HW 9

12.1

In my job in advertising we often design experiments to understand the true effectiveness of a media campaign. For example, we might design a match market test where we divide the US up into Designated Metro Area (DMAs) and then gather client data for the success metric we care about for each DMA. If we care about sales, we will want to understand historical sales in the dmas. We then randomize all of the DMAs into two cells: one for the control where we make no change to the advertisers media, and one for the control where we introduce a singular change. There are many techniques to ensure the randomization is done well. We then run media in the test DMAs and analyze results at the end of the test. We are hoping there is a statistically significant lift in the test DMAs and can thus conclude that the advertising media we introduced into the test DMAs caused the lift in sales.

12.2

If you have 10 features, full factorial design implies that it would be 2^10 or 1024 different variations. If you want to get down to 16, you would solve the equation 2^(10-x) = 16. In this case, x would equal 6, and we are doing 1/64 factorial design. When you are doing this reduction to only 16 different factors, you want to make sure that all of the factors are independent. For example, if one of your factors is having a driveway and another is having a garage, these two might not be totally independent, since it doesn't make too much sense to have a garage without a driveway. You would also want to make sure you eliminate any variables that are absolutely must haves. For example, if some of the features are things like a kitchen, it might make no sense to show a version of a house with no kitchen!

Below is my code and output. Note that I put column names as letters for ease of printout

```
install.packages("FrF2")
library(FrF2)

factors <- 10
runs <- 16
design <- FrF2(nfactors = factors, nruns = runs)

custom_colnames <- c("(no large yard)", "(no solar roof)",
                     "(no pool)", "(no garage)", "(no fireplace)",
                     "(no basement)", "(no air conditioning)",
                     "(no new appliances)", "(no updated kitchen)",
                     "(no updated bathroom)")

colnames(design) <-custom_colnames
design
```

```
    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
```

13.1

Binomial: The number of successful attempts in a fixed number of trials, where each trial has a known probability of success. For example, the number of heads obtained when flipping a coin 10 times, where the probability of heads is 0.5.

Geometric: The number of trials needed to obtain the first success in a sequence of independent trials, where each trial has a known probability of success. For example, the number of times a hockey player needs to attempt a slapshot to record a speed of over 100mph, where the probability of going over 100mph is 30%.

Poisson: The number of rare events occurring in a fixed amount of time or space, where the events occur independently of each other and the rate of occurrence is constant. For example, the number of typos on a page of a book, where the rate of typos is 0.1 per page.

Exponential: The time between occurrences of rare events that follow a Poisson process, where the rate of occurrence is constant. For example, the time between successive earthquakes on a particular fault line, where the rate of occurrence is 0.5 per year.

Weibull: The time until failure of a device or system, where the failure rate follows a Weibull distribution. For example, the time until failure of a light bulb, where the failure rate increases over time due to wear and tear.

13.2

In [5]:
```
import simpy
import random
```

In [133]:
```
# Constants
ARRIVAL_RATE = 50 # passengers per minute
ID_CHECK_SERVERS = 2
ID_CHECK_TIME = 0.75 # mean time per server
MIN_PERSONAL_CHECK_TIME = 0.5
MAX_PERSONAL_CHECK_TIME = 1
PERSONAL_CHECK_QUEUES = 2
```

In [134]:
```
# Variables
wait_times = []
```

In [135]:
```
# Passenger process
def passenger(env, id_check_servers, personal_check_queues):
    # Arrival at ID check
    arrival_time = env.now
    with id_check_servers.request() as req:
        yield req
        id_check_time = random.expovariate(ID_CHECK_TIME)
        yield env.timeout(id_check_time)

    # Arrival at personal check
    personal_check_queue_lengths = [queue.count for queue in personal_check_queues]
    shortest_queue = personal_check_queues[personal_check_queue_lengths.index(min(personal_check_queue_lengths))]
    with shortest_queue.request() as req:
        yield req
        personal_check_time = random.uniform(MIN_PERSONAL_CHECK_TIME, MAX_PERSONAL_CHECK_TIME)
        yield env.timeout(personal_check_time)

    # Departure from personal check
    wait_times.append(env.now - arrival_time)
```

```
# Environment setup
env = simpy.Environment()
```

In [136]:
```
# Resource setup
id_check_servers = simpy.Resource(env, capacity=ID_CHECK_SERVERS)
personal_check_queues = [simpy.Resource(env) for i in range(PERSONAL_CHECK_QUEU
```

In [137]:
```
# Passenger arrival process
def passenger_arrivals(env):
    while True:
        interarrival_time = random.expovariate(1/ARRIVAL_RATE)
        yield env.timeout(interarrival_time)
        env.process(passenger(env, id_check_servers, personal_check_queues))
```

In [138]:
```
# Start simulation
env.process(passenger_arrivals(env))
env.run(until=1000)
```

In [139]:
```
# Print results
avg_wait_time = sum(wait_times) / len(wait_times)
print("Average wait time: %.2f minutes" % avg_wait_time)
```

```
Average wait time: 2.09 minutes
```