Problem 1. Simulation Function for Hungry Dawg [30 points]

Hungry Dawg is a growing restaurant chain with a self-insured employee health plan. An employee needs to pay a certain amount for their premium to stay in the plan. When there is a health claim filed by a covered employee, the company covers all the claim expenses. The costs of the health plan during a given period is the difference between the total claim expenses and the total premium revenue from the same period. The actual costs of the health plan for a future period are uncertain because the number of employees and the average claim per employee are unknown. As a manager who oversees the health plan, you want to evaluate the risks of the plan in all possible settings using simulation techniques. To streamline your analysis, you need a function for this task.

The goal of this problem is to define a function which simulates the costs of the plan in a given setting. The function should let users specify the problem setting (e.g., current number of employees and insurance premium). The function returns a vector of simulated costs for the health plan for a future period (in months). Each cost in the vector is computed in one simulation scenario.

Specifically, to compute the number of employees in Month n, we use the following formula:

$$No.of\ current\ employees * (1 + changing\ rate\ of\ No.of\ employees)^n$$

To compute the average claim per employee in Month n, we use the following formula:

$$Current\ avg.claim\ per\ employee * (1 + changing\ rate\ of\ avg.claim\ per\ employee)^n$$

In the simulation, we assume that the changing rate of number of employees follows a uniform distribution (parameters are specified by users) and that the changing rate of average claim per employee follows a normal distribution (parameters are specified by users).

In what follows, you can find the detailed function requirements:

- Function name: HungryDawg_sim_fun
- Arguments (input):
    - n_employee: current number of employees
    - avg_claim_per_employee: current average claim per employee
    - amt_paid_per_employee: amount paid per employee per month
    - sim_mon: forecast time span (in months)
    - sim_times: number of simulated scenarios
    - incre_rate_employee_min: lower limit of the uniform distribution for the monthly changing rate of the number of employees
    - incre_rate_employee_max: upper limit of the uniform distribution for the monthly changing rate of the number of employees
    - incre_rate_avg_claim_mean: mean of the normal distribution for the monthly changing rate of the average claim per employee

    - incre_rate_avg_claim_sd: standard deviation of the normal distribution for the monthly changing rate of the average claim per employee
- Value (output): a vector of company costs for each simulation scenario
- Hint:
    - Do not include set.seed() within the function.
    - For each iteration in your simulation (you have sim_times number of iterations):
        - you need to generate one number for the changing rate of number of employees using a uniform distribution and one number for the changing rate of average claim per employee using a normal distribution
        - then forecast the number of employees in future months, the monthly amount paid by employees in the future months, the average claim per employee in future months, the monthly total claims in the future months.
        - The above forecasts can be used to calculate the total company costs
- Validate your function – run the following code:

```
set.seed(1)
test1 <- HungryDawg_sim_fun(n_employee = 10
                            , avg_claim_per_employee = 250, amt_paid_per_employee = 125
                            , sim_mon = 12, sim_times = 5
                            , incre_rate_employee_min = -0.01, incre_rate_employee_max = 0.01
                            , incre_rate_avg_claim_mean = 0.01, incre_rate_avg_claim_sd = 0.02)
```

- Then, type test1 in console, and you should get:

```
[1] 15210.56 14453.44 20010.26 13037.14 16208.30
```

Grading: Include the following code in your R file.

```
set.seed(1)
P1_test_1 <- HungryDawg_sim_fun(n_employee = 18533, avg_claim_per_employee = 250,
                            amt_paid_per_employee = 125, sim_mon = 12,
                            sim_times = 10000, incre_rate_employee_min = -0.03,
                            incre_rate_employee_max = 0.07, incre_rate_avg_claim_mean = 0.01,
                            incre_rate_avg_claim_sd = 0.015)
set.seed(2)
P1_test_2 <- HungryDawg_sim_fun_sol(n_employee = 18533, avg_claim_per_employee = 250,
                            amt_paid_per_employee = 140, sim_mon = 12,
                            sim_times = 10000, incre_rate_employee_min = -0.03,
                            incre_rate_employee_max = 0.07, incre_rate_avg_claim_mean = 0.01,
                            incre_rate_avg_claim_sd = 0.015)

set.seed(3)
P1_test_3 <- HungryDawg_sim_fun_sol(n_employee = 10000, avg_claim_per_employee = 250,
                            amt_paid_per_employee = 125, sim_mon = 24,
                            sim_times = 10000, incre_rate_employee_min = -0.01,
                            incre_rate_employee_max = 0.01, incre_rate_avg_claim_mean = 0.01,
                            incre_rate_avg_claim_sd = 0.015)
set.seed(4)
P1_test_4 <- HungryDawg_sim_fun_sol(n_employee = 10000, avg_claim_per_employee = 250,
```

```
                    amt_paid_per_employee = 125, sim_mon = 24,
                    sim_times = 10000, incre_rate_employee_min = -0.01,
                    incre_rate_employee_max = 0.01, incre_rate_avg_claim_mean = 0,
                    incre_rate_avg_claim_sd = 0.02)
set.seed(5)
P1_test_5 <- HungryDawg_sim_fun_sol(n_employee = 10000, avg_claim_per_employee = 250,
                    amt_paid_per_employee = 125, sim_mon = 24,
                    sim_times = 10000, incre_rate_employee_min = -0.01,
                    incre_rate_employee_max = 0.02, incre_rate_avg_claim_mean = 0.01,
                    incre_rate_avg_claim_sd = 0.02)
```

- **Display your results with**
  - P1_test_mean, which is a vector of the mean/average of each of your simulation results
  - P1_test_sd, which is a vector of the standard deviation of each of your simulation results
- Each correct value in the P1_test_mean P1_test_sd will be 1 point. The function itself is 20 points.

Problem 2. Risk Analysis and Insurance Policy Design [20 points]

At the end of September, Hungry Dawg has 19,735 employees who have enrolled in the health plan, the current average claim per employee is $224, and the monthly premium per employee is $147. The company budgets $20,000,000 to cover employees' total claims in the coming 10 months. Based on historical data, the monthly changing rate of the number of employees follows a uniform distribution between -0.02 and 0.04, and the monthly changing rate of the average claim per employee follows a normal distribution with mean 0.005 and standard deviation 0.01.

Please answer the following questions:

a. [4 points] Call `HungryDawg_sim_fun` to simulate 10000 10-month costs of the current health plan (set the random seed to 2). Then, save the returned vector of costs in object `vec_sim_cost_10000`.

b. [4 points] Compute the mean and standard deviation of the simulated costs (`vec_sim_cost_10000`), and save them in object `mean_cost_10000` and object `sd_cost_10000`, respectively.

c. [4 points] Compute the percentage of simulated scenarios in `vec_sim_cost_10000` whose cost exceeds the budget. This percentage is often used to estimate the probability that the company will deplete their budget for the health plan. Save the percentage in object `pct_budget_empty_147`.

d. [4 points] If the company increases the monthly premium to $150 per employee, what is the estimated probability that the company will deplete their budget for the health plan? Construct your estimate by simulating the costs 10000 times with random seed 2, and save the estimated probability in object `pct_budget_empty_150`.

e. [4 points] To lower the risk of running out of budget, the company decides to increase the monthly premium per employee. Suppose the company can increase the premium to one of the following amounts: $152, $154, $156, $158, $160, $162, or $164. Use simulations to select the lowest premium so that the risk of depleting the budget is less than 5%. Save the selected premium in a numeric scalar `slct_premium`. (Hint: for each premium, set the random seed to 2 and simulate 10000 costs to evaluate the risk).

Problem 3. Descriptive Analysis of Transactional Dataset [15 points]

In this problem, we perform a descriptive analysis for the products in the competition's transactional dataset. First, load the transactional dataset retail_transaction_train.

a. [3 point] Compute the total sold quantity *of each product* in retail_transaction_train. Save the data frame returned in object df_prod_tot_quan.
   - Display your results with: head(df_prod_tot_quan)

b. [2 points] Use the data frame df_prod_tot_quan to compute the average and standard deviation of the total transacted quantity per product. Save the mean in object mean_tot_quan, and save the standard deviation in sd_tot_quan.

c. [2 point] Use the order function to sort the rows of df_prod_tot_quan so the values in the total transacted quantity column are in descending order. Save the sorted data frame in object df_prod_tot_quan_sort.
   - Display your results with: head(df_prod_tot_quan_sort)

d. [2point] Use the data frame df_prod_tot_quan_sort to select the top 50 products with the largest transacted quantities, and save the product IDs in vector vec_top50_quan_prod. Make sure the order of product IDs in vec_top50_quan_prod is consistent with the order in df_prod_tot_quan_sort.
   - Display your results with: head(vec_top50_quan_prod)

e. [2 point] Add a column to retail_transaction_train named Revenue to save the revenues generated by a product in a given transaction. (Hint: A product's revenue for a transaction is equal to the transacted quantity times the price per unit)
   - Display your results with: head(retail_transaction_train$Revenue)

f. [3 points] Compute the total revenue generated by each product in retail_transaction_train. Save the data frame returned in object df_prod_tot_reve.
   - Display your results with:
     1. head(df_prod_tot_reve)
     2. dim(df_prod_tot_reve)

g. [1 points] Use the data frame df_prod_tot_reve and order function to select the top 50 products with the highest revenues, and save the product IDs in vector vec_top50_reve_prod. Make sure the order of product IDs in vec_top50_reve_prod is in descending order of revenues.
   - Display your results with: head(vec_top50_reve_prod)

Problem 4. `merge` Function [5 points]

Look up and read the documentation for the `merge` function (`?merge`). Create a new data frame by merging `df_prod_tot_quan` and `df_prod_tot_reve` using the `ProductID` column as the key. Save the data frame returned by `merge` function in object `df_prod_quan_reve`. It should have three columns and 3919 rows. The first column should be `ProductID`, the second column should be your total quantity, and third column should be your total revenue.

- Display your results using: head(df_prod_quan_reve)

Problem 5. Fitting a probability distribution [30 points]

We will look at a built-in data set this time. groundbeef is a data set that only has one column – serving, which is the serving size for ground beef patties consumed y children under 5 years old collected in a French survey. We will use data("groundbeef", package = "fitdistrplus") to load the data set. **In this problem, I require you to use ggplot from the ggplot2 package.**
Check out this website for tutorials: http://www.cookbook-r.com/Graphs/Plotting_distributions_(ggplot2)/

1) [2 points] Plot the histogram of the data
    a. 1point for correct labeling title to "Serving size of groundbeef patties", x-axis to "Serving Size" and y-axis to "Count"
2) [5 point] Plot the empirical density by adding a density curve to the histogram you created in number 1.
3) [5 point] Create a graph that shows the skewness and kuotosis of different distributions and your data to help us make a decision. We created such graphs in class.
4) [9 points] Based on your graph from part3, fit three different distributions of your chosen and explain why.
    a. Each distribution you fit together with your reasons why you chose to fit them worth 3 points.
5) [6 points] Plot the distributions you fitted in one single plot. Two points for each curve in the plot.
6) [3 points] Make a final decision which distribution fits your data the best. Provide reasons.