1 Prescriptive Modeling - Linear Programming

1.1 Transportation Problem

```
# We are using the JuMP Package for Optimization
using JuMP

# Start with a new Model
m = Model()

# List of Plants
plants = ["P1", "P2", "P3"]

# List of Regions
regions = ["D1", "D2"]

# Variables for the Linear Program
@variable(m, x[plants, regions] >= 0);

# Capacity for each plant
capacity = Dict([("P1", 4000), ("P2", 4000), ("P3", 4000)])

# Demand for each region
demand = Dict([("D1", 4000), ("D2", 4000)])


# Transportation costs from plants to regions per unit of supply
costs = Dict([("P1", Dict([("D1", 22), ("D2", 22)])),
              ("P2", Dict([("D2", 22), ("D3", 22)])),
              ("P3", Dict([("D1", 22), ("D3", 22)]))])

# Objective Function Definition - Minimize Transportation cost
@objective(m, Min,
sum{x[i,j]*costs[i][j], i=plants, j= regions})

# Demand Constraint
for j in regions
       @constraint(m, sum{x[i,j], i=plants} >= demand[j])
end

# Supply Constraint
for i in plants
       @constraint(m, sum{x[i,j], j=regions} <= capacity[i])
end

print(m)

# Solve
solve(m)
getvalue(x)
```

```
print(m.objVal)
```

Listing 1: Transportation Problem

1.2 Transshipment Problem

```
using JuMP

m = Model()

plants = ["p1", "p2"]
dcs = ["a","b"]
regions = ["r1", "r2", "r3"]

plants_dc = Dict([("plant1", Dict([("a", 44), ("b", 55)])),
("plant2", Dict([("a", 43), ("b", 11)]))])
dc_regions = Dict([("a", Dict([("r1", 55), ("r2", 55), ("r3"
    , 44)])),
("b", Dict([("r1",33), ("r2", 44), ("r3", 55)]))])

supply = Dict([("plant1", 222), ("plant2", 444)])
demand = Dict([("r1", 444), ("r2", 444), ("r3", 444)])

# From plants to DCs
@variable(m, x[plants, dcs] >= 0)
# From DCs to regions
@variable(m, y[dcs, regions] >= 0)

# Include both costs
@objective(m, Min, sum{x[i,j]*plants_dc[i][j], i=plants, j=
    dcs} + sum{y[i,j]*dc_regions[i][j],i=dcs,j=regions})

# Supply Constraint
for i in plants
  @constraint(m, sum{x[i,j], j=dcs} <= supply[i])
end

# Demand Constraint
for j in regions
  @constraint(m, sum{y[i,j], i=dcs} >= demand[j])
end

# Conservation Constraint
for i in dcs
  @constraint(m, sum{x[k,i], k=plants} - sum{y[i,j],j=
     regions} == 0)
end

# Solve
solve(m)
getvalue(x)
getvalue(y)
print(m.objVal)
```

Listing 2: Transshipment Problem

1.3 Fixed costs for setting up DCs

```
using JuMP

m = Model()
plants = ["P1", "P2", "P3"]
dcs = ["D1", "D2"]
M = 1000000000

# Linear variable
@variable(m,x[plants, dcs] >= 0)
# Binary Variable
@variable(m,y[plants], Bin)

# Transportation Costs
costs = Dict([("P1", Dict([("D1", 22), ("D2", 22)])),
              ("P2", Dict([("D2", 22), ("D3", 22)])),
              ("P3", Dict([("D1", 22), ("D3", 22)]))])

# Supply Constraint values
capacities = Dict([("P1", 22), ("P2", 22), ("P3", 22)]);

# Demand Constraint values
demands = Dict([("D1", 22), ("D2", 22)])

# Fixed costs
fixed_costs = Dict([("P1", 22), ("P2", 22), ("P3", 2)])


# Minimize total costs + Fixed cost
@objective(m, Min, sum{x[i,j]*costs[i][j],i=plants,j=dcs} +
    sum{y[i]*fixed_costs[i], i=plants})

# Supply Constraint
for (i,k) in capacities
    @constraint(m, sum{x[i,j], j=dcs} <= capacities[i])
end

# Demand Constraints
for (j,k) in demands
    @constraint(m, sum{x[i,j], i=plants} >= k)
end

# Linking Constraints
for i in plants
    @constraint(m, sum{x[i,j], j=dcs} - M*y[i] <= 0)
end

# Solve
print(m)
solve(m)
getvalue(x)
```

```
getvalue(y)
```

Listing 3: Fixed Cost Problem - MILP

2 Managing Uncertainty - Distributions and Random Variables

2.1 Summary Statistics

```
# Read Data
df <- read.csv("data.csv")

# Show structure of data
str(df)

# Show summary
summary(df)

# Standard deviation
sd(df$col)
```

Listing 4: Read data from csv file, general stats

2.2 Distributions

- *p* for "probability", the cumulative distribution function (c. d. f.)
- *q* for "quantile", the inverse c. d. f.
- *d* for "density", the density function (p. f. or p. d. f.)
- *r* for "random", a random variable having the specified distribution

```


# Calculate p.f.
dnorm(x, mean=mu, sd=sigma)

# Calculate cdf
pnorm(p, mean=mu, sd=sigma)

# Get the q quantile from probability p
qnorm(p, mean=mu, sd=sigma, lower.tail=FALSE)

# Generate n random numbers picked from this distribution
rnorm(n, mean=mu, sd=sigma)
```

Listing 5: Probability Distributions

3 Predictive Modeling

3.1 Hypothesis Tests

```
# T test for hypothesis
t.test(x, alternative=c("two.sided", "less", "greater"), mu=
    testMean, conf.level=0.95)

# Calculate Goodness of fit using Chi-square
chisq.test(x, p=probablilties)
```

Listing 6: Hypothesis Tests

3.2 Ordinary Least Squares Linear Regression

```
# least square
result <- lm(Dependent ~ Independent Variable1 + Independent
    Variable 2, data=data)

# Get the result
summary(result)

# Predict using model
predict(result, data.frame(v1=x,v2=y))
```

Listing 7: Regression

4 Descriptive Modeling

4.1 Queue Model

Indebted to *nirski* for recommending *simmer*

```r
library(simmer)
library(triangle)
library(DT)

set.seed(1234)

# Create customer
customer <- create_trajectory("Customer Path") %>%
     seize("Kiosk", 1) %>%
     timeout(function() rtriangle(1, a=0.5, b=15, c=5)) %>%
     release("Kiosk", 1)

logan <- simmer("Airport") %>%
     add_resource("Kiosk", capacity=6, queue_size=16) %>%
     add_generator("Customer", customer, function() rexp(1, 0.5))

logan %>% run(until=9000)

average_times <- function(m) {
    m %>% get_mon_arrivals(per_resource = TRUE) %>%
         mutate(
              flow_time = end_time - start_time,
              waiting_time = flow_time - activity_time
         ) %>%
         group_by(resource) %>%
         summarise_each(funs(mean), flow_time, activity_time, waiting_time)
}

resource_popularity <- function(m) {
    m %>% get_mon_arrivals(per_resource = TRUE) %>%
         group_by(name) %>%
         summarise(resource = resource %>% paste(collapse = " -> ")) %>%
         count(resource)
}

logan %>% resource_popularity
logan %>% average_times
logan %>% plot_evolution_arrival_times("waiting_time")
logan %>% plot_resource_utilization("Kiosk")
```

Listing 8: Discrete Event Simulation

4.2 Simulation

```
runs <- 200

one.trial <- function() {
     samples <- 1000
     #define your simulation function
}

mc.trial <- replicate(runs, one.trial())

result <- mean(mc.trial) %>% round

t.test(mc.trial, conf.level=0.95)
```

Listing 9: Simulation