

로또 시뮬레이션 및 조작설 검증

데이터 과학 기초

시뮬레이션에 필요한 의미 있는 데이터 추출

https://colab.research.google.com/drive/1RGvsnxfbCtHBC2H4duD4_slT-AJIsKSN#scrollTo=UV49DsSxe3SI

- 시뮬레이션을 위해 회차별 1등 당첨 번호와 총 판매 로또 수, 1등 당첨자 수 가 필요
- 필요한 데이터만 추출 (lotto_data_table)

	회차	1등 당첨 번호	판매 로또 수	로또 1등 당첨자 수
0	1	[10, 23, 29, 33, 37, 40]	1840891	0
1	2	[9, 13, 21, 25, 32, 42]	2452137	1
2	3	[11, 16, 19, 21, 27, 31]	2364671	1
3	4	[14, 27, 30, 31, 40, 42]	2635732	0
4	5	[16, 24, 29, 40, 41, 42]	3138551	0
...
1086	1087	[13, 14, 18, 21, 34, 44]	113960897	16
1087	1088	[11, 21, 22, 30, 39, 44]	109089305	11
1088	1089	[4, 18, 31, 37, 42, 43]	108725932	9
1089	1090	[12, 19, 21, 29, 40, 45]	108364862	11
1090	1091	[6, 20, 23, 24, 28, 30]	109335581	9

1091 rows × 4 columns

시뮬레이션에 필요한 의미 있는 데이터 추출

- 1091회 까지 로또는 약 692억장 가량 판매됨 (total_sales)
- 1091회 까지 로또 1등 당첨자 수는 8405명 (total_1st_sales)

```
# 1091회 까지의 로또 총 판매량
total_sales = lotto_data_table['판매 로또 수'].sum()
total_sales
```

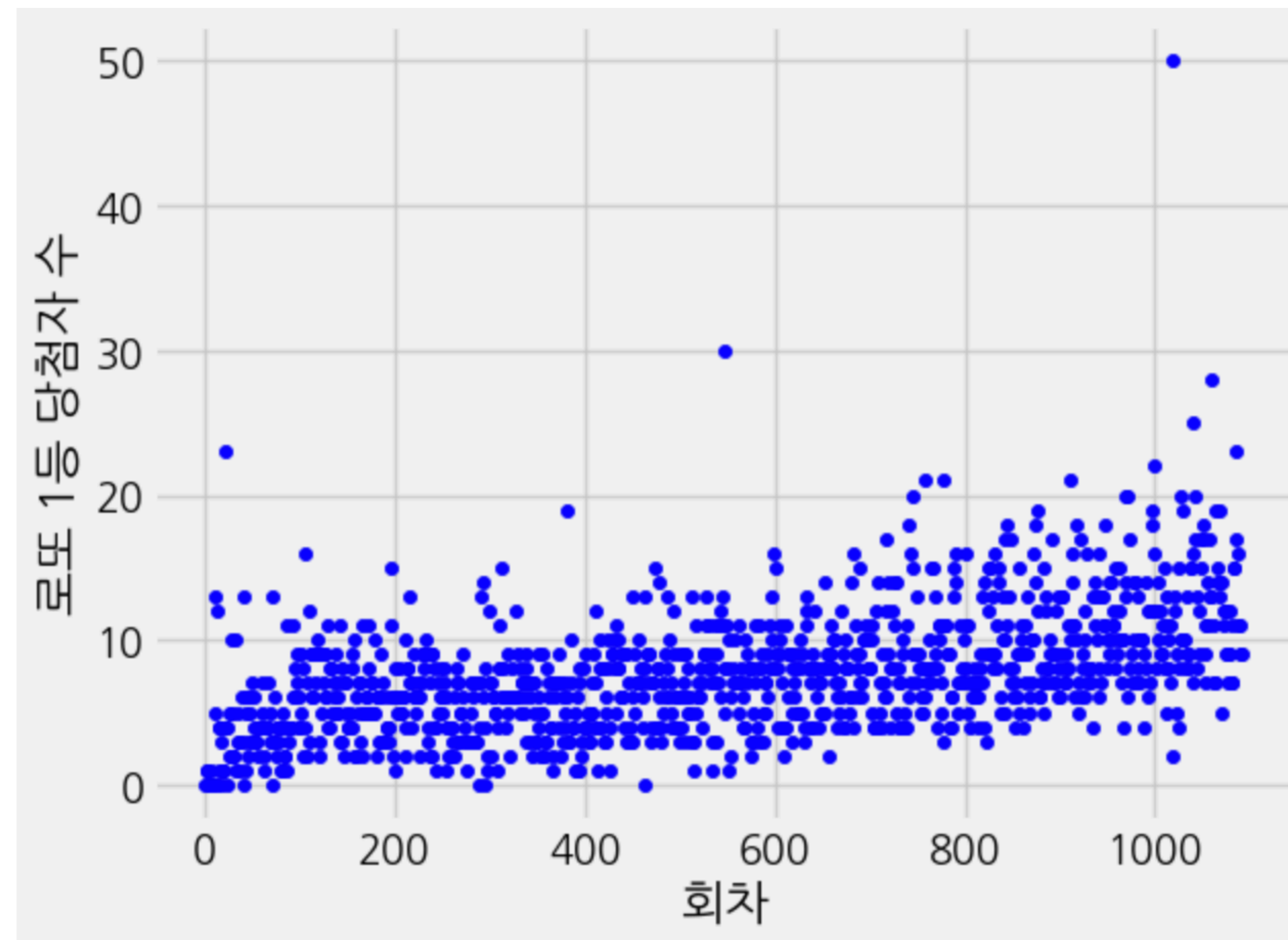
69274867722

```
# 1091회 까지의 로또 1등 당첨자 수
total_1st_sales = lotto_data_table['로또 1등 당첨자 수'].sum()
total_1st_sales
```

8405

실제 로또 회차별 1등 당첨자 수

- 대부분 고르게 분포 되어 있으나, '1019회차' 에서 특이점 발생
- 여태 많아봐야 2~30명이던 1등 당첨자가 50명? → 로또 조작설 제기



로또 당첨이 조작될 수 있는가?

- 실제로 1019회 직전 회차(1018회) 판매 금액은 약 1011억, 하지만 1019회 판매 금액은 약 1028억으로 1.63% 증가
- 반면에 1등 당첨자 수는 2건에서 50건으로 25배 상승
- 1등 당첨 50건 중 수동 추첨 42건, 자동 6건, 반자동 2건
- 1019회차 당첨 번호가 과거 당첨 번호로 자주 등장했던 숫자?
- **시뮬레이션의 필요성** 제기!

로또 시뮬레이션 프로세스

- 파이썬으로 작성된 'Lottery.py' 로 실제 로또 결과 데이터 (csv) 파일을 읽음
- c언어로 작성된 'simulation.c' 파일로 데이터를 전송함 (연산속도 📈)
- 과거 로또 회차별 당첨 번호와 랜덤 생성된 값을 비교 후 일치하면 1등 당첨 카운트 증가
- 1091회차 까지 진행 후 각 회차별 당첨 카운트를 다시 파이썬 파일로 전송
- 파이썬에서 최종적으로 각 회차별 시뮬레이션 결과를 csv 파일로 저장

Lottery.py 로직 설명

- Csv 파일로 부터 읽어온 데이터를 회차 별로 분리(= iterrows() 사용)
- 컴파일 된 simulation 실행 파일로 회차별 정보를 넘겨줌
- Simulation 실행 결과로 회차별 시뮬레이션 결과 1등 당첨자 수를 반환함
- 반환값을 딕셔너리 형태로 저장한 뒤 1091회차까지 수집이 완료되면 csv 파일로 저장
- 실행 결과 생성 파일 → 'SimulationResult.csv'

simulation.c 로직 설명

- 중복을 허용하지 않고 6개의 난수를 생성 후 'Quick sort' 를 이용하여 **오름차순 정렬**
- 실제 1등 당첨번호 배열과 랜덤으로 생성한 번호 배열이 **일치**하면 1등 당첨 **카운트 증가**
- Gcc -o simulation simulation.c 를 통해 컴파일

```
// 오름 차순 정렬
qsort(my_num, 6, sizeof(my_num[0]), compare);
same = is_equal(win_num, my_num, 6);
```

```
int compare(const void *a , const void *b)
{
    if( *(int*)a > *(int*)b )
        return 1;

    else if( *(int*)a < *(int*)b )
        return -1;

    else
        return 0;
}

int is_equal(int* a, int* b, int size)
{
    for (int i = 0; i < size; i++){
        if (a[i] != b[i])
            return 0;
    }
    return 1;
}
```

로또 시뮬레이션 결과

https://colab.research.google.com/drive/1RGvsnxfbCtHBC2H4duD4_sIT-AJIsKSN#scrollTo=SuRVVxcZij1b

- 앞의 두 코드 실행 결과 생성된 csv 파일을 읽어와 테이블 생성 (simulation_lotto)

```
simulation_lotto = Table.read_table('SimulationResult.csv')
simulation_lotto
```

drwNo	Win
-------	-----

1	1
---	---

2	0
---	---

3	1
---	---

4	0
---	---

5	0
---	---

6	1
---	---

7	1
---	---

8	1
---	---

9	4
---	---



10	12
----	----

... (1081 rows omitted)

로또 시뮬레이션 결과

- 판다스 데이터 프레임으로 변환 (simulation_lotto_data_table)

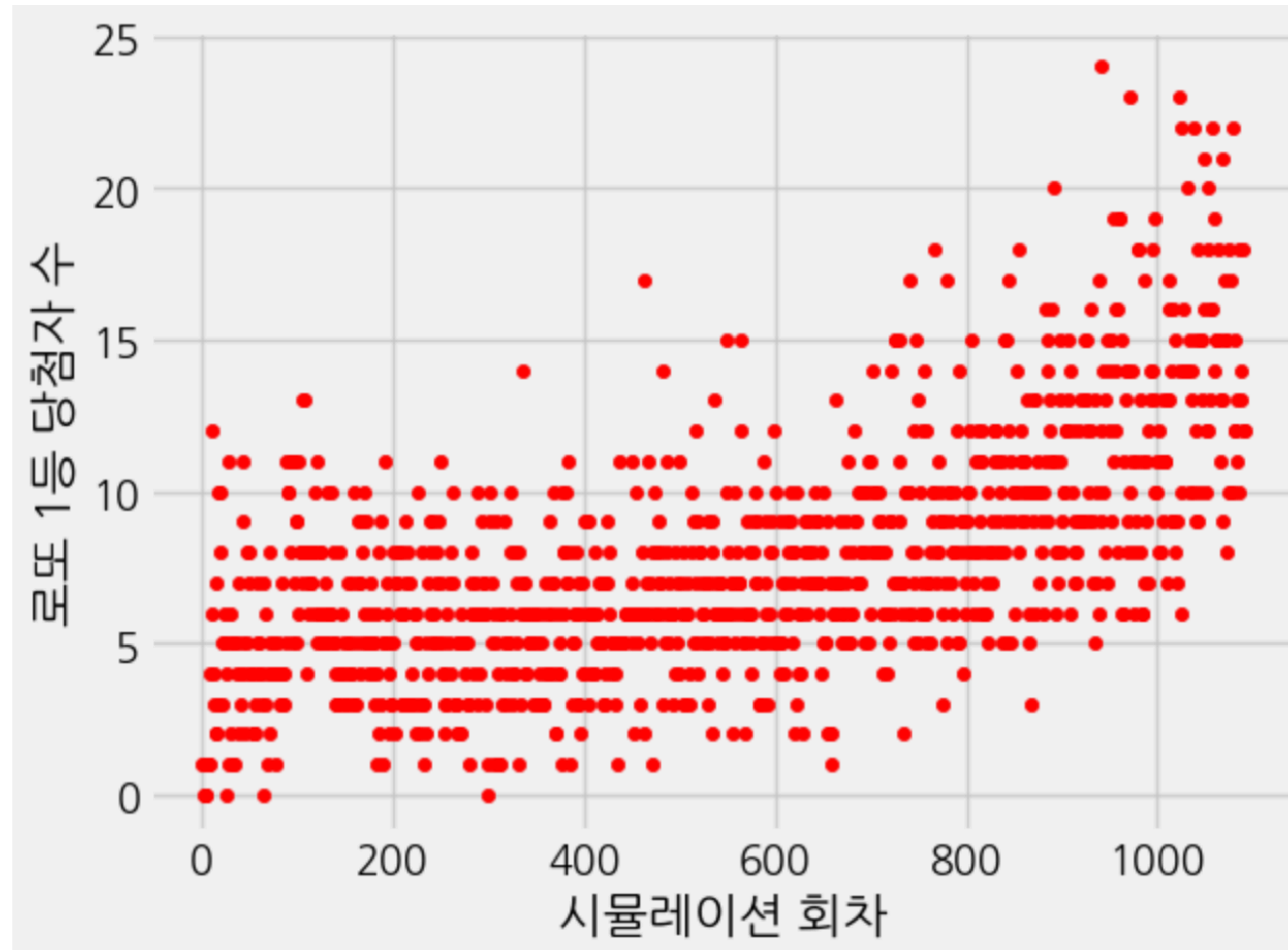
```
simulation_lotto_data_table = pd.DataFrame({'시뮬레이션 회차' : simulation_lotto.column('drwNo'), '로또 1등 당첨자 수' : simulation_lotto.column('Win')})
simulation_lotto_data_table
```

시뮬레이션 회차 로또 1등 당첨자 수			
0	1	1	
1	2	0	
2	3	1	
3	4	0	
4	5	0	
...	
1086	1087	13	
1087	1088	14	
1088	1089	18	
1089	1090	12	
1090	1091	12	

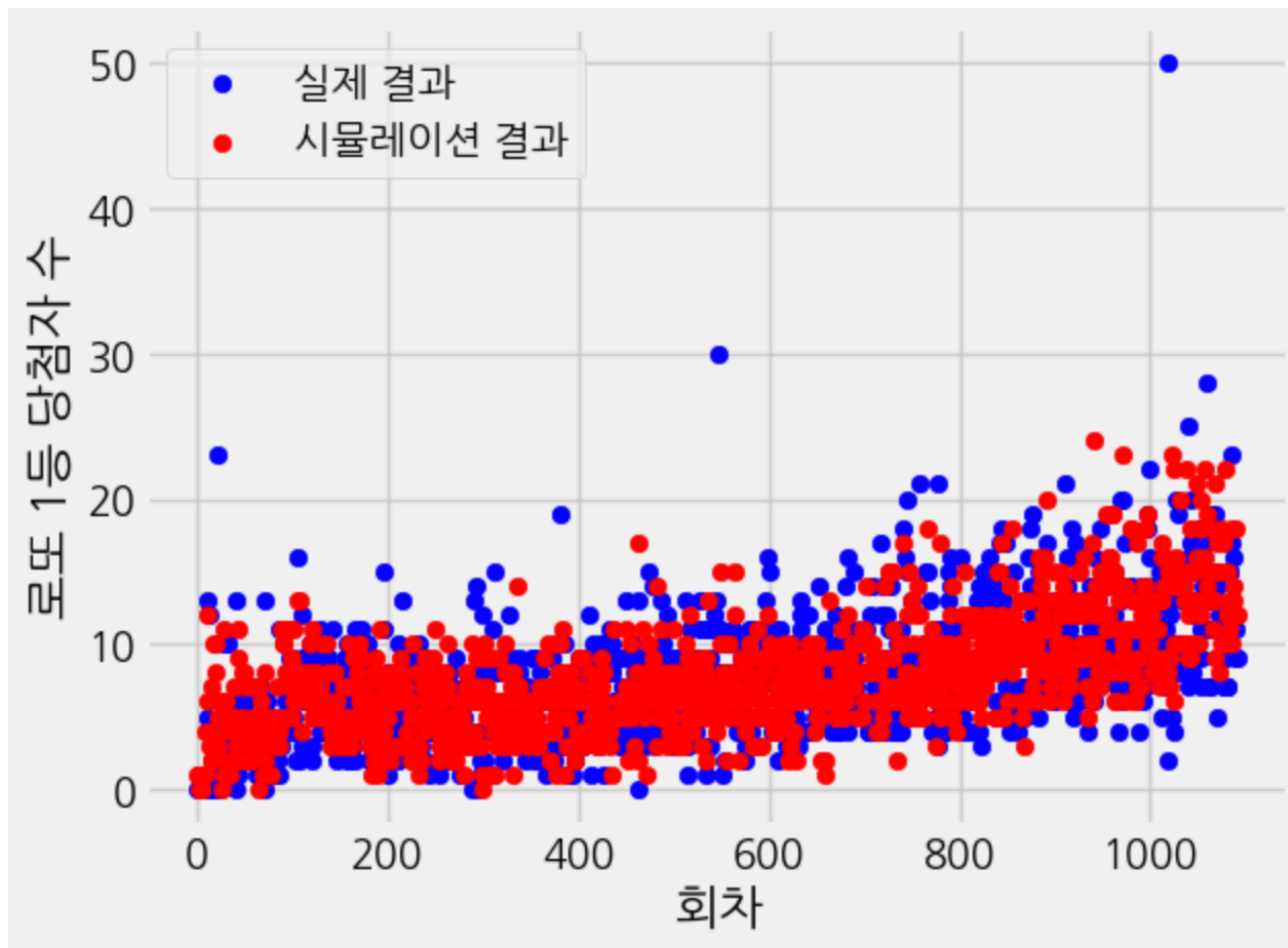
1091 rows × 2 columns

로또 시뮬레이션 결과

- 시뮬레이션 결과 회차별 로또 1등 당첨자 수



실제 데이터와 시뮬레이션 결과와 비교



로또 조작설에 대한 의견

- 매 회차 꾸준히 1등 당첨자가 나오는 이유는 나올 수 있는 로또 조합번호의 모든 경우의 수보다 더 많은 로또가 팔리고 있기 때문이다.
- 단순히 생각해도 평균적으로 게임당 천원, 매주 5500만 게임이 시행되는데 800만분의 1정도의 당첨 확률을 고려할 때, 회차마다 평균 7명의 당첨자가 발생하는 것은 놀랄 일이 아니다.
- 실제로 1등 당첨자가 없던 회차도 있었으므로, 평균보다 많은 1등 당첨자 수가 생길 수도 있다. 시뮬레이션 결과에서도 평균치보다 유독 많은 당첨자가 나온 회차가 존재한다.
- 번외로, 로또 1등 당첨 번호 배열에 자주 등장하는 숫자들이 있는데 그냥 우연의 일치이다.
- 나의 의견은 ‘로또는 조작되고 있지 않다.’ 이다.