**关于时间序列模型的调研**

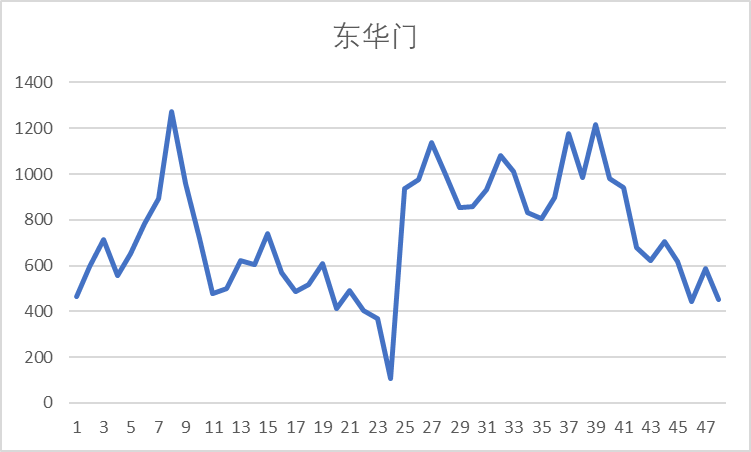
**一、常用方法**

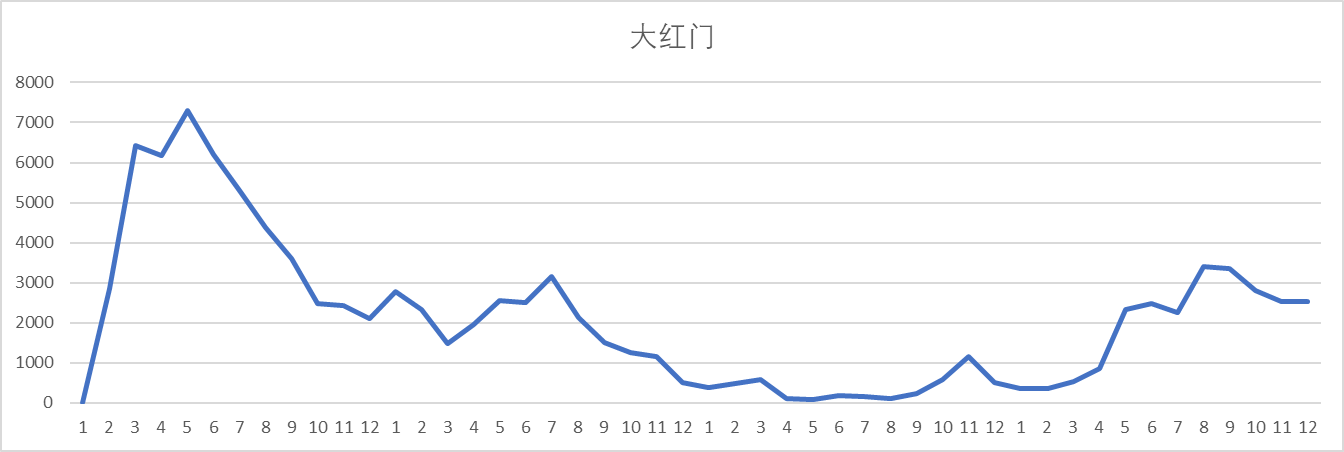
|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 特点 | 优点 | 缺点 |
| AR(Auto Regressive Model)自回归模型 | 自回归模型（Autoregressive Model）是用自身做回归变量的过程，即利用前期若干时刻的随机变量的线性组合来描述以后某时刻随机变量的线性回归模型，适用于平稳随机过程，对于非平稳的时间序列，首先应当将其平稳化。 | 1. 是线性时间序列分析模型中能最简单的模型。 2. 原理和计算都简洁实用，容易实现和掌握。 | 1.模型为线性自回归，对于复杂情况模拟效果不好  2.适用于平稳随机过程，对于非平稳的时间序列表现一般 |
| MA模型(moving average model)滑动平均模型 | MA模型和AR相似，但它并非是历史时序值的线性组合而是历史白噪声的线性组合。与AR最大的不同之处在于，AR模型中历史白噪声的影响是间接影响当前预测值的（通过影响历史时序值） | 1. 模型简单，原理和计算都简洁实用，容易实现和掌握。 | 1. 适用于平稳随机过程，对于非平稳的时间序列表现一般 |
| ARMA(Auto Regressive and Moving Average Model)自回归移动平均模型 | 自回归移动平均模型综合了AR和MA两个模型的优势，在ARMA模型中，自回归过程负责量化当前数据与前期数据之间的关系，移动平均过程负责解决随机变动项的求解问题，因此，该模型更为有效和常用。 | 1. 综合了AR和MA两个模型的优势，AR可以解决当前数据与后期数据之间的关系，MA则可以解决随机变动也就是噪声的问题。 | 1. 适用于平稳随机过程，对于非平稳的时间序列表现一般 |
| ARIMA(Auto Regressive Integrate Moving Average Model)差分自回归移动平均模型 | ARIMA模型是在ARMA模型的基础上解决非平稳序列的模型，因此在模型中会对原序列进行差分。ARIMA模型能够用于齐次非平稳时间序列的分析，这里的齐次指的是原本不平稳的时间序列经过d次差分后成为平稳时间序列。 | 1. 在ARMA模型的基础上解决非平稳序列问题 | 1. 能解决的不平稳的时间序列指的是经过d次差分后可以成为平稳时间序列的时间序列。 |
| Long Short Term Memory(LSTM)：长短时记忆神经网络 | Long Short Term Memory(LSTM)：长短时记忆神经网络，是一种特殊的循环神经网络（RNN），优势在于解决RNN的梯度消失和梯度爆炸的问题，目前广泛应用于序列数据处理和预测，比如文本上下文感情分析，股票预测等 | 1. LSTM神经元在其管道中可以保持记忆，以允许解决顺序和时间问题，而不会出现影响其性能的消失梯度问题 2. LSTM能够以非线性方式处理具有多个维度的数据 | 1. LSTM存在一些局限性，特别是在使用金融时间序列时，该系列本身具有很难建模的非平稳特性。 |

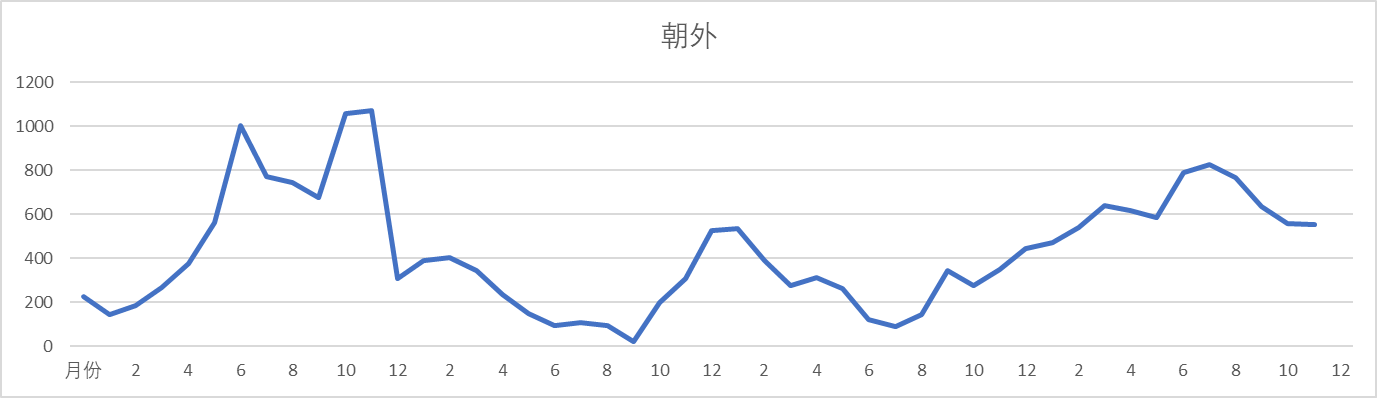
通过分析调研并结合数据本身特性，目前想法是**以LSTM时间序列模型为主对所有区域进行分析，对部分具有季节性、周期性、年变化较为平稳的区域采用ARIMA模型进行对比，对比分析其预测效果**。

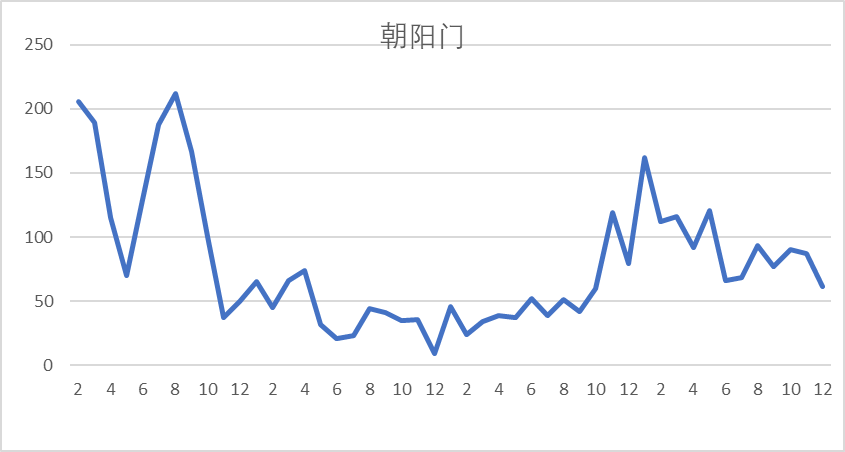
1. **数据筛选**

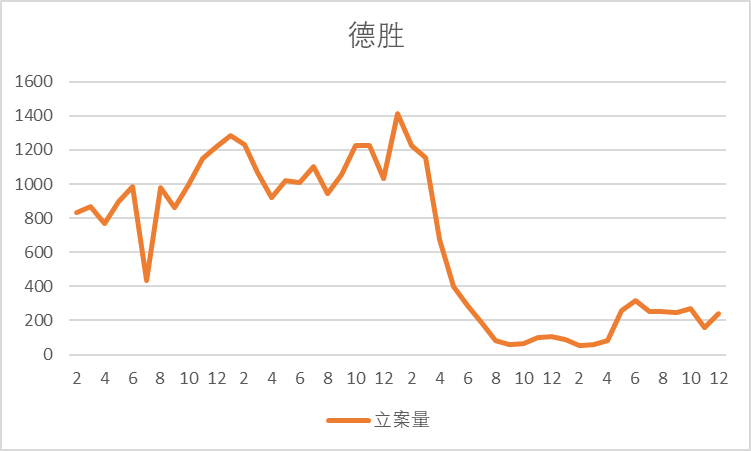
根据目前调查的模型，以及数据整体情况，目前准备筛选具有**完整48期**数据且具有**一定规律**的数据作为实验数据。例如，在下述几个区域中，由于年与年之间数据波动过大，拟淘汰**白纸坊和德胜**区域的数据，保留剩下几个区域。

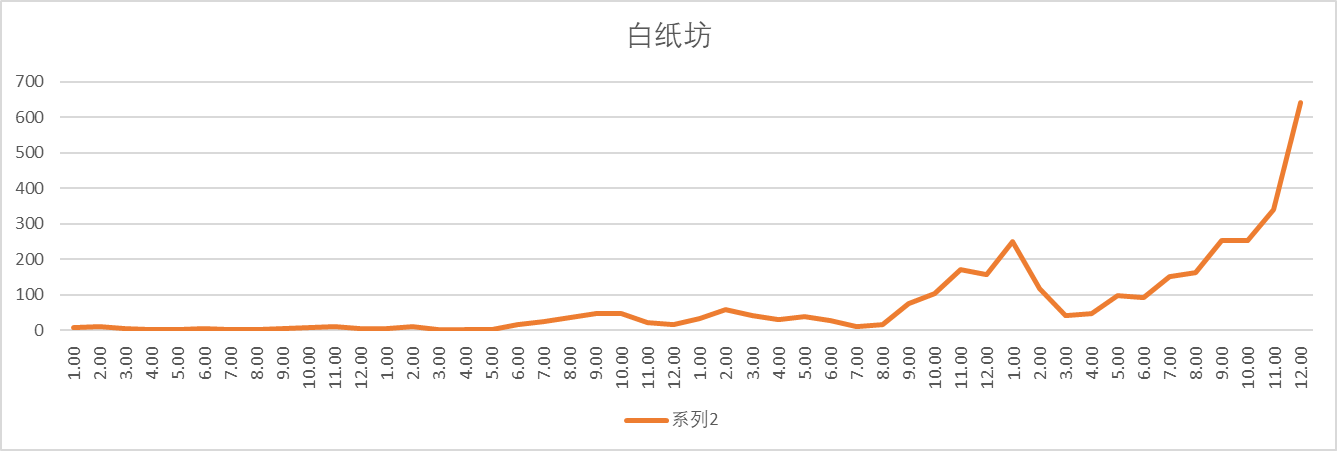








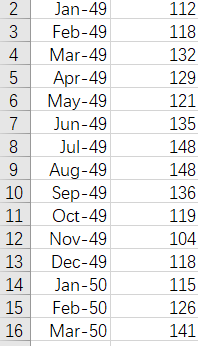




1. **下阶段工作安排**

继续完成50个社区5种问题的数据筛选工作，初步把握数据规律，选取其中数据量充足，具有一定规律性的适合模型的数据；辅助师姐使用LSTM模型进行实验；进一步进行时序模型调研，积累理论知识。

**附：LSTM时间序列模型预测举例**

此数据是1949 到 1960 一共 12 年，每年 12 个月的航班乘客数据，一共 144 个数据，单位是 1000。  
**数据如图所示**  
  
第一列为时间 第二列为数据

**编写代码**

**头文件**

import numpy

import matplotlib.pyplot as plt

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

import pandas as pd

import os

from keras.models import Sequential, load\_model

**加载数据**  
在这里我们设置时序数据的前65%为训练数据 后35%为测试数据

dataframe = read\_csv('./international-airline-passengers.csv', usecols=[1], engine='python', skipfooter=3)

dataset = dataframe.values

# 将整型变为float

dataset = dataset.astype('float32')

train\_size = int(len(dataset) \* 0.65)

trainlist = dataset[:train\_size]

testlist = dataset[train\_size:]

**对数据进行处理**  
LSTM进行预测需要的是时序数据 根据前timestep步预测后面的数据  
假定给一个数据集  
{  
A,B,C->D  
B,C,D->E  
C,D,E->F  
D,E,F->G  
E,F,G->H  
}  
这时timestep为3，即根据前三个的数据预测后一个数据的值  
所以我们需要对数据进行转化  
举一个简单的情况 假设一个list为[1,2,3,4,5],timestep = 2  
我们转化之后要达到的效果是

|  |  |
| --- | --- |
| **train\_X** | **train\_Y** |
| [1,2] | [3] |
| [2,3] | [4] |
| [3,4] | [5] |

**即依据前两个值预测下一个值**

**对数据进行归一化**  
LSTM可以不进行归一化的操作，但是这样会让训练模型的loss下降很慢。本教程如果不进行归一化，100次迭代后loss还是很高

scaler = MinMaxScaler(feature\_range=(0, 1))

dataset = scaler.fit\_transform(dataset)

**对数据进行处理**

def create\_dataset(dataset, look\_back):

#这里的look\_back与timestep相同

dataX, dataY = [], []

for i in range(len(dataset)-look\_back-1):

a = dataset[i:(i+look\_back)]

dataX.append(a)

dataY.append(dataset[i + look\_back])

return numpy.array(dataX),numpy.array(dataY)

#训练数据太少 look\_back并不能过大

look\_back = 1

trainX,trainY = create\_dataset(trainlist,look\_back)

testX,testY = create\_dataset(testlist,look\_back)

**LSTM模型**  
LSTM的输入为 [samples, timesteps, features]  
这里的timesteps为步数，features为维度 这里我们的数据是1维的

trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))

testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1] ,1 ))

# create and fit the LSTM network

model = Sequential()

model.add(LSTM(4, input\_shape=(None,1)))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

model.fit(trainX, trainY, epochs=100, batch\_size=1, verbose=2)

model.save(os.path.join("DATA","Test" + ".h5"))

# make predictions

**进行预测**

#model = load\_model(os.path.join("DATA","Test" + ".h5"))

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

#反归一化

trainPredict = scaler.inverse\_transform(trainPredict)

trainY = scaler.inverse\_transform(trainY)

testPredict = scaler.inverse\_transform(testPredict)

testY = scaler.inverse\_transform(testY)

**查看结果**

plt.plot(trainY)

plt.plot(trainPredict[1:])

plt.show()

plt.plot(testY)

plt.plot(testPredict[1:])

plt.show()

**这个时候结果为**  
