2.3 Import and Generate Network Data in Stata, R, and Python

This section explains how to import network data into the three software platforms used throughout this book: Stata, R, and Python. The first step involves importing the dataset using specific libraries and packages available in each environment, which will be introduced at the beginning of the relevant sections. This section is organized into three parts: (1) data import from various sources, (2) data processing and network creation, and (3) network visualization, with an emphasis on key components.

In Stata, we primarily rely on the macro package *nwcommands*, supplemented by the *datanet*, *netsis*, and *netsummarize* methods (Zinilli & Cerulli, 2017). We will also illustrate how to create adjacency matrices using *Mata*, a matrix programming language that augments Stata's functionality.

R offers two popular packages for network manipulation: *statnet* and *igraph* (Kolaczyk & Csárdi, 2014). Python, on the other hand, boasts two major modules—*pandas* and *networkX* —that facilitate manipulation, visualization, and analysis of network data (McKinney, 2012; Al-Taie & Kadry, 2017) .

All the packages for Network Science in Stata, R, and Python comprise graph object classes, generators for creating full graphs, routines for reading in real data, algorithms for analyzing the resulting networks, and basic graphical tools.

2.3.1 Stata Code and Output

From a MATA Adjacency Matrix

```
// Search for nwcommands
. findit nwcommands

// Having information on the nwcommands routines
. help nwcommands

// Creating matrix A in Mata
. mata A=(0,10,1 \5,0,0 \0,2,0)

// Observing the new matrix A
. mata A
```

```
        1   2   3
    +-------------+
  1 |  0  10   1  |
  2 |  5   0   0  |
  3 |  0   2   0  |
    +-------------+
```

```
// Export matrix A from MATA to Stata. Stub option assigns the name to variable identifying the membership
. nwtostata, mat (A) stub (n)

// Declare data to be network data
. nwset , mat(A)

// Details on the new matrix
. nwnodelab, nodeid(3) detail

// Declare data to be network data and with name() we assign a name to STATA network
. nwset, mat(A) name(netA)
```

From a Stata Adjacency Matrix

```
// In Stata, we create a directed adjacency matrix. label identifies the name of vertices

n1      n2      n3      label
0       10      1       x
5       0       0       y
0       2       0       z

// Plot the network just created. We use the option arcsyle() for having curved edges
. nwplot,label(matrix) arcstyle(curved)

// Edge size by weight
. nwplot netA, edgesize(netA)
```

From a Stata Edge List

```
// Clear all networks and variables from memory
. nwset, nwclear

// Create two variables (ego and alter) that will be used to identify the relationships: (1, 2), (2, 3), (2, 4), (3, 4)

ego     alter
 1       2
 2       3
 2       4
 3       4
 5       5


// Declare data in edge list format
. nwset ego alter, edgelist

// Plot the network with labels
. nwplot, lab
```

List of all Networks in Memory

```
// List networks in memory
. nwds

// Display more details about existing networks
. nwset, detail

// Show the current network. The current network is just the most recently set network through the command nwset
. nwcurrent
```

Converting an Edge List to an Adjacency Matrix

```
// The example below shows how to make an edge list in Stata and convert it to an adjacency matrix and vice versa.

// Edge list creation
. nwset, nwclear
. gen ego=.
. set obs 1
. replace ego = 1 in 1
.
.

// Edge list converted to adjacency matrix
. nwfromedge ego alter, name(mynet)

// To create an edge list from the newly formed adjancecy matrix
. nwtoedge mynet
```

ego	alter		_nodelab	_nodevar	_nodeid	net1	net2	net3	net4	net5
1	2		1	net1	1	0	1	0	0	0
2	3	⟶	2	net2	2	0	0	1	1	0
2	4		3	net3	3	0	0	0	1	0
3	4		4	net4	4	0	0	0	0	0
5	5		5	net5	5	0	0	0	0	1

2.3.2 R Code and Output

Edge List Creation

```
# Install and upload the library
> install.packages("igraph")
> library(igraph)

# We set the seed (random number), which is useful for creating random objects that can be reproduced
> set.seed(99)

# Create a dataframe “list ”with two columns
> list <- data.frame(source=c("A", "A", "A", "A", "B", "F", "B"), target=c("B", "B", "C", "D", "E", "A", "F"))

# We build the graph object that we call “network”
> network <- graph_from_data_frame(d=list, directed=F)

# Plot the network
> plot(network)
```

Adjacency Matrix Creation #1

```
# Install and upload the library
> install.packages("igraph")
> library(igraph)

# We set the seed (random number), which is useful for creating random objects that can be reproduced
> set.seed(99)

# Create a 5x5 matrix with values from 0 to 1
> matrix <- matrix(sample(0:1, 25, replace=TRUE), nrow=5)

# The first five alphabetic letters are used to label the columns and rows.
> colnames(matrix) = rownames(matrix) = LETTERS[1:5]

# We build the graph object that we call “network”
> network <- graph_from_adjacency_matrix(data)

# Plot the network
> plot(network)
```

Adjacency Matrix Creation #2

```
# Install and upload the library
> library(statnet)
> library(sna)
> library(readr)
> library(readxl)

# We set the seed (random number)
> set.seed(999)

# We set the number of nodes
> num_nodes <- 26

# The runif() function is used to generate n uniform random numbers. nrow and ncol must be the same.
> matrix<-matrix(round(runif(num_nodes*num_nodes)), nrow = num_nodes, ncol = num_nodes)

# We replace the diagonal elements of the matrix with 0 for avoiding self -loops
> diag(matrix) <- 0

# We build the graph object that we call “net”. We specify whether the network is directed or undirected. Option matrix.type is the
typology of input
> net <- as.network(x = matrix, directed = FALSE, loops = FALSE, matrix.type = "adjacency")

# We give each of the 26 nodes in my network a name. The labels in this case are the ISO codes of several European countries.
> network.vertex.names(net) <- c("AT", "BE", "CH", "CY", "DE", "DK", "EE", "EL", "ES", "FI", "FR", "HU", "IE",
"IL", "IT", "LT", "LU", "LV", "MT", "NL", "NO", "PL", "PT", "RO", "SE", "SK")
```

Import Data from External Sources

```
# haven package allows R to read and write a variety of data formats used by other statistical tools. readr package provides a fast way to read data formats like 'txt', 'csv', 'tsv', and 'fwf'
> install.packages("haven")
> install.packages("readr ")

# We download the libraries
> library(haven)
> library("readr")

# setwd() function sets the working directory
> setwd("C:\\mypath")

# We import the dataset we want to use
> dataset<- read.csv("C:\\mypath\\dataset-name.txt")

# Splitting the dataset by the "year" variable
> split<-split(dataset, dataset$year)

# We can see that several datasets have been created
> str(split)

# Now we can create four different datasets
> y2011<-split$'2011'
> y2012<-split$'2012'
> y2013<-split$'2013'
> y2014<-split$'2014'
```

2.3.3 Python Code and Output

Graph Creation #1

```
# Loading Python Libraries. 'as' is a more practical way to use the libraries. It can be used to call the specific library, related functions and data types
>>> import pandas as pd
>>> import numpy as np
>>> import networkx as nx
>>> import matplotlib.pyplot as plt

# Create an empty graph G (undirected graph). We can write G = nx.DiGraph() to create an empty direct graph
>>> G = nx.Graph()

# Add one node
>>> G.add_node(1)

# Add nodes from a list
>>> G.add_nodes_from([2, 3])

# We add one edge between node 1 and 2
>>> G.add_edge(1, 2)

# Add an edge between node 2 and 3
>>> e = (2, 3)
>>> G.add_edge(*e)

# Plot the network with label option
>>> nx.draw(G, pos=nx.circular_layout(G), node_color='r', edge_color='b', with_labels=True)
```

Graph Creation #2

```
# Loading Python Libraries
>>> import networkx as nx
>>> import matplotlib.pyplot as plt
>>> %matplotlib inline
>>> import warnings; warnings.simplefilter('ignore')

# We build the symmetric network. The 'weight' option allows you to add the relationship's intensity to the network; each edge has a weight. We use nx.DiGraph() for directed networks
>>> MyGraph = nx.Graph()
>>> MyGraph.add_edge('Antonio', 'Valentina')
>>> MyGraph.add_edge('Antonio', 'Claudia')
>>> MyGraph.add_edge('Claudia','Valentina')

# Some network info are visualized
>>> print(nx.info(MyGraph))

# We use the draw_networkx() function to visualize the network and save it to the folder.nx.draw_networkx(MyGraph, font_size=7)
>>> fig.savefig('myGraph.png', dpi=100)
```

Edge List Creation

```
# Loading Python libraries
>>> import pandas as pd
>>> import numpy as np
>>> import networkx as nx
>>> import matplotlib.pyplot as plt

# Build a dataframe with 4 connections
>>> df = pd.DataFrame({ 'from':['A', 'B', 'C','A'], 'to':['D', 'A', 'E','C']})

# Build the graph
>>> G=nx.from_pandas_edgelist(df, 'from', 'to')

# Plot it
>>> nx.draw(G, with_labels=True)
>>> plt.show()
```

Adjacency Matrix Creation

```
# Loading Python Libraries
>>> import pandas as pd
>>> import numpy as np
>>> import networkx as nx
>>> import matplotlib.pyplot as plt

# Create a graph in adjacency format
>>> graph = {'A': set(['B', 'C']),
             'B': set(['A', 'D', 'E']),
             'C': set(['A', 'F']),
             'D': set(['B']),
             'E': set(['B', 'F']),
             'F': set(['C', 'E'])
            }

# Save the generated figure in .png format. We can also save it as EPS, JPEG and other formats
>>> plt.savefig("graph.png")
```

From Edge List to Adjacency Matrix

```
# We create a list of 4 items
>>> E = [[0, 0], [0, 1], [1, 0], [1, 1]]

# We indicate the size of each item in E
>>> size = len(set([n for e in E for n in e]))

# Make an empty adjacency list
>>> adjacency = [[0]*size for _ in range(size)]

# Populate the list for each edge
>>> for sink, source in E:
    adjacency[sink][source] = 1

# Adjacency matrix visualization
>>> print(adjacency)
```