# 实验报告

## 实验1：随机漫步

### 实验目的： 通过模拟随机漫步学习如何运用数组运算

### 实验步骤：

- 通过内置的random模块以纯Python的方式实现随机漫步并生成折线图
- 使用np.random模块一次性随机产生
- 一次模拟多个随机漫步

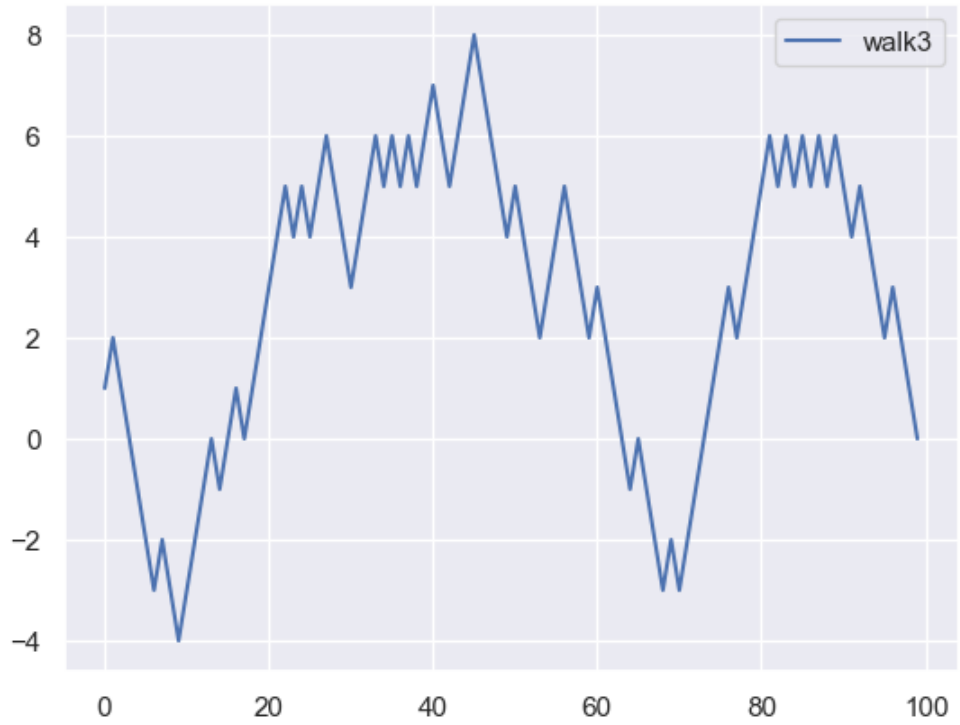
In [10]:
```
import random

import pandas as pd

random.seed(5)
# 法1
position = 0
walk = [position]
steps = 100
for i in range(steps):
    step = 1 if random.randint(0, 1) else -1
    position += step
    walk.append(position)
print(walk)
# 法二
step2 = [1 if random.randint(0, 1) else -1 for _ in range(steps)]
walk2 = [sum(step2[:i + 1]) for i in range(steps)]
print(walk2)
```

```
[0, 1, 2, 1, 2, 1, 0, -1, -2, -1, 0, -1, 0, -1, -2, -3, -4, -3, -2, -3, -2, -3, -4, -5, -4, -5, -6, -7, -8, -9, -10, -11, -12, -11, -10, -11, -12, -13, -14, -13, -12, -13, -12, -11, -12, -13, -12, -13, -12, -11, -12, -11, -12, -11, -10, -9, -8, -7, -8, -7, -6, -7, -8, -7, -8, -7, -6, -7, -6, -5, -4, -5, -4, -5, -6, -7, -8, -9, -8, -7, -6, -5, -4, -5, -4, -3, -4, -3, -4, -5, -4, -3, -4, -3, -2, -3, -2, -1, 0, -1, -2]
[-1, 0, 1, 0, -1, -2, -1, -2, -3, -2, -1, 0, 1, 0, -1, -2, -1, 0, -1, -2, -3, -2, -3, -4, -3, -2, -3, -4, -3, -2, -3, -2, -3, -2, -1, 0, 1, 2, 3, 4, 5, 4, 3, 4, 3, 4, 5, 4, 3, 4, 5, 6, 5, 6, 5, 6, 7, 8, 9, 10, 11, 12, 11, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 21, 22, 23, 24, 23, 22, 21, 22, 23, 24, 23, 24, 23, 24, 25, 26, 25, 24, 23, 24, 23, 22, 21, 20]
```

In [17]:
```
import matplotlib.pyplot as plt
import seaborn as sn
sn.set()
# plt.subplot(2, 1, 2)
plt.plot(walk, label='walk')
plt.plot(walk2, label='walk2')
plt.legend()
plt.show()
```

In [31]:
```
import numpy as np
# 法三
nsteps = 100
draws = np.random.randint(0, 2, size=nsteps)
steps = np.where(draws > 0, 1, -1)
walk3 = steps.cumsum()
plt.plot(walk3, label='walk3')
plt.legend()
```

Out[31]: <matplotlib.legend.Legend at 0x207a502b070>



In [40]:
```
# 一次生成多个
nwalks = 5
nsteps = 100
draws = np.random.randint(0, 2, size=(nwalks, nsteps))
steps = np.where(draws > 0, 1, -1)
walks = steps.cumsum(axis=1)
walks
```

Out[40]:
```
array([[  1,   2,   1,   0,  -1,   0,   1,   2,   1,   0,  -1,   0,   1,
          2,   3,   2,   3,   2,   1,   0,   1,   2,   1,   0,  -1,   0,
         -1,  -2,  -3,  -2,  -1,  -2,  -1,  -2,  -1,  -2,  -1,   0,   1,
          2,   1,   2,   3,   4,   5,   6,   7,   6,   7,   6,   7,   8,
          9,   8,   7,   6,   5,   4,   3,   2,   1,   0,  -1,   0,  -1,
         -2,  -3,  -2,  -1,  -2,  -1,  -2,  -1,  -2,  -3,  -4,  -3,  -4,
         -5,  -6,  -7,  -6,  -7,  -8,  -9,  -8,  -9, -10,  -9, -10, -11,
        -10, -11, -12, -13, -14, -13, -14, -13, -12],
       [ -1,   0,  -1,  -2,  -3,  -2,  -3,  -2,  -3,  -4,  -5,  -6,  -7,
         -6,  -5,  -4,  -5,  -6,  -5,  -6,  -5,  -4,  -5,  -4,  -5,  -4,
         -5,  -6,  -5,  -4,  -5,  -4,  -3,  -4,  -5,  -6,  -7,  -6,  -5,
         -6,  -5,  -6,  -5,  -6,  -5,  -6,  -5,  -4,  -3,  -2,  -1,  -2,
         -1,  -2,  -1,   0,  -1,  -2,  -1,   0,   1,   2,   3,   2,   3,
          4,   5,   4,   5,   6,   5,   4,   3,   4,   3,   4,   5,   6,
          5,   6,   5,   6,   5,   6,   7,   8,   7,   6,   5,   4,   5,
          4,   3,   4,   5,   4,   3,   2,   1,   2],
       [  1,   0,  -1,   0,   1,   2,   3,   4,   3,   2,   3,   4,   3,
          4,   5,   4,   5,   4,   5,   4,   5,   6,   5,   6,   5,   6,
          7,   8,   9,   8,   7,   6,   5,   4,   5,   4,   3,   4,   3,
          2,   1,   2,   1,   2,   3,   4,   5,   4,   3,   2,   1,   0,
         -1,   0,  -1,  -2,  -3,  -4,  -3,  -2,  -3,  -2,  -1,  -2,  -3,
         -2,  -1,  -2,  -1,   0,   1,   2,   1,   2,   1,   2,   1,   0,
          1,   2,   3,   4,   3,   2,   1,   0,  -1,   0,  -1,   0,   1,
          2,   3,   4,   5,   6,   5,   4,   5,   6],
       [  1,   0,   1,   0,  -1,   0,  -1,   0,  -1,   0,   1,   0,  -1,
          0,   1,   0,   1,   0,  -1,  -2,  -1,  -2,  -3,  -2,  -1,   0,
          1,   0,  -1,  -2,  -1,  -2,  -1,   0,   1,   2,   1,   0,  -1,
          0,  -1,   0,  -1,  -2,  -1,  -2,  -1,  -2,  -1,   0,  -1,  -2,
         -3,  -2,  -3,  -4,  -3,  -4,  -5,  -4,  -3,  -2,  -3,  -4,  -3,
         -2,  -3,  -2,  -3,  -4,  -3,  -2,  -3,  -4,  -3,  -2,  -1,   0,
          1,   2,   3,   2,   3,   2,   3,   4,   5,   4,   3,   4,   5,
          4,   5,   4,   5,   6,   5,   6,   5,   6],
       [  1,   2,   3,   2,   1,   2,   3,   2,   3,   4,   3,   2,   1,
          2,   3,   4,   3,   2,   1,   0,   1,   0,   1,   2,   3,   4,
          3,   4,   5,   6,   5,   6,   7,   8,   7,   6,   5,   6,   7,
          8,   9,   8,   9,   8,   7,   6,   7,   8,   9,   8,   9,  10,
          9,  10,  11,  12,  11,  10,   9,  10,  11,  10,  11,  10,  11,
         10,  11,  12,  11,  10,  11,  12,  13,  12,  13,  12,  13,  12,
         11,  10,   9,   8,   9,   8,   9,   8,   7,   6,   5,   4,   3,
          4,   3,   4,   3,   4,   5,   4,   3,   2]])
```

In [42]:
```
plt.plot(walks.T)
```

Out[42]:
```
[<matplotlib.lines.Line2D at 0x207a8ae32b0>,
 <matplotlib.lines.Line2D at 0x207a8b1b460>,
 <matplotlib.lines.Line2D at 0x207a8b1b490>,
 <matplotlib.lines.Line2D at 0x207a8b1b580>,
 <matplotlib.lines.Line2D at 0x207a8b1b670>]
```

In [43]:
```
steps = np.random.normal(loc=0, scale=0.25,size=(nwalks, nsteps))
normal_walk = steps.cumsum(axis=1)
normal_walk
```

```
7.56231043e-01,  1.27171955e+00,  1.35820830e+00,
1.69339581e+00,  1.31780486e+00,  1.06414042e+00,
1.11833391e+00,  1.46536686e+00,  1.37117161e+00,
9.79101293e-01,  1.13124610e+00,  1.13579611e+00,
1.16638716e+00,  1.15279538e+00,  1.44755181e+00,
1.29360123e+00,  1.01645716e+00,  5.18389575e-01,
6.16279318e-01,  3.87935527e-01,  3.47389649e-01,
6.18108905e-01,  7.93520159e-01,  8.96951387e-01,
6.53581131e-01,  8.10757681e-01,  1.12580444e+00,
1.04757634e+00,  7.24986332e-01,  8.09389704e-01,
6.42449747e-01,  3.46847903e-01,  4.92857914e-01,
6.99874579e-01,  7.07040235e-01,  7.99164936e-01,
8.44763974e-01,  5.40323080e-01,  5.13186048e-01,
2.91648112e-01,  2.16460556e-01,  4.45908292e-01,
1.81334713e-02, -2.15214392e-01,  4.41640763e-02,
-2.49280004e-01, -3.82455932e-01, -3.86118669e-01,
-3.49222335e-01, -9.81280297e-02,  1.15418205e-02,
1.89052437e-01,  5.71321014e-01,  5.37959354e-01,
2.62105370e-01,  1.89344060e-01, -1.49582854e-01,
-1.69883365e-01,  3.20904919e-02, -2.57883854e-01,
```

In [47]:
```
plt.plot(normal_walk.T, label='normal')
```

Out[47]:
```
[<matplotlib.lines.Line2D at 0x207a9052f80>,
 <matplotlib.lines.Line2D at 0x207a90535e0>,
 <matplotlib.lines.Line2D at 0x207a9051e10>,
 <matplotlib.lines.Line2D at 0x207a8ed1600>,
 <matplotlib.lines.Line2D at 0x207a8ed16f0>]
```


**心得：**

在数据处理上，numpy库确实比Python有更大的优势。

## 实验二：计算指标/哑变量

**实验目的： 通过MovieLens 1M数据集体会如何对数据进行编码**

**实验步骤：**

- 从文件读取数据
- 从数据中抽取特征
- 通过迭代对数据进行编码

In [35]:
```
import pandas as pd
import numpy as np
```

### 1. 导入数据

In [17]:
```
names = ['movie_id', 'title', 'genres']
movies = pd.read_table('movies.dat', sep='::',header=None, names=names, engine='python')
movies.head()
```

Out[17]:

| | movie_id | title | genres |
|---|---|---|---|
| **0** | 1 | Toy Story (1995) | Animation\|Children's\|Comedy |
| **1** | 2 | Jumanji (1995) | Adventure\|Children's\|Fantasy |
| **2** | 3 | Grumpier Old Men (1995) | Comedy\|Romance |
| **3** | 4 | Waiting to Exhale (1995) | Comedy\|Drama |
| **4** | 5 | Father of the Bride Part II (1995) | Comedy |

### 2. 查看数据信息

In [52]:
```
movies.describe()
```

Out[52]:

| | title | genres |
|---|---|---|
| **count** | 3883 | 3883 |
| **unique** | 3883 | 301 |
| **top** | Toy Story (1995) | Drama |
| **freq** | 1 | 843 |

In [53]:
```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3883 entries, 1 to 3952
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   title   3883 non-null   object
 1   genres  3883 non-null   object
dtypes: object(2)
memory usage: 91.0+ KB
```

### 3. 重设索引

In [18]:
```
movies.set_index('movie_id', inplace=True)
movies
```

Out[18]:

| movie_id | title | genres |
|---|---|---|
| 1 | Toy Story (1995) | Animation\|Children's\|Comedy |
| 2 | Jumanji (1995) | Adventure\|Children's\|Fantasy |
| 3 | Grumpier Old Men (1995) | Comedy\|Romance |
| 4 | Waiting to Exhale (1995) | Comedy\|Drama |
| 5 | Father of the Bride Part II (1995) | Comedy |
| ... | ... | ... |
| 3948 | Meet the Parents (2000) | Comedy |
| 3949 | Requiem for a Dream (2000) | Drama |
| 3950 | Tigerland (2000) | Drama |
| 3951 | Two Family House (2000) | Drama |
| 3952 | Contender, The (2000) | Drama\|Thriller |

3883 rows × 2 columns

### 4. 判断数据是否缺失

In [59]:
```
movies.isnull().any()
```

Out[59]:
```
title     False
genres    False
dtype: bool
```

### 5. 判断数据是否重复

In [64]:
```
movies.duplicated().any()
```

Out[64]:
```
False
```

### 6. 提取特征

In [32]:
```
all_genres = []
for x in movies.genres:
    all_genres.extend(x.split('|'))
genres = pd.unique(all_genres)
genres
```

Out[32]:
```
array(['Animation', "Children's", 'Comedy', 'Adventure', 'Fantasy',
       'Romance', 'Drama', 'Action', 'Crime', 'Thriller', 'Horror',
       'Sci-Fi', 'Documentary', 'War', 'Musical', 'Mystery', 'Film-Noir',
       'Western'], dtype=object)
```

### 7. 编码

In [65]:
```
# 构建指标DataFrame
zero_matrix=np.zeros(shape=(len(movies), len(genres)))
dummies = pd.DataFrame(zero_matrix, columns=genres)
dummies.head()
```

Out[65]:

| | Animation | Children's | Comedy | Adventure | Fantasy | Romance | Drama | Action | Crime | Thriller | Horror | Sci-Fi | Documen |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **1** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **2** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **3** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |

In [66]:
```
for num, gen in enumerate(movies.genres):
    indices = dummies.columns.get_indexer(gen.split('|'))
    dummies.iloc[num, indices] = 1
dummies.head()
```

Out[66]:

| | Animation | Children's | Comedy | Adventure | Fantasy | Romance | Drama | Action | Crime | Thriller | Horror | Sci-Fi | Documen |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1.0 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **1** | 0.0 | 1.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **2** | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **3** | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **4** | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |

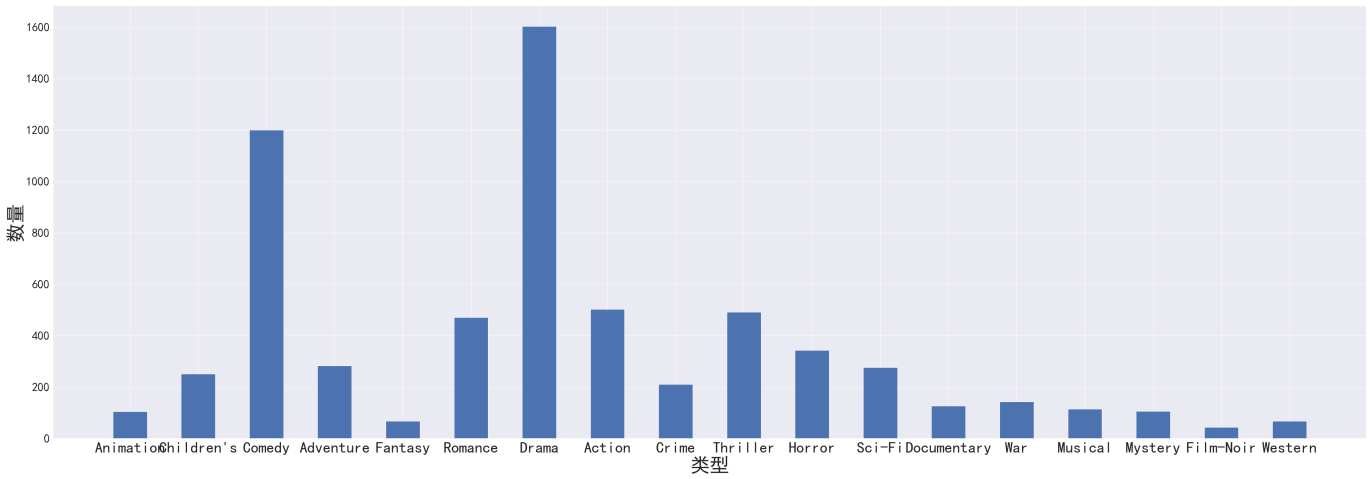
In [77]:
```
# 每种特征所包含的书籍数量
dummies.sum()
```

Out[77]:
```
Animation       105.0
Children's      251.0
Comedy         1200.0
Adventure       283.0
Fantasy          68.0
Romance         471.0
Drama          1603.0
Action          503.0
Crime           211.0
Thriller        492.0
Horror          343.0
Sci-Fi          276.0
Documentary     127.0
War             143.0
Musical         114.0
Mystery         106.0
Film-Noir        44.0
Western          68.0
dtype: float64
```

### 8. 图像显示

In [96]:
```
import seaborn as sn
import matplotlib.pyplot as plt
sn.set()
plt.figure(figsize=(60, 20), dpi=100)
plt.rcParams['font.family'] = 'SimHei'
plt.bar(dummies.sum().index, height=dummies.sum(), label=dummies.columns,width=0.5)
plt.xlabel('类型', fontsize=50)
plt.ylabel('数量', fontsize=50)
plt.xticks(fontsize=40)
plt.yticks(fontsize=30)
```

Out[96]:
```
(array([   0.,  200.,  400.,  600.,  800., 1000., 1200., 1400., 1600.,
        1800.]),
 [Text(0, 0.0, '0'),
  Text(0, 200.0, '200'),
  Text(0, 400.0, '400'),
  Text(0, 600.0, '600'),
  Text(0, 800.0, '800'),
  Text(0, 1000.0, '1000'),
  Text(0, 1200.0, '1200'),
  Text(0, 1400.0, '1400'),
  Text(0, 1600.0, '1600'),
  Text(0, 1800.0, '1800')])
```




## 心得:

通过使用Numpy和Pandas库，我们可以更加高效的准备数据，从而将更多的时间用于数据分析。从长远看，这样可以极大地提高生产力，快速完成数据分析工作。在使用Python 中，我发现自己有很多地方仍然不熟练，我还需要多加练习，提高自己的熟练程度。