Day 5: Demand II

In this lecture, we talked about the potential uneven impacts of the energy transition, with a focus on the electricity sector.

We will replicate here some results from "The Distributional Impact of Real-Time Pricing" by Fabra, Rapson, Reguant, and Wang.

```
begin
    using DataFrames
    using CSV
    using JuMP
    using Ipopt   , Cbc
    using Clustering
    using Plots
    using StatsPlots
    using Binscatters
    using Statistics   , StatsBase
    using Printf
end
```

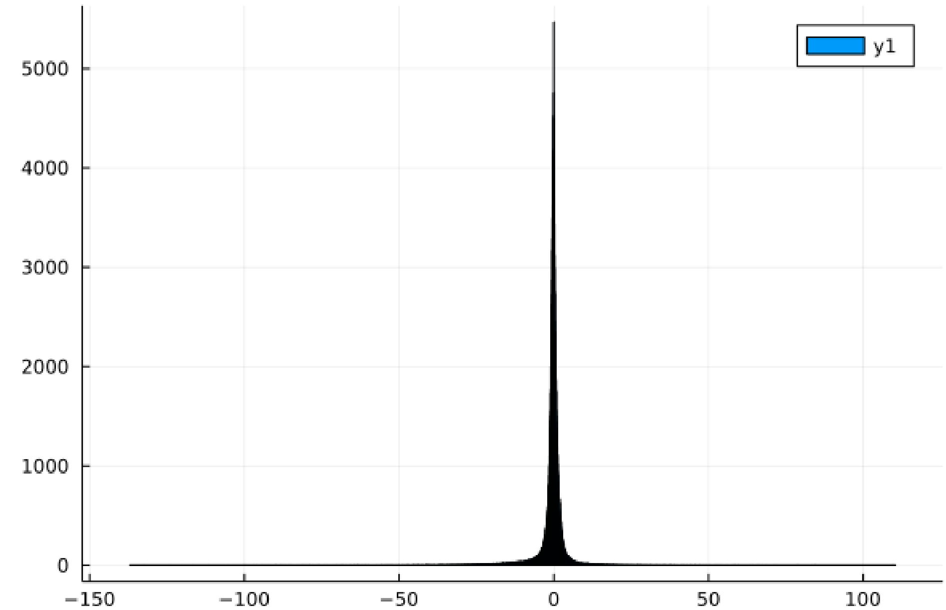
The data prepared for this exercise are **Monte Carlo** data. They are never observed in practice, as in this fake data we will pretend that we have income available to us.

We will then pretend we cannot see it, and see how far we can go with the two-step methodology explained in the paper.

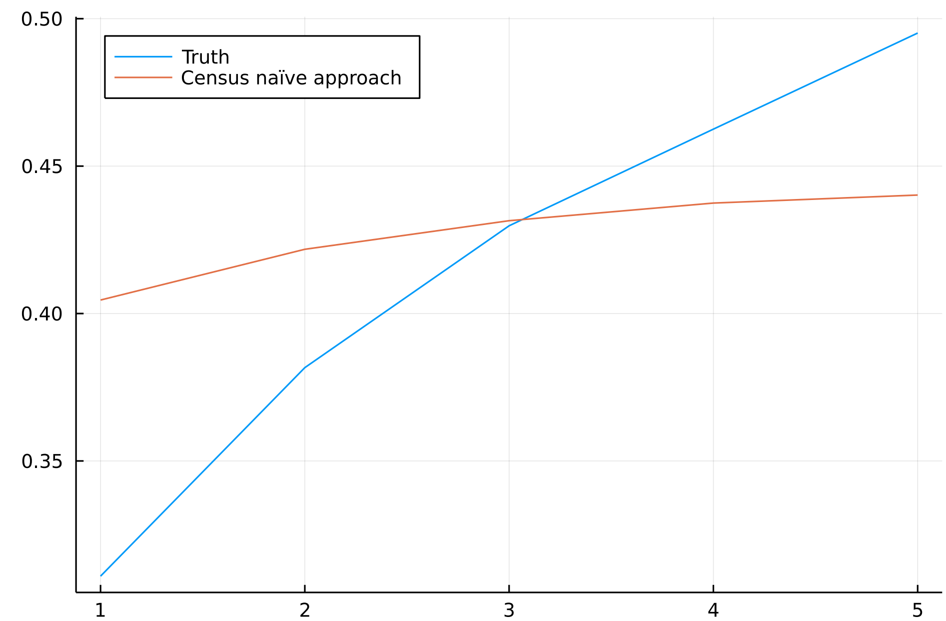
	id	N	zip_id	zip_group	theta_id	inck	cost_rtp	cost_flat	more
1	15	454	2	1	2	5	43.5463	43.6214	
2	16	568	9	2	4	3	184.969	186.096	
3	36	454	5	1	3	3	52.6711	52.8742	
4	37	451	15	3	4	4	176.644	175.378	
5	43	450	8	2	5	4	130.884	129.589	
6	53	441	10	2	3	2	133.092	133.134	
7	57	450	3	1	2	5	22.4435	22.7575	
8	62	454	8	2	4	5	207.941	206.815	
9	66	570	10	2	4	4	182.461	184.725	
10	77	570	13	3	4	5	175.765	176.51	
more									
198598	1311900	454	4	1	1	5	268.701	268.076	

```
begin
    df = CSV.read("data_kmeans.csv", DataFrame)
    df.loose = (df.cost_rtp.-df.cost_flat.>0)
    df
end
```

Impacts of the policy with observed income




```
@df df histogram(:cost_rtp.-:cost_flat)
```




```
let
    df_plt1 = combine(groupby(df, :inck), :loose => mean)
    @df df_plt1 plot(:inck, :loose_mean, label="Truth", legend=:topleft)

    # we assume all households have the same income distribution as the zip code
    # equivalent to assuming all households have the same losing rate
    df_plt2 = transform!(groupby(df, :zip_id), :loose => mean)
    df_plt2 = combine(groupby(df, :inck), :loose_mean => mean)
    @df df_plt2 plot!(:inck, :loose_mean_mean,
        label="Census naïve approach")
end
```

Pretending we don't know! (we really don't)

Now the fun part. We will assume our data does not have data income `inck` and using a naïve approach, the best we can get is the red line in the graph above...

Step 1

In Step 1, we will get at consumer heterogeneity by classifying households into types.

Here we will be using k-means clustering based on consumer load profiles and average consumption.

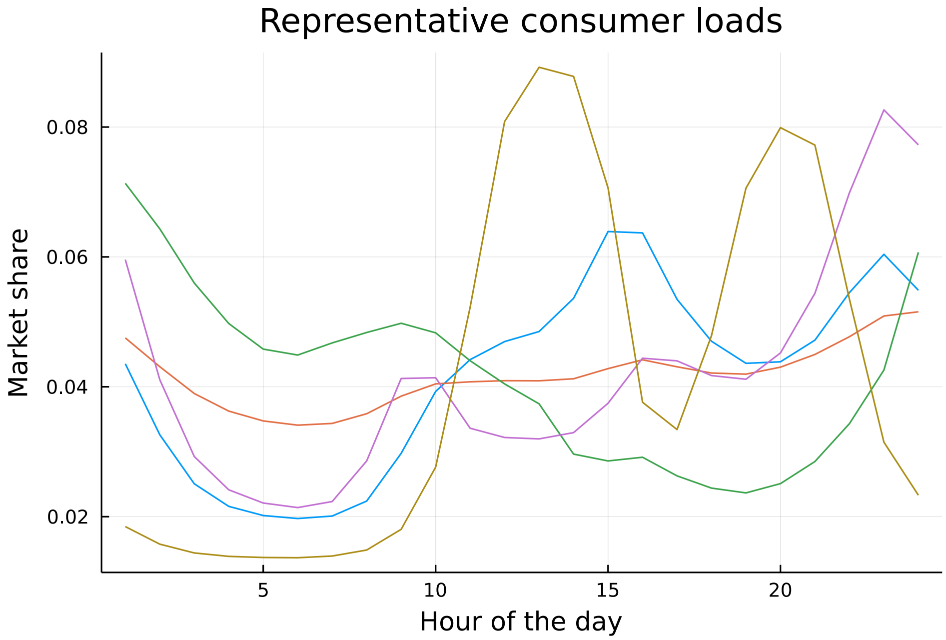
Here is an example of how to do it for all the data at once. Later, we will do it by groups of zip codes to allow for greater heterogeneity.

Note: The data here have already been prepped to do that. I have summarized the smart meter data into one line per household, so that we can easily apply the clustering technique.

```
KmeansResult(26×5 Matrix{Float64}:                                              , [2, 1, 4, 4, 1
               0.129537    -0.357847     1.46373    -0.126386    0.386198
               0.167335    -0.406733     1.5009     -0.149797    0.599714
              -0.326856    -0.119572     1.11829     0.507003   -1.63038
              -0.450267     0.226198     1.59688     0.0971038  -1.53784
              -0.541489     0.512411     1.80225    -0.223277   -1.34895
              -0.570888     0.643927     1.76083    -0.359867   -1.20846
              -0.570769     0.692086     1.65102    -0.403821   -1.13111
               ⋮
               0.121192    -0.00945723  -1.44474    -0.0711638   2.23796
              -0.00423541  -0.0632225   -1.34567     0.0931882   2.57413
              -0.057673    -0.19739     -1.24262     0.400287    1.84687
              -0.0222302   -0.397358    -1.13674     0.826874   -0.0777025
              -0.0250594   -0.487821    -0.89335     1.05751    -1.4336
              -0.199625    -0.366893     0.0955449   0.93083    -1.79278
```

```
begin
    X = transpose(Array(select(df,Between(:kwh,:s_24_mean0))));

    # We scale variables to improve kmeans performance
    Xs = (X.- repeat(mean(X,dims=2),1,nrow(df)))./repeat(std(X,dims=2),1,nrow(df));
    R = kmeans(Xs, 5, maxiter=500, tol=1e-8);
end
```

We can plot the profiles picked up by k-means.




```
let
    df.theta_together = assignments(R);
    df_plt = combine(groupby(df, :theta_together),
        propertynames(df[:,Between(:s_1_mean0,:s_24_mean0)]) .=> mean);

    to_plot = transpose(Array(df_plt[:,Between(:s_1_mean0_mean,:s_24_mean0_mean)]));
    plot(1:24,to_plot,legend=false,
        title="Representative consumer loads",
        xlabel="Hour of the day", ylabel="Market share")
end
```

Step 2

Here we define the Step 2 GMM function, which takes the zip-code level income distribution and the zip-code level type distribution as given.

The algorithm solves for η:

$$
\begin{aligned}
& min_\eta \sum_z \omega_z \sum_k \Big(Pr_z(inc_k) - \sum_n \eta_n^k Pr_z(\theta_n) \Big)^2 \\
& \text{s.t. } \sum_k \eta_n^k = 1, \forall n, \\
& \quad \eta_n^k \in [0,1], \forall n, k.
\end{aligned}
$$

```
gmm_zip (generic function with 1 method)
```

```
function gmm_zip(inc::Array{Float64,2}, theta::Array{Float64,2})

    # We declare a model
    model = Model(
        optimizer_with_attributes(
            Ipopt.Optimizer)
        );

    Z = size(inc,1);
    K = size(inc,2);
    N = size(theta,2);

    @variable(model, 0.0 <= eta[1:K,1:N] <= 1.0);  # pred income prob. for each type
    @variable(model, 0.0 <= fit[1:Z,1:K] <= 1.0);  # pred income prob. for each zip

    @objective(model, Min, sum((inc[z,k] - fit[z,k])^2 for k=1:K, z=1:Z));

    @constraint(model, [n=1:N], sum(eta[k,n] for k=1:K)==1.0);
    @constraint(model, [k=1:K,z=1:Z],
        fit[z,k] == sum(eta[k,n]*theta[z,n] for n=1:N));

    optimize!(model);

    status = @sprintf("%s", JuMP.termination_status(model));

    if (status=="LOCALLY_SOLVED")
        return JuMP.value.(eta),  JuMP.value.(fit)
    else
        @sprintf("%s",JuMP.termination_status(model))
    end

end
```

Putting the two steps together

Here we will run the algorithm for every group of similar zip codes.

- Step 1: Cluster types.
- Step 2: Recover η.

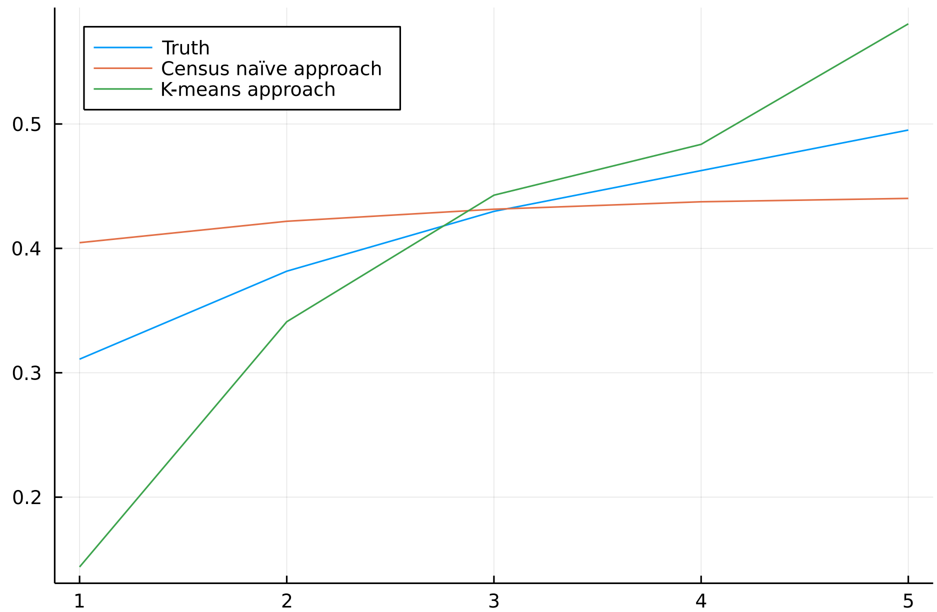
```
begin

    # Number of clusters per group of zip codes
    N = 5;

    # set up some matrix to store results and help
    df.index = rownumber.(eachrow(df));
    for k=1:5
        df[!,string("inc_imp",k)] = zeros(nrow(df));
    end
    for k=1:5
        df[!,string("inc",k)] = (df.inck.==k);
    end

    # This line will put all zip codes together in the same group
    #df.zip_group = 0 * df.zip_group

    # Loop k-means over groups of zip codes
    for zg in unique(df.zip_group)

        df_zip = filter(row -> row.zip_group==zg, df);

        # STEP 1 ####
        X = transpose(Array(select(df_zip,Between(:kwh,:s_24_mean0))));

        # We scale variables to improve kmeans performance
        Xs = (X.-repeat(mean(X,dims=2),1,nrow(df_zip))) ./
                repeat(std(X,dims=2),1,nrow(df_zip));
        R = kmeans(Xs, N; tol=1e-8, maxiter=500);

        # Store theta assignments
        df_zip[!,"theta"] = assignments(R);

        # STEP 2 ####
        # create dummies for types to get type-zip code distribution
        for n=1:N
            df_zip[!,string("theta",n)] = (df_zip.theta.==n);
        end

        inc_dist = Array(combine(groupby(df_zip, :zip_id),
                propertynames(df_zip[:,Between(:inc1,:inc5)]) .=> mean))[:,2:6];
        theta_dist = Array(combine(groupby(df_zip, :zip_id),
                propertynames(df_zip[:,Between(:theta1,string("theta",N))])
                .=> mean))[:,2:N+1];

        eta_fit, inc_fit = gmm_zip(inc_dist,theta_dist)

        # assign back to main dataframe
        ind_zip = df_zip.index;
        for k=1:5
            [df[ind_zip[i],string("inc_imp",k)] =
                eta_fit[k,df_zip[i,:theta]] for i in 1:nrow(df_zip)];
        end
    end

end
```

[2, 9, 5, 15, 8, 10, 3, 13, 12, 6, 1, 7, 14, 11, 4]

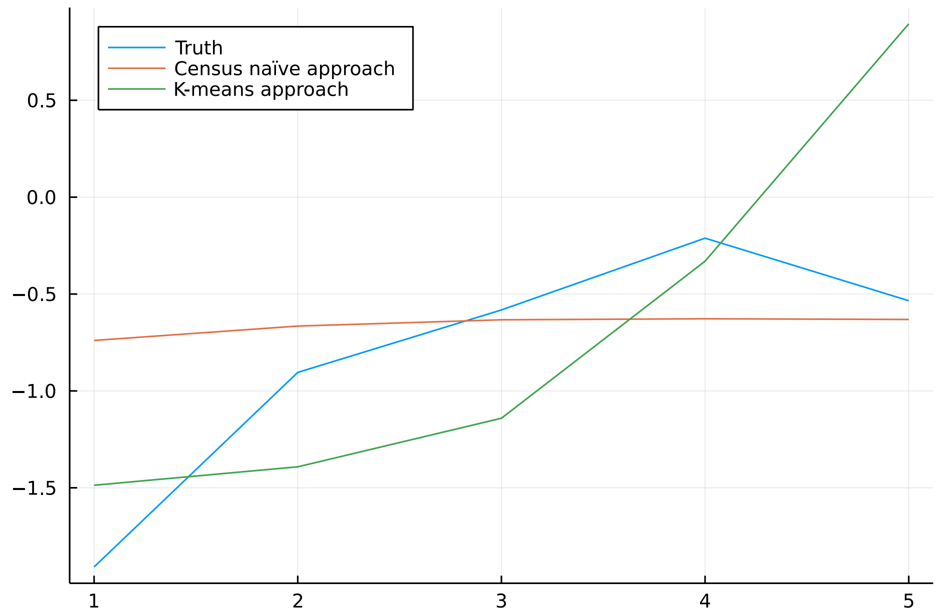
```
unique(df.zip_id)
```

Examining the results

We can now check how the method is doing...!




```
let
    df_plt1 = combine(groupby(df, :inck), :loose => mean)
    @df df_plt1 plot(:inck, :loose_mean, label="Truth", legend=:topleft)

    df_plt2 = transform!(groupby(df, :zip_id), :loose => mean)
    df_plt2 = combine(groupby(df, :inck), :loose_mean => mean)
    @df df_plt2 plot!(:inck, :loose_mean_mean,
        label="Census naïve approach")

    # we assume all households have the same income distribution as the zip code
    # equivalent to assuming all households have the same losing rate
    new_fit = [mean(df.loose, weights(df[!,string("inc_imp",k)])) for k=1:5]
    # replicates truth again
    # new_fit = [mean(df.loose, weights((df[!,"inck"].==k))) for k=1:5]
    plot!(new_fit, label="K-means approach")

end
```




```
let
    df.euros = df.cost_rtp.-df.cost_flat;
    df_plt1 = combine(groupby(df, :inck), :euros => mean)
    @df df_plt1 plot(:inck, :euros_mean, label="Truth", legend=:topleft)

    df_plt2 = transform!(groupby(df, :zip_id), :euros => mean)
    df_plt2 = combine(groupby(df, :inck), :euros_mean => mean)
    @df df_plt2 plot!(:inck, :euros_mean_mean,
        label="Census naïve approach")

    # we assume all households have the same income distribution as the zip code
    # equivalent to assuming all households have the same losing rate
    new_fit = [mean(df.euros, weights(df[!,string("inc_imp",k)])) for k=1:5]
    plot!(new_fit, label="K-means approach")
end
```