

# Python之建模图论篇





图-  
Dijkstra

图-Floyd

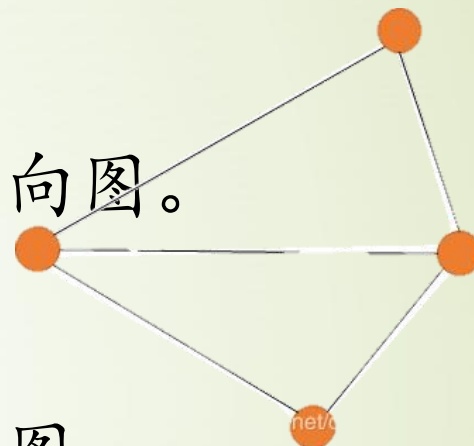
机场航线  
设计

# 1 图论模型-Dijkstra

Dijkstra算法能求一个顶点到另一顶点最短路径。

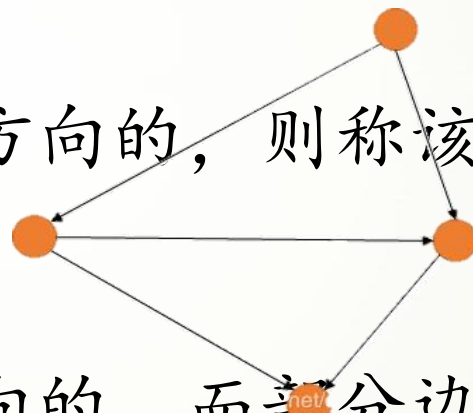
✓ 无向图:

若图中的每条边都是没有方向的, 则称该图为无向图。



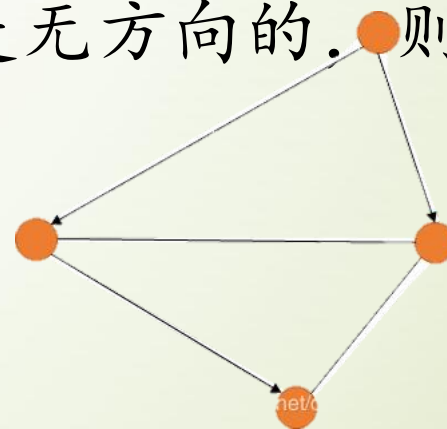
✓ 有向图:

若图中的每条边都是有方向的, 则称该图为有向图。



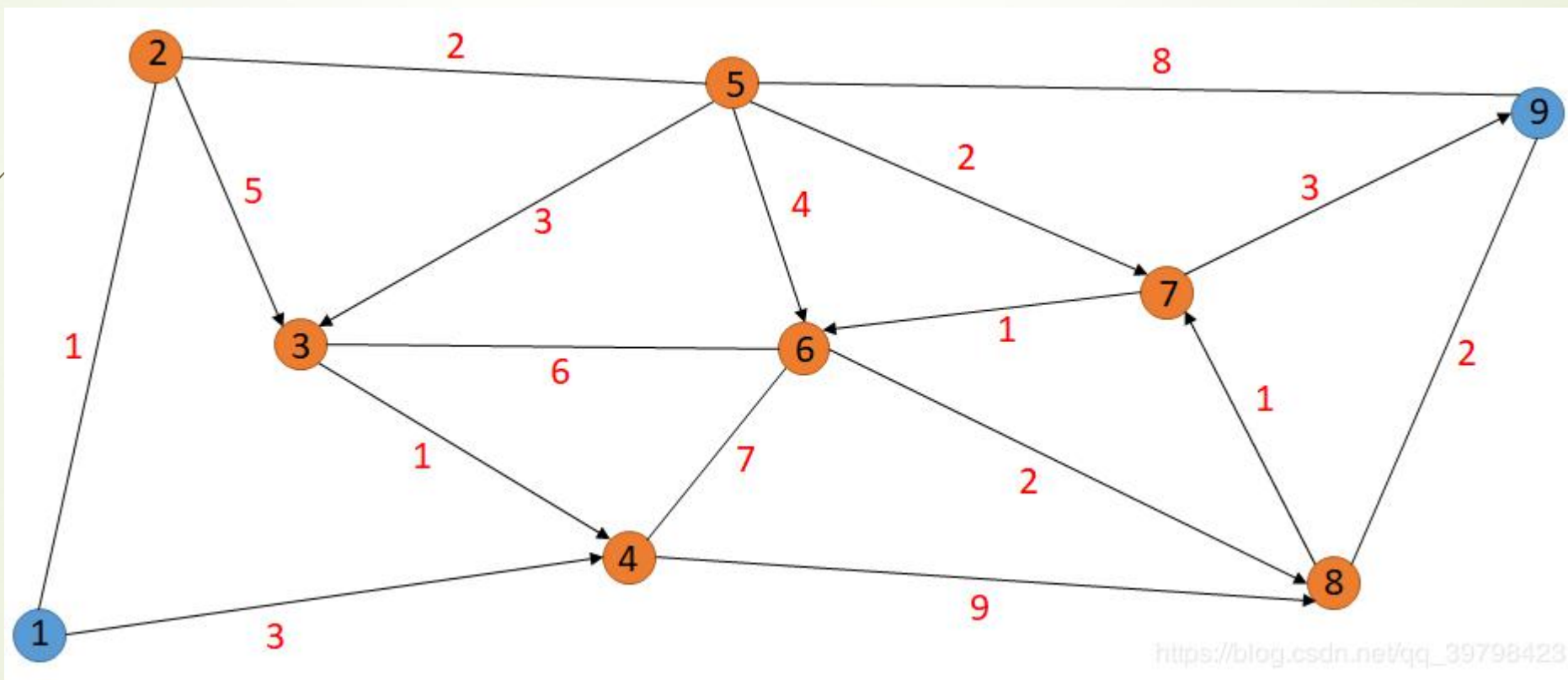
✓ 混合图:

若图中的部分边是有方向的, 而部分边是无方向的, 则称该图为混合图。



## 1 图论模型-Dijkstra

样例1：如下图所示，我们需要从①点走到⑨点，每条边的红色数字代表这条边的长度，我们如何找到①到⑨的最短路径呢？



# 1 图论模型-Dijkstra

➤ 步骤:

1. 将①标记为P, 其它标记为T, 找出从①出发当前最短的边所到的点, 将该点的T改为P
2. 将所有P点找到可以到达的T标记点上最短的边, 将到达的点T改为P
3. 重复步骤, 指导终点的T变为P

➤ 过程展示: 圈加数字代表每个顶点, ()内数字代表当前行走的距离

1.①(0)

2.①②(1)

3.①②⑤(3)

4.①②⑤(3)  
①④(3)

5.①②⑤⑦(5)

①④(3)

6.①②⑤⑦⑥(6)

①④(3)

7.①②⑤⑦⑥(6)  
①②⑤③(6)  
①④(3)

8.①②⑤⑦⑥(6)

①②⑤③④(7)

①④(3)

9.①②⑤⑦⑥⑧(8)

①②⑤③④(7)  
①④(3)

10.①②⑤⑦⑥⑧(8)

①②⑤⑦⑨(8)

①②⑤③④(7)

①④(3)

所以最短的路径是  
① ② ⑤ ⑦ ⑨,  
长度为8

# 1 图论模型-Dijkstra

## ➤ 带权邻接矩阵:

带权邻接矩阵是表示顶点相邻关系的矩阵，例如上面那个图的带权邻接矩阵如下

	①	②	③	④	⑤	⑥	⑦	⑧	⑨
①	0	1	inf	3	inf	inf	inf	inf	inf
②	1	0	5	inf	2	inf	inf	inf	inf
③	inf	inf	0	1	inf	6	inf	inf	inf
④	inf	inf	inf	0	inf	7	inf	9	inf
⑤	inf	2	3	inf	0	4	2	inf	8
⑥	inf	inf	6	7	inf	0	inf	2	inf
⑦	inf	inf	inf	inf	inf	1	0	inf	3
⑧	inf	inf	inf	inf	inf	inf	1	0	2
⑨	inf	inf	inf	inf	8	inf	3	2	0

- ✓ 每个点和自己的距离为0（主对角线上元素都是零）
- ✓ 在图上相邻两个点的如果是连通的，距离就是矩阵的值，无向的关于主对角线对称；有向的只有可以过去的路有数值
- ✓ 无法连通的距离就是无穷，记为inf

# 1 图论模型-Dijkstra

➤ 代码:

```
from collections import defaultdict
```

```
from heapq import *
```

```
inf = 99999 # 不连通值
```

```
mtx_graph = [[0, 1, inf, 3, inf, inf, inf, inf, inf],  
[1, 0, 5, inf, 2, inf, inf, inf, inf],  
[inf, inf, 0, 1, inf, 6, inf, inf, inf],  
[inf, inf, inf, 0, inf, 7, inf, 9, inf],  
[inf, 2, 3, inf, 0, 4, 2, inf, 8],  
[inf, inf, 6, 7, inf, 0, inf, 2, inf],  
[inf, inf, inf, inf, inf, inf, 1, 0, inf, 3],  
[inf, inf, inf, inf, inf, inf, inf, 1, 0, 2],  
[inf, inf, inf, inf, 8, inf, 3, 2, 0]]
```

<续下页>



# 1 图论模型-Dijkstra



```
m_n = len(mtx_graph) #带权连接矩阵的阶数
edges = [] #保存连通的两个点之间的距离 (点A、点B、距离)
for i in range(m_n):
    for j in range(m_n):
        if i!=j and mtx_graph[i][j]!=inf:
            edges.append(i,j,mtx_graph[i][j])
```

```
def dijkstra(edges, from_node, to_node):
```

```
    go_path = []
```

```
    to_node = to_node-1
```

```
    g = defaultdict(list)
```

```
    for l,r,c in edges:
```

```
        g[l].append((c,r))
```

```
    q, seen = [(0, from_node-1, ())], set()
```

<续下页>



# 1 图论模型-Dijkstra

while q:

(cost, v1, path) = heappop(q) # 堆弹出当前路径最小成本

if v1 not in seen:

    seen.add(v1)

    path = (v1, path)

    if v1 == to\_node:

        break

    for c, v2 in g.get(v1, ()):

        if v2 not in seen:

            heappush(q, (cost+c, v2, path))

→ if v1 != to\_node: # 无法到达

    return float['inf'], []

heapq

seen

path

cost

v1

v2

c

g

to\_node

float['inf'], []

堆中压入新的成本和路径

<续下页>

# 1 图论模型-Dijkstra

```
if len(path)>0:
    left=path[0]
    go_path.append(left)
    right=path[1]
    while len(right)>0:
        left=right[0]
        go_path.append(left)
        right=right[1]
    go_path.reverse() #逆序变换
    for i in range(len(go_path)): #标号加1
        go_path[i]=go_path[i]+1
    return cost, go_path
```

```
leght, path = Dijkstra(edges, 1, 9)
print('最短距离为: '+str(leght))
print('前进路径为: '+str(path))
```

1-9  
7-9

## 1 图论模型-Dijkstra

➤ 输出结果为：

- 最短距离为：8
- 前进路径为：[1, 2, 5, 7, 9]

## 2 图论模型-Floyd

Floyd算法主要通过动态规划解决任意两点的最短路径（多源最短路径）的问题，可以正确处理负权的最短路径问题

➤ 关键原理：

$$d[i][j] = \min_{1 \leq k \leq n} (d[i][k] + d[k][j])$$

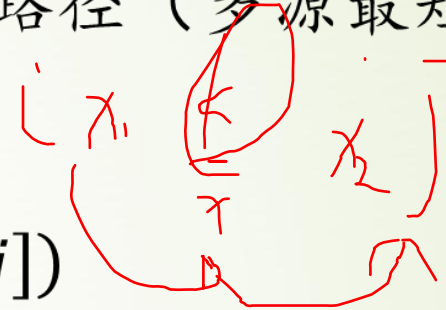
枚举中间点 $k$ ，找到最小的 $d[i][k] + d[k][j]$ ，作为 $d[i][j]$ 的最小值

➤ 关键结论：假设 $i$ 和 $j$ 之间的最短路径上的结点集里（不包含 $i, j$ ），编号最大的一个是 $x$ 。那么在外循环 $k = x$ 时， $d[i][j]$ 肯定得到了最小值。

<强归纳法>：设 $i$ 到 $x$ 中间编号最大的是 $x_1$ ， $x$ 到 $j$ 中间编号最大的是 $x_2$ 。由于 $x$ 是 $i$ 到 $j$ 中间编号最大的，那么显然 $x_1 < x$ ， $x_2 < x$ 。

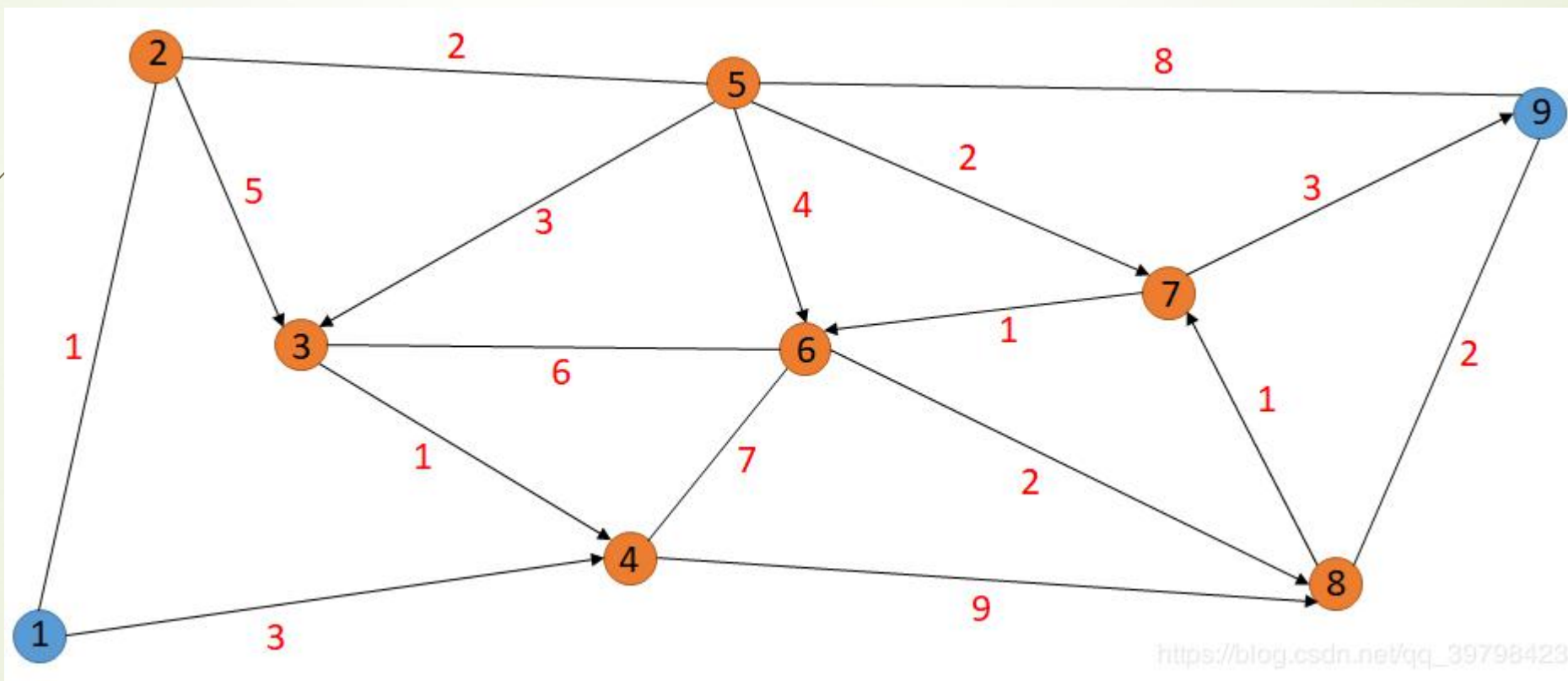
根据结论， $k = x_1$ 时候 $d[i][x]$ 已经取得最小值， $k = x_2$ 时候 $d[x][j]$ 已经取得最小值，那么 $k = x$ 时， $d[i][x]$ 和 $d[x][j]$ 已经都取得最小值。

因此 $k = x$ 时，执行 $d[i][j] = \min(d[i][j], d[i][k] + d[k][j])$ 得 $d[i][j]$



## 2 图论模型-Floyd

样例2：与例1相似，只是这次我们求出从每一个点到其他点的最短距离和路径，每条边的红色数字代表这条边的长度。




## 2 图论模型-Floyd

➤ 代码:

```
import numpy as np
inf = 99999 # 不连通值
mtx_graph = [[0, 1, inf, 3, inf, inf, inf, inf, inf],
              [1, 0, 5, inf, 2, inf, inf, inf, inf],
              [inf, inf, 0, 1, inf, 6, inf, inf, inf],
              [inf, inf, inf, 0, inf, 7, inf, 9, inf],
              [inf, 2, 3, inf, 0, 4, 2, inf, 8],
              [inf, inf, 6, 7, inf, 0, inf, 2, inf],
              [inf, inf, inf, inf, inf, 1, 0, inf, 3],
              [inf, inf, inf, inf, inf, inf, 1, 0, 2],
              [inf, inf, inf, inf, 8, inf, 3, 2, 0]]
```

〈续下页〉

## 2 图论模型-Floyd

```
def Floyd(graph):  
    N=len(graph)  
    A=np.array(graph)   
    path=np.zeros((N,N))  
    for i in range(0,N):  
        for j in range(0,N):  
            if A[i][j]!=inf:  
                path[i][j]=j  
    for k in range(0,N):  
        for i in range(0,N):  
            for j in range(0,N):  
                if A[i][j]+A[k][j]<A[i][j]:  
                    A[i][j]=A[i][k]+A[k][j]  
                    path[i][j]=path[i][k]
```

<续下页>



## 2 图论模型-Floyd

```
for k in range(0,N):  
    for i range(0,N):  
        for j in range(0,N):  
            if A[i][j]+A[k][j]<A[i][j]:  
                A[i][j]=A[i][k]+A[k][j]  
                path[i][j]=path[i][k]  
  
for i in range(0,N):  
    for j in range(0,N):  
        path[i][j]=path[i][j]+1  
  
print('距离 = ')  
print(A)  
print('路径 = ')  
print(path)  
Floyd(mtx_graph)
```

## 2 图论模型-Floyd

➤ 输出结果为:

距离 =

[ [ 0	1	6	3	3	6	5	8	8 ]
[ 1	0	5	4	2	5	4	7	7 ]
[ 21	20	0	1	18	6	9	8	10 ]
[ 22	21	13	0	19	7	10	9	11 ]
[ 3	2	3	4	0	3	2	5	5 ]
[ 15	14	6	7	12	0	3	2	4 ]
[ 14	13	7	8	11	1	0	3	3 ]
[ 13	12	8	9	10	2	1	0	2 ]
[ 11	10	10	11	8	4	3	2	0 ] ]

路径 =

[ [ 1.	2.	2.	4.	2.	2.	2.	2.	2 ]
[ 1.	2.	3.	1.	5.	5.	5.	5.	5 ]
[ 6.	6.	3.	4.	6.	6.	6.	6.	6 ]
[ 8.	8.	6.	4.	8.	6.	8.	8.	8 ]
[ 2.	2.	3.	3.	5.	7.	7.	7.	7 ]
[ 8.	8.	3.	4.	8.	6.	8.	8.	8 ]
[ 9.	9.	6.	6.	9.	6.	7.	6.	9 ]
[ 9.	9.	7.	7.	9.	7.	7.	8.	9 ]
[ 5.	5.	7.	7.	5.	7.	7.	8.	9 ] ]

➤ 结论:

✓ 从①到⑨的距离为8 (距离矩阵第一行第九列数值)

✓ 路径为[1-2-5-7-9] (看路径矩阵从①到⑨看第一行第九列的数值为2, 这是中间位置②, 再转到第二行第九列看数值为5, 得到第二个中间位置⑤, 再转到第五行第九列, 一直找到⑨, 就能得到路径为【1 - 2 - 5 - 7 - 9】)

### 3 机场航线设计

Airvlines

- 数据集来自航空业，有一些关于航线的基本信息。有某段旅程的起始点和目的地。还有一些列表示每段旅程的到达和起飞时间。这个数据集非常适合作为图进行分析。想象一下通过航线（边）连接的几个城市（节点）。如果你是航空公司，你可以问如下几个问题：
- ✓ 从A到B的最短途径是什么？分别从距离和时间角度考虑。
  - ✓ 有没有办法从C到D？
  - ✓ 哪些机场的交通最繁忙？
  - ✓ 哪个机场位于大多数其他机场“之间”？这样它就可以变成当地的一个中转站。

### 3 机场航线设计

#### ➤ 数据导入、观察变量

```
import numpy as np
import pandas as pd
→ data = pd.read_csv('data/Airlines.csv')
data.shape
```

```
> (100, 16)
```

```
data.dtypes
```

```
> year int64
```

```
> month int64
```

```
> day int64
```

```
> dep_time float64
```

```
→ sched_dep_time int64
```

```
> dep_delay float64
```

```
> arr_time float64
```

```
> sched_arr_time int64
```

```
> arr_delay float64
```

```
...
```

date\_from

- ◆ 起始点和目的地可以作为节点，其他信息应当作为节点或边属性；单条边可以被认为是一段旅程。这样的旅程将有不同的时间，航班号，飞机尾号等相关信息。
- ◆ 注意到年，月，日和日期时间信息分散在许多列；想创建一个包含所有这些信息的日期时间列，还需要将预计的(scheduled)和实际的(actual)到达离开时间分开；最终应该有4个日期时间列（预计到达时间、预计起飞时间、实际到达时间和实际起飞时间）
- ◆ 时间格式问题
- ◆ 数据类型问题
- ◆ NaN值的麻烦

### 3 机场航线设计

#### ➤ 数据清洗

#将sched\_dep\_time转换为'std'—预定的出发时间

```
data['std'] = data.sched_dep_time.astype(str).str.replace('(\d{2}$)', '') +  
' :' + data.sched_dep_time.astype(str).str.extract('(\d{2}$)', expand=False) +  
' :00'
```

#将sched\_arr\_time转换为“sta”—预定到达时间

```
data['sta'] = data.sched_arr_time.astype(str).str.replace('(\d{2}$)', '') +  
' :' + data.sched_arr_time.astype(str).str.extract('(\d{2}$)', expand=False) +  
' :00'
```

#将dep\_time转换为'atd' —实际出发时间

```
data['atd'] =  
data.dep_time.fillna(0).astype(np.int64).astype(str).str.replace('(\d{2}$)',  
'') + ' :' +  
data.dep_time.fillna(0).astype(np.int64).astype(str).str.extract('(\d{2}$)',  
expand=False) + ' :00'
```

#将arr\_time转换为'ata' —实际到达时间

```
data['ata'] =  
data.arr_time.fillna(0).astype(np.int64).astype(str).str.replace('(\d{2}$)',  
'') + ' :' +  
data.arr_time.fillna(0).astype(np.int64).astype(str).str.extract('(\d{2}$)',
```

String 15

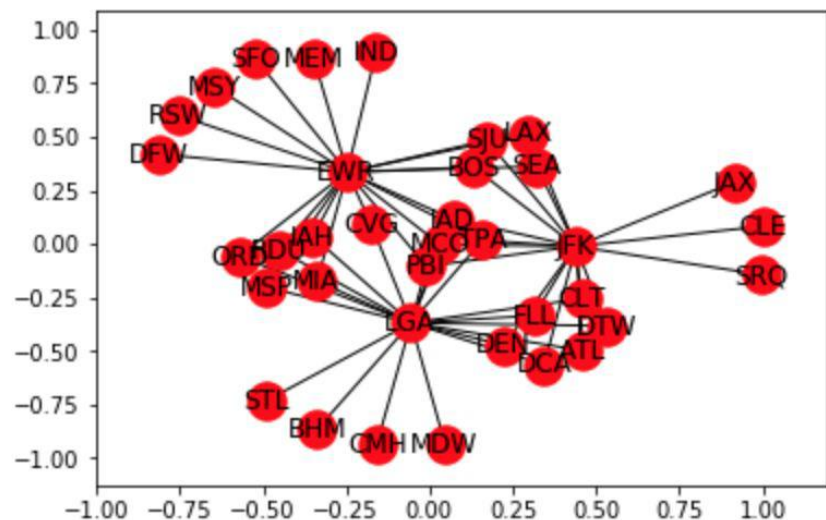
### 3 机场航线设计

#### ➤ 时间信息合并

```
data['date'] = pd.to_datetime(data[['year', 'month', 'day']])  
data = data.drop(columns = ['year', 'month', 'day'])
```

#### ➤ 创建图

```
import networkx as nx  
FG = nx.from_pandas_edgelist(data, source='origin', target='dest', edge_attr=True,)  
FG.nodes() # 查看所有节点  
FG.edges() # 查看所有边  
nx.draw_networkx(FG, with_labels=True) # 快速查看图表, 发现3个十分繁忙的机场
```



```
nx.algorithms.degree_centrality(FG) # 100条记录3机场  
nx.density(FG) # 图的平均边密度 0.09047619047619047  
nx.average_shortest_path_length(FG)  
# 图中所有路径的平均最短路径长度2.36984126984127  
nx.average_degree_connectivity(FG)  
# 对于一个度为k的节点-它的邻居度的平均值是多少?  
#{1: 19.307692307692307, 2: 19.0625, 3: 19.0, 17:  
2.0588235294117645, 20: 1.95}
```



### 3 机场航线设计

Drui

#### ➤ 两个机场最短路线

- 距离最短的路径
- 飞行时间最短的路径

我们可以通过距离或飞行时间来给路径赋予权重，并用算法计算最短路径。

实际问题是计算当你到达中转机场时的航班可用性加候机的等待时间，这是人们计划旅行的方式。

出于简化，假设你到达机场时可以随时使用航班并使用飞行时间作为权重，从而计算最短路径。

以JAX和DFW机场为例，先按照求取距离最短的路径

```
for path in nx.all_simple_paths(FG, source='JAX', target='DFW'):  
    print(path)  
dijpath = nx.dijkstra_path(FG, source='JAX', target='DFW')  
dijpath
```

输出: ['JAX', 'JFK', 'SEA', 'EWR', 'DFW']

再按照求取飞行时间最短的路径

```
shortpath = nx.dijkstra_path(FG, source='JAX', target='DFW', weight='air_time')  
shortpath
```

输出: ['JAX', 'JFK', 'BOS', 'EWR', 'DFW']



### 3 机场航线设计

12 →

#### ➤ 更多可以尝试的问题：

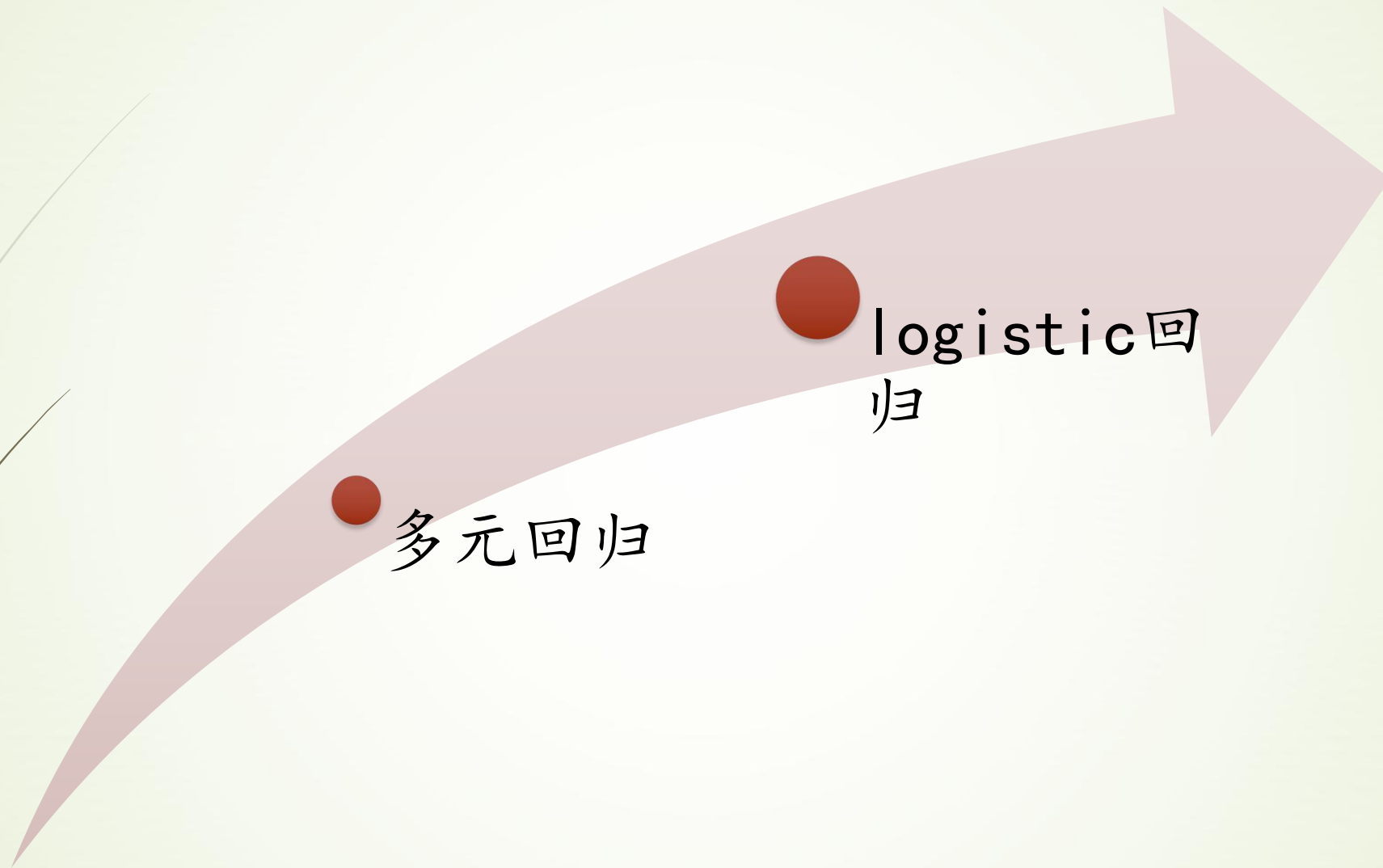
- 在给定成本，飞行时间和可用性的情况下，找到两个机场之间的最短路径？
- 作为一家航空公司，你们拥有一队飞机。你了解航班的需求。假设你有权再运营2架飞机（或者为你的机队添加2架飞机），把这两架飞机投入到哪条航线可以最大限度地提高盈利能力？
- 你可以重新安排航班和时刻表以优化某个参数吗？（如时效性或盈利能力等）

可行的

是否可行成本

# Python之建模回归篇





多元回归



logistic回  
归

# 1 python实现多元回归

## 1.1 选取数据

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as mpl    #显示中文
def mul_lr():
    pd_data=pd.read_excel('C:\\Users\\lenovo\\Desktop\\test.xlsx')
    mpl.rcParams['font.sans-serif'] = ['SimHei']    #配置显示中文，否则乱码
    mpl.rcParams['axes.unicode_minus']=False #用来正常显示负号
    sns.pairplot(pd_data, x_vars=['中证500','沪深300','上证50','上证180'], y_vars='
    上证指数',kind="reg", size=5, aspect=0.7)
    plt.show()
```

日期	中证500	沪深300	上证50	上证180	上证指数	大V指数
2013年11月	6.2148	2.8586	2.0446	3.3844	3.7168	17.71896
2013年12月	-2.7939	-4.4463	-5.2215	-4.811	-4.7284	-17.5809
2014年1月	1.6272	-5.5126	-6.2815	-5.1455	-3.9151	-17.1323
2014年2月	2.5048	-0.9411	-0.7577	-0.9174	1.2255	0.210567
2014年3月	-3.2479	-1.3509	0.2709	-0.6597	-1.0057	-5.60884
2014年4月	-1.813	0.6817	2.1168	0.8143	-0.258	1.594701
2014年5月	1.7678	-0.0378	-0.7972	-0.2607	0.6812	1.30814
2014年6月	2.5614	0.458	0.0805	0.4252	0.4897	3.997903
2014年7月	8.2138	8.3028	9.1111	8.724	7.2834	39.38692
2014年8月	4.0253	-0.4312	-2.4434	-0.8538	0.7689	0.459126
2014年9月	10.6109	4.7972	2.4936	4.5789	6.4858	28.62751
2014年10月	1.5257	2.3893	2.3794	2.6099	2.4228	9.178061
2014年11月	5.1817	11.4593	14.6701	12.9085	10.4117	49.88368
2014年12月	1.8753	23.6541	31.0665	26.513	19.3253	91.44228
2015年1月	5.9681	-2.2713	-6.2428	-4.1344	-0.0959	-6.70726
2015年2月	6.7223	4.1075	3.0376	3.7433	3.2089	18.52537
2015年3月	18.8486	12.7949	11.071	12.3981	12.6127	58.23848
2015年4月	15.9286	16.1768	16.8956	17.0223	17.2677	73.73779
2015年5月	17.0032	2.4921	-3.7975	-0.0995	4.4024	19.31175

# 1 python实现多元回归

## 1.2 构建训练集与测试级，并构建模型

```
from sklearn.model_selection import train_test_split #这里是引用了交叉验证
from sklearn.linear_model import LinearRegression #线性回归
from sklearn import metrics
import numpy as np
def mul_lr(): #剔除日期数据
    X=pd_data.loc[:,('中证500','沪深300','上证50','上证180')]
    y=pd_data.loc[:, '上证指数']
    X_train,X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=100)
    linreg = LinearRegression()
    model=linreg.fit(X_train, y_train)
    print (model)
    # 训练后模型截距
    print (linreg.intercept_)
    # 训练后模型权重 (特征个数无变化)
    print (linreg.coef_)
```

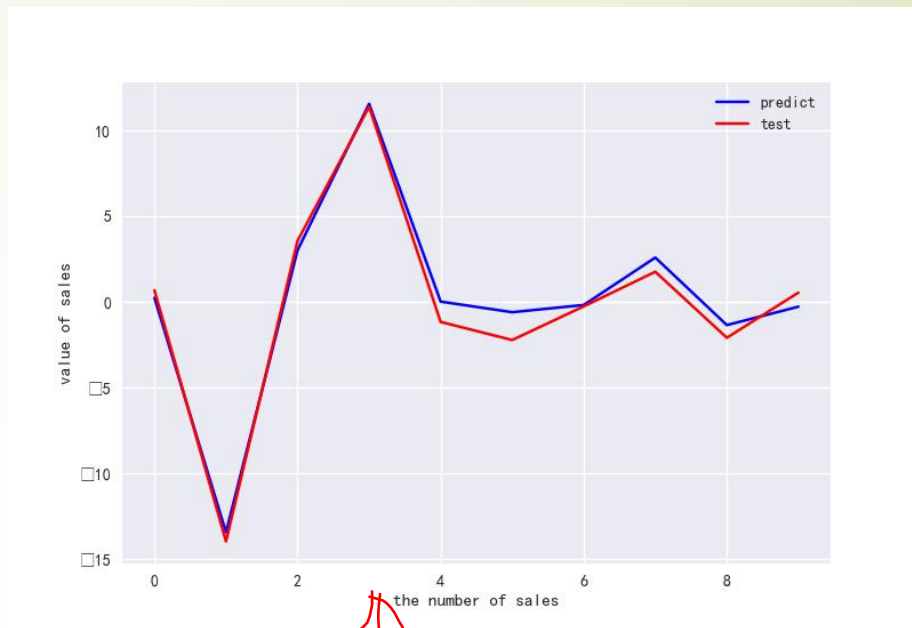
# 1 python实现多元回归

## 1.3 模型预测

```
y_pred = linreg.predict(X_test)
print (y_pred) #10个变量的预测结果
```

## 1.4 模型评估

```
sum_mean=0
for i in range(len(y_pred)):
    sum_mean+=(y_pred[i]-y_test.values[i])**2
sum_erro=np.sqrt(sum_mean/10) #这个10是你测试级的数量
# RMSE
print ("RMSE:",sum_erro)
#做预测对比曲线
plt.figure()
plt.plot(range(len(y_pred)),y_pred,'b',label="predict")
plt.plot(range(len(y_test)),y_test,'r',label="test")
plt.legend(loc="upper right") #显示图中的标签
plt.xlabel("the number of sales")
plt.ylabel('value of sales')
plt.show()
```



## 2 鸢尾花数据集的Logistic回归

### 2.1 鸢尾花数据集

鸢尾花有三个亚属，分别是山鸢尾（Iris-setosa）、变色鸢尾（Iris-versicolor）和维吉尼亚鸢尾（Iris-virginica）。该数据集一共包含4个特征变量，1个类别变量。共有150个样本，iris是鸢尾植物，这里存储了其萼片和花瓣的长宽，共4个属性，鸢尾植物分三类。

### 2.2 绘制散点图

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris
```

```
iris = load_iris()
```

```
#获取花卉两列数据集
```

```
DD = iris.data
```

```
X = [x[0] for x in DD]
```

```
Y = [x[1] for x in DD]
```

```
plt.scatter(X[:50], Y[:50], color='red', marker='o', label='setosa')
```

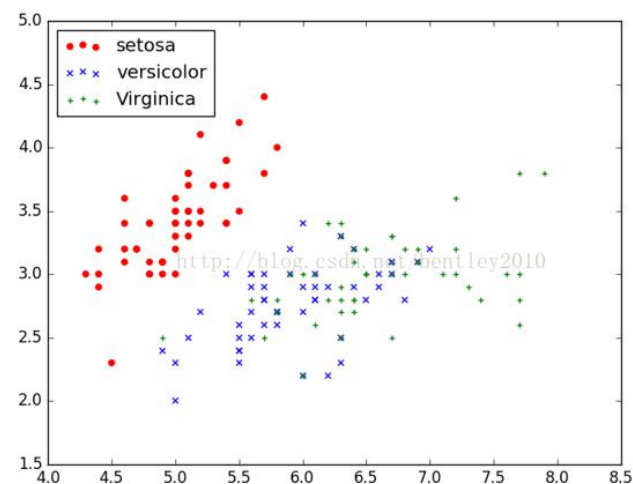
```
plt.scatter(X[50:100], Y[50:100], color='blue', marker='x', label='versicolor')
```

```
plt.scatter(X[100:], Y[100:], color='green', marker='+', label='Virginica')
```

```
plt.legend(loc=2) #左上角
```

```
plt.show()
```

列名	说明	类型
SepalLength	花萼长度	float
SepalWidth	花萼宽度	float
PetalLength	花瓣长度	float
PetalWidth	花瓣宽度	float
Class	类别变量。0 表示山鸢尾，1 表示变色鸢尾，2 表示维吉尼亚鸢尾。	int





## 2 鸢尾花数据集的Logistic回归

### 2.3逻辑回归分析

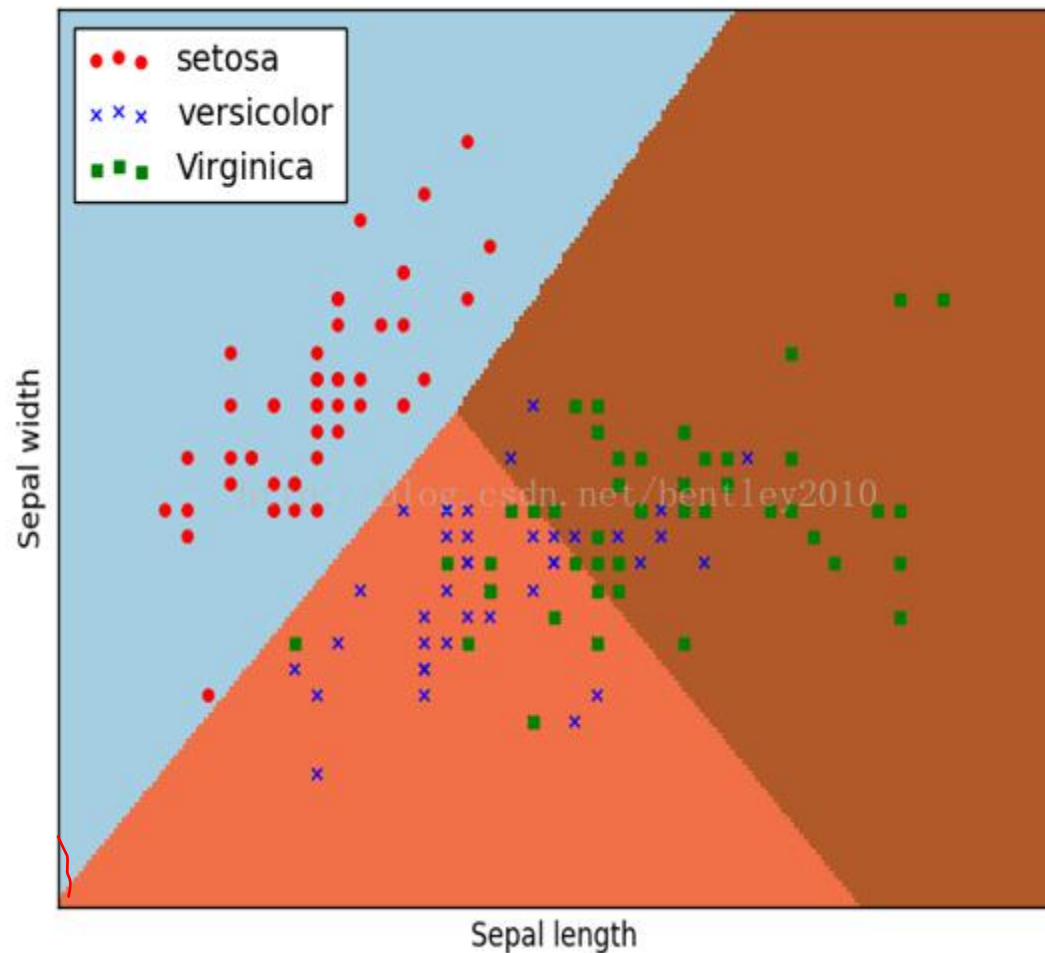
```
from sklearn.linear_model import LogisticRegression
iris = load_iris()
X = iris.data[:, :2] #获取花卉两列数据集
Y = iris.target
lr = LogisticRegression(C=1e5)
lr.fit(X, Y)
#meshgrid函数生成两个网格矩阵
h = .02
x_min, x_max = X[:, 0].min()-.5, X[:, 0].max()+.5
y_min, y_max = X[:, 1].min()-.5, X[:, 1].max()+.5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = lr.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(8, 6))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)
plt.scatter(X[:50, 0], X[:50, 1], color='red', marker='o', label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 1], color='blue', marker='x', label='versicolor')
plt.scatter(X[100:, 0], X[100:, 1], color='green', marker='s', label='Virginica')
```

<续下页>

## 2 鸢尾花数据集的Logistic回归

### 2.3逻辑回归分析

```
plt.xlabel('Sepal length')  
plt.ylabel('Sepal width')  
plt.xlim(xx.min(), xx.max())  
plt.ylim(yy.min(), yy.max())  
plt.xticks(())  
plt.yticks(())  
plt.legend(loc=2)  
plt.show()
```





下课，谢谢！