## 2.1.4 简单回归与多元回归[1][2][3]

在相关性分析部分计算了两两之间的相关系数，但在相关系数计算中并没有区分自变量（independent variable）或预测变量（predictor variable）亦或解释变量，与因变量(dependent variable)或结果变量（outcome variable）亦或效标变量（criterion variable），而回归分析除了可以反映变量间关系的性质和强度，也可以根据自变量预测因变量，进行统计推断，其预测的准确程度取决于相关的程度，相关性越强，预测就越准确。根据自变量的数量可以将回归分为简单回归和多元回归。多元回归包括两个或两个以上的自变量及一个因变量，那么这多个自变量作为一个整体与因变量有多大关系；做为个体，各个自变量与因变量的关系强度如何；同时，自变量之间的相对强度又如何，及自变量之间是否存在交互效应等，回答上述问题后，基本就解释了自变量和因变量之间的关系。

回归涉及的内容异常丰富，回归的类型丰富多彩，在实际应用回归来预测因变量时，并不会从最基础的开始一步步计算，而是直接使用Python中集成的库，这时就涉及到了著名的机器学习库[scikit-learn（sklearn）](https://scikit-learn.org/stable/index.html)①库，该库包含了数据的预处理、降维、分类模型、聚类模型和回归模型，及模型选择等内容，能够辅助处理众多城市空间数据分析问题。同时也可以使用[statsmodels](https://www.statsmodels.org/stable/index.html)②库等。scikit-learn库倾向于预测，而statsmodels库倾向于统计推断。但如果一开始就使用集成的模型，对于理解回归，乃至任何统计学的知识点或者数据分析的算法都是不利的，往往囫囵吞枣、一知半解，因此亦然一步步，借助Python语言来把这个最为基础部分的脉络梳理清楚。

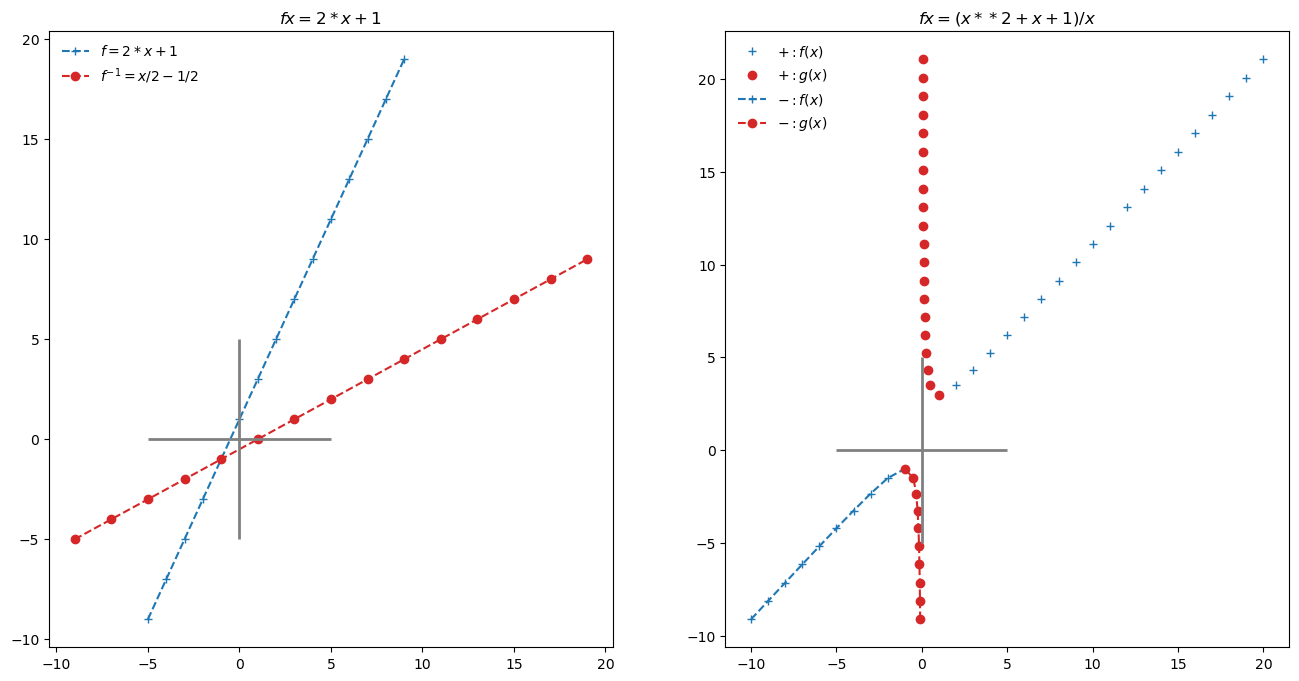
### 2.1.4.1 预备知识

#### 1）反函数（inverse function）

如果函数f(x)有y=f(x)， f是y关于自变量x的函数；如果存在一个函数g，使得， g(y)=g(f(x))=x，则g是x关于自变量y的函数，称g为f的反函数。若一函数有反函数，此函数便是可逆的。可记作函数f和它的反函数f−1。假设函数f=2x+1，则x=(f/2−1/2)，替换x为f−1， f为x，则结果为反函数f−1=x/2−1/2。在下述代码表述假设函数及其反函数时，使用了[SymPy](https://docs.sympy.org/latest/index.html" \t "_blank)③代数计算库，轻量型的SymPy的语法方式保持了Python语言本身语法形式特点，使得数学公示的表述和计算上清晰简便。打印图表f和它的反函数f−1，可以看到，二者的相对横纵坐标发生了置换。求一个函数的反函数，当前有很多在线自动转换的平台，可以搜索输入公式获得其反函数的公式。例如[Symbolab](https://www.symbolab.com/" \t "_blank)④，如果，则其反函数为*g*(*x*)= ，及*g*(*x*)= 

import sympy  
from sympy import init\_printing,pprint,sqrt  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
init\_printing() # sympy提供有多种公式打印模式  
  
# 示例-A  
# 定义字符  
x=sympy.Symbol('x')  
  
# 定义表达式  
f=2\*x+1 # 函数fx  
g=x/2-1/2 # fx的反函数  
  
# 转换表达式为等价的numpy函数实现数值计算  
x\_array=np.arange(-5,10)  
f\_=sympy.lambdify(x,f,"numpy")  
g\_=sympy.lambdify(x,g,"numpy")  
  
# 求解函数并绘制图表  
fig, axs=plt.subplots(1,2,figsize=(16,8))  
axs[0].plot(x\_array,f\_(x\_array),'+--',color='tab:blue',label='$f=2\*x+1$')  
axs[0].plot(f\_(x\_array),g\_(f\_(x\_array)),'o--',color='tab:red',label='$f^{-1}=x/2-1/2$')  
  
axs[0].set\_title('$fx=2\*x+1$')  
axs[0].legend(loc='upper left', frameon=False)  
axs[0].hlines(y=0,xmin=-5,xmax=5,lw=2,color='gray')  
axs[0].vlines(x=0,ymin=-5,ymax=5,lw=2,color='gray')  
  
# 示例-B  
f\_B=(x\*\*2+x+1)/x  
print("使用pprint方式打印公式：")  
pprint(f\_B,use\_unicode=True) # 使用pprint方式打印公式  
g\_B\_negative=(-1+x+sqrt(x\*\*2-2\*x-3))/2  
g\_B\_positive=(-1+x-sqrt(x\*\*2-2\*x-3))/2  
  
f\_B\_=sympy.lambdify(x,f\_B,"numpy")  
g\_B\_positive\_=sympy.lambdify(x,g\_B\_positive,"numpy")  
g\_B\_negative\_=sympy.lambdify(x,g\_B\_negative,"numpy")  
  
x\_B\_array=np.arange(-10,21)  
x\_B\_array\_positive=x\_B\_array[x\_B\_array>0]  
axs[1].plot(x\_B\_array\_positive,f\_B\_(x\_B\_array\_positive),'+',color='tab:blue',label='$+:f(x)$')  
axs[1].plot(g\_B\_positive\_(f\_B\_(x\_B\_array\_positive)),f\_B\_(x\_B\_array\_positive),'o',color='tab:red',label='$+:g(x)$')  
  
x\_B\_array\_negative=x\_B\_array[x\_B\_array<0]  
axs[1].plot(x\_B\_array\_negative,f\_B\_(x\_B\_array\_negative),'+--',color='tab:blue',label='$-:f(x)$')  
axs[1].plot(g\_B\_negative\_(f\_B\_(x\_B\_array\_negative)),f\_B\_(x\_B\_array\_negative),'o--',color='tab:red',label='$-:g(x)$')  
  
axs[1].hlines(y=0,xmin=-5,xmax=5,lw=2,color='gray')  
axs[1].vlines(x=0,ymin=-5,ymax=5,lw=2,color='gray')  
axs[1].legend(loc='upper left', frameon=False)  
axs[1].set\_title('$fx=(x\*\*2+x+1)/x$')  
  
plt.show()  
print("JupyterLab直接输出公式:g\_B\_negative=")  
g\_B\_negative # 用JupyterLab直接输出公式

使用pprint方式打印公式：  
 2   
x + x + 1  
──────────  
 x



JupyterLab直接输出公式:g\_B\_negative=

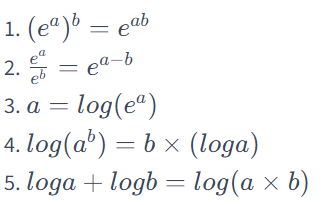
#### 2）指数函数与自然对数函数

指数函数（Exponential function）是形式为的数学函数，其中b是底数（或称基数，base），而x是指数（index/exponent）。

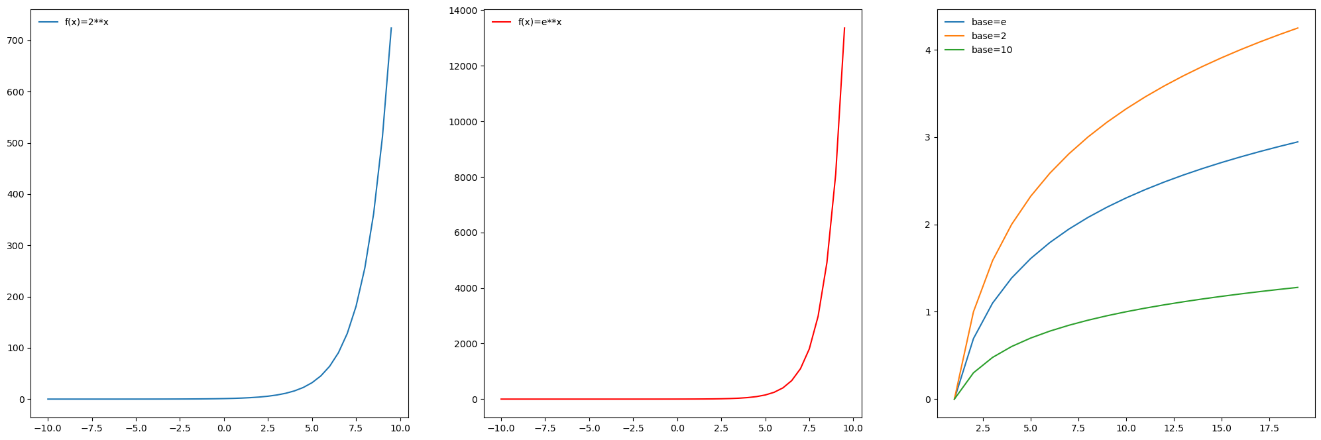
对数（logarithm）是幂运算的逆运算，假如，则有y=logb​x，其中b是对数的底（或称基数），而y就是x对于底数b， x的对数。典型的底数有e、10或2。

自然对数（Natural logarithm）为以数学常数e为底数的对数函数，标记为lnx或，其反函数为指数函数。

* 指数函数与对数函数的性质：



import math  
from sympy import ln,log,Eq  
  
x=sympy.Symbol('x')  
f\_exp=2\*\*x  
f\_exp\_=sympy.lambdify(x,f\_exp,"numpy")  
  
fig, axs=plt.subplots(1,3,figsize=(25,8))  
exp\_x=np.arange(-10,10,0.5,dtype=float)  
axs[0].plot(exp\_x,f\_exp\_(exp\_x),label="f(x)=2\*\*x")  
axs[0].legend(loc='upper left', frameon=False)  
  
f\_exp\_e=math.e\*\*x  
f\_exp\_e\_=sympy.lambdify(x,f\_exp\_e,"numpy")  
axs[1].plot(exp\_x,f\_exp\_e\_(exp\_x),label="f(x)=e\*\*x",color='r')  
axs[1].legend(loc='upper left', frameon=False)  
  
log\_x=np.arange(1,20,dtype=float)  
f\_ln=ln(x)   
f\_ln\_=sympy.lambdify(x,f\_ln,"numpy")  
axs[2].plot(log\_x,f\_ln\_(log\_x),label='base=e')  
  
f\_log\_2=log(x,2)  
f\_log\_2\_=sympy.lambdify(x,f\_log\_2,"numpy")  
axs[2].plot(log\_x,f\_log\_2\_(log\_x),label="base=2")  
  
f\_log\_10=log(x,10)  
f\_log\_10\_=sympy.lambdify(x,f\_log\_10,"numpy")  
axs[2].plot(log\_x,f\_log\_10\_(log\_x),label="base=10")  
axs[2].legend(loc='upper left', frameon=False)  
  
plt.show()

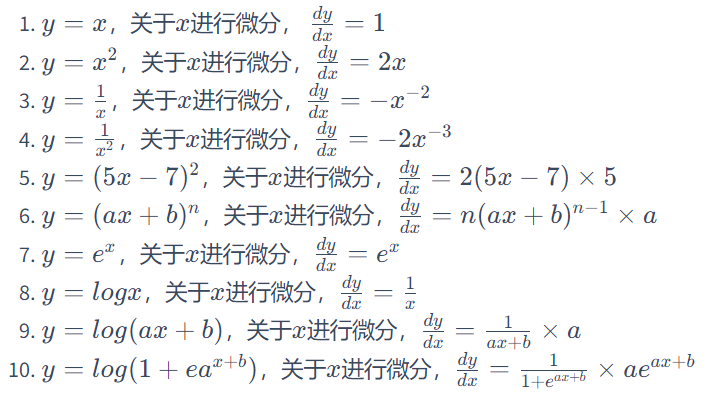


#### 3）回归与微分

首先根据《漫画统计学之回归分析》中美羽的年龄和身高数据建立数据集，实现计算年龄和身高的相关系数，结果p-value<0.05，即pearson’s r=0.942的相关系数能够说明年龄和身高直接存在强相关关系。既然二者之间存在相关性，就可以建立回归方程。在下述代码中给出了三种回归模型（方程），一种是《漫画统计学之回归分析》给出的方程，另外两种是直接使用sklearn库Linear Models线性模型中的LinearRegression线性回归，和基于LinearRegression的Polynomial regression多项式回归。关于sklearn的语法规则，可以参考官方网站scikit-learn给出的指南，sklearn的语法结构秉承了Python自身语法特点，具有很强的易读性，代码编写流畅自然。三种回归模型中，以多项式回归拟合的最好，《漫画统计学之回归分析》中给出的公式次之，而简单粗暴的简单线性回归因为呈现线性，与真实值近似对数函数曲线的形状相异。

微分是对函数的局部变化率的一种线性描述，可以近似的描述当函数自变量的取值足够小时的改变时，函数的值是怎样变化的。关于x求微分，即是求x岁到x岁之后极短时间内，身高的年平均增长量（自变量以年为单位），​，对于微分的计算，可以直接使用SymPy提供的diff工具计算，计算结果记作。

* 常用函数求微分示例：

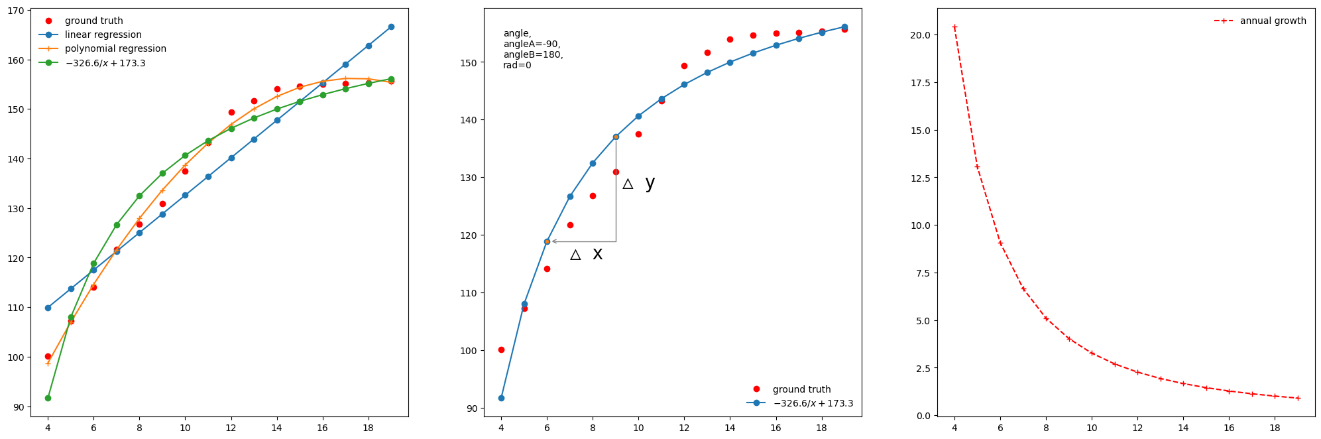


在代码的领域里直接用SymPy的diff方法，或其它库提供的方法计算。

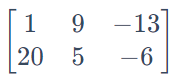
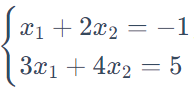
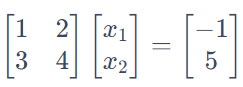
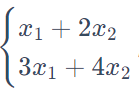
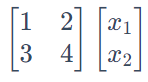
微分可以阅读该书微积分基础的代码表述部分

import pandas as pd  
from scipy import stats  
  
emma\_statureAge={"age":list(range(4,20)),"stature":[100.1,107.2,114.1,121.7,126.8,130.9,137.5,143.2,149.4,151.6,154.0,154.6,155.0,155.1,155.3,155.7]}  
emma\_statureAge\_df=pd.DataFrame(emma\_statureAge)  
  
r\_=stats.pearsonr(emma\_statureAge\_df.age,emma\_statureAge\_df.stature)  
print(  
 "pearson's r:",r\_[0],"\n",  
 "p\_value:",r\_[1]  
 )  
  
# 原始数据散点图  
fig, axs=plt.subplots(1,3,figsize=(25,8))  
axs[0].plot(emma\_statureAge\_df.age,emma\_statureAge\_df.stature,'o',label='ground truth',color='r')  
  
# A - 使用sklearn库sklearn.linear\_model.LinearRegression()，Ordinary least squares Linear Regression-普通最小二乘线性回归，获取回归方程  
from sklearn.linear\_model import LinearRegression  
X=emma\_statureAge\_df.age.to\_numpy().reshape(-1,1)  
y=emma\_statureAge\_df.stature.to\_numpy()  
  
# 拟合模型  
LR=LinearRegression().fit(X,y)  
  
# 模型参数  
print("slop:%.2f,intercept:%.2f"%(LR.coef\_, LR.intercept\_))  
print(LR.get\_params())  
  
# 模型预测  
axs[0].plot(emma\_statureAge\_df.age,LR.predict(X),'o-',label='linear regression')  
  
# B - 多项式回归 Polynomial regression  
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.linear\_model import LinearRegression  
from sklearn.pipeline import Pipeline  
model=Pipeline([('poly', PolynomialFeatures(degree=2)),  
 ('linear', LinearRegression(fit\_intercept=False))])  
reg=model.fit(X,y)  
axs[0].plot(emma\_statureAge\_df.age,reg.predict(X),'+-',label='polynomial regression')  
  
# C - 使用'漫画统计学之回归分析'给出的公式  
from sympy import Symbol  
x=Symbol('x')  
f\_emma=-326.6/x+173.3  
f\_emma\_=sympy.lambdify(x,f\_emma,"numpy")  
axs[0].plot(emma\_statureAge\_df.age,f\_emma\_(emma\_statureAge\_df.age),'o-',label='$-326.6/x+173.3$')  
  
axs[1].plot(emma\_statureAge\_df.age,emma\_statureAge\_df.stature,'o',label='ground truth',color='r')  
axs[1].plot(emma\_statureAge\_df.age,f\_emma\_(emma\_statureAge\_df.age),'o-',label='$-326.6/x+173.3$')  
  
def demo\_con\_style(a\_coordi,b\_coordi,ax,connectionstyle):  
 '''  
 function - 在matplotlib的子图中绘制连接线。参考： matplotlib官网Connectionstyle Demo  
   
 Params:  
 a\_coordi - a点的x，y坐标；tuple  
 b\_coordi - b点的x，y坐标；tuple  
 ax - 子图；ax(plot)  
 connectionstyle - 连接线的形式；string  
   
 Returns:  
 None  
 '''  
 x1, y1=a\_coordi[0],a\_coordi[1]  
 x2, y2=b\_coordi[0],b\_coordi[1]  
  
 ax.plot([x1, x2], [y1, y2], ".")  
 ax.annotate("",  
 xy=(x1, y1), xycoords='data',  
 xytext=(x2, y2), textcoords='data',  
 arrowprops=dict(arrowstyle="->", color="0.5",  
 shrinkA=5, shrinkB=5,  
 patchA=None, patchB=None,  
 connectionstyle=connectionstyle,  
 ),  
 )  
  
 ax.text(.05, .95, connectionstyle.replace(",", ",\n"),  
 transform=ax.transAxes, ha="left", va="top")  
  
dx=3  
demo\_con\_style((6,f\_emma.evalf(subs={x:6})),(6+dx,f\_emma.evalf(subs={x:6+dx})),axs[1],"angle,angleA=-90,angleB=180,rad=0")   
axs[1].text(7, f\_emma.evalf(subs={x:6})-3, "△ x", family="monospace",size=20)  
axs[1].text(9.3, f\_emma.evalf(subs={x:9.3})-10, "△ y", family="monospace",size=20)  
  
# 用sympy提供的diff方法求微分  
from sympy import diff  
print("f\_emma=-326.6/x+173.3关于x求微分：")  
pprint(diff(f\_emma),use\_unicode=True)   
diff\_f\_emma\_=sympy.lambdify(x,diff(f\_emma),"numpy")  
axs[2].plot(emma\_statureAge\_df.age,diff\_f\_emma\_(emma\_statureAge\_df.age),'+--',label='annual growth',color='r')  
  
axs[2].legend(loc='upper right', frameon=False)  
axs[1].legend(loc='lower right', frameon=False)  
axs[0].legend(loc='upper left', frameon=False)  
plt.show()

pearson's r: 0.9422225583501309   
 p\_value: 4.943118398567093e-08  
slop:3.78,intercept:94.82  
{'copy\_X': True, 'fit\_intercept': True, 'n\_jobs': None, 'normalize': 'deprecated', 'positive': False}  
f\_emma=-326.6/x+173.3关于x求微分：  
326.6  
─────  
 2   
 x



#### 4）矩阵

一个*m*×*n*的矩阵是一个由*m*行（row）*n*列（column）元素排列成的矩形阵列。矩阵里的元素可以是数字、符号或数学式。例如：，如果​可以写作：，而如果，可以写作：。

矩阵的操作和运算可以直接应用SymPy库的Matrices部分方法，或者其它库。更多的内容需要参考官方教程，此处不再赘述。

矩阵可以阅读该书线性代数基础的代码表述部分

from sympy import Matrix,init\_printing,pprint  
  
init\_printing()  
M\_a=Matrix([[1, -1], [3, 4], [0, 2]])  
pprint(M\_a)

⎡1 -1⎤  
⎢ ⎥  
⎢3 4 ⎥  
⎢ ⎥  
⎣0 2 ⎦

### 2.1.4.2 简单线性回归

在统计学中，线性回归（linear regression）是利用称为线性回归方程的最小平方函数对一个或多个自变量和因变量之间关系进行建模的一种回归分析。这种函数是一个或多个称为回归系数的模型参数的线性组合。只有一个自变量的情况称为简单线性回归（simple linear regression），大于一个自变量情况的叫多元回归（multivariable linear regression）。

* （一元）回归分析的流程:

1.数据可视化：画出自变量和因变量的散点图，并求解相关系数，确定自变量因变量之间是否可以用（一元）线性回归方程y=Ax+b解释；

2.求解回归方程中A和b的值；

3.确认回归方程的精度；

4.进行回归系数的检验；

5.计算总体回归y=Ax+b的估计；

6.进行预测。

#### 1）建立数据集并计算相关系数

使用《漫画统计学之回归分析》中最高温度（）与冰红茶销售量(杯)的数据，首先建立基于DataFrame格式的数据集，该数据集使用了时间戳（timestamp）作为索引。

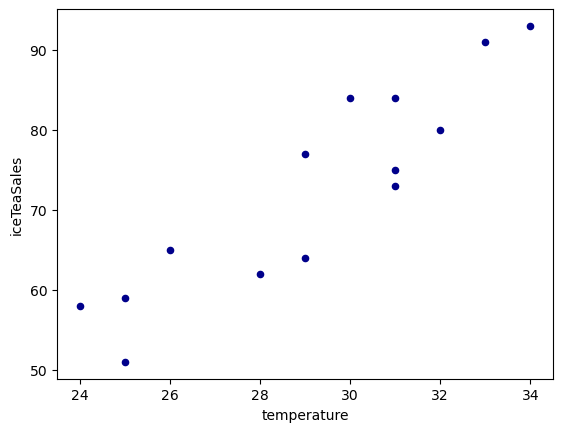
import pandas as pd  
import util\_misc  
from scipy import stats  
  
dt=pd.date\_range('2020-07-22', periods=14, freq='D')  
dt\_temperature\_iceTeaSales={"dt":dt,"temperature":[29,28,34,31,25,29,32,31,24,33,25,31,26,30],"iceTeaSales":[77,62,93,84,59,64,80,75,58,91,51,73,65,84]}  
iceTea\_df=pd.DataFrame(dt\_temperature\_iceTeaSales).set\_index("dt")  
util\_misc.print\_html(iceTea\_df,14)

|  | **temperature** | **iceTeaSales** |
| --- | --- | --- |
| **dt** |  |  |
| **2020-07-22** | 29 | 77 |
| **2020-07-23** | 28 | 62 |
| **2020-07-24** | 34 | 93 |
| **2020-07-25** | 31 | 84 |
| **2020-07-26** | 25 | 59 |
| **2020-07-27** | 29 | 64 |
| **2020-07-28** | 32 | 80 |
| **2020-07-29** | 31 | 75 |
| **2020-07-30** | 24 | 58 |
| **2020-07-31** | 33 | 91 |
| **2020-08-01** | 25 | 51 |
| **2020-08-02** | 31 | 73 |
| **2020-08-03** | 26 | 65 |
| **2020-08-04** | 30 | 84 |

计算温度与销量之间的相关系数，确定二者之间存在关联，其p-value=7.661412804450245e-06，小于0.05的显著性水平，确定pearson’s r=0.90，能够表明二者之间是强相关性，适合采用线性拟合模型求解二者之间的关联公式。

from scipy import stats  
  
r\_=stats.pearsonr(iceTea\_df["temperature"],iceTea\_df["iceTeaSales"])  
print(  
 "pearson's r:",r\_[0],  
 "\np-value:",r\_[1]  
 )  
  
iceTea\_df.plot.scatter(x="temperature",y="iceTeaSales",c="DarkBLue")  
plt.show

pearson's r: 0.9069229780508894   
p-value: 7.661412804450245e-06  
  
  
  
  
  
<function matplotlib.pyplot.show(close=None, block=None)>



#### 2）求解回归方程

求解回归方程使用了两种方法，一种是逐步计算的方式；另一种是直接使用sklearn库的LinearRegression模型。逐步计算的方式可以更深入的理解回归模型，熟悉基本计算过程之后，直接应用sklearn机器学习库中的模型也会对各种参数的配置有个比较清晰的了解。

求解回归方程即是使所有真实值与预测值之差的和为最小，求出a和b，就是所有变量残差residual的平方s\_residual的和S\_residual为最小。因为温度与销量为线性相关，因此使用一元一次方程式： y=ax+b， x为自变量温度， y为因变量销量， a和b为回归系数（参数），分别称为斜率（slop）和截距（intercept），求解a和b的过程，可以使用最小二乘法（least squares method），又称最小平方法，通过最小化误差的平方（残差平方和）寻找数据的最佳函数匹配。残差平方和为：(−34𝑎−𝑏+93)2+(−33𝑎−𝑏+91)2+(−32𝑎−𝑏+80)2+(−31𝑎−𝑏+73)2+(−31𝑎−𝑏+75)2+(−31𝑎−𝑏+84)2+(−30𝑎−𝑏+84)2+(−29𝑎−𝑏+64)2+(−29𝑎−𝑏+77)2+(−28𝑎−𝑏+62)2+(−26𝑎−𝑏+65)2+(−25𝑎−𝑏+51)2+(−25𝑎−𝑏+59)2+(−24𝑎−𝑏+58)2(−34a−b+93)2+(−33a−b+91)2+(−32a−b+80)2+(−31a−b+73)2+(−31a−b+75)2+(−31a−b+84)2+(−30a−b+84)2+(−29a−b+64)2+(−29a−b+77)2+(−28a−b+62)2+(−26a−b+65)2+(−25a−b+51)2+(−25a−b+59)2+(−24a−b+58)2，先对a和b分别求微分和，是△a，即a在横轴上的增量，及△b，即b在横轴上的增量趋近于无穷小，无限接近a和b时，因变量的变化量，这个因变量就是残差平方和的值。残差平方和的值是由a和b确定的，当a和b取不同的值时，残差平方和的值随之变化，当残差平方和的值为0时，说明由自变量温度所有值通过回归方程预测的销量，与真实值的差值之和为0；单个温度值通过回归模型预测的销量与真实值之差则趋于0。在实际计算中，手工推算时，对残差平方和关于a和b求微分，是对公式进行整理，最终获得求解回归方程回归系数的公式为：，式中Sxy​即变量SS\_xy是x和y的离差积， Sxx​即变量SS\_x是x的离差平方和。求得a后，可以根据推导公式：计算b，式中和​分别为x和y的均值，分别对应iceTea\_df.temperature.mean()和iceTea\_df.iceTeaSales.mean()。

在Python语言中，使用相关库则可以避免上述繁琐的手工推导过程，在逐步计算中，使用SymPy库约简残差平方和公式为： ， 并直接分别对a和b微分，获得结果为：和，另二者分别为0，使用SymPy库的solve求解二元一次方程组，计算获取a和b值。

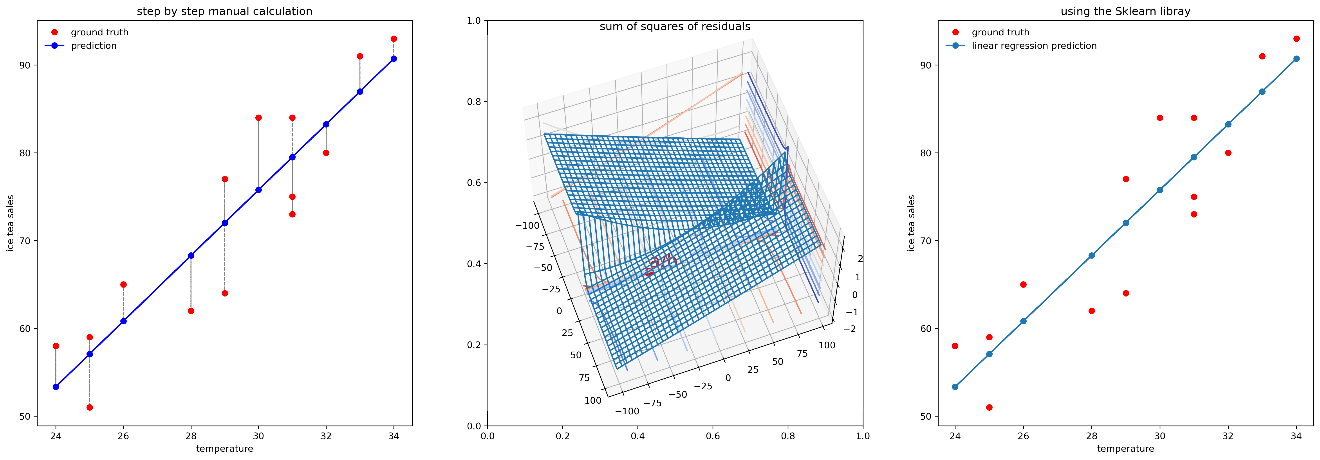
最后使用sklearn库的LinearRegression模型求解决回归模型，仅需要几行代码，所得结果与上述同。可以用sklearn返回的参数，建立回归方程公式，但是在实际的应用中并不会这么做，而是直接应用以变量形式代表的回归模型直接预测值。

import math  
import sympy  
from sympy import diff,Eq,solveset,solve,simplify,pprint  
import matplotlib.pyplot as plt  
from matplotlib import cm  
import numpy as np  
  
# 原始数据散点图  
fig, axs=plt.subplots(1,3,figsize=(25,8),dpi=300)  
axs[0].plot(iceTea\_df.temperature,iceTea\_df.iceTeaSales,'o',label='ground truth',color='r')  
axs[0].set(xlabel='temperature',ylabel='ice tea sales')  
  
  
# A - 使用‘最小二乘法’逐步计算  
# 1 - 求出x和y的离差及离差平方和  
iceTea\_df["x\_deviation"]=iceTea\_df.temperature.apply(lambda row: row-iceTea\_df.temperature.mean())  
iceTea\_df["y\_deviation"]=iceTea\_df.iceTeaSales.apply(lambda row: row-iceTea\_df.iceTeaSales.mean())  
iceTea\_df["S\_x\_deviation"]=iceTea\_df.temperature.apply(lambda row: math.pow(row-iceTea\_df.temperature.mean(),2))  
iceTea\_df["S\_y\_deviation"]=iceTea\_df.iceTeaSales.apply(lambda row: math.pow(row-iceTea\_df.iceTeaSales.mean(),2))  
SS\_x=iceTea\_df["S\_x\_deviation"].sum()  
SS\_y=iceTea\_df["S\_y\_deviation"].sum()  
  
# 2 - 求出x和y的离差积及其其和  
iceTea\_df["S\_xy\_deviation"]=iceTea\_df.apply(lambda row: (row["temperature"]-iceTea\_df.temperature.mean())\*(row["iceTeaSales"]-iceTea\_df.iceTeaSales.mean()),axis=1)  
SS\_xy=iceTea\_df["S\_xy\_deviation"].sum()  
  
# 3 - 运算过程  
a,b=sympy.symbols('a b')  
iceTea\_df["prediciton"]=iceTea\_df.temperature.apply(lambda row: a\*row+b)  
iceTea\_df["residual"]=iceTea\_df.apply(lambda row: row.iceTeaSales-(a\*row.temperature+b),axis=1)  
iceTea\_df["s\_residual"]=iceTea\_df.apply(lambda row: (row.iceTeaSales-(a\*row.temperature+b))\*\*2,axis=1)  
S\_residual=iceTea\_df["s\_residual"].sum()  
S\_residual\_simplify=simplify(S\_residual)  
print("S\_residual simplification(Binary quadratic equation):")  
pprint(S\_residual\_simplify) # 残差平方和为一个二元二次函数  
print("\_"\*50)  
  
# 打印残差平方和图形  
S\_residual\_simplif\_=sympy.lambdify([a,b],S\_residual\_simplify,"numpy")  
a\_=np.arange(-100,100,5)  
a\_3d=np.repeat(a\_[:,np.newaxis],a\_.shape[0],axis=1).T  
b\_=np.arange(-100,100,5)  
b\_3d=np.repeat(b\_[:,np.newaxis],b\_.shape[0],axis=1)  
z=S\_residual\_simplif\_(a\_3d,b\_3d)  
from sklearn import preprocessing  
z\_scaled=preprocessing.scale(z) # 标准化z值，同 from scipy.stats import zscore方法  
  
axs[1]=fig.add\_subplot(1,3,2, projection='3d')  
axs[1].plot\_wireframe(a\_3d,b\_3d,z\_scaled)  
axs[1].contour(a\_3d,b\_3d,z\_scaled, zdir='z', offset=-2, cmap=cm.coolwarm)  
axs[1].contour(a\_3d,b\_3d,z\_scaled, zdir='x', offset=-100, cmap=cm.coolwarm)  
axs[1].contour(a\_3d,b\_3d,z\_scaled, zdir='y', offset=100, cmap=cm.coolwarm)  
  
# 4 - 对残差平方和S\_residual关于a和b求微分，并使其为0  
diff\_S\_residual\_a=diff(S\_residual,a)  
diff\_S\_residual\_b=diff(S\_residual,b)  
print("diff\_S\_residual\_a=",)  
pprint(diff\_S\_residual\_a)  
print("\n")  
print("diff\_S\_residual\_b=",)  
pprint(diff\_S\_residual\_b)  
  
# 对 a 和 b 的微分公式构造二元一次方程组，即diff\_S\_residual\_a=0，diff\_S\_residual\_b=0  
Eq\_residual\_a=Eq(diff\_S\_residual\_a,0)   
Eq\_residual\_b=Eq(diff\_S\_residual\_b,0)   
# 设置二元一次方差组中的变量为a和b，并求解  
slop\_intercept=solve((Eq\_residual\_a,Eq\_residual\_b),(a,b))   
print("\_"\*50)  
print("slop and intercept:\n")  
pprint(slop\_intercept)  
slop=slop\_intercept[a]  
intercept=slop\_intercept[b]  
  
# 用求解回归方程回归系数的推导公式之间计算斜率slop和截距intercept  
print("\_"\*50)  
slop\_=SS\_xy/SS\_x  
print("derivation formula to calculate the slop=",slop\_)  
intercept\_=iceTea\_df.iceTeaSales.mean()-iceTea\_df.temperature.mean()\*slop\_  
print("derivation formula to calculate the intercept=",intercept\_)  
print("\_"\*50)  
  
# 5 - 建立简单线性回归方程  
x=sympy.Symbol('x')  
fx=slop\*x+intercept  
print("linear regression\_fx=:\n")  
pprint(fx)  
fx\_=sympy.lambdify(x,fx,"numpy")  
  
# 在残差平方和图形上标出a,b的位置  
axs[1].text(slop,intercept,-1.7,"a/b",color="red",size=20)  
axs[1].scatter(slop,intercept,-2,color="red",s=80)  
axs[1].view\_init(60,340) # 可以旋转图形的角度，方便观察  
  
# 6 - 绘制简单线性回归方程的图形  
axs[0].plot(iceTea\_df.temperature,fx\_(iceTea\_df.temperature),'o-',label='prediction',color='blue')  
  
# 绘制真实值与预测值之间的连线  
i=0  
for t in iceTea\_df.temperature:  
 axs[0].arrow(t, iceTea\_df.iceTeaSales[i], t-t, fx\_(t)-iceTea\_df.iceTeaSales[i], head\_width=0.1, head\_length=0.1,color="gray",linestyle="--" )  
 i+=1  
  
# B - 使用sklearn库sklearn.linear\_model.LinearRegression()，Ordinary least squares Linear Regression-普通最小二乘线性回归，获取回归方程  
from sklearn.linear\_model import LinearRegression  
X,y=iceTea\_df.temperature.to\_numpy().reshape(-1,1),iceTea\_df.iceTeaSales.to\_numpy()  
  
# 拟合模型  
LR=LinearRegression().fit(X,y)  
# 模型参数  
print("\_"\*50)  
print("Sklearn slop:%.2f,intercept:%.2f"%(LR.coef\_, LR.intercept\_))  
# 模型预测  
axs[2].plot(iceTea\_df.temperature,iceTea\_df.iceTeaSales,'o',label='ground truth',color='r')  
axs[2].plot(X,LR.predict(X),'o-',label='linear regression prediction')  
axs[2].set(xlabel='temperature',ylabel='ice tea sales')  
  
axs[0].legend(loc='upper left', frameon=False)  
axs[2].legend(loc='upper left', frameon=False)  
  
axs[0].set\_title('step by step manual calculation')  
axs[1].set\_title('sum of squares of residuals')  
axs[2].set\_title('using the Sklearn libray')  
plt.show()  
util\_misc.print\_html(iceTea\_df,14)

S\_residual simplification(Binary quadratic equation):  
 2 2   
12020⋅a + 816⋅a⋅b - 60188⋅a + 14⋅b - 2032⋅b + 75936  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
diff\_S\_residual\_a=  
24040⋅a + 816⋅b - 60188

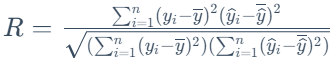
​  
diff\_S\_residual\_b= 816⋅a + 28⋅b - 2032 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ slop and intercept:

⎧ 1697 -8254 ⎫  
⎨a: ────, b: ──────⎬  
⎩ 454 227 ⎭  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
derivation formula to calculate the slop= 3.7378854625550666  
derivation formula to calculate the intercept= -36.361233480176224  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
linear regression\_fx=:  
  
1697⋅x 8254  
────── - ────  
 454 227   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Sklearn slop:3.74,intercept:-36.36



|  | **temperature** | **iceTeaSales** | **x\_deviation** | **y\_deviation** | **S\_x\_deviation** | **S\_y\_deviation** | **S\_xy\_deviation** | **prediciton** | **residual** | **s\_residual** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **dt** |  |  |  |  |  |  |  |  |  |  |
| **2020-07-22** | 29 | 77 | -0.142857 | 4.428571 | 0.020408 | 19.612245 | -0.632653 | 29\*a + b | -29\*a - b + 77 | (-29\*a - b + 77)\*\*2 |
| **2020-07-23** | 28 | 62 | -1.142857 | -10.571429 | 1.306122 | 111.755102 | 12.081633 | 28\*a + b | -28\*a - b + 62 | (-28\*a - b + 62)\*\*2 |
| **2020-07-24** | 34 | 93 | 4.857143 | 20.428571 | 23.591837 | 417.326531 | 99.224490 | 34\*a + b | -34\*a - b + 93 | (-34\*a - b + 93)\*\*2 |
| **2020-07-25** | 31 | 84 | 1.857143 | 11.428571 | 3.448980 | 130.612245 | 21.224490 | 31\*a + b | -31\*a - b + 84 | (-31\*a - b + 84)\*\*2 |
| **2020-07-26** | 25 | 59 | -4.142857 | -13.571429 | 17.163265 | 184.183673 | 56.224490 | 25\*a + b | -25\*a - b + 59 | (-25\*a - b + 59)\*\*2 |
| **2020-07-27** | 29 | 64 | -0.142857 | -8.571429 | 0.020408 | 73.469388 | 1.224490 | 29\*a + b | -29\*a - b + 64 | (-29\*a - b + 64)\*\*2 |
| **2020-07-28** | 32 | 80 | 2.857143 | 7.428571 | 8.163265 | 55.183673 | 21.224490 | 32\*a + b | -32\*a - b + 80 | (-32\*a - b + 80)\*\*2 |
| **2020-07-29** | 31 | 75 | 1.857143 | 2.428571 | 3.448980 | 5.897959 | 4.510204 | 31\*a + b | -31\*a - b + 75 | (-31\*a - b + 75)\*\*2 |
| **2020-07-30** | 24 | 58 | -5.142857 | -14.571429 | 26.448980 | 212.326531 | 74.938776 | 24\*a + b | -24\*a - b + 58 | (-24\*a - b + 58)\*\*2 |
| **2020-07-31** | 33 | 91 | 3.857143 | 18.428571 | 14.877551 | 339.612245 | 71.081633 | 33\*a + b | -33\*a - b + 91 | (-33\*a - b + 91)\*\*2 |
| **2020-08-01** | 25 | 51 | -4.142857 | -21.571429 | 17.163265 | 465.326531 | 89.367347 | 25\*a + b | -25\*a - b + 51 | (-25\*a - b + 51)\*\*2 |
| **2020-08-02** | 31 | 73 | 1.857143 | 0.428571 | 3.448980 | 0.183673 | 0.795918 | 31\*a + b | -31\*a - b + 73 | (-31\*a - b + 73)\*\*2 |
| **2020-08-03** | 26 | 65 | -3.142857 | -7.571429 | 9.877551 | 57.326531 | 23.795918 | 26\*a + b | -26\*a - b + 65 | (-26\*a - b + 65)\*\*2 |
| **2020-08-04** | 30 | 84 | 0.857143 | 11.428571 | 0.734694 | 130.612245 | 9.795918 | 30\*a + b | -30\*a - b + 84 | (-30\*a - b + 84)\*\*2 |

#### 3）确认回归方程的精度

回归方程（模型）的精度可以通过计算判断系数确认。判断系数（也称决定系数，coefficient of determination），记为或，用于度量因变量的变异中可由自变量解释部分所占的比例，以此来判断回归模型的解释力，来表示实测值（图表中的点）与回归方程拟合的程度。对于简单线性回归而言，判断系数为样本相关系数R的平方。其复（重）相关系数计算公式为：，式中y为观测值，​为观测值的均值， ​为预测值，为预测值的均值。而判定系数则为重相关系数的平方。判定系数的取值在0到1，其值越接近于1，回归方程的精度越高。第二种计算公式为：，其中SSres​为残差平方和， SStot​为观测值离差平方和（(总平方和，或总的离差平方和)），ei​为残差， yi​为观测值，​为预测值，为观测值均值。第三种是直接使用sklearn库提供的r2\_score方法直接计算。

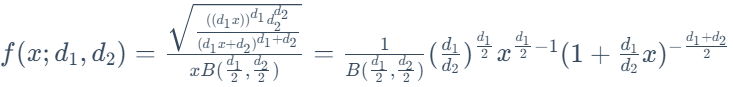
根据计算结果第1，2，3种方法结果一致。在后续的实验中，直接使用sklearn提供的方法进行计算。

def coefficient\_of\_determination(observed\_vals,predicted\_vals):   
 '''  
 function - 回归方程的决定系数  
   
 Params:  
 observed\_vals - 观测值（实测值）；list(float)  
 predicted\_vals - 预测值；list(float)  
   
 Returns:  
 R\_square\_a - 决定系数，由观测值和预测值计算获得；float  
 R\_square\_b - 决定系数，由残差平方和和总平方和计算获得；float  
 '''  
 import pandas as pd  
 import numpy as np  
 import math  
   
 vals\_df=pd.DataFrame({'obs':observed\_vals,'pre':predicted\_vals})  
 # 观测值的离差平方和(总平方和，或总的离差平方和)  
 obs\_mean=vals\_df.obs.mean()  
 SS\_tot=vals\_df.obs.apply(lambda row:(row-obs\_mean)\*\*2).sum()  
 # 预测值的离差平方和  
 pre\_mean=vals\_df.pre.mean()  
 SS\_reg=vals\_df.pre.apply(lambda row:(row-pre\_mean)\*\*2).sum()  
 # 观测值和预测值的离差积和  
 SS\_obs\_pre=vals\_df.apply(lambda row:(row.obs-obs\_mean)\*(row.pre-pre\_mean), axis=1).sum()  
   
 # 残差平方和  
 SS\_res=vals\_df.apply(lambda row:(row.obs-row.pre)\*\*2,axis=1).sum()  
   
 # 判断系数  
 R\_square\_a=(SS\_obs\_pre/math.sqrt(SS\_tot\*SS\_reg))\*\*2  
 R\_square\_b=1-SS\_res/SS\_tot  
   
 return R\_square\_a,R\_square\_b  
   
R\_square\_a,R\_square\_b=coefficient\_of\_determination(iceTea\_df.iceTeaSales.to\_list(),fx\_(iceTea\_df.temperature).to\_list())   
print("R\_square\_a=%.5f,R\_square\_b=%.5f"%(R\_square\_a,R\_square\_b))  
  
from sklearn.metrics import r2\_score  
R\_square\_=r2\_score(iceTea\_df.iceTeaSales.to\_list(),fx\_(iceTea\_df.temperature).to\_list())  
print("using sklearn libray to calculate r2\_score=",R\_square\_)

R\_square\_a=0.82251,R\_square\_b=0.82251  
using sklearn libray to calculate r2\_score= 0.8225092881166944

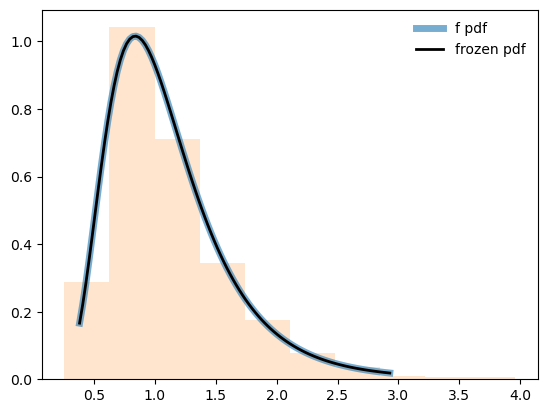
#### 4）回归系数的检验（回归显著性检验） | F分布与方差分析

F-分布（F-distribution）是两个服从卡方分布的独立随机变量各除以其自由度后的比值的抽样分布，是一种非对称分布，且位置不可互换。F-分布是一种连续概率分布，广泛应用于似然比率检验，特别是方差分析（Analysis of variance, ANOVA，或变异数分析）中。对于F-分布的阐释使用scipy.stats.f的官方案例。函数方法基本同正态分布和t分布。

如果随机变量X有参数d1​和d2​的F-分布，写作X∼F(d1​,d2​)，那么对于实数x≥0， X的概率密度函数（pdf）为：，式中B是B函数。在很多应用中，参数d1​和d2​是正整数，但对于这些参数为正实数时也有定义。一个 F-分布的随机变量是两个卡方分布变量除以自由度的比率：，式中U1​和U2​呈卡方分布，它们的自由度（degree of freedom）分别是d1​和d2​；且， U1​和U2​是相互独立的。

from scipy.stats import f  
import matplotlib.pyplot as plt  
fig, ax=plt.subplots(1, 1)  
  
dfn, dfd=29, 18 # 分别是第1自由度，即分子中卡方分布的自由度，和第2自由度，即分母中卡方分布的自由度  
mean, var, skew, kurt=f.stats(dfn, dfd, moments='mvsk')  
print("mean=%f, var=%f, skew=%f, kurt=%f"%(mean, var, skew, kurt))  
  
# 打印概率密度函数(probability density function,pdf)  
x=np.linspace(f.ppf(0.01, dfn, dfd),f.ppf(0.99, dfn, dfd), 100) #取服从自由度dfn和dfd，位于1%到99%的100个取值  
ax.plot(x, f.pdf(x, dfn, dfd),'-', lw=5, alpha=0.6, label='f pdf')  
  
# 固定分布形状，即固定自由度  
rv = f(dfn, dfd)  
ax.plot(x, rv.pdf(x), 'k-', lw=2, label='frozen pdf')  
  
vals = f.ppf([0.001, 0.5, 0.999], dfn, dfd)  
print("验证累计分布函数CDF返回值与其PPF返回值是否相等或近似：",np.allclose([0.001, 0.5, 0.999], f.cdf(vals, dfn, dfd)))  
  
# 生成服从F-分布的随机数，并打印直方图  
r=f.rvs(dfn, dfd, size=1000)  
ax.hist(r, density=True, histtype='stepfilled', alpha=0.2)  
ax.legend(loc='best', frameon=False)  
plt.show()

mean=1.125000, var=0.280557, skew=1.806568, kurt=7.074636  
验证累计分布函数CDF返回值与其PPF返回值是否相等或近似： True



·SStot​=SSreg​+SSres​，即总平方和=回归平方和+残差平方和

公式为：，式中SSreg​为回归平方和，其它同上。回归平方和，是预测值（回归值）与观测值（真实值、实测值）均值之差的平方和，该统计量反映了自变量x1​,x2​,…,xm​,的变化引起的y(yk​(k=1,2,…,n))的波动，其自由度为dfreg​=m，其中m为自变量的个数，温度与销量求解的一元一次线性方程只有一个自变量，因此其自由度为1，即只有这一个因素可以自由变化；残差平方和，是观测值与预测值之差的平方和，残差的存在是由实验误差及其它因素引起的，其自由度为dfres​=n−m−1，其中n为样本数量，即对应的y的取值数量。总的离差平方和SStot​的自由度为n−1。

观测值（样本）通常是给定的，因此总的离差平方和是固定的，构成总的离差平方和的因素为回归平方和和残差平方和，分别代表所求得的回归方程，或实验误差和其它因素引起值的变化，当残差平方和越小（就是实验误差和其它因素影响小），则回归平方和越大，则说明所求得的回归方程的预测值越准确。

自由度的再讨论（参考Wikipedia） 在统计学中，自由度（defree of freedom, df）是指当以样本的统计量估计总体的参数时，样本中独立或能自由变化的数据的个数，称为该统计量的自由度。范例：

1. 若存在两个变量x和y，如果y=x+c，其中c为常量，则其自由度为1，因为实际上只有x才能真正的自由变化， y会被x取值的不同所限制；
2. 估计总体的平均数μ时，由于样本中n个数都是相互独立的，任一个尚未抽出的数都不受已抽出任何数值的影响，所以自由度为n;
3. 估计总体的方差σ2时所使用的统计量是样本的方差s2，而s2必须用到样本平均数来计算，在抽样完成后已确定，所以大小为n的样本中只要n−1个数确定，第n个数就只有一个能使样本符合的数值。也就是说，样本中只有n−1个数可以自由变化，只要确定了这n−1个数，方差也就确定了。这里，平均数就相当于一个限制条件，由于加了这个限制条件，样本方差s2的自由度为n−1；
4. 统计模型的自由度等于可自由取值的自变量的个数。如在回归方程中，如果共有p个参数需要估计，则其中包括了p−1个自变量（与截距对应的自变量是常量），因此该回归方程的自由度为p−1。

无偏估计（unbiased estimator）在统计学中，一个总体的标准差通常是由总体中随机抽取的样本的估计，样本标准差的定义为：，其中 x1​,x2​,…,xn​为样本，样本容量为n，为样本均值。使用n−1替代n，被称为Bessel's correction（贝塞尔矫正），纠正了总体方差估计中的偏差（总体方差估计是使用随机抽取的样本的估计，不等于总体方差），及总体标准差估计中的部分偏差，但不是全部偏差。因为偏差取决于特定的分布，不可能找到对所有总体分布无偏的标准偏差的估计。

·方差分析（Analysis of variance, ANOVA，或变异数分析）

上述总平方和=回归平方和+残差平方和分析实际上是在分析因变量（总平方和，即总的离差平方和）与影响因变量变化的两个因素（或称为两个类别），即回归平方和及残差平方和的关系探索，这个过程既称之为方差分析。在求解上述回归方程之前，温度与销量的关系不一定是线性的，可能存在两种情况，一种是不管温度（x）取什么值，销量（y）都在一条水平线上下波动；二是，温度和销量存在除线性外其它类型的关系，例如非线性等。

对于上述所求得回归方程​（样本回归模型），对于总体而言， Fx​=Ax+B（总体回归模型），斜率A约为a(A∼a)，截距B约为b(B∼b)， （无偏估计量，残差平方和有n−2个自由度，这是因为两个自由度与得到预测值的估计值A和B相关），的平方根有时称为回归标准误差。(由残差平方和求得的推导过程，可以参考《线性回归分析导论》(Introduction to linear regression analysis)，简单线性回归部分)。

总体回归方程Fx​=Ax+B非常重要的特例是， H0​:A=0,H1​:A≠0，原假设意味x和y之间不存在线性关系， x对解释y的方差几乎是无用的；如果拒绝原假设，而接受备择假设，意味x对解释y的方差是有用的，可能意味线性模型是合适的，但是也可能存在需要用高阶多项式拟合的非线性模型。对于回归显著性检验可以使用t统计量，也可以使用方差分析。回归系数检验的F统计量为： ​​ ，式中SSreg​为回归平方和，自由度dfreg​=m为1， SSres​为残差平方和，其自由度dfres​=n−m−1为14−1−1=12。如果原假设成立，那么检验统计量就服从第1自由度m=1，第2自由度n−m−1=12的F分布。p-value=0.000008，小于显著性水平0.05，拒绝原假设，备择假设成立。

def ANOVA(observed\_vals,predicted\_vals,df\_reg,df\_res):  
 '''  
 function - 简单线性回归方程-回归显著性检验（回归系数检验）  
   
 Params:  
 observed\_vals - 观测值（实测值）；list(float)  
 predicted\_vals - 预测值；list(float)  
   
 Returns:  
 None  
 '''  
 import pandas as pd  
 import numpy as np  
 import math  
 from scipy.stats import f   
   
 vals\_df=pd.DataFrame({'obs':observed\_vals,'pre':predicted\_vals})  
 # 观测值的离差平方和(总平方和，或总的离差平方和)  
 obs\_mean=vals\_df.obs.mean()  
 SS\_tot=vals\_df.obs.apply(lambda row:(row-obs\_mean)\*\*2).sum()  
   
 # 残差平方和  
 SS\_res=vals\_df.apply(lambda row:(row.obs-row.pre)\*\*2,axis=1).sum()  
   
 # 回归平方和  
 SS\_reg=vals\_df.pre.apply(lambda row:(row-obs\_mean)\*\*2).sum()  
   
 print("总平方和=%.6f,回归平方和=%.6f,残差平方和=%.6f"%(SS\_tot,SS\_reg,SS\_res))  
 print("总平方和=回归平方和+残差平方和：SS\_tot=SS\_reg+SS\_res=%.6f+%.6f=%.6f"%(SS\_reg,SS\_res,SS\_reg+SS\_res))  
   
 Fz=(SS\_reg/df\_reg)/(SS\_res/df\_res)  
 print("F-分布统计量=%.6f;p-value=%.6f"%(Fz,f.sf(Fz,df\_reg,df\_res)))  
  
ANOVA(iceTea\_df.iceTeaSales.to\_list(),fx\_(iceTea\_df.temperature).to\_list(),df\_reg=1,df\_res=12)

总平方和=2203.428571,回归平方和=1812.340466,残差平方和=391.088106  
总平方和=回归平方和+残差平方和：SS\_tot=SS\_reg+SS\_res=1812.340466+391.088106=2203.428571  
F-分布统计量=55.609172;p-value=0.000008

利用F检验对回归方程进行显著性检验的方法就是方程分析，将上述过程可以归结为一个方程分析表，从而更容易缕清脉络。

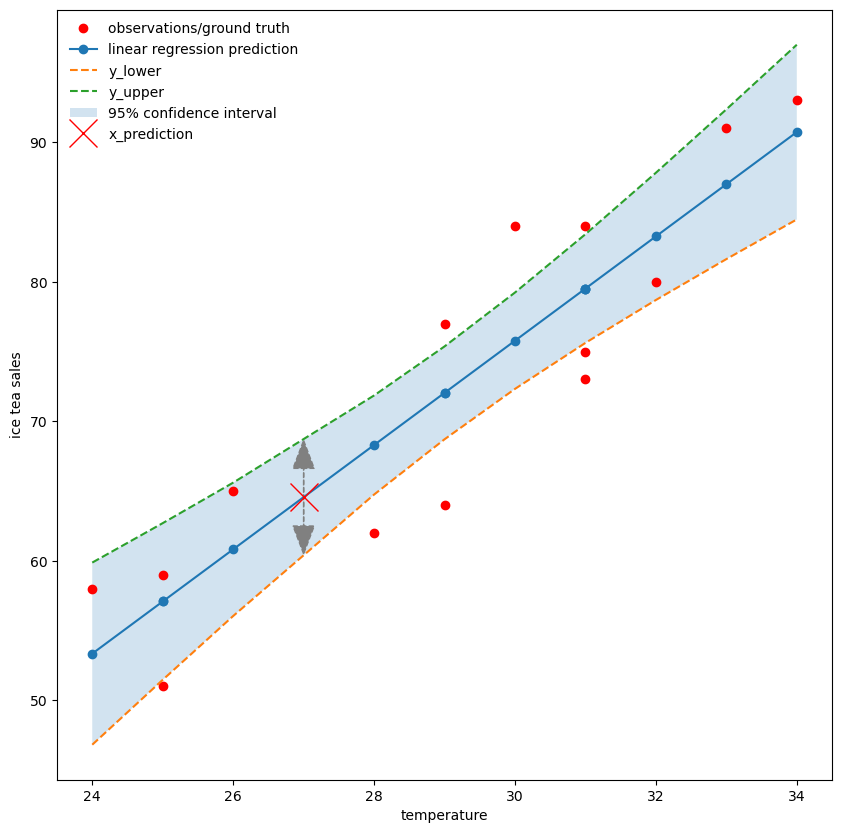
#### 

#### 5）总体回归的估计——置信区间估计

对于温度与销量的回归模型，温度为任意值时，所对应的销量不是一个固定的值，而是服从平均值为Ax+B（总体回归），标准差为σ的正态分布，因此在给定置信度（95%，99%等），总体回归Ax+B（即预测值）一定会在某个值以上，某个值以下的区间中，计算任意温度所对应销量的置信区间，是由预测值加减一个区间，该区间的计算公式为：，其中n为样本个数， m为自变量x（温度）的个数， xi​为自变量（温度）样本取值，为样本均值， Sxx​为自变量x（温度）样本的离差平方和， SSres​为残差平方和。

def confidenceInterval\_estimator\_LR(x,sample\_num,X,y,model,confidence=0.05):  
 '''  
 function - 简单线性回归置信区间估计，及预测区间  
   
 Params:  
 x - 自变量取值；float  
 sample\_num - 样本数量；int  
 X - 样本数据集-自变量；list(float)  
 y - 样本数据集-因变量；list(float)  
 model -使用sklearn获取的线性回归模型；model  
 confidence - 置信度。The default is 0.05" ；float  
   
 Returns:  
   
 CI - 置信区间；list(float)  
 '''  
 import numpy as np  
 import math  
 from scipy.stats import f  
 import matplotlib.pyplot as plt  
   
 X\_=X.reshape(-1)  
 X\_mu=X\_.mean()  
 s\_xx=(X\_-X\_mu)\*\*2  
 S\_xx=s\_xx.sum()  
 ss\_res=(y-LR.predict(X))\*\*2  
 SS\_res=ss\_res.sum()  
 probability\_val=f.ppf(q=1-confidence,dfn=1, dfd=sample\_num-2) # dfn=1, dfd=sample\_num-2  
 CI=[math.sqrt(probability\_val\*(1/sample\_num+(x-X\_mu)\*\*2/S\_xx)\*SS\_res/(sample\_num-2)) for x in X\_]  
 y\_pre=LR.predict(X)  
   
 fig, ax=plt.subplots(figsize=(10,10))  
 ax.plot(X\_,y,'o',label='observations/ground truth',color='r')  
 ax.plot(X\_,y\_pre,'o-',label='linear regression prediction')  
 ax.plot(X\_,y\_pre-CI,'--',label='y\_lower')  
 ax.plot(X\_,y\_pre+CI,'--',label='y\_upper')  
 ax.fill\_between(X\_, y\_pre-CI, y\_pre+CI, alpha=0.2,label='95% confidence interval')   
   
 # 给定值的预测区间  
 x\_ci=math.sqrt(probability\_val\*(1/sample\_num+(x-X\_mu)\*\*2/S\_xx)\*SS\_res/(sample\_num-2))  
 x\_pre=LR.predict(np.array([x]).reshape(-1,1))[0]  
 x\_lower=x\_pre-x\_ci  
 x\_upper=x\_pre+x\_ci  
 print("x prediction=%.6f;confidence interval=[%.6f,%.6f]"%(x\_pre,x\_lower,x\_upper))  
 ax.plot(x,x\_pre,'x',label='x\_prediction',color='r',markersize=20)  
 ax.arrow(x, x\_pre, 0, x\_upper-x\_pre, head\_width=0.3, head\_length=2,color="gray",linestyle="--" ,length\_includes\_head=True)  
 ax.arrow(x, x\_pre, 0, x\_lower-x\_pre, head\_width=0.3, head\_length=2,color="gray",linestyle="--" ,length\_includes\_head=True)  
   
 ax.set(xlabel='temperature',ylabel='ice tea sales')  
 ax.legend(loc='upper left', frameon=False)   
 plt.show()   
 return CI  
  
sample\_num=14  
confidence=0.05  
iceTea\_df\_sort=iceTea\_df.sort\_values(by=['temperature'])  
X,y=iceTea\_df\_sort.temperature.to\_numpy().reshape(-1,1),iceTea\_df\_sort.iceTeaSales.to\_numpy()  
CI=confidenceInterval\_estimator\_LR(27,sample\_num,X,y,LR,confidence)

x prediction=64.561674;confidence interval=[60.496215,68.627133]



#### 6）预测区间

给定特定值例如温度为31，则预测值为79.51，但是实际值不一定为该值，而是在置信度（置信水平，置信系数）为95%，对应的置信区间[66.060470,92.965962][66.060470,92.965962]内浮动，这个区间称为预测区间。

### 2.1.4.3 多元线性回归

包含多于一个回归变量的回归模型称为多元回归模型，如果为线性则为多元线性回归（multivariable linear regression）。实际问题处理上，尤其大数据，会涉及到很多自变量，例如sklearn机器学习库经典的鸢尾花（Iris）数据集包含的自变量有花萼长度、花萼宽度，花瓣长度和花瓣宽度（Sepal Length, Sepal Width, Petal Length and Petal Width），总共4个，其因变量为鸢尾花的种类，如果要根据自变量与因变量建立回归模型，则需要5个参数。

#### 1）建立数据集

在应用Python语言解析该部分内容时，仍然使用比较简单的数据集，用《漫画统计学之回归分析》中店铺的数据集，自变量包括店铺的面积(m2)、 最近的车站距离（m），因变量为月营业额（万元）。

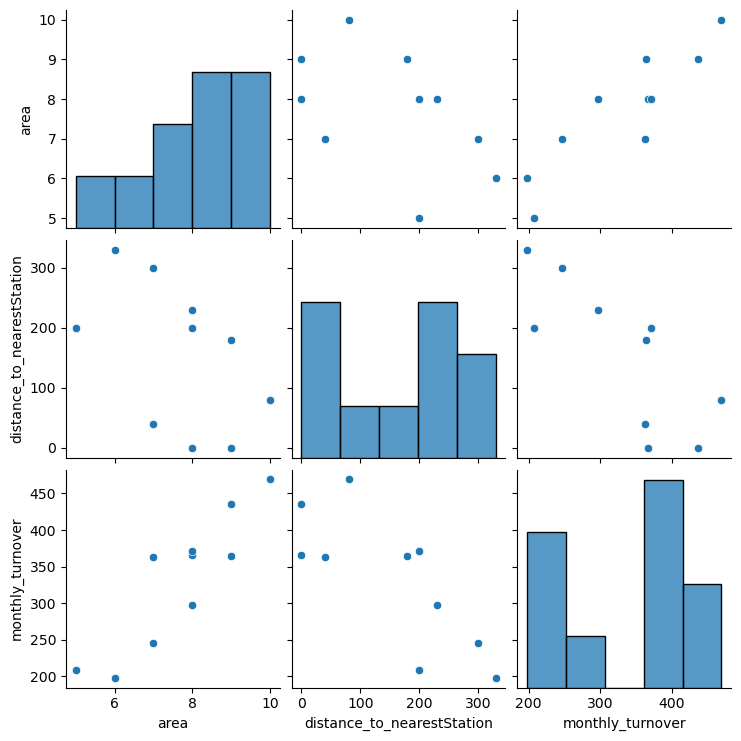
import pandas as pd  
import util\_misc  
from scipy import stats  
  
store\_info={"location":['Ill.','Ky.','Lowa.','Wis.','MIch.','Neb.','Ark.','R.I.','N.H.','N.J.'],"area":[10,8,8,5,7,8,7,9,6,9],"distance\_to\_nearestStation":[80,0,200,200,300,230,40,0,330,180],"monthly\_turnover":[469,366,371,208,246,297,363,436,198,364]}  
storeInfo\_df=pd.DataFrame(store\_info)  
util\_misc.print\_html(storeInfo\_df,10)

|  | **location** | **area** | **distance\_to\_nearestStation** | **monthly\_turnover** |
| --- | --- | --- | --- | --- |
| **0** | Ill. | 10 | 80 | 469 |
| **1** | Ky. | 8 | 0 | 366 |
| **2** | Lowa. | 8 | 200 | 371 |
| **3** | Wis. | 5 | 200 | 208 |
| **4** | MIch. | 7 | 300 | 246 |
| **5** | Neb. | 8 | 230 | 297 |
| **6** | Ark. | 7 | 40 | 363 |
| **7** | R.I. | 9 | 0 | 436 |
| **8** | N.H. | 6 | 330 | 198 |
| **9** | N.J. | 9 | 180 | 364 |

#### 2）相关性分析

为了判断依据上述数据是否具有建立多元线性回归模型的意义，同样需要进行相关性分析。因为涉及的变量增加，需要计算两两之间的相关系数，及对应的p-value，为了方便日后对此种类型数据的相关性分析，建立correlationAnalysis\_multivarialbe()函数。自变量与因变量之间的相关系数反映了自变量所能解释因变量的程度，其相关系数分别为0.8924,-0.7751，两个自变量均与因变量具有较强的相关关系，能够解释因变量，可以建立回归模型；同时，自变量之间的相关关系，可以初步判断自变量之间是否存在多重共线性，即自变量之间存在精确相关关系或高度相关关系，使得模型估计失真，或者难以估计准确。根据计算结果，两个自变量之间的相关系数为-0.4922，但是对应p-value=0.1485，即拒绝原假设，说明两个自变量之间不存在线性相关关系，因此同时使用这两个自变量解释因变量，初步判断不会使回归模型失真。

import seaborn as sns  
import matplotlib.pyplot as plt  
sns.pairplot(storeInfo\_df)  
plt.show()  
  
def correlationAnalysis\_multivarialbe(df):  
 '''  
 function - DataFrame数据格式，成组计算pearsonr相关系数  
   
 Params:  
 df - DataFrame格式数据集；DataFrame(float)  
   
 Returns:  
 p\_values - P值；DataFrame(float)  
 correlation - 相关系数；DataFame(float)  
 '''  
 from scipy.stats import pearsonr  
 import pandas as pd  
   
 df=df.dropna().\_get\_numeric\_data()  
 df\_cols=pd.DataFrame(columns=df.columns)  
 p\_values=df\_cols.transpose().join(df\_cols, how='outer')  
 correlation=df\_cols.transpose().join(df\_cols, how='outer')  
 for r in df.columns:  
 for c in df.columns:  
 p\_values[r][c]=round(pearsonr(df[r], df[c])[1], 4)  
 correlation[r][c]=round(pearsonr(df[r], df[c])[0], 4)  
   
 return p\_values,correlation  
  
p\_values,correlation=correlationAnalysis\_multivarialbe(storeInfo\_df)  
print("p\_values:")  
print(p\_values)  
print("\_"\*78)  
print("correlation:")  
print(correlation)

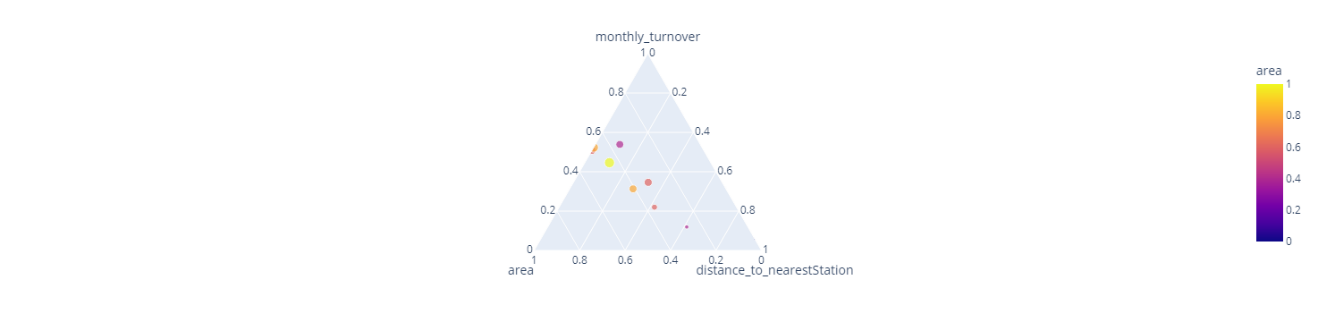


​

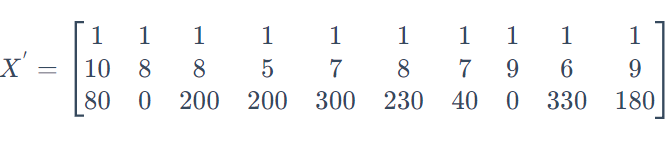
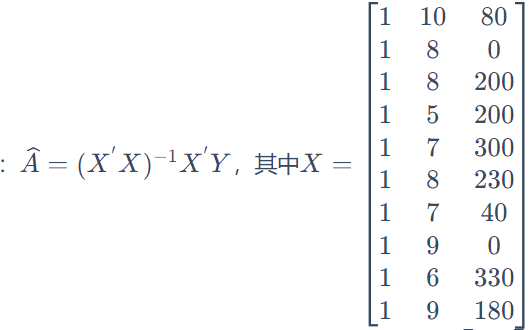
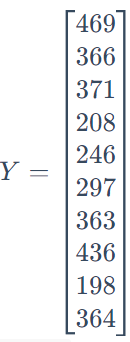
p\_values:  
 area distance\_to\_nearestStation monthly\_turnover  
area 0.0 0.1485 0.0005  
distance\_to\_nearestStation 0.1485 0.0 0.0084  
monthly\_turnover 0.0005 0.0084 0.0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
correlation:  
 area distance\_to\_nearestStation monthly\_turnover  
area 1.0 -0.4922 0.8924  
distance\_to\_nearestStation -0.4922 1.0 -0.7751  
monthly\_turnover 0.8924 -0.7751 1.0

因为总共涉及到了3个变量，可以使用Plotly库提供的三元图（Ternary Plot），查看两个自变量与一个因变量之间的分布关系。可能变量之间数值的取值范围相差较大，在三元图打印时，某些变量的值可能全部贴近图形边缘，无法清晰表述变量间的关系，因此使用： ​方法分别标准化各个变量。从图示可得店铺面积（颜色表示面积）逐渐增加，月营业额逐渐增加（点的大小表示月营业额数值大小）；而最近的车站距离逐步减小时，月营业额逐渐增加。

pd.options.mode.chained\_assignment=None  
  
columns=['area','distance\_to\_nearestStation','monthly\_turnover']  
storeInfo\_plot=storeInfo\_df[columns]  
normalize\_df=storeInfo\_plot.T.apply(lambda row:(row-row.min())/(row.max()-row.min()) , axis=1).T  
normalize\_df["location"]=storeInfo\_df.location  
  
import plotly.express as px  
fig=px.scatter\_ternary(normalize\_df,a="monthly\_turnover", b="area",c="distance\_to\_nearestStation",hover\_name="location",  
 color="area",size="monthly\_turnover", size\_max=8)   
fig.show()

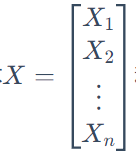
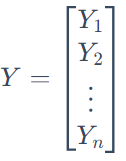
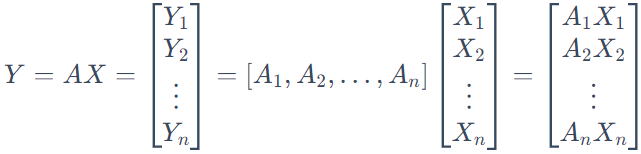


#### 3）求解多元回归方程

求解多元回归方程的方法与简单线性回归求解方式相类似，使用最小二乘法对偏回归系数进行求解。求解过程中，使用了三种方法，一是，使用SymPy分别对残差平方和SSr​es的a1、a2和b求微分，当各自微分的值等于0时，所反映的残差平方和为0，即观测值和预测值差值的平方和为0，而单个观测值与对应的预测值之间的差值趋于0；二是，使用矩阵计算的方式求解参数，其计算公式为  
即X的的转置，也可记作,等，。对于一个矩阵X，其逆矩阵（inverse matrix）为。使用矩阵的计算方法时，仍然是使用SymPy库，该库提供了建立矩阵和矩阵计算的功能。最后一种求解多元线性回归方程的方式是直接使用sklearn.linear\_model.LinearRegression计算，并获得回归模型。

偏回归系数（partial regression coefficient），是多元回归问题出现的一个特殊性质。设自变量x1​,x2​,…,xm​,，与因变量y具有线性关系，有y=a1​x1​+a2​x2​+…+an​xn​+b，则a1​,a2​,…,an​为相对于各自变量的偏回归系数，表示当其他的各自变量都保持一定时，指定的某一自变量每变动一个单位，因变量y增加或减少的数值。

* 对矩阵表示的求解参数公式再理解：

多元线性回归模型公式： y=a1​x1​+a2​x2​+…+an​xn​+b， b首先可以被看做每一个样本中固定的一个自变量x0​，再引入一个可以学习的a0​，即y=a0​x0​+a1​x1​+a2​x2​+…+an​xn​，为进一步方便表示和学习，设公式中的x0​为不需要学习的值，即x0​=1，此时a0​值的意义则代表原公式中的偏置b，因此多元线性回归模型公式可简单表示为： Y=AX，对于所有样本​​​和，其矩阵的表示方式为： ​​​。因为矩阵不能相除，因此不能直接对Y=AX两边同时除以X，以求取A，但是可以两边同时乘以X的逆矩阵X′避免除法（矩阵乘以自身的逆矩阵结果为1）。同时只有方阵才可能可逆，而样本的数量是无法控制的，因此用X乘以其转置产生一个可以求逆的方阵X′X。完整推导过程如下：

*Y*=*AX*

首先右乘*X*‘，得到*YX*‘=*A*^*XX*‘

然后右乘 (*XX*‘)−1，得到*YX*‘(*XX*‘)−1=*A*^*XX*‘(*XX*‘)−1

因为*XX*‘(*XX*‘)−1=1，所以化简如下：

*YX*‘(*XX*‘)−1=**

即=(*X*‘*X*)−1*X*‘*Y*

import sympy,math  
from sympy import diff,Eq,solveset,solve,simplify,pprint,Matrix  
  
a1,a2,b=sympy.symbols('a1 a2 b')  
# 计算残差平方和  
storeInfo\_df["ss\_res"]=storeInfo\_df.apply(lambda row:(row.monthly\_turnover-(row.area\*a1+row.distance\_to\_nearestStation\*a2+b))\*\*2,axis=1)  
util\_misc.print\_html(storeInfo\_df,10)  
SS\_res=storeInfo\_df["ss\_res"].sum()  
  
# A- 使用sympy求解多元回归方程  
# 对残差平方和SS\_res关于a1，a1和b求微分，并使微分值为0  
diff\_SSres\_a1=diff(SS\_res,a1)  
diff\_SSres\_a2=diff(SS\_res,a2)  
diff\_SSres\_b=diff(SS\_res,b)  
  
# 当微分值为0时，解方程组，获得a1，a2和b的值  
Eq\_residual\_a1=Eq(diff\_SSres\_a1,0) # 设所求a1微分为0  
Eq\_residual\_a2=Eq(diff\_SSres\_a2,0) # 设所求a2微分为0  
Eq\_residual\_b=Eq(diff\_SSres\_b,0) # 设所求a2微分为0  
slop\_intercept=solve((Eq\_residual\_a1,Eq\_residual\_a2,Eq\_residual\_b),(a1,a2,b)) # 计算三元一次方程组  
print("diff\_a1,a2 and b:\n")  
pprint(slop\_intercept)  
print("\_"\*50)  
  
# B - 使用矩阵（基于sympy）求解多元回归方程  
if 'one' not in storeInfo\_df.columns:  
 X\_m=Matrix(storeInfo\_df.insert(loc=1,column='one',value=1)[['one','area','distance\_to\_nearestStation']])  
else:  
 X\_m=Matrix(storeInfo\_df[['one','area','distance\_to\_nearestStation']])  
y\_m=Matrix(storeInfo\_df.monthly\_turnover)  
  
parameters\_reg=(X\_m.T\*X\_m)\*\*-1\*X\_m.T\*y\_m # 注意在矩阵计算时，矩阵相乘不能任意变化位置  
print("matrix\_a1,a2 and b:\n")  
pprint(parameters\_reg)  
  
# C - 使用sklearn求解多元回归方程  
# B - 使用sklearn库sklearn.linear\_model.LinearRegression()，Ordinary least squares Linear Regression-普通最小二乘线性回归，获取回归方程  
from sklearn.linear\_model import LinearRegression  
X=storeInfo\_df[['area','distance\_to\_nearestStation']].to\_numpy()  
y=storeInfo\_df['monthly\_turnover'].to\_numpy()  
  
# 拟合模型  
LR\_multivariate=LinearRegression().fit(X,y)  
# 模型参数  
print("\_"\*50)  
print("Sklearn a1=%.2f,a2=%.2f,b=%.2f"%(LR\_multivariate.coef\_[0],LR\_multivariate.coef\_[1], LR\_multivariate.intercept\_))  
  
# 建立回归方程  
x1,x2=sympy.symbols('x1,x2')  
fx\_m=slop\_intercept[a1]\*x1+slop\_intercept[a2]\*x2+slop\_intercept[b]  
print("linear regression\_fx=:\n")  
pprint(fx\_m)  
fx\_m=sympy.lambdify([x1,x2],fx\_m,"numpy")

diff\_a1,a2 and b:  
  
⎧ 4073344 -44597 6409648⎫  
⎨a₁: ───────, a₂: ───────, b: ───────⎬  
⎩ 98121 130828 98121 ⎭  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
matrix\_a1,a2 and b:  
  
⎡6409648⎤  
⎢───────⎥  
⎢ 98121 ⎥  
⎢ ⎥  
⎢4073344⎥  
⎢───────⎥  
⎢ 98121 ⎥  
⎢ ⎥  
⎢-44597 ⎥  
⎢───────⎥  
⎣ 130828⎦  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Sklearn a1=41.51,a2=-0.34,b=65.32  
linear regression\_fx=:  
  
4073344⋅x₁ 44597⋅x₂ 6409648  
────────── - ──────── + ───────  
 98121 130828 98121

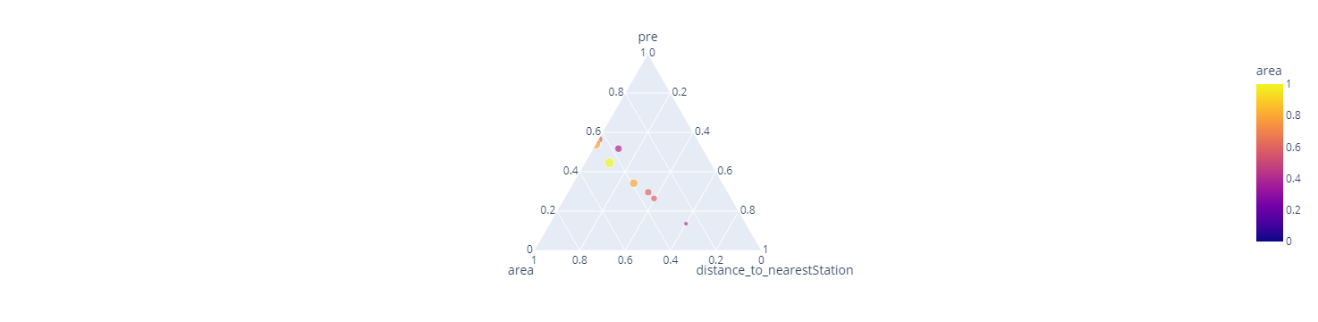
可以将矩阵打印为Latex格式的数学表达式，方便在markdown中表述，避免自行输入。

from sympy import latex  
print(latex(X\_m.T))

\left[\begin{matrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\10 & 8 & 8 & 5 & 7 & 8 & 7 & 9 & 6 & 9\\80 & 0 & 200 & 200 & 300 & 230 & 40 & 0 & 330 & 180\end{matrix}\right]

同样使用三元图打印两个自变量，及预测值之间的图表，观察变量之间的关系。

pd.options.mode.chained\_assignment = None  
storeInfo\_df['pre']=LR\_multivariate.predict(X)  
columns=['area','distance\_to\_nearestStation','monthly\_turnover','pre']  
storeInfo\_plot=storeInfo\_df[columns]  
normalize\_df=storeInfo\_plot.T.apply(lambda row:(row-row.min())/(row.max()-row.min()) , axis=1).T  
normalize\_df["location"]=storeInfo\_df.location  
  
import plotly.express as px  
fig = px.scatter\_ternary(normalize\_df, a="pre", b="area",c="distance\_to\_nearestStation",hover\_name="location",  
 color="area",size="pre", size\_max=8,  
 )   
  
fig.show()



#### 4）确认多元回归方程的精度

非修正自由度的判定系数计算同简单线性回归，将定义的计算函数coefficient\_of\_determination放置于util\_A.py文件（模块）中，直接调用计算，同时也使用sklearn提供的r2\_score计算，其计算结果约为0.94，表示实测值与回归方程的预测值拟合程度的指标较高，能够比较好的根据店铺面积和最近车站距离预测月营业额。

# 计算复相关系数R  
import util\_A  
R\_square\_a,R\_square\_b=util\_A.coefficient\_of\_determination(storeInfo\_df.monthly\_turnover.to\_list(),storeInfo\_df.pre.to\_list())   
print("R\_square\_a=%.5f,R\_square\_b=%.5f"%(R\_square\_a,R\_square\_b))  
  
from sklearn.metrics import r2\_score  
R\_square\_=r2\_score(storeInfo\_df.monthly\_turnover.to\_list(),storeInfo\_df.pre.to\_list())  
print("using sklearn libray to calculate r2\_score=",R\_square\_)

R\_square\_a=0.94524,R\_square\_b=0.94524  
using sklearn libray to calculate r2\_score= 0.945235852681711

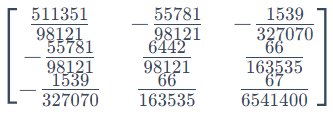
* 修正自由度的判定系数

直接使用判定系数时，其自变量的数量越多，判定系数的值越高，但是并不是每一个自变量都是有效的，因此通常使用修正自由度的判定系数，其公式为：，其中ns​为样本个数， nv​为自变量个数， SSres​为残差平方和， SStot​为总的离差平方和。

def coefficient\_of\_determination\_correction(observed\_vals,predicted\_vals,independent\_variable\_n):  
 '''  
 function - 回归方程修正自由度的判定系数  
   
 Params:  
 observed\_vals - 观测值（实测值）；list(float)  
 predicted\_vals - 预测值；list(float)  
 independent\_variable\_n - 自变量个数；int  
   
 Returns:  
 R\_square\_correction - 正自由度的判定系数；float  
 '''  
 import pandas as pd  
 import numpy as np  
 import math  
   
 vals\_df=pd.DataFrame({'obs':observed\_vals,'pre':predicted\_vals})  
 # 观测值的离差平方和(总平方和，或总的离差平方和)  
 obs\_mean=vals\_df.obs.mean()  
 SS\_tot=vals\_df.obs.apply(lambda row:(row-obs\_mean)\*\*2).sum()  
   
 # 残差平方和  
 SS\_res=vals\_df.apply(lambda row:(row.obs-row.pre)\*\*2,axis=1).sum()  
   
 # 判断系数  
 sample\_n=len(observed\_vals)  
 R\_square\_correction=1-(SS\_res/(sample\_n-independent\_variable\_n-1))/(SS\_tot/(sample\_n-1))  
   
 return R\_square\_correction  
  
R\_square\_correction=coefficient\_of\_determination\_correction(storeInfo\_df.monthly\_turnover.to\_list(),storeInfo\_df.pre.to\_list(),2)  
print("修正自由度的判定系数=",R\_square\_correction)

修正自由度的判定系数= 0.929588953447914

#### 5） 回归显著性检验

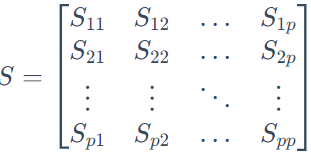
在简单回归模型中的回归系数检验，只需要给定H0​:A=0,H1​:A≠0。但是在多元回归中，就总体而言0Fx​=A1​x1​+A2​x2​+B，其中A1​∼a1​,A2​∼a2​,B∼b，∼为约为。包括A1​和 A2​两个偏相关系数，因此可以分为两种情况，一种是全面讨论偏回归系数的检验，原假设： A1​=A2​=0，备择假设: A1​=A2​=0不成立，即以下任意一组关系成立， A1​≠0且A2​≠0， A1​≠0且A2​=0，或A1​=0且A2​≠0。另一种是分别讨论偏回归系数的检验，例如原假设： A1​=0，备择假设： A1​≠0。在这两种方式中，检验统计量是不同的，对于全面检验，其统计量为：，式中SStot​为总平方和SSres​为残差平方和， ns​为样本个数， nv​为自变量个数;对于单个回归系数的检验，其统计量为：，式中Cjj​为 (X′X)−1对角线相交位置的值，即 (X′X)−1=​​​​，对角线的值为。

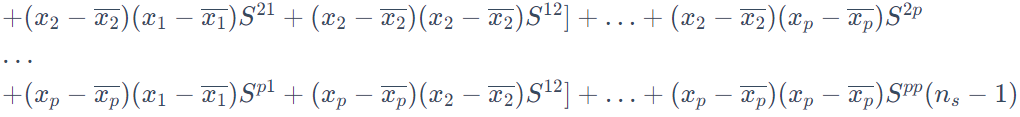
对于全部回归系数的总体检验，及单个回归系数的检验，其结果p-value均小于0.05，意味着所求得的多元线性回归模型是合适的。

def ANOVA\_multivarialbe(observed\_vals,predicted\_vals,independent\_variable\_n,a\_i,X):  
 '''  
 function - 多元线性回归方程-回归显著性检验（回归系数检验），全部回归系数的总体检验，及单个回归系数的检验  
   
 Paras:   
 observed\_vals - 观测值（实测值）；list(float)  
 predicted\_vals - 预测值；list(list)  
 independent\_variable\_n - 自变量个数；int  
 a\_i - 偏相关系数列表；list(float)  
 X - 样本数据集\_自变量；array(numpy)  
   
 Returns:  
 None  
 '''  
 import pandas as pd  
 import numpy as np  
 import math  
 from scipy.stats import f  
 from sympy import Matrix,pprint  
   
 vals\_df=pd.DataFrame({'obs':observed\_vals,'pre':predicted\_vals})  
 # 总平方和，或总的离差平方和  
 obs\_mean=vals\_df.obs.mean()  
 SS\_tot=vals\_df.obs.apply(lambda row:(row-obs\_mean)\*\*2).sum()  
   
 # 残差平方和  
 SS\_res=vals\_df.apply(lambda row:(row.obs-row.pre)\*\*2,axis=1).sum()  
   
 # 回归平方和  
 SS\_reg=vals\_df.pre.apply(lambda row:(row-obs\_mean)\*\*2).sum()  
   
 # 样本个数  
 n\_s=len(observed\_vals)  
 dfn=independent\_variable\_n  
 dfd=n\_s-independent\_variable\_n-1  
   
 # 计算全部回归系数的总体检验统计量  
 F\_total=((SS\_tot-SS\_res)/dfn)/(SS\_res/dfd)  
 print("F-分布统计量\_total=%.6f;p-value=%.6f"%(F\_total,f.sf(F\_total,dfn,dfd)))  
   
 # 逐个计算单个回归系数的检验统计量  
 X=np.insert(X,0,1,1)  
 X\_m=Matrix(X)  
 M\_inverse=(X\_m.T\*X\_m)\*\*-1  
 C\_jj=M\_inverse.row(1).col(1)[0]  
 pprint(C\_jj)  
   
 F\_ai\_list=[]  
 i=0  
 for a in a\_i:  
 F\_ai=(a\*\*2/C\_jj)/(SS\_res/dfd)  
 F\_ai\_list.append(F\_ai)  
 print("a%d=%.6f时，F-分布统计量\_=%.6f;p-value=%.6f"%(i,a,F\_ai,f.sf(F\_total,1,dfd)))  
 i+=1  
   
a1\_,a2\_=LR\_multivariate.coef\_[0],LR\_multivariate.coef\_[1]  
X=storeInfo\_df[['area','distance\_to\_nearestStation']].to\_numpy()  
ANOVA\_multivarialbe(storeInfo\_df.monthly\_turnover.to\_list(),storeInfo\_df.pre.to\_list(),2,a\_i=[a1\_,a2\_],X=X)

F-分布统计量\_total=60.410426;p-value=0.000038  
 6442  
─────  
98121  
a0=41.513478时，F-分布统计量\_=44.032010;p-value=0.000110  
a1=-0.340883时，F-分布统计量\_=0.002969;p-value=0.000110

#### 6）[总体回归A1​X1​+A2​X2​+…+An​Xn​+B的估计——置信区间](https://richiebao.github.io/USDA_CH_endup/#/./markdown/2_1_4_%E7%AE%80%E5%8D%95%E5%9B%9E%E5%BD%92%E4%B8%8E%E5%A4%9A%E5%85%83%E5%9B%9E%E5%BD%92?id=_6%ef%bc%89%e6%80%bb%e4%bd%93%e5%9b%9e%e5%bd%92a1x1a2x2anxnba_1-x_1-a_2-x_2-ldots-a_n-x_nba1x1a2x2anxnb%e7%9a%84%e4%bc%b0%e8%ae%a1%e7%bd%ae%e4%bf%a1%e5%8c%ba%e9%97%b4)

多元线性回归模型的预测值置信区间估计使用了两种计算方式，一是，自定义函数逐步计算，其计算公式为：，其中ns​为样本个数， nv​为自变量个位数， D²为马氏距离（Mahalanobis distance）的平方， SSres​为残差平方和；D²马氏距离的平方计算公式为：先求的逆矩阵，式中， S22​代表第2个自变量的离差平方和， S25​代表第2个自变量和第5个自变量的离差积和， S25​与S52​是相等的，以此类推；然后根据，求取马氏距离的平方公式为：

，其中ns​为样本个数。

二是，使用[statsmodels](https://www.statsmodels.org/stable/index.html)的statsmodels.regression.linear\_model.OLS普通最小二乘法（Ordinary Least Squares，OLS）求得多元线性回归方程，其语法结构与sklearn基本相同。所求的的回归模型包含有置信区间的属性，可以通过dt=res.get\_prediction(X).summary\_frame(alpha=0.05)的方式提取。可以打印statsmodels计算所得回归模型的概要（summary），比较求解回归方程的偏回归系数和截距（coef\_const/area/distance\_to\_nearestStation ），及确认多元回归方程的精度R-squared（R²）和修正自由度的判定系数Adj. R-squared，和回归显著性检验全面讨论偏回归系数的检验F-分布统计量F-statistic，对应p-value/Prob (F-statistic)，全部相等，互相印证了所使用的方法是否保持一致。

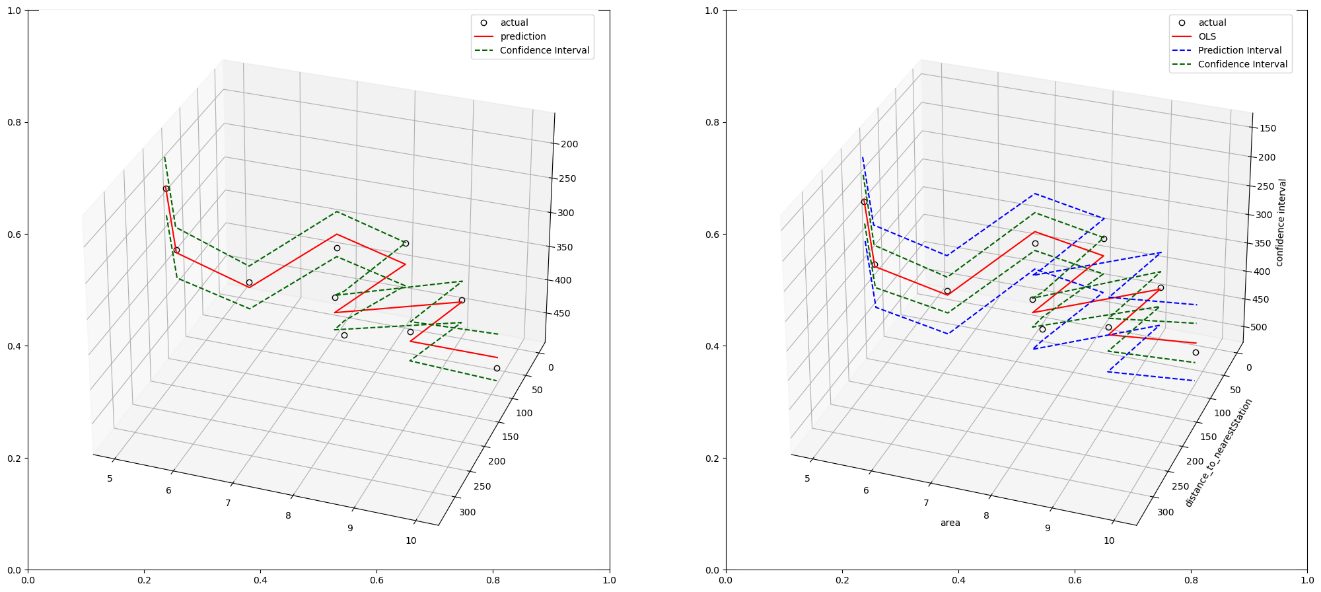
对于两种方法在预测变量置信区间比较上，分别打印了各自的三维分布图，其结果显示二者的图形保持一致，即通过statsmodels求解多元回归方程与逐步计算所得结果保持一致。

* 马氏距离（Mahalanobis distance）

[statsmodels](https://www.statsmodels.org/stable/index.html) 提供了一些类和函数，用于估计许多不同的统计模型，及执行统计测试和统计数据研究。每个估计器都有一个广泛的结果统计信息列表，可以用以查看相关信息，以确保所求得的估计器（模型）的准确性、正确性。

import numpy as np  
import statsmodels.api as sm  
import warnings  
warnings.filterwarnings("ignore")  
  
# 使用statsmodels库求解回归方程，与获得预测值的置信区间  
storeInfo\_df\_sort=storeInfo\_df.sort\_values(by=['area'])  
X=storeInfo\_df\_sort[['area','distance\_to\_nearestStation']]  
X=sm.add\_constant(X) # 因为在上述逐步计算或者使用Sklearn求解回归方程过程中，多元回归方程均增加了常量截距的参数，因此此处增加一个常量 adding a constant  
y=storeInfo\_df\_sort['monthly\_turnover']  
mod=sm.OLS(y,X) # 构建最小二乘模型 Describe model  
res=mod.fit() # 拟合模型 Fit model  
print(res.summary()) # Summarize model  
  
dt=res.get\_prediction(X).summary\_frame(alpha=0.05)  
y\_prd = dt['mean']  
yprd\_ci\_lower = dt['obs\_ci\_lower']  
yprd\_ci\_upper = dt['obs\_ci\_upper']  
ym\_ci\_lower = dt['mean\_ci\_lower']   
ym\_ci\_upper = dt['mean\_ci\_upper']  
  
# 逐步计算  
def confidenceInterval\_estimator\_LR\_multivariable(X,y,model,confidence=0.05):  
 '''  
 function - 多元线性回归置信区间估计，及预测区间  
   
 Params:  
 X - 样本自变量；DataFrame数据格式  
 y - 样本因变量；list(float)  
 model - 多元回归模型；model  
 confidence - 置信度，The default is 0.05；float  
   
 return:  
 CI- 预测值的置信区间；list(float)  
 '''  
 import pandas as pd  
 from sympy import Matrix,pprint  
 import numpy as np  
   
 # 根据指定数目，划分列表的函数  
 def chunks(lst, n):  
 for i in range(0, len(lst), n):  
 yield lst[i:i + n]  
   
 X\_deepCopy=X.copy(deep=True) # 不进行深度拷贝，如果传入的参数变量X发生了改变，则该函数外部的变量值也会发生改变  
 columns=X\_deepCopy.columns  
 n\_v=len(columns)  
 n\_s=len(y)  
   
 # 求S，用于马氏距离的计算  
 SD=[]  
 SD\_name=[]  
 for col\_i in columns:  
 i=0  
 for col\_j in columns:  
 SD\_column\_name=col\_i+'S'+str(i)  
 SD\_name.append(SD\_column\_name)  
 if col\_i==col\_j:  
 X\_deepCopy[SD\_column\_name]=X\_deepCopy.apply(lambda row: (row[col\_i]-X\_deepCopy[col\_j].mean())\*\*2,axis=1)  
 SD.append(X\_deepCopy[SD\_column\_name].sum())  
 else:  
 X\_deepCopy[SD\_column\_name]=X\_deepCopy.apply(lambda row: (row[col\_i]-X\_deepCopy[col\_i].mean())\*(row[col\_j]-X\_deepCopy[col\_j].mean()),axis=1)  
 SD.append(X\_deepCopy[SD\_column\_name].sum())   
 i+=1  
 M=Matrix(list(chunks(SD,n\_v)))  
   
 # 求S的逆矩阵  
 M\_invert=M\*\*-1  
 # pprint(M\_invert)  
 M\_invert\_list=[M\_invert.row(row).col(col)[0] for row in range(n\_v) for col in range(n\_v)]  
 X\_mu=[X\_deepCopy[col].mean() for col in columns]  
   
 # 求马氏距离的平方  
 SD\_array=X\_deepCopy[SD\_name].to\_numpy()   
 D\_square\_list=[sum([x\*y for x,y in zip(SD\_selection,M\_invert\_list)])\*(n\_s-1) for SD\_selection in SD\_array]   
   
 # 计算CI-预测值的置信区间  
 print(columns)  
 ss\_res=(y-model.predict(X\_deepCopy[columns].to\_numpy()))\*\*2  
 SS\_res=ss\_res.sum()  
 print(SS\_res)  
 probability\_val=f.ppf(q=1-confidence,dfn=1, dfd=n\_s-n\_v-1)   
 CI=[math.sqrt(probability\_val\*(1/n\_s+D\_square/(n\_s-1))\*SS\_res/(n\_s-n\_v-1)) for D\_square in D\_square\_list]  
  
 return CI  
  
X\_=storeInfo\_df\_sort[['area','distance\_to\_nearestStation']]  
y\_=storeInfo\_df\_sort['monthly\_turnover']  
CI=confidenceInterval\_estimator\_LR\_multivariable(X\_,y\_,LR\_multivariate,confidence=0.05)  
  
# 打印图表  
fig, axs=plt.subplots(1,2,figsize=(25,11))  
x\_=X.area  
y\_=X.distance\_to\_nearestStation  
  
# 由自定义函数，逐步计算获得的置信区间  
axs[0]=fig.add\_subplot(1,2,1, projection='3d')  
axs[0].plot(x\_,y\_, y, linestyle = "None", marker = "o",markerfacecolor = "None", color = "black",label = "actual")  
X\_array=X\_.to\_numpy()  
LR\_pre=LR\_multivariate.predict(X\_array)  
axs[0].plot(x\_, y\_,LR\_pre, color = "red",label = "prediction")  
axs[0].plot(x\_,y\_, LR\_pre+CI, color = "darkgreen", linestyle = "--", label = "Confidence Interval")  
axs[0].plot(x\_,y\_, LR\_pre-CI, color = "darkgreen", linestyle = "--")  
  
# 由statsmodels库计算所得的置信区间  
axs[1]=fig.add\_subplot(1,2,2, projection='3d')  
axs[1].plot(x\_,y\_, y, linestyle = "None", marker = "o",markerfacecolor = "None", color = "black",label = "actual")  
axs[1].plot(x\_, y\_,y\_prd, color = "red",label = "OLS")  
axs[1].plot(x\_, y\_,yprd\_ci\_lower, color = "blue", linestyle = "--",label = "Prediction Interval")  
axs[1].plot(x\_, y\_,yprd\_ci\_upper, color = "blue", linestyle = "--")  
  
axs[1].plot(x\_,y\_, ym\_ci\_lower, color = "darkgreen", linestyle = "--", label = "Confidence Interval")  
axs[1].plot(x\_,y\_, ym\_ci\_upper, color = "darkgreen", linestyle = "--")  
  
axs[1].view\_init(210,250) # 可以旋转图形的角度，方便观察  
axs[1].set\_xlabel('area')  
axs[1].set\_ylabel('distance\_to\_nearestStation')  
axs[1].set\_zlabel('confidence interval')  
  
axs[0].legend()  
axs[1].legend()  
axs[0].view\_init(210,250) # 可以旋转图形的角度，方便观察  
axs[1].view\_init(210,250) # 可以旋转图形的角度，方便观察  
plt.show()

OLS Regression Results   
==============================================================================  
Dep. Variable: monthly\_turnover R-squared: 0.945  
Model: OLS Adj. R-squared: 0.930  
Method: Least Squares F-statistic: 60.41  
Date: Mon, 24 Oct 2022 Prob (F-statistic): 3.84e-05  
Time: 13:22:21 Log-Likelihood: -44.358  
No. Observations: 10 AIC: 94.72  
Df Residuals: 7 BIC: 95.62  
Df Model: 2   
Covariance Type: nonrobust   
==============================================================================================  
 coef std err t P>|t| [0.025 0.975]  
----------------------------------------------------------------------------------------------  
const 65.3239 55.738 1.172 0.280 -66.476 197.124  
area 41.5135 6.256 6.636 0.000 26.720 56.307  
distance\_to\_nearestStation -0.3409 0.078 -4.362 0.003 -0.526 -0.156  
==============================================================================  
Omnibus: 0.883 Durbin-Watson: 3.440  
Prob(Omnibus): 0.643 Jarque-Bera (JB): 0.448  
Skew: 0.479 Prob(JB): 0.799  
Kurtosis: 2.603 Cond. No. 1.40e+03  
==============================================================================  
  
Notes:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.  
[2] The condition number is large, 1.4e+03. This might indicate that there are  
strong multicollinearity or other numerical problems.  
Index(['area', 'distance\_to\_nearestStation'], dtype='object')  
4173.006119994701



注释（Notes）：

① scikit-learn，也称为sklearn，针对Python编程语言的免费软件机器学习库（<https://scikit-learn.org/stable/index.html>）。

② statsmodels，为许多不同统计模型的估计提供了Python类和函数，进行统计测试和统计数据探索。每个估计器都返回一个包含广泛结果的统计列表，这些结果与现有的统计包进行了测试，确保其正确性（<https://www.statsmodels.org/stable/index.html>）。

③ SymPy, 用于符号数学（symbolic mathematics）的Python库（<https://docs.sympy.org/latest/index.html>）。

④ Symbolab，为一个在线数学教学工具，允许用户使用数学符号和科学符号及文本来学习、练习和发现数学知识，例如代数、三角和微积分。同时提供了丰富的智能计算器，包括：方程（equations）、同步方程（simultaneous equations）、不等式（inequalities）、积分（integrals）、导数（derivatives）、极限（limits）、切线（tangent line）、三角方程（trigonometric equations）、函数（functions）等（<https://www.symbolab.com/>）。

参考文献（References）:

[1] [日]高桥 信著作,Inoue Iroha,株式会社 TREND-PRO漫画制作,张仲恒译.漫画统计学之回归分析[M].科学出版社.北京.2009.08.

[2] Timothy C.Urdan(蒂莫西•C•厄丹),彭志文译.白话统计学[M].中国人民大学出版社.2013,12.第3版. <Timothy C. Urdan.Statistics in plain English[M]. New York: Routledge, 2010>.

[3] Douglas C.Montgomery,Elizabeth A.Peck,G.Geoffrey Vining著.王辰勇译.线性回归分析导论[M].机械工业出版社.2016.04(第5版). <Douglas C.Montgomery,Elizabeth A.Peck,G.Geoffrey Vining. Introduction to linear regression analysis[M]. John Wiley & Sons, 2013.>