## 天气预报：数据采集+可视化+单变量线性回归模型

[天气预报：数据采集+可视化+单变量线性回归模型](#header-n60)  
 [一、数据采集](#header-n63)  
 [1.审查网页元素](#header-n66)  
 [2.建立请求头](#header-n74)  
 [3.数据爬取+数据存储](#header-n80)  
 [需要模块](#header-n83)  
 [具体过程](#header-n84)  
 [代码](#header-n95)  
 [二、数据可视化](#header-n97)  
 [需要模块](#header-n101)  
 [数据清洗](#header-n155)  
 [可视化](#header-n162)  
 [三、单变量线形回归](#header-n182)  
 [需要模块](#header-n191)  
 [建立模型](#header-n184)  
 [定义自变量和因变量](#header-n207)  
 [调用模型创建预测模型](#header-n188)   
 [模型评测](#header-n205)  
 [模型评价](#header-n237)  
 [四、其他想法](#header-n241)

### 一、数据采集

#### 1.审查网页元素

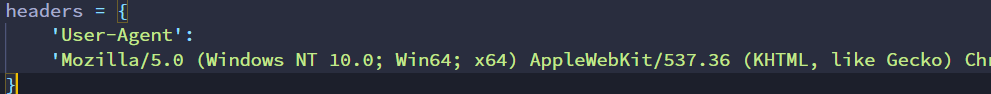
打开杭州市8月份的天气预报(http://www.tianqihoubao.com/lishi/hangzhou/month/202008.html)，审查元素。

发现信息包括：日期、天气状况、气温、风力状况，这四个信息都存在<tr>元素中，每个<tr>元素中有四个<td>元素， 分别存放这四项信息。所以在爬取8月份的数据时，只需要对着30个<tr>元素进行循环存储就可以了。



#### 2.建立请求头

分析好页面后，就可以开始爬虫，在正式爬虫前，需要建立一些请求头(这个网站暂时没有发现反爬的措施，不加请求头也是可以的)。使用postman工具可以快速伪造出请求头的参数。伪造的请求头形如下：



#### 3.数据爬取+数据存储

##### 需要模块

* request(建立对网页的请求)
* bs4(对网页的元素进行解析)
* pandas(对获得数据进行清洗存放)

##### 具体过程

使用request获取网页的内容，用beautiful库将获取的内容进行解析成为课寻找元素的对象。在这个对象中寻找所有的<tr>元素，在周到的所有元素中获取text内容，并将这些文本使用split()方法切割成列表，然后再经过调试确定需要寻找的内容所在的位置，使用切片和字符串连接分别存入三个变量(日期，天气情况，气温)中，再将这三个变量打包成一个列表，每次循环将该列表存入20天的大列表中，最后在主函数中使用pandas库将列表中的数据存入表格中。将这些代码封装成函数即可。

##### 代码

# -\*- encoding: utf-8 -\*-   
# here put the import lib   
import requests   
from bs4 import BeautifulSoup   
import pandas as pd   
   
# 获取天气数据   
def get\_data():   
 url = "http://www.tianqihoubao.com/lishi/hangzhou/month/202008.html"   
 Weathers = []   
 headers = {   
 'User-Agent':   
 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36',   
 }   
 # get方式请求网页 获取源码文本   
 strpage = requests.request("GET", url, headers = headers).text   
 # 将源码解析 缩小范围 用'lxml'加快解析速度   
 Soup = BeautifulSoup(strpage, 'lxml')   
 # 寻找所有tr表格元素   
 All\_tr = Soup.find\_all('tr')   
 # 在获取的tr元素中查找相对应的信息元素   
 for x in All\_tr[1:21]:   
 # 拆分字符串，形成数组   
 List = x.text.split()   
 # 日期   
 date = ''.join(List[0])   
 # 天气情况   
 weather = ''.join(List[1:3])   
 # 气温   
 T = ''.join(List[3:6])   
 # 将三大信息加入到/list中   
 Weathers.append([date, weather, T])   
 return Weathers

### 二、数据可视化

#### 需要模块

* matplotlib.pyplot (python图形库)
* numpy (将数据转换成矩阵)
* import pandas (对表格的处理)

#### 数据清洗

获取的数据有些不需要放在图像中，所以需要对表格中的数据进行一定的处理，主要用到的是split()、replace()方法。其中有一个比较特殊的情况是得到的日期为字符，不是数字，所以需要将字符转换成数字，这样才能在坐标轴中正确的排序。

data['最高气温'] = data['气温'].str.split('/', expand=True)[0]   
data['最高气温'] = data['最高气温'].map(lambda x: int(x.replace('℃', '')))   
data['最低气温'] = data['气温'].str.split('/', expand=True)[1]   
data['最低气温'] = data['最低气温'].map(lambda x: int(x.replace('℃', '')))   
data['日期'] = data['日期'].map(lambda x: int(x.replace('2020年08月', '')[:-1]))

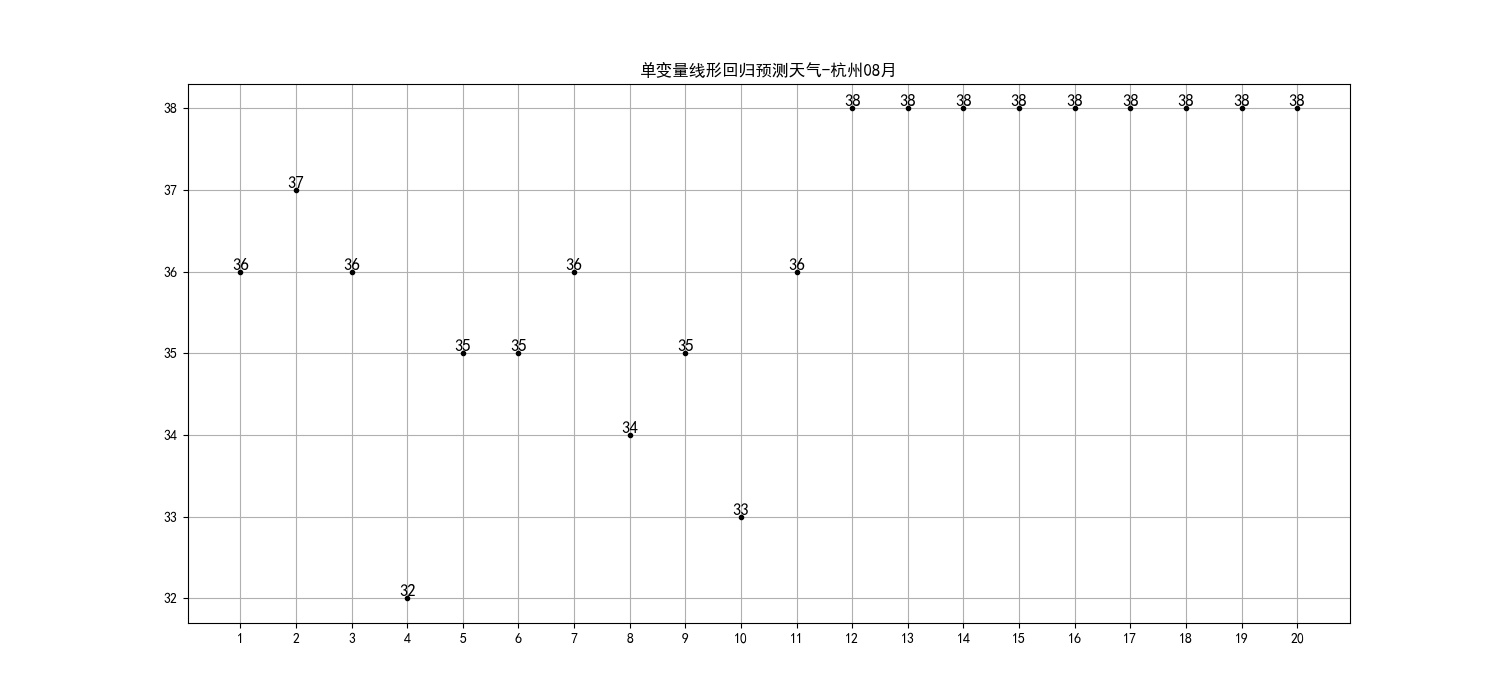
#### 可视化

在画图之前，需要创建画板，并对他进行初始化：

def initplot():   
 plt.figure(figsize=(15, 7))   
 plt.grid(True)   
 x\_major\_locator = MultipleLocator(1)   
 y\_major\_locator = MultipleLocator(1)   
 # ax为两条坐标轴的实例   
 ax = plt.gca()   
 # 把x轴的主刻度设置为1的倍数   
 ax.xaxis.set\_major\_locator(x\_major\_locator)   
 #把y轴的主刻度设置为1的倍数   
 ax.yaxis.set\_major\_locator(y\_major\_locator)   
 plt.yticks(np.arange(30, 40))   
 plt.xticks(np.arange(1, 24))   
 plt.title("单变量线形回归预测天气-杭州08月")

初始化面板后，开始画图。在画图结束后在每个点添加数据标签，使数据可视化更加清晰：

plt.plot(data['日期'], data['最高气温'], 'k.')   
for a, b in zip(data['日期'], data['最高气温']):   
 plt.text(a, b, b, ha="center", va="bottom", fontsize=12)   
plt.show()



### 三、单变量线形回归

#### 需要模块

* *from* sklearn.linear\_model *import* LinearRegression

#### 建立模型

##### 定义自变量和因变量

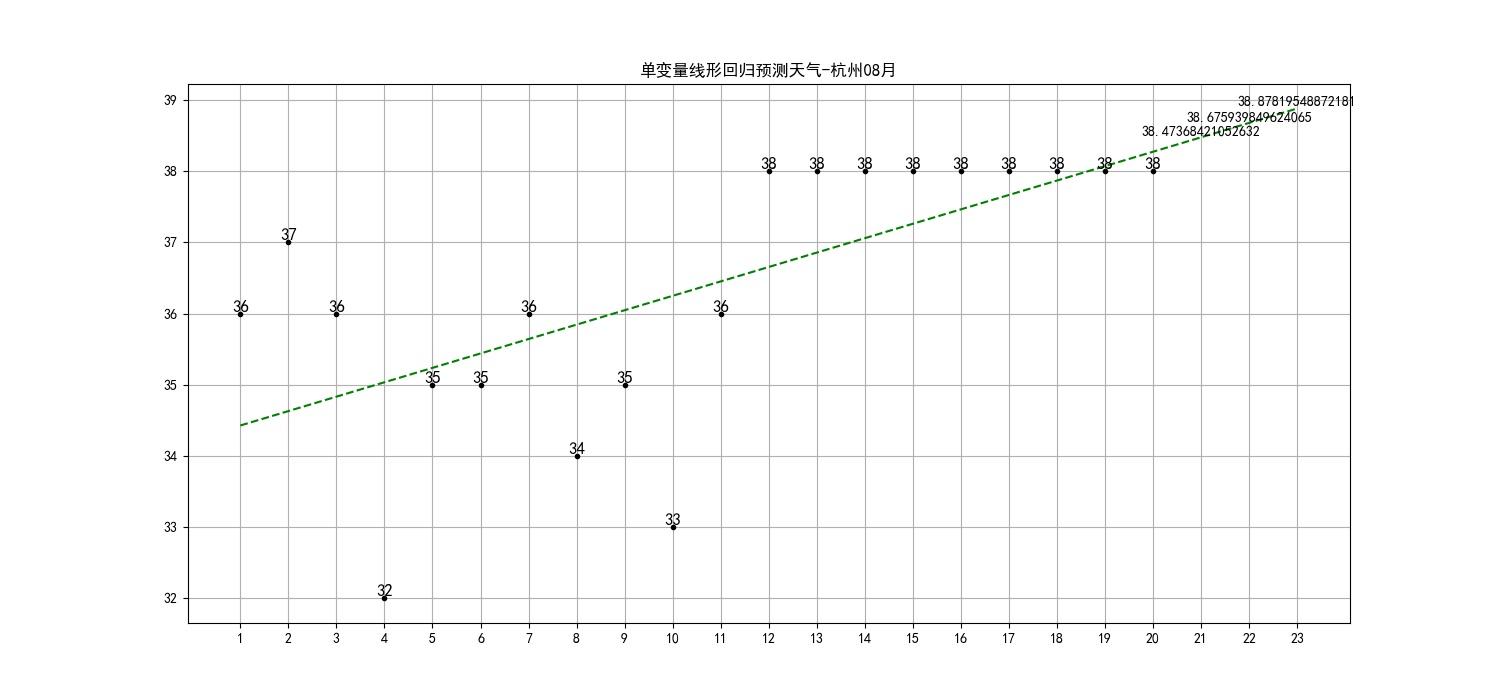
首先将清洗好的日期和最高气温使用numpy模块转换成矩阵 xTrain为训练数据的因变量，yTrain为训练数据的结果值。

# 将日期转为矩阵   
xTrain = np.array(data['日期'])[:, np.newaxis]   
# 将最高气温转换成列向量   
yTrain = np.array(data['最高气温'])

##### 调用模型创建预测模型

调用LinearRegression()对象预测,得到截距斜率等数据,通过该对象的predict()方法获取预测数据,并转换成矩阵传入plot中得到对象图像.

model = LinearRegression()   
# 根据训练数据拟合出直线(以得到假设函数)   
hypothesis = model.fit(xTrain, yTrain)   
# 截距   
print("直线截距=", hypothesis.intercept\_)   
# 斜率   
print("直线斜率=", hypothesis.coef\_)   
print("单变量线性回归预测:2020年8月21日的最高气温：", model.predict([[21]])[0])   
   
# 将预测值在图像中显示出来   
for i in [21, 22, 23]:   
 plt.text(i,   
 model.predict([[i]])[0],   
 model.predict([[i]])[0],   
 ha="center",   
 va="bottom",   
 fontsize=10)   
# 将从开始到预测的日期转换成矩阵形式   
xNew = np.array([   
 1,   
 23,   
])[:, np.newaxis]   
# 传入日期矩阵，获取预测值   
yNew = model.predict(xNew)   
# 用虚线画出图像   
plt.plot(xNew, yNew, 'g--')



##### 模型评测

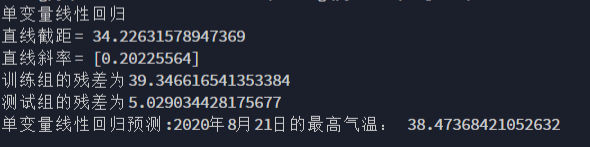
评测的方法主要依靠训练组和测试组的残差平方和和R方评价。

训练组的残差平方和：ssResTrain = sum((hpyTrain - yTrain) \*\* 2) --- model.\_residues

测试组的残差平方和：ssResTest = sum((hpyTest - yTest) \*\* 2) --- model.score(xTest, yTest)

R方：Rsquare = 1 - ssResTest / ssTotTest

# 引入测试组横坐标，并转换成矩阵   
xtest = np.array([3, 6, 9, 12, 15, 18])[:, np.newaxis]   
# 引入测试组的纵坐标，转换成矩阵   
ytest = np.array([36, 35, 35, 38, 38, 38])   
# 测试组的预测值   
hpyTest = model.predict(xtest)   
# 计算测试组的残差   
ssResTest = sum((ytest - hpyTest)\*\* 2)   
print(f"训练组的残差为{model.\_residues}")   
print(f"测试组的残差为{ssResTest}")



##### 模型评价

对于该模型，其实日期和温度没有很大的关系，座椅该模型的残差会很大，而且预测得到的数据没有具体意义。

### 四、其他想法

日期和温度几乎没有关联性，在设计模型的时候可以设置权重，按照月份每个月份都可以设置一个基础温度，然后根据天气情况(晴，多云，下雨等)和湿度对温度预测。

基础温度：可以从前几年的数据中的所有月进行平均值处理，然后根据气温上升得到趋势，预测今年每个月的基础温度。

天气情况：天气情况这四个天气可以设置权重值a，并根据天气具体情况x进行哈希处理 {x：a}组成字典，用于表示基础温度变化的影响程度，可正可负。

湿度：湿度和天气情况的道理一样，设置一个权重b，并根据湿度的范围y进行哈希{y：b} ，用于表示对基础温度变化的影响程度，可正可负。