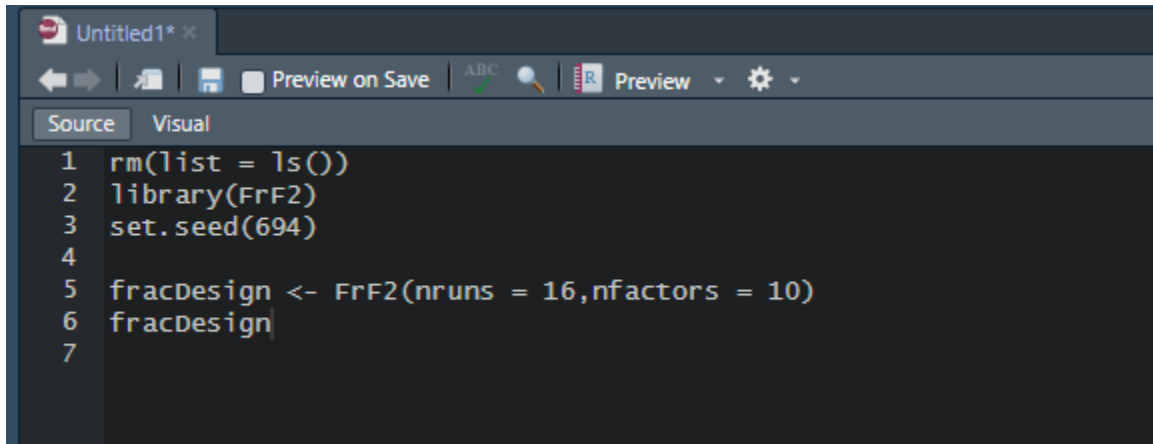


Question 12.1

My family and I are deciding where we want to host Thanksgiving for the holidays. My family lives all across the northern east coast with most of us scattered around the tri-state area. By using A-B testing, we can take a survey of everyone who is free on Thanksgiving holiday and who would go to whose house if they hosted. In this way we can determine whoever has the greatest amounts of people going to an available host, and thus determine the best option for the majority of the family to get together for Thanksgiving holiday.

Question 12.2

In Figures 1 & 2 you will find the code used to generate the 16 different houses and the features (labeled A-K) that should be associated with each listing. There is not much to say about this solution as it is simply just an output of the FrF2 function, and it does not provide much insight without a detailed dataset to analyze.

A screenshot of the RStudio code editor. The window title is 'Untitled1*'. The toolbar shows navigation icons, a 'Preview on Save' button, and a 'Preview' button. The 'Source' tab is active, displaying the following R code:

```
1 rm(list = ls())
2 library(FrF2)
3 set.seed(694)
4
5 fracDesign <- FrF2(nruns = 16, nfeatures = 10)
6 fracDesign
7
```

Figure 1. This is the code used to generate the 16 different houses and the features associated with each one as seen in the figure below.

```

> fracDesign <- FrF2(nruns = 16,nfactors = 10)
> fracDesign
  A  B  C  D  E  F  G  H  J  K
1   1  1 -1  1  1 -1 -1  1 -1 -1
2  -1 -1  1 -1  1 -1 -1  1  1 -1
3   1  1 -1 -1  1 -1 -1 -1  1  1
4  -1 -1  1  1  1 -1 -1 -1 -1  1
5  -1 -1 -1 -1  1  1  1  1 -1  1
6  -1  1  1  1 -1 -1  1 -1  1 -1
7  -1  1  1 -1 -1 -1  1  1 -1  1
8   1 -1  1 -1 -1  1 -1 -1  1  1
9   1  1  1 -1  1  1  1 -1 -1 -1
10  1 -1 -1  1 -1 -1  1  1  1  1
11 -1  1 -1  1 -1  1 -1 -1 -1  1
12  1  1  1  1  1  1  1  1  1  1
13 -1 -1 -1  1  1  1  1 -1  1 -1
14 -1  1 -1 -1 -1  1 -1  1  1 -1
15  1 -1  1  1 -1  1 -1  1 -1 -1
16  1 -1 -1 -1 -1 -1  1 -1 -1 -1
class=design, type= FrF2

```

Figure 2. This is the output of the code in Figure 1. As stated in the HW preview, 1 means “include” and -1 means “not include.”

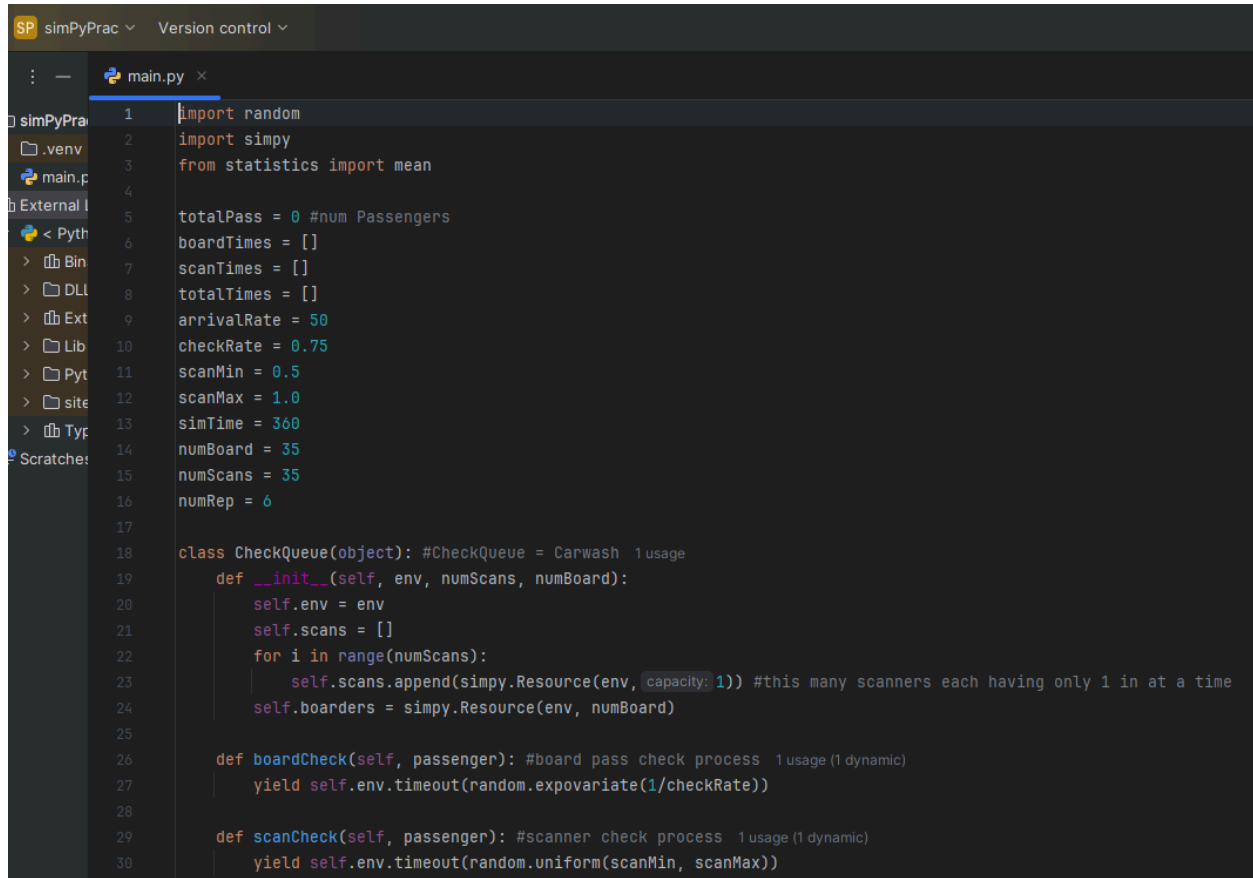
Question 13.1

- a) Binomial
 - i) An example of data I would expect the Binomial distribution to follow is a dataset such as the number of spam emails per day. This has the properties of “yes” or “no” that the binomial distribution requires (i.e. is it spam or not).
- b) Geometric
 - i) An example of data I would expect the Geometric distribution to follow is a dataset such as how many kicks would a soccer ball take before it becomes flat. This has the property of the number of tries between failures that Dr. Sokol mentioned in the lecture.
- c) Poisson
 - i) An example of data I would expect the Poisson distribution to follow is a dataset such as the number of network failures per week. This has the properties of the Poisson distribution as the events occur independently and the possible events of the network can not occur at the same time (i.e. working or failing)
- d) Exponential
 - i) An example of data I would expect the Exponential distribution to follow is a dataset such as the time between volcanic eruptions. This has the properties of the time we must wait for until a certain event (e.g. volcanic eruptions) occurs.
- e) Weibull
 - i) An example of data I would expect the Weibull distribution to follow is a dataset such as how long would it take for a soccer ball to go flat. This has the property of time between failures that Dr. Sokol mentioned in the lecture.

Question 13.2

Using the SimPy module in Python, I simulated a simplified version of the time spent waiting at an airport security line, and the time it would take for that wait to be cut down to at most 15 minutes by modifying the amount of scanners/ID checkers. The process for airline security arrivals follows a Poisson distribution with λ being equal to 50 to simulate 50 passengers per minute. Given a mean interval rate of 0.20 (i.e. $1/\lambda$) for the time it would take to get through the ID/boarding pass check in, and the TSA agents following an exponential distribution at 0.75 minutes/person, I deduced that the amount of scanners and ID checkers had to be at least 35 each for the wait time to be less than 15 minutes. The following figures and their captions go into more detail about the methodology and the code used to find this value. However with that being said, the main method for determining the wait time was done by incrementing the number of scanners and the number of checkers by 5, starting at 5. Additionally, one problem I have encountered with this code was that some of the values for calculating the wait time (total time in process - time spent checking - time spent scanning) ended up being negative. Although these values were extremely close to 0 (e.g. 1.324×10^{-16}), I could not find a proper reason as to why the values were negative. That should not be possible as

the passenger function takes the start time before beginning any scanning/checking, and taking the end time after all of the processes have been completed. With these values being so close to 0, perhaps this could be a rounding issue as the values are on the -16th order, and python rounding issues can be prevalent due to the IEEE method of storing floats. Regardless, the final numbers do fall in line with what would typically be expected of an airport to reduce waiting times, and the decrease in number of scanners/checkers does proportionally increase the wait times.



```
SP simPyPrac Version control
main.py x
1 import random
2 import simpy
3 from statistics import mean
4
5 totalPass = 0 #num Passengers
6 boardTimes = []
7 scanTimes = []
8 totalTimes = []
9 arrivalRate = 50
10 checkRate = 0.75
11 scanMin = 0.5
12 scanMax = 1.0
13 simTime = 360
14 numBoard = 35
15 numScans = 35
16 numRep = 6
17
18 class CheckQueue(object): #CheckQueue = Carwash 1 usage
19     def __init__(self, env, numScans, numBoard):
20         self.env = env
21         self.scans = []
22         for i in range(numScans):
23             self.scans.append(simpy.Resource(env, capacity=1)) #this many scanners each having only 1 in at a time
24         self.boarders = simpy.Resource(env, numBoard)
25
26     def boardCheck(self, passenger): #board pass check process 1 usage (1 dynamic)
27         yield self.env.timeout(random.expovariate(1/checkRate))
28
29     def scanCheck(self, passenger): #scanner check process 1 usage (1 dynamic)
30         yield self.env.timeout(random.uniform(scanMin, scanMax))
```

Figure 3. This figure is the global variables and the class used to initiate the process of checking/scanning the passengers. The variables are aptly named to the significance they have in the real life application. In addition, the variables numBoard and numScans were the values incremented by 5 until a successful solution was reached.

```

32 def passenger(env, name, CheckQ, numScans): 1 usage
33     global boardTimes
34     global scanTimes
35     global totalTimes
36
37     arrivalTime = env.now
38     boardTime = 0 #total time for boarding
39     scanTime = 0 #total time spent getting scanning
40
41
42     #check boarding time
43     with CheckQ.boarders.request() as request:
44         yield request
45         btimeIn = env.now #time at start of boarding check
46         yield env.process(CheckQ.boardCheck(name))
47         btimeOut = env.now #time at end of boarding check
48         boardTime = btimeOut - btimeIn #time spent boarding
49
50     #pick shortest queue for scanning
51     minScan = 0
52     for i in range(1, numScans):
53         if (len(CheckQ.scans[i].queue) < len(CheckQ.scans[minScan].queue)):
54             minScan = i
55
56     #check scanning time
57     with CheckQ.scans[minScan].request() as request:
58         yield request
59         timeIn = env.now #time at start of boarding check
60         yield env.process(CheckQ.scanCheck(name))
61         timeOut = env.now #time at end of boarding check
62         scanTime = timeOut - timeIn #time spent scanning
63
64     leavingTime = env.now #time at leaving the entire process
65     boardTimes.append(boardTime)
66     scanTimes.append(scanTime)
67     totalTimes.append(leavingTime-arrivalTime)
68
69     # print('arrival Time: {}, board Time: {}, scan Time: {}, leaving Time: {}'.format(arrivalTime, boardTime, scanTime, leavingTime))

```

Figure 4. This figure is the passenger function used to actually process the boarding/checking system. It is in this block of code that the difference of the total time and the board+scanning value was rarely negative. Based on the layout of code this should be impossible, but perhaps the rounding values is what could be causing this anomaly.

```

70
71 ✓ def passengerArrival(env, CheckQ, arrivalRate, numScans): 1 usage
72     global totalPass
73
74     #internal tracker for each rep
75     numPass = 0
76
77     while True:
78         yield env.timeout(random.expovariate(arrivalRate)) #arrival time
79
80         numPass+=1
81         totalPass+=1
82
83         #passenger creation after arrival time is initiated
84         env.process(passenger(env, name=f'pass {numPass}', CheckQ, numScans))
85
86
87
88 ✓ for i in range(numRep):
89
90     env = simpy.Environment()
91     checkQ = CheckQueue(env, numScans=numScans, numBoard=numBoard)
92
93     #start simulation by pass arriving
94     env.process(passengerArrival(env, checkQ, arrivalRate=arrivalRate, numScans=numScans))
95
96     #run sim for 6 hrs
97     env.run(simTime)
98
99 ✓ # print(boardTimes[:5])
100 # print('-----')
101 # print(scanTimes[:5])
102 # print('-----')
103 # print(totalTimes[:5])
104
105 totalWaitTime = [] #each index is a passenger and their times boarding/scanning/waiting
106 for i in range(len(boardTimes)):
107     totalWaitTime.append((totalTimes[i] - scanTimes[i] - boardTimes[i]))
108
109 print('The average wait time is: {}'.format(mean(totalWaitTime)))

```

Figure 5. This final part of the codebase begins the arrival of the passenger and starts the process of determining their time in the system.

```
The average wait time is: 14.699692043655775
```

```
Process finished with exit code 0
```

Figure 6. This is the output after running the above program.