Week 9 Homework - Part 1

Question 12.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a design of experiments approach would be appropriate.

Answer 12.1

A Mobile Applications development company might develop a new productivity application or health application with multiple features. If the company would like to know which features to be included for free and which ones should be provided as a part of a paid upgrade, a design of experiments approach could provide the required insights.
The experiment would have trial versions provided to potential users with different sets of features as paid and others as free. The objective is to see which features will encourage users to upgrade and which would not. Also the company would know which features are more important to invest in its development.

Question 12.2

To determine the value of 10 different yes/no features to the market value of a house (large yard, solar roof, etc.), a real estate agent plans to survey 50 potential buyers, showing a fictitious house with different combinations of features. To reduce the survey size, the agent wants to show just 16 fictitious houses.
Use R's FrF2 function (in the FrF2 package) to find a fractional factorial design for this experiment: what set of features should each of the 16 fictitious houses have? Note: the output of FrF2 is "1" (include) or "-1" (don't include) for each feature.

Answer 12.2

In [4]:

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

In [2]:

```
# Fix Seed Number
set.seed(0)
# Create Experiment using 16 runs (16 fictitious houses) and 10 factors (10 different ye
experiment <- FrF2(nruns = 16, nfactors = 10)
# Convert to Data Frame
experiment <- as.data.frame(experiment)
# Display Results of Each experiment (1 = included, -1 = Not included)
experiment
```

A data.frame: 16 × 10

	A	B	C	D	E	F	G	H	J	K
	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>	<fct>
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

Note that Features names are labeled A through K

In [3]:

```r
# Store factors in each fictitious house
for (i in seq(1,16)){
    features <- colnames(experiment)[experiment[i,]==1]
    cat("Experiment", i, "features included:", features, "\n")
}
```

```
Experiment 1 features included: A C D F H
Experiment 2 features included: D E F G J
Experiment 3 features included: A B E J K
Experiment 4 features included: B C G H K
Experiment 5 features included: E F G H K
Experiment 6 features included: A G
Experiment 7 features included: C D E K
Experiment 8 features included: B D F K
Experiment 9 features included: B F H J
Experiment 10 features included: A B C E F G
Experiment 11 features included: A B D E H
Experiment 12 features included: C E H J
Experiment 13 features included: A C F J K
Experiment 14 features included: B C D G J
Experiment 15 features included: A D G H J K
Experiment 16 features included: A B C D E F G H J K
```

Question 13.1

For each of the following distributions, give an example of data that you would expect to follow this distribution (besides the examples already discussed in class).
a. Binomial b. Geometric c. Poisson d. Exponential e. Weibull

Answer 13.1

a. Binomial Distribution:
Asking 100 of our Non-American classmates if they have ever been to Georgia. The response might have a binomial distribution.

b. Geometric Distribution:
The number of Petroleum Exploration wells that have to be drilled before a successful hydrocarbon discovery is made could follow a geometric distribution.

c. Poisson Distribution:
The number of expected customers that will arrive at the restaurant per day might have a poisson distribution.

d. Exponential Distribution:
The amount of time between traffic accidents on a certain road might have an exponential distribution

e. Weibull Distribution:
The amount of time before a sensor fails might follow a Weibull distribution. With $k>1$ the more time passes the more likely it is the sensor would fail assuming no manufacturing defects.

Week 9 Homework - Part 2

Question 13.2

In this problem you, can simulate a simplified airport security system at a busy airport. Passengers arrive according to a Poisson distribution with λ1 = 5 per minute (i.e., mean interarrival rate λ1 = 0.2 minutes) to the ID/boarding-pass check queue, where there are several servers who each have exponential service time with mean rate λ2 = 0.75 minutes. [Hint: model them as one block that has more than one resource.]
After that, the passengers are assigned to the shortest of the several personal-check queues, where they go through the personal scanner (time is uniformly distributed between 0.5 minutes and 1 minute).

Use the Arena software (PC users) or Python with SimPy (PC or Mac users) to build a simulation of the system, and then vary the number of ID/boarding-pass checkers and personal-check queues to determine how many are needed to keep average wait times below 15 minutes. [If you're using SimPy, or if you have access to a non-student version of Arena, you can use λ1 = 50 to simulate a busier airport.]

In [1]:

```
# import SimPy Package
import simpy
# import random package for distributions
import random
import numpy as np
# Package for plotting
%matplotlib inline
import matplotlib.pyplot as plt
```

Create Airport Class

In [2]:

```
random.seed(10)
class Airport(object):
    def __init__(self, env, num_servers, num_scanners):
        # assign environment as SimPy environment
        self.env = env

        # Create the Servers resource
        # Modelling servers as one block that has more than one resource
        self.servers = simpy.Resource(env, num_servers)

        # Create the Scanners resources
        # Note a list of Multiple blocks
        self.scanners = []
        for i in range(num_scanners):
            block = simpy.Resource(env, capacity = 1)
            self.scanners.append(block)

    # define the process of passing a server
    def pass_server(self, passenger):
        # estimate and return an iterable of the time needed to pass the servers
        yield self.env.timeout(np.random.exponential(servers_inv_rate))  # Exponential d

    # define the process of passing a server
    def pass_scanner(self, passenger):
        # estimate and return an iterable of the time needed to pass the scanners
        yield self.env.timeout(random.uniform(scan_min_time, scan_max_time))  # Uniform
```

Define go to Airport Function

In [3]:

```
def go_to_airport(env, passenger, airport):
    # Moviegoer arrives at the theater
    arrival_time = env.now

    # Simulate passing a server
    with airport.servers.request() as request:
        # wait for a server to become available if all are currently in use
        start_server_wait =  env.now
        yield request
        end_server_wait =  env.now
        # Pass the server
        yield env.process(airport.pass_server(passenger))
        exit_server_time = env.now
        servers_wait_time.append(end_server_wait-start_server_wait)

    # Simulate passing a Scanner
    # Finding shortest Queue Scanner
    min_q_scanner = 0
    for a_scanner in range(0, num_scanners):
        if len(airport.scanners[a_scanner].queue) < len(airport.scanners[min_q_scanner].
            min_q_scanner = a_scanner

    # Simulate passing the lowest queue scanner
    with airport.scanners[min_q_scanner].request() as request:
        # wait for a scanner to become available if all are currently in use
        start_scanner_wait =  env.now
        yield request
        end_scanner_wait =  env.now
        # Pass the scanner
        yield env.process(airport.pass_scanner(passenger))
        # Store scanner wait time
        exit_scan_time = env.now
        scanners_wait_time.append(end_scanner_wait-start_scanner_wait)

    # Passenger Passes all check points
    total_wait_times.append(env.now - arrival_time)
```

Define Run Simulation

In [4]:

```
random.seed(10)
def run_simulation(env, num_servers, num_scanners, arrival_rate):
    # Define Airport
    airport = Airport(env, num_servers, num_scanners)

    # initialize passenger
    global passenger
    passenger = 0

    while True:
        # Generate New passengers based on Poisson distribution
        new_passenger_arrival = np.random.poisson(arrival_rate)
        while new_passenger_arrival == 0:
            new_passenger_arrival = np.random.poisson(arrival_rate)
        yield env.timeout(1/new_passenger_arrival)
        passenger += 1
        env.process(go_to_airport(env, passenger, airport))
```

Define Inputs and run Base Case Simulation

In [5]:

```
# define input data
arrival_rate = 5  # passengers per minute
servers_inv_rate = 0.75  # minute per passenger
scan_min_time = 0.5  # minute per passenger
scan_max_time = 1.0  # minute per passenger

# Assumptions
num_servers = 4
num_scanners = 4
```

In [6]:

```
# empty lists to store results
total_wait_times = []
servers_wait_time = []
scanners_wait_time = []

# Set up the environment
env = simpy.Environment()
env.process(run_simulation(env, num_servers, num_scanners, arrival_rate))

# Run simulation for simulating 60 minutes for 24 hours (1 Day)
sim_duration = 60*24
env.run(until=sim_duration)

# calculate Mean results
avg_system_time = round(sum(total_wait_times)/len(total_wait_times), 3)
avg_servers_wait_time = round(sum(servers_wait_time)/len(servers_wait_time), 3)
avg_scanners_wait_time = round(sum(scanners_wait_time)/len(scanners_wait_time), 3)
total_avg_wait_time = round(avg_servers_wait_time + avg_scanners_wait_time, 3)

# get results
print("Number of Passengers", passenger, "Mean", round(passenger/sim_duration), "Passeng
print("Average Total System time:", avg_system_time, "mins")
print("Average Wait time for the Servers:", avg_servers_wait_time, "mins")
print("Average Wait time for the Scanners:", avg_scanners_wait_time, "mins")
print("Total Average Wait time:", total_avg_wait_time, "mins")
```

```
Number of Passengers 5573 Mean 4 Passengers/Minute
Average Total System time: 2.194 mins
Average Wait time for the Servers: 0.224 mins
Average Wait time for the Scanners: 0.463 mins
Total Average Wait time: 0.687 mins
```

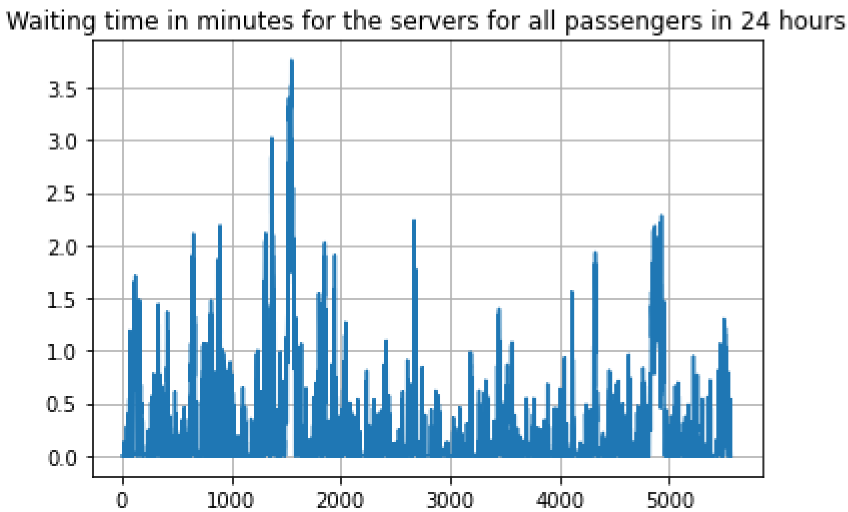
The results shown above are after a sensivity analysis process for a 1 day simulation model. The results are:

1. Optimum number of ID/boarding-pass check servers is between 3 and 4 (4 was used to avoid Max wait time exceeding 15 mins as well as different impacts of random seeds).
2. Optimum number of personal scanners is 3 and 4 (4 was used to avoid Max wait time exceeding 15 mins as well as different impacts of random seeds).

This reduces the average wait time to around 0.7 minutes with peak waiting times in both servers and scanners queues being around 1 minute to 3 minutes as shown in the plots below

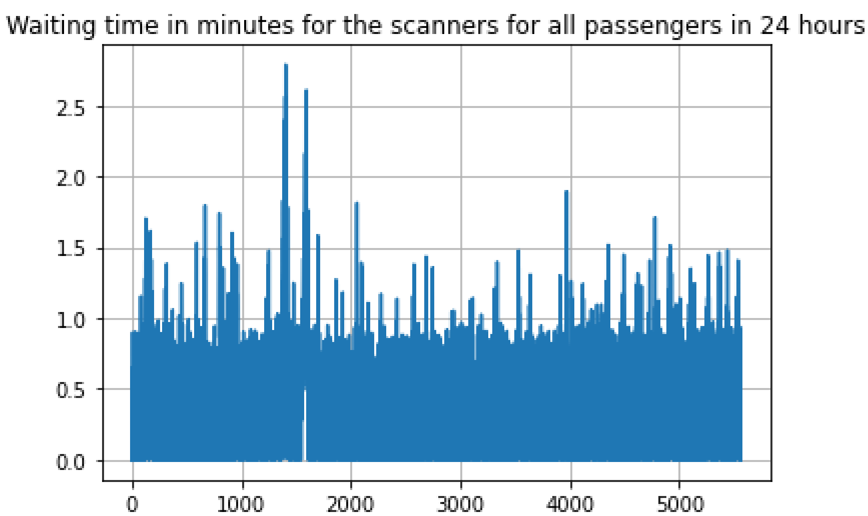
In [7]:

```
plt.plot(servers_wait_time)
plt.title("Waiting time in minutes for the servers for all passengers in 24 hours")
plt.grid()
plt.show()
```



In [8]:

```
plt.plot(scanners_wait_time)
plt.title("Waiting time in minutes for the scanners for all passengers in 24 hours")
plt.grid()
plt.show()
```



Repeating the process for an airport with an arrival rate of 50 passengers per minute.

In [9]:

```
# define input data
arrival_rate = 50  # passengers per minute
servers_inv_rate = 0.75  # minute per passenger
scan_min_time = 0.5  # minute per passenger
scan_max_time = 1.0  # minute per passenger

# Assumptions
num_servers = 37
num_scanners = 37
```

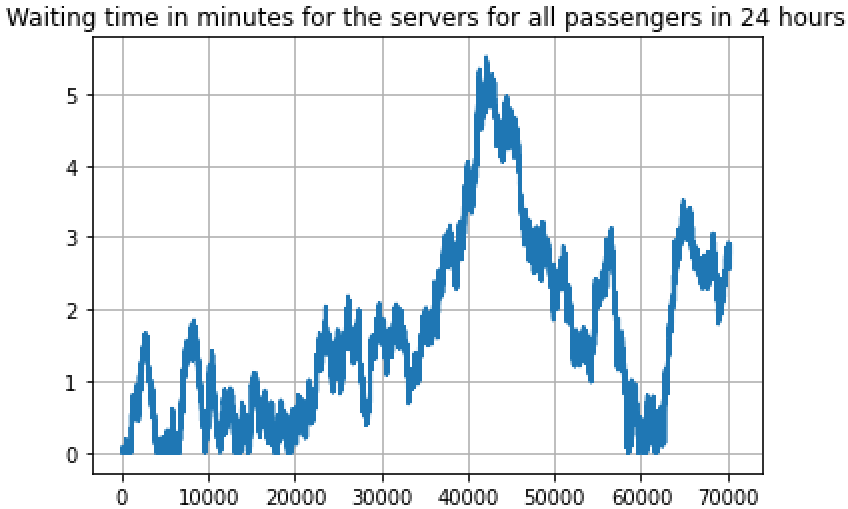
In [10]:

```
# empty lists to store results
total_wait_times = []
servers_wait_time = []
scanners_wait_time = []

# Set up the environment
env = simpy.Environment()
env.process(run_simulation(env, num_servers, num_scanners, arrival_rate))

# Run simulation for simulating 60 minutes for 24 hours (1 Day)
sim_duration = 60*24
env.run(until=sim_duration)

# calculate Mean results
avg_system_time = round(sum(total_wait_times)/len(total_wait_times), 3)
avg_servers_wait_time = round(sum(servers_wait_time)/len(servers_wait_time), 3)
avg_scanners_wait_time = round(sum(scanners_wait_time)/len(scanners_wait_time), 3)
total_avg_wait_time = round(avg_servers_wait_time + avg_scanners_wait_time, 3)

# get results
print("Number of Passengers", passenger, "Mean", round(passenger/sim_duration), "Passeng
print("Average Total System time:", avg_system_time, "mins")
print("Average Wait time for the Servers:", avg_servers_wait_time, "mins")
print("Average Wait time for the Scanners:", avg_scanners_wait_time, "mins")
print("Total Average Wait time:", total_avg_wait_time, "mins")
```

```
Number of Passengers 70550 Mean 49 Passengers/Minute
Average Total System time: 4.528 mins
Average Wait time for the Servers: 1.68 mins
Average Wait time for the Scanners: 1.346 mins
Total Average Wait time: 3.026 mins
```

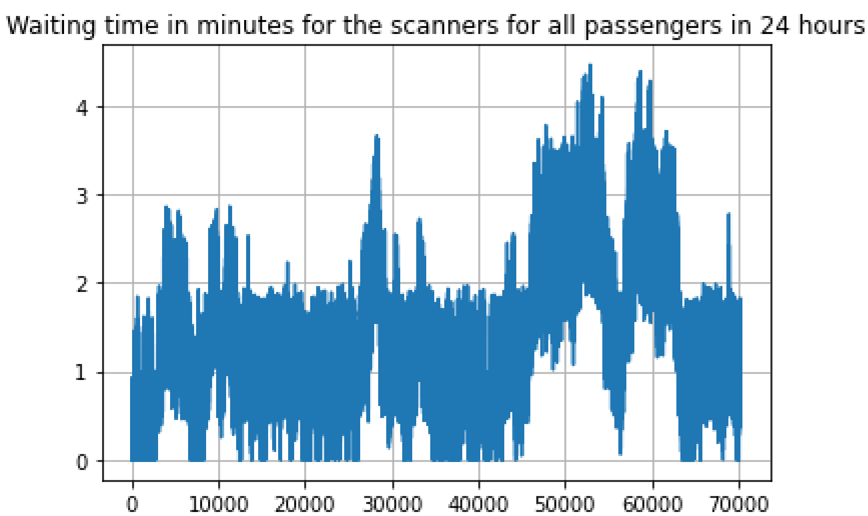
In [11]:

```
plt.plot(servers_wait_time)
plt.title("Waiting time in minutes for the servers for all passengers in 24 hours")
plt.grid()
plt.show()
```



In [12]:

```
plt.plot(scanners_wait_time)
plt.title("Waiting time in minutes for the scanners for all passengers in 24 hours")
plt.grid()
plt.show()
```



The results shown above are after a sensivity analysis process for a 1 day simulation model. The results are:

1. Optimum number of ID/boarding-pass check servers is 37
2. Optimum number of personal scanners is 37

conclusion, simulating a busier airport (Passengers arrive according to a Poisson distribution with λ1 = 50 per minute) 10X the first airport arrival's rate, lead to increasing the number of servers & Scanners needed by 10 times.

Some Notes on the results:

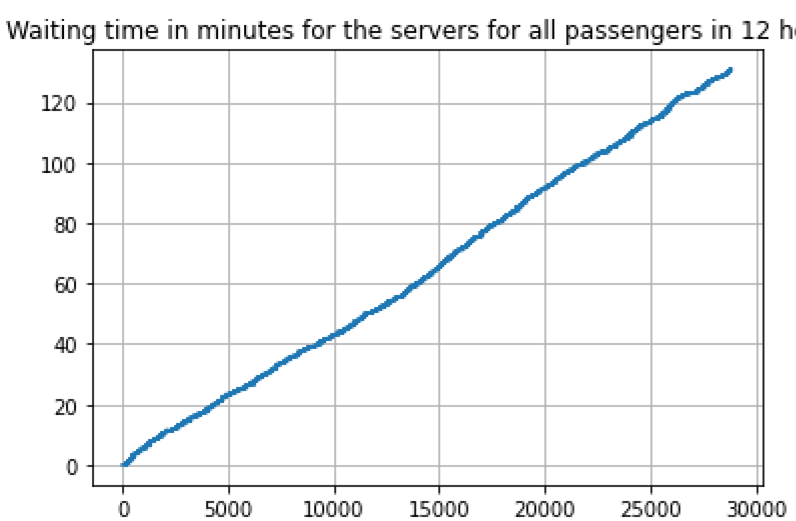
1. Reducing the Number of Scanners or Servers to 30 results in a continous increase in waiting time with increasing number of passengers due to overflow compared to system capacity.
2. Working on small time frames e.g. 1 hour would lead to misleading results as the average might still be less than 15 minutes however there would be a continous increase in the system that was not captured. (Note at 1 hour +/- 2500 passenger should pass on the graph below)

In [13]:

```
# Assumptions
num_servers = 30
num_scanners = 37

# empty lists to store results
total_wait_times = []
servers_wait_time = []
scanners_wait_time = []

# Set up the environment
env = simpy.Environment()
env.process(run_simulation(env, num_servers, num_scanners, arrival_rate))

# Run simulation for simulating 60 minutes for 12 hours
sim_duration = 60*12
env.run(until=sim_duration)
```

In [14]:

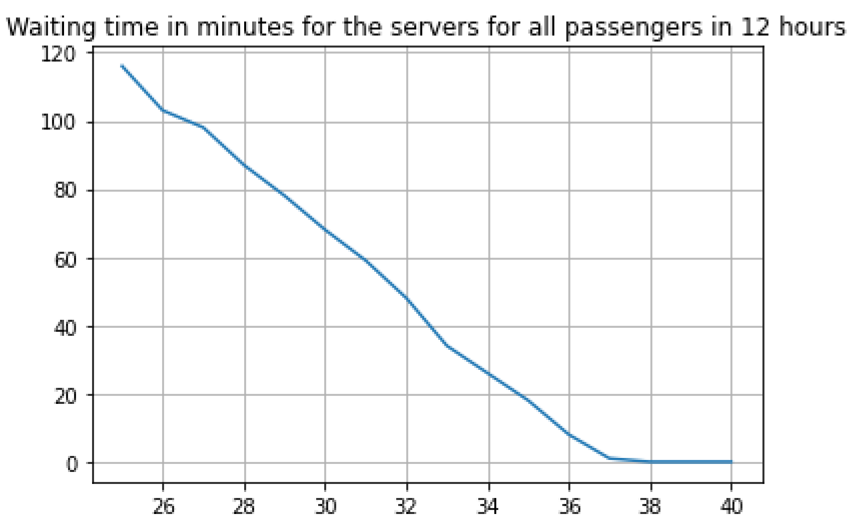
```
plt.plot(servers_wait_time)
plt.title("Waiting time in minutes for the servers for all passengers in 12 hours")
plt.grid()
plt.show()
```



Sensitivity Analysis on the number of Servers

In [15]:

```
# initialize Assumptions
num_servers = 50
num_scanners = 50

# Vector to store results
avg_servers_wait_time_list = np.repeat(0,16,axis=0)

# Loop with different number of Servers
for i in range(25, 40):
    num_servers = i
    # empty lists to store results
    total_wait_times = []
    servers_wait_time = []
    scanners_wait_time = []

    # Set up the environment
    env = simpy.Environment()
    env.process(run_simulation(env, num_servers, num_scanners, arrival_rate))

    # Run simulation for simulating 60 minutes for 12 hours
    sim_duration = 60*12
    env.run(until=sim_duration)

    # Store Results
    avg_servers_wait_time_list[i-25] = round(sum(servers_wait_time)/len(servers_wait_tir

# plot results
plt.plot(list(range(25,41)), avg_servers_wait_time_list)
plt.title("Waiting time in minutes for the servers for all passengers in 12 hours")
plt.grid()
plt.show()
```



Sensitivity Analysis on the number of Scanners

In [16]:

```
# initialize Assumptions
num_servers = 100
num_scanners = 100

# Vector to store results
avg_scanners_wait_time_list = np.repeat(0,16,axis=0)

# Loop with different number of Scanners
for i in range(25,40):
    num_scanners = i
    # empty lists to store results
    total_wait_times = []
    servers_wait_time = []
    scanners_wait_time = []

    # Set up the environment
    env = simpy.Environment()
    env.process(run_simulation(env, num_servers, num_scanners, arrival_rate))

    # Run simulation for simulating 60 minutes for 12 hours
    sim_duration = 60*12
    env.run(until=sim_duration)

    # Store Results
    avg_scanners_wait_time_list[i-25] = round(sum(scanners_wait_time)/len(scanners_wait_

# plot results
plt.plot(list(range(25,41)), avg_scanners_wait_time_list)
plt.title("Waiting time in minutes for the servers for all passengers in 12 hours")
plt.grid()
plt.show()
```