Step 1: Setup the working environment

- Rstudio uses a default directory (folder) to acess the data it needs. In this initial step you will tell Rstudio where it can find the data to run the model.
- This can be done using the setwd() function.
- This function takes as an input the direction where the data is stored in your computer and it changes the working directory to that direction.

```
#A. Setup the working environment
  setwd("Copy here the Path to the <Demo> folder on your computer")
  rm(list = ls())
```

Step 2: Install (or load) the IGC.CSM package

- You need to install the IGC.CSM package if it the first time you use it. This can be done using the function install.packages() function.
- Once you have installed it, you need to load it every time you want to use it. This can be done using the function library().

```
#B. Install the package from the CRAN repository
  install.packages('IGC.CSM')
  #library('IGC.CSM')
```

Step 3: Read the data

- The next step consists in reading the data. This can be done using the function read.csv(). Notice that you must read both data files.
- Once the data is read, you must save every characteristic as a variable. In this way, you can call them independently.

```
#C. Upload the required data
    data_chars = read.csv("1. Data for model/1. Chars.csv")
    data_times = read.csv("1. Data for model/2. TravelTimes.csv")

    L_j = as.data.frame(data_chars$L_j)           # Number of workers in each location
    L_i = as.data.frame(data_chars$L_i)           # Number of residents in each locatiobn
    L_i = L_i*sum(L_j)/sum(L_i)                   # Normalize the number of residents in each location

    K = as.data.frame(data_chars$K)               # Area in sq. km from each location
    Q = as.data.frame(data_chars$Q)               # Average floorspace price in each location

    N = dim(L_i)[1]                                       # Define the number of locations in the model
    t_ij = as.matrix(data_times[,2:(N+1)], dim=c(N,N))    # Use the travel times matrix without the index column
```

Step 4: Invert the model

- Next, you will use the function inversionModel().
- This function takes as input the data you read in the previous step and it returns the unobserved characteristics of the city (amenities, productivities, wages and density of development).

```
#D. Invert the model
    inversion_m_bl  = inversionModel(N=N,
                                     L_i=L_i,
                                     L_j=L_j,
                                     Q=Q,
                                     K=K,
                                     t_ij=t_ij)
```

Step 5: Solve the model

- Once you know both the observed and unobserved characteristics of the city, you can solve the model using the function solveModel().
- This function takes as input the observed and unobserved characteristics and it returns features of the city such as the share of resients who commute i to j or the expected utility of living in the city.

```
#E. Solve model
    results_m_bl  = solveModel(N=N,
                               L_i=L_i,
                               L_j=L_j,
                               varphi=inversion_m_bl$varphi,
                               t_ij=t_ij,
                               K=K,
                               a=inversion_m_bl$a,
                               b=inversion_m_bl$b,
                               w_eq=inversion_m_bl$w,
                               u_eq=inversion_m_bl$u,
                               Q_eq=inversion_m_bl$Q_norm,
                               ttheta_eq=inversion_m_bl$ttheta)
```

Step 6: Run counterfactuals

- The function solveModel() can also be used to run counterfactuals.
- The idea is to load new values for either the observed or unobserved variables to assess how the remaining variables change.
- In the code template, we input a new commuting time matrix as a counterfactual scenario.

```
#F. Solve the model after changes in travel times

    data_times_policy = read.csv("1. Data for model/3. TravelTimes_Policy.csv")

    # Use the travel times matrix without the index column
    t_ij_policy = as.matrix(data_times_policy[,2:(N+1)], dim=c(N,N))

    results_m_trans  = solveModel(N=N,
                                  L_i=L_i,
                                  L_j=L_j,
                                  varphi=inversion_m_bl$varphi,
                                  t_ij=t_ij_policy,
                                  K=K,
                                  a=inversion_m_bl$a,
                                  b=inversion_m_bl$b,
                                  w_eq=inversion_m_bl$w,
                                  u_eq=inversion_m_bl$u,
                                  Q_eq=inversion_m_bl$Q_norm,
                                  ttheta_eq=inversion_m_bl$ttheta)
```