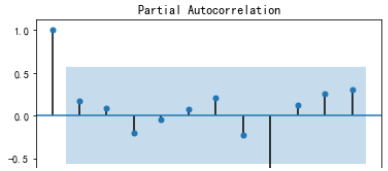


不同月份，不同的城市的销售数据曲线可视化，首先有几个城市处于非常高的销售量水准，大约在1000以上。其余城市的销售量情况基本处于500以下，可能是没有普及到这些城市，也有可能是这些城市在内陆。

```
#%%
data_1 = data['Beijing']

#%% --画出自相关性和偏相关性图，并做平稳性检验
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
data = data_1.values.astype(float)
plot_acf(data) #--自相关
plt.savefig('acf.png')
plot_pacf(data) #--偏相关
plt.savefig('pacf.png')
plt.show()

#%% --由检验结果可知原始序列是非平稳序列
from statsmodels.tsa.stattools import adfuller as ADF
print('原始序列的ADF检验结果为：', ADF(data_1))
#%%
#差分后的结果
D_data_1 = data_1.diff().dropna()
print("差分序列的ADF 检验结果为", ADF(D_data_1))

#%% 一阶差分可视化
data = D_data_1.values.astype(float)
plot_acf(data) #--自相关
plt.savefig('acf.png')
plot_pacf(data) #--偏相关
plt.savefig('pacf.png')
plt.show()
```

```
In [6]: from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
   ...: data = data_1.values.astype(float)
   ...: plot_acf(data) #--自相关
   ...: plt.savefig('acf.png')
   ...: plot_pacf(data) #--偏相关
   ...: plt.savefig('pacf.png')
   ...: plt.show()
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\regression\linear_model.py:1358: RuntimeWarning: invalid value encountered in sqrt
  return rho, np.sqrt(sigmasq)
```





对北京市进行单独分析，可视化自相关和偏相关图，并尝试平稳性检测，发现是不平稳序列，差分，发现数据为一阶存在白噪音的非平稳序列

```python
#%% --对定阶前的处理
columns = df.columns.tolist()
test = {}
for i in columns:
    city = df[i].values.astype(float)
    print(i,city)
    test[i] = city

#%%
#从一阶差分后的序列是平稳的非白噪声序列可以看出ARIMA模型中的d=1
from statsmodels.tsa.arima_model import ARIMA
for k,i in test.items():
    data = i
    pmax = int(len(data_1)/10) #一般阶数不超过length/10
    qmax = int(len(data_1)/10) #一般阶数不超过length/10
    bic_matrix = [] #bic矩阵
    for p in range(pmax+1):
        tmp = []
        for q in range(qmax+1):
            try:
                #存在部分报错，所以用try来跳过报错。
                tmp.append(ARIMA(data, (p,1,q)).fit().bic)
            except:
                tmp.append(None)
        bic_matrix.append(tmp)
    bic_matrix = pd.DataFrame(bic_matrix) #从中可以找出最小值
    p,q = bic_matrix.stack().idxmin() #先用stack展平，然后用idxmin找出最小值位置。
    print(k,p,q)
```

```
  "Check mle_retvals", ConvergenceWarning)
Guangzhou 0 1
Changsha 1 0
Nanning 0 1
Nanchang 0 1
Shenzhen 0 1
Xiamen 0 1
Haikou 0 1
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\base\mo
Inverting hessian failed, no bse or cov_params available
  'available', HessianInversionWarning)
Dongguan 0 1
Chengdu 0 1
Chongqing 0 1
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\base\mo
Inverting hessian failed, no bse or cov_params available
  'available', HessianInversionWarning)
```

由于使用ARIMA模型，需要确定三个参数，所以在这里选择BIC最低的模型，确认每个城市的不同参数取值。

```
#%%
#确定了ARIMA模型的三个参数就可以构建模型
#这里确实没找到合适的方法来进行自动化
#只能一个城市一个城市的进行预测
#预测返回的结果是为期6天的预测，返回了结果，标准误差，置信区间。
#%%
#Beijing --0 1 1
model_Beijing = ARIMA(test['Beijing'],(0,1,1)).fit()
print(model_Beijing.summary2(),'\n',model_Beijing.forecast(6)[0])
list_1 = pd.DataFrame(model_Beijing.forecast(6)[0])
```

这里对每个城市都进行预测，使用不同的参数，预测结果返回为期为6天的预测，返回了结果。（这里把12个月份作为了12天来处理。）

```
#%% --合并
predict =  pd.concat([list_1,list_2,list_3,list_4,list_5,list_6,list_7,
           list_8,list_9,list_10,list_11,list_12,list_13,list_14,
           list_15,list_16,list_17,list_18,list_19,list_20,list_21,
           list_22,list_23,list_24,list_25],axis=1)

#%% -- 改变列名
predict.columns = columns
#%% --合并原始数据和预测数据
data = pd.concat([df,predict],axis=0)

#%% 把数据帧输出为csv文件
data.to_csv("task2.csv",index=False)
```

把预测都合并为一个dataframe之后，合并原视数据和预测数据，最后保存为CSV文件输入结果。