```julia
# ------------------------------------------------------------------------------
# Author: Philip Coyle
# Date Created: 06/26/2020
# ps4_Q1.jl
# ------------------------------------------------------------------------------
using Distributed

# Add cores for parallization
if Sys.isapple()
    addprocs(5)
elseif Sys.islinux()
    addprocs(19)
end

@everywhere using CSV, Optim, Random, LinearAlgebra, Statistics, SharedArrays

# Get & Sort data
function get_data()
    # Get & Sort data
    data = CSV.read("/home/p/pcoyle/CodingBootcamp/ProblemSets/PS4/lwage.csv")

    lwage =  data[:,1]
    college =  data[:,2]
    experience = data[:,3]
    experience2 = experience.^2
    n = length(lwage)

    Y = lwage
    X = [ones(n) college experience experience2]

    return X, Y, n
end

@everywhere function solve_mle(X::Array{Float64, 2}, Y::Array{Float64, 1},
β_in::Array{Float64, 1}, n::Int64)
    nvar = size(X,2)
    func(vars) = Log_Likelihood(X, Y, vars[1:nvar], vars[nvar + 1], n)
    opt = optimize(func, β_in, NelderMead())
    β_opt = opt.minimizer
    β_opt[end] = exp(β_opt[end])

    return β_opt
end

@everywhere function Log_Likelihood(X, Y, β, log_σ, n)
    σ = exp(log_σ)
    llike = -n/2*log(2*π) - n*log(σ) - sum((Y - X * β).^2)/(2σ^2)
    llike = -llike

    return llike
end

function bootstrap_mle(X::Array{Float64, 2}, Y::Array{Float64, 1}, β::Array{Float64,
1}, n::Int64, nsim::Int64)
    nvar = size(X,2)
    n_boot = convert(Int64,floor(n/2))
    β_boot = zeros(nvar+1, nsim)
    β_in = β

    for i = 1:nsim
        println("At bootstrap simulation ", i, " of ", nsim)
        boot_inx = randperm(n)[1:n_boot]
        Y_boot = Y[boot_inx]
        X_boot = X[boot_inx,:]

        β_mle = solve_mle(Y_boot, X_boot, β_in, n_boot)
```

```
        β_mle = solve_mle(X_boot, Y_boot, β_in, n_boot)
        β_boot[:,i] = β_mle

        β_in = β_mle
    end

    return β_boot
end

function bootstrap_mle_parallel(X::Array{Float64, 2}, Y::Array{Float64, 1},
β::Array{Float64, 1}, n::Int64, nsim::Int64)
    nvar = size(X,2)
    n_boot = convert(Int64,floor(n/2))
    β_boot = SharedArray{Float64}(nvar+1,nsim)
    β_in = β

    @sync @distributed for i = 1:nsim
        println("At bootstrap simulation ", i, " of ", nsim)
        boot_inx = randperm(n)[1:n_boot]
        Y_boot = Y[boot_inx]
        X_boot = X[boot_inx,:]

        β_mle = solve_mle(X_boot, Y_boot, β_in, n_boot)
        β_boot[:,i] = β_mle

        β_in = β_mle
    end

    return β_boot
end


X, Y, n = get_data()
β_mle = solve_mle(X, Y, ones(5), n)

println("Running Bootstrap Simulation on 1 core")
@time β_boot = bootstrap_mle(X, Y, β_mle, n, 100)

println("Running Simulation on 20 cores")
@time β_boot = bootstrap_mle_parallel(X, Y, β_mle, n, 100)

σ = std(β_boot,dims=2)

```