Seminario

Stefano Bilotta

TensorFlow and Rstudio for Traffic Flow Reconstruction

all'interno del Corso di Big Data Architectures (prof. Paolo Nesi)

Table of Contents

1. An Introduction to R language
2. An Introduction to TensorFlow
3. Computational view: Trends and examples
4. Traffic flow reconstruction problem:
   - Theoretical background
   - Data ingestion
   - Data Structures
   - Method
   - Learning approach
   - Parallelizing computing
   - Visual view

An introduction to R

- R can be regarded as an implementation of the S language which was developed at Bell Laboratories by Rick Becker, John Chambers and Allan Wilks, and also forms the basis of the S-Plus systems (commercial licence).

- R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

An introduction to R

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. Among other things it has:

- an effective data handling and storage facility
- a suite of operators for calculations on arrays, in particular matrices
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either directly at the computer or on hardcopy
- simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.

An introduction to R

- Free software environment
- Mathematical function and graphic module embedded
- R is functional programming language
- R is an interpreted language
- R is object oriented language

http://cran.r-project.org/src/base/

An introduction to R: *packages*

- R is very much a vehicle for newly developing methods of interactive data analysis. It has developed rapidly, and has been extended by a large collection of *packages*.
- However, most programs written in R are essentially ephemeral, written for a single piece of data analysis.
- All R functions and datasets are stored in packages. Only when a package is loaded, then its contents is available. This is done for efficiency (the full list would take more memory).
- The process of developing packages is described by formalized procedure.

An introduction to R: *packages*

- The standard (or base) packages are considered part of the R source code. They contain the basic functions that allow R to work, and the datasets and standard statistical and graphical functions. They should be automatically available in any R installation.

- There exist a lot of dedicated packages regarding, for example, Data Import/Export, Data Mining, Linear and non-linear modelling, Time-Series Analysis, Classification models, Clustering procedures, Machine Learning, etc.

An introduction to R: *R Datasets Package*

The R Datasets Package

Documentation for package 'datasets' version 3.6.0

- DESCRIPTION file.

Help Pages

A B C D E F H I J L M N O P Q R S T U V W

datasets-package	The R Datasets Package
-- A --	
ability.cov	Ability and Intelligence Tests
airmiles	Passenger Miles on Commercial US Airlines, 1937-1960
AirPassengers	Monthly Airline Passenger Numbers 1949-1960
airquality	New York Air Quality Measurements
anscombe	Anscombe's Quartet of 'Identical' Simple Linear Regressions
attenu	The Joyner-Boore Attenuation Data
attitude	The Chatterjee-Price Attitude Data
austres	Quarterly Time Series of the Number of Australian Residents
-- B --	
beaver1	Body Temperature Series of Two Beavers
beaver2	Body Temperature Series of Two Beavers
beavers	Body Temperature Series of Two Beavers
BJsales	Sales Data with Leading Indicator
BJsales.lead	Sales Data with Leading Indicator
BOD	Biochemical Oxygen Demand
-- C --	
cars	Speed and Stopping Distances of Cars
ChickWeight	Weight versus age of chicks on different diets
chickwts	Chicken Weights by Feed Type
CO2	Carbon Dioxide Uptake in Grass Plants
co2	Mauna Loa Atmospheric CO2 Concentration
crimtab	Student's 3000 Criminals Data
-- D --	
datasets	The R Datasets Package
discoveries	Yearly Numbers of Important Discoveries
DNase	Elisa assay of DNase
-- E --	

An introduction to R: *packages*

- All packages have rich documentation:

RDocumentation | Search for packages, functions, etc | R Enterprise Training | R package | Leaderboard

jsonlite v1.5

Other versions

2,259 Monthly downloads | 99.5th Percentile

by Jeroen Ooms | View Source | https://www.rdocumentation.org/packages/jsonlite | Copy

A Robust, High Performance JSON Parser and Generator for R

A fast JSON parser and generator optimized for statistical data and the web. Started out as a fork of 'RJSONIO', but has been completely rewritten in recent versions. The package offers flexible, robust, high performance tools for working with JSON in R and is particularly powerful for building pipelines and interacting with a web API. The implementation is based on the mapping described in the vignette (Ooms, 2014). In addition to converting JSON data from/to R objects, 'jsonlite' contains functions to stream, validate, and prettify JSON data. The unit tests included with the package verify that all edge cases are encoded and decoded consistently for use with dynamic data in systems and applications.

Functions in jsonlite

Search

Name	Description
unbox	Unbox a vector or data frame
validate	Validate JSON
base64	Encode/decode base64
flatten	Flatten nested data frames
rbind_pages	Combine pages into a single data frame
read_json	Read/write JSON
serializeJSON	serialize R objects to JSON
stream_in, stream_out	Streaming JSON input/output
toJSON, fromJSON	Convert R objects to/from JSON
prettify, minify	Prettify or minify a JSON string

An introduction to R: *packages*

- All packages have rich documentation:

arXiv:1403.2805v1 [stat.CO] 12 Mar 2014

The **jsonlite** Package: A Practical and Consistent Mapping Between JSON Data and R Objects

Jeroen Ooms
UCLA Department of Statistics

Abstract

A naive realization of JSON data in R maps JSON *arrays* to an **unnamed list**, and JSON *objects* to a **named list**. However, in practice a list is an awkward, inefficient type to store and manipulate data. Most statistical applications work with (homogeneous) vectors, matrices or data frames. Therefore JSON packages in R typically define certain special cases of JSON structures which map to simpler R types. Currently there exist no formal guidelines, or even consensus between implementations on how R data should be represented in JSON. Furthermore, upon closer inspection, even the most basic data structures in R actually do not perfectly map to their JSON counterparts and leave some ambiguity for edge cases. These problems have resulted in different behavior between implementations and can lead to unexpected output. This paper explicitly describes a mapping between R classes and JSON data, highlights potential problems, and proposes conventions that generalize the mapping to cover all common structures. We emphasize the importance of type consistency when using JSON to exchange dynamic data, and illustrate using examples and anecdotes. The jsonlite R package is used throughout the paper as a reference implementation.

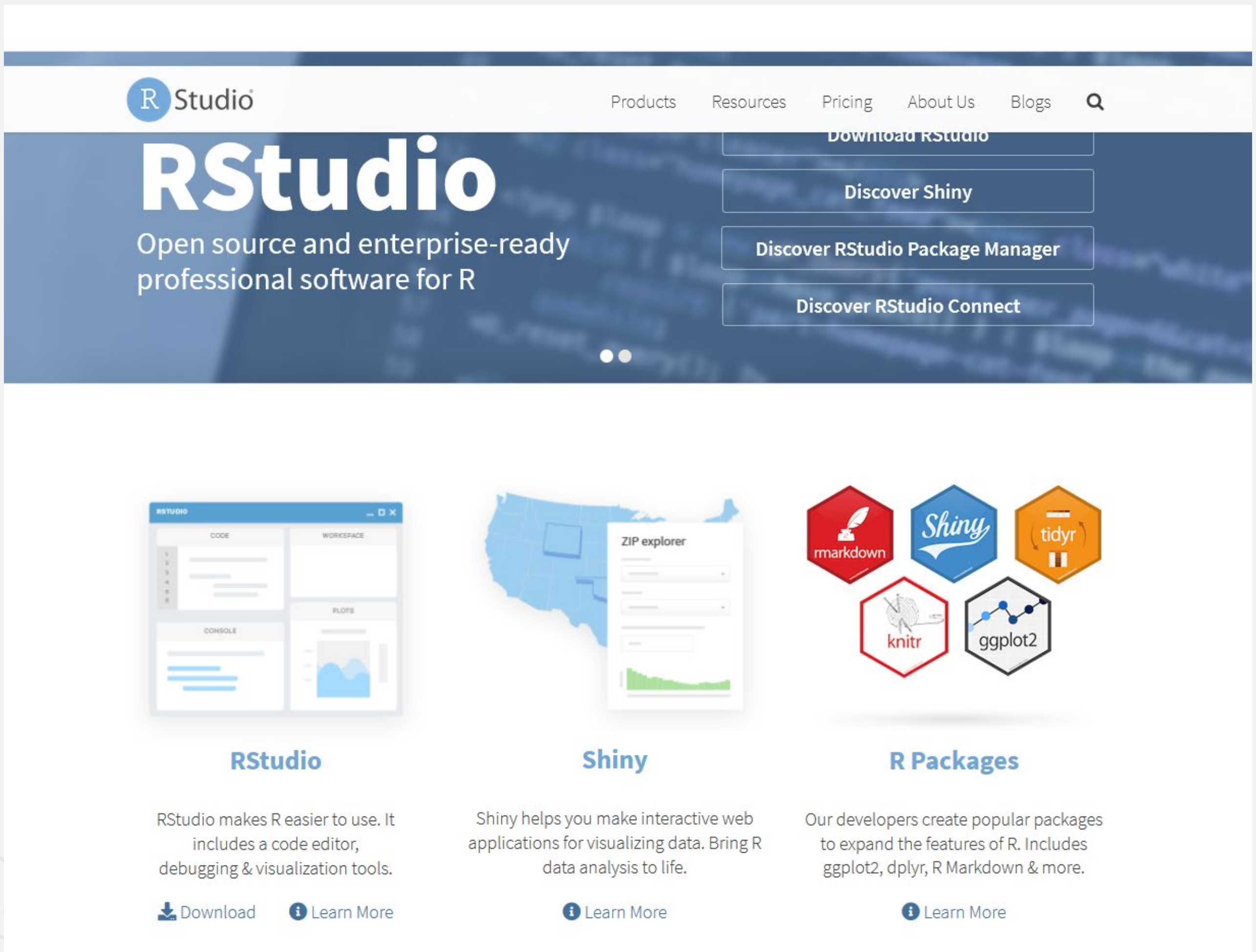
1 Introduction

JavaScript Object Notation (JSON) is a text format for the serialization of structured data (Crockford, 2006a). It is derived from the object literals of JavaScript, as defined in the ECMAScript Programming Language Standard, Third Edition (ECMA, 1999). Design of JSON is simple and concise in comparison with other text based formats, and it was originally proposed by Douglas Crockford as a "fat-free alternative to XML"

An introduction to R: *Rstudio*

- The most convenient way to use R is at a graphics workstation running a windowing system
- Rstudio: Take control of your code
- RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.
- RStudio is available in open source and commercial editions and runs on the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server or RStudio Server Pro (Debian/Ubuntu, RedHat/CentOS, and SUSE Linux).

An introduction to R: *Rstudio*



Rstudio Desktop

RStudio Desktop

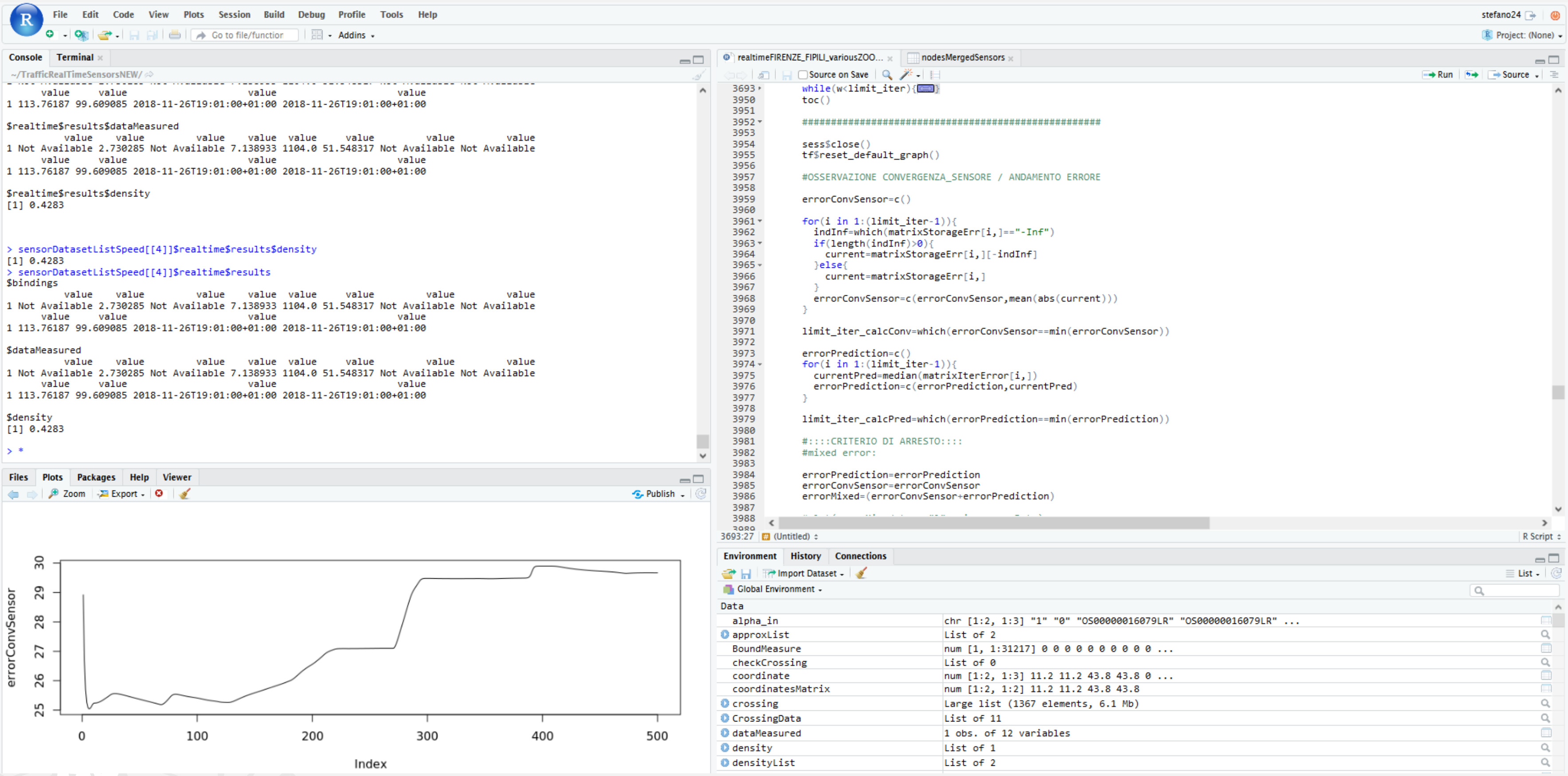
	Open Source Edition	Commercial License
Overview	• Access RStudio locally • Syntax highlighting, code completion, and smart indentation • Execute R code directly from the source editor • Quickly jump to function definitions • Easily manage multiple working directories using projects • Integrated R help and documentation • Interactive debugger to diagnose and fix errors quickly • Extensive package development tools	All of the features of open source; plus: • A commercial license for organizations not able to use AGPL software • Access to priority support
Support	Community forums only	• Priority Email Support • 8 hour response during business hours (ET)
License	AGPL v3	RStudio License Agreement
Pricing	Free	$995/year

Rstudio Server

RStudio Server

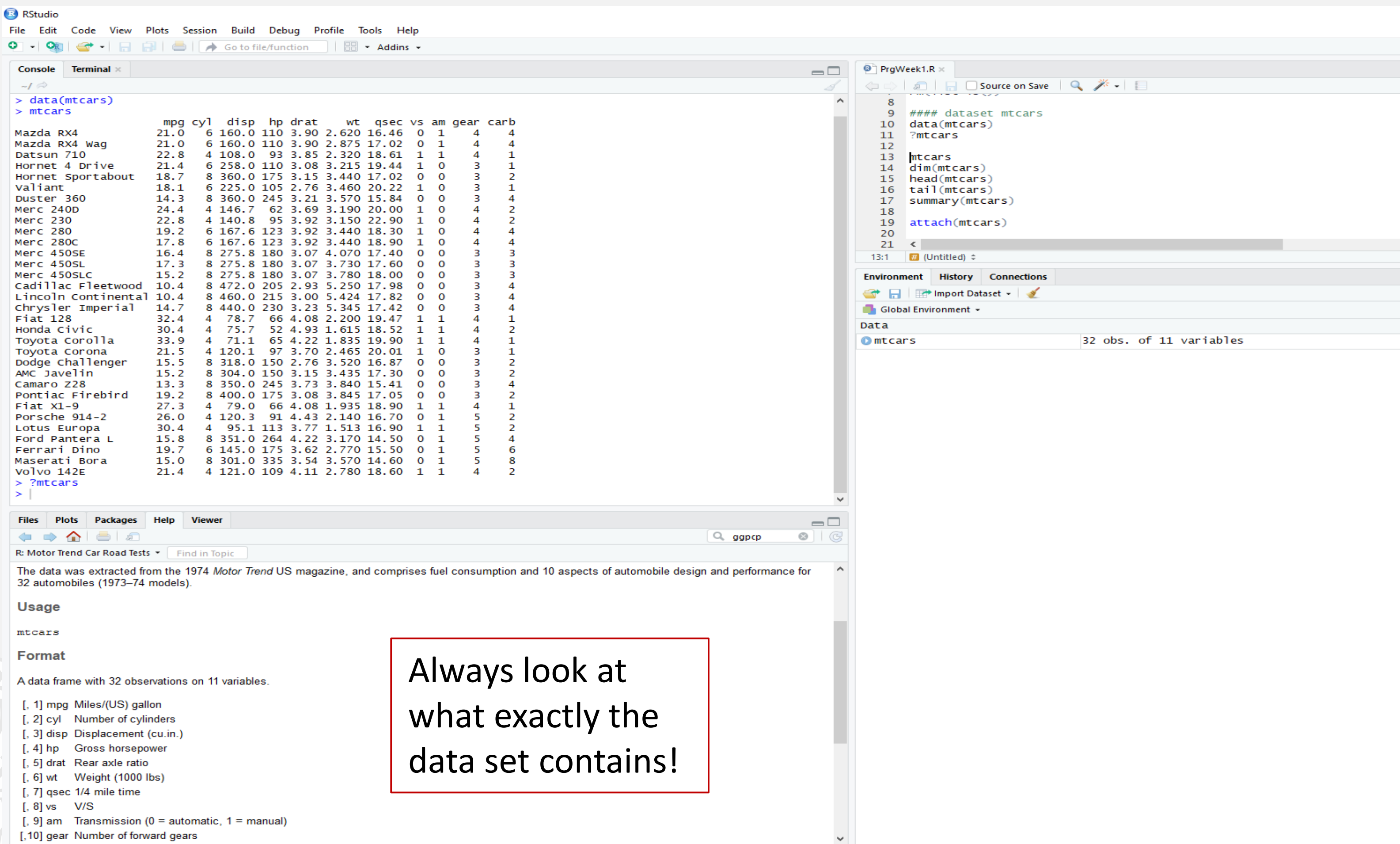
	Open Source Edition	Commercial License
Overview	• Access via a web browser • Move computation closer to the data • Scale compute and RAM centrally	All of the features of open source; plus: • Administrative Tools • Enhanced Security and Authentication • Metrics and Monitoring • Advanced Resource Management
Documentation	Getting Started with RStudio Server	RStudio Server Professional Admin Guide
Support	Community forums only	• Priority Email Support • 8 hour response during business hours (ET)
License	AGPL v3	RStudio License Agreement
Pricing	Free	$9,995/server/year Academic and Small Business discounts available
	DOWNLOAD SERVER	DOWNLOAD FREE RSTUDIO PRO EVAL
		Purchase Contact Sales Info

Rstudio: graphical interface



An introduction to R: *powerful calculator*

- mtcars: popular data set for examples



Always look at what exactly the data set contains!

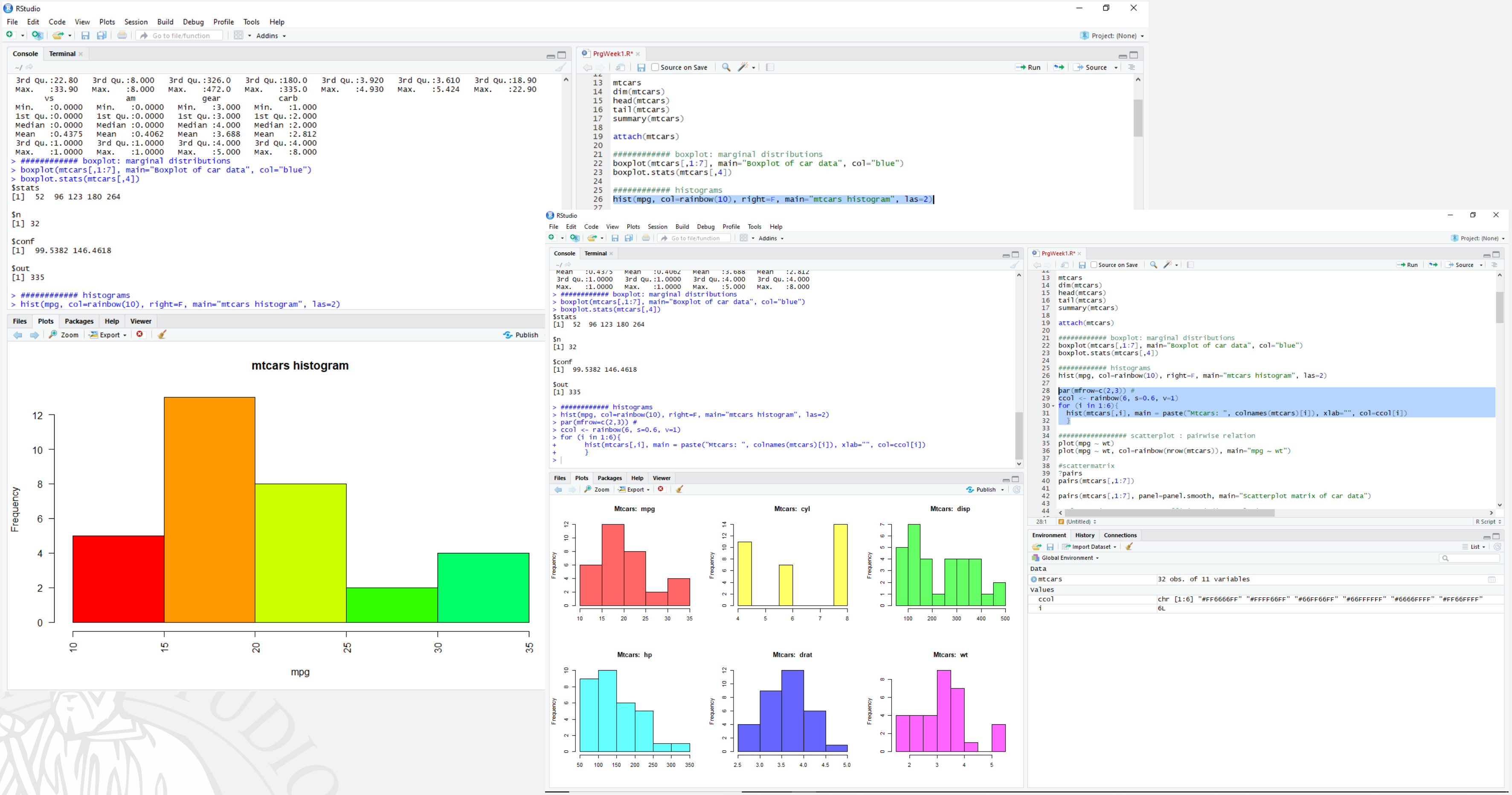
An introduction to R: *powerful calculator*

- First look at the data : Summarizing statistics

```
> summary(mtcars)
      mpg             cyl             disp             hp             drat      
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0   Min.   :2.760  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:3.080  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0   Median :3.695  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7   Mean   :3.597  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.920  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0   Max.   :4.930  
       wt             qsec             vs               am        
 Min.   :1.513   Min.   :14.50   Min.   :0.0000   Min.   :0.0000  
 1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000   1st Qu.:0.0000  
 Median :3.325   Median :17.71   Median :0.0000   Median :0.0000  
 Mean   :3.217   Mean   :17.85   Mean   :0.4375   Mean   :0.4062  
 3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000   3rd Qu.:1.0000  
 Max.   :5.424   Max.   :22.90   Max.   :1.0000   Max.   :1.0000  
      gear            carb      
 Min.   :3.000   Min.   :1.000  
 1st Qu.:3.000   1st Qu.:2.000  
 Median :4.000   Median :2.000  
 Mean   :3.688   Mean   :2.812  
 3rd Qu.:4.000   3rd Qu.:4.000  
 Max.   :5.000   Max.   :8.000  
> 
```

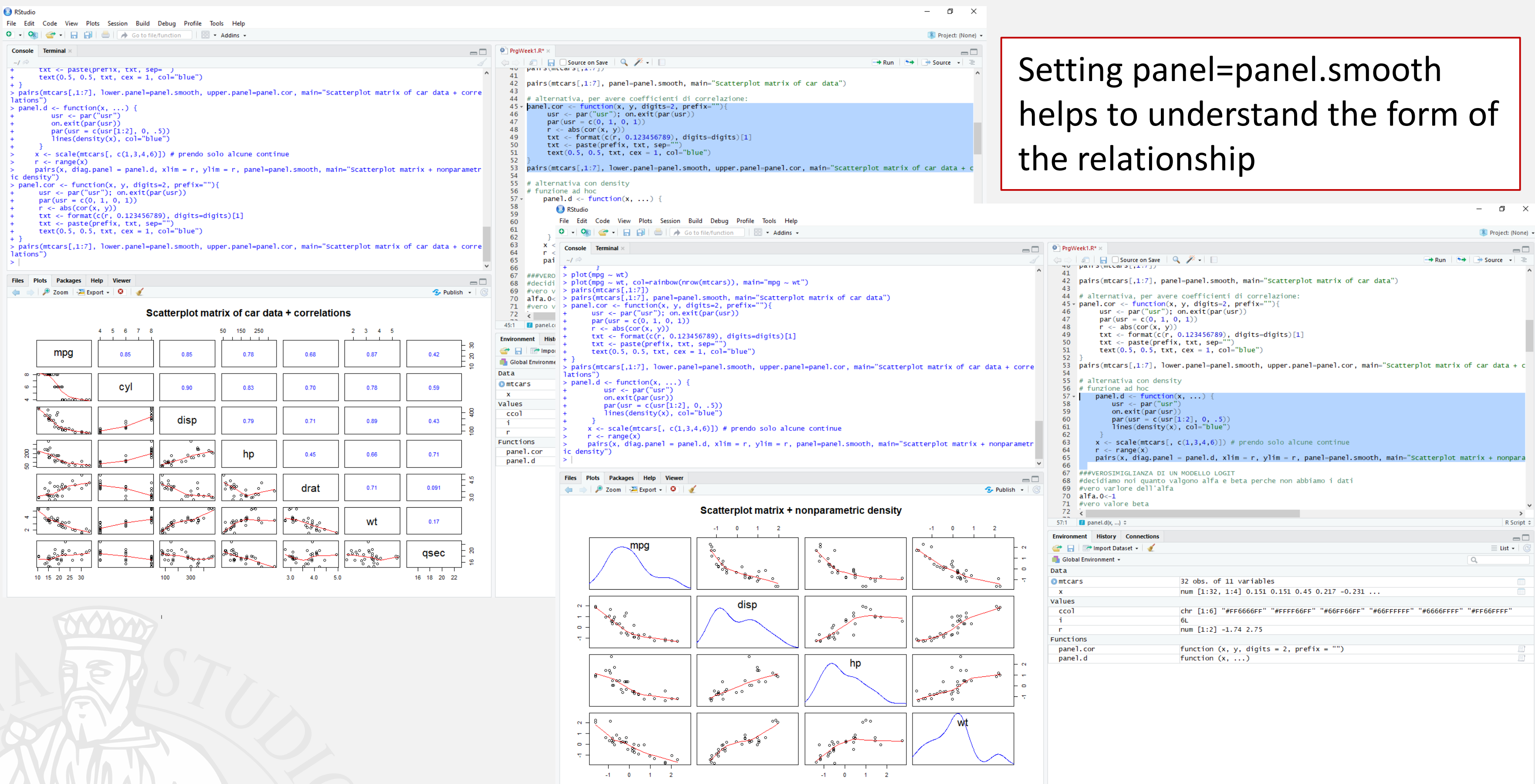
An introduction to R: *powerful calculator*

- Histograms



An introduction to R: *powerful calculator*

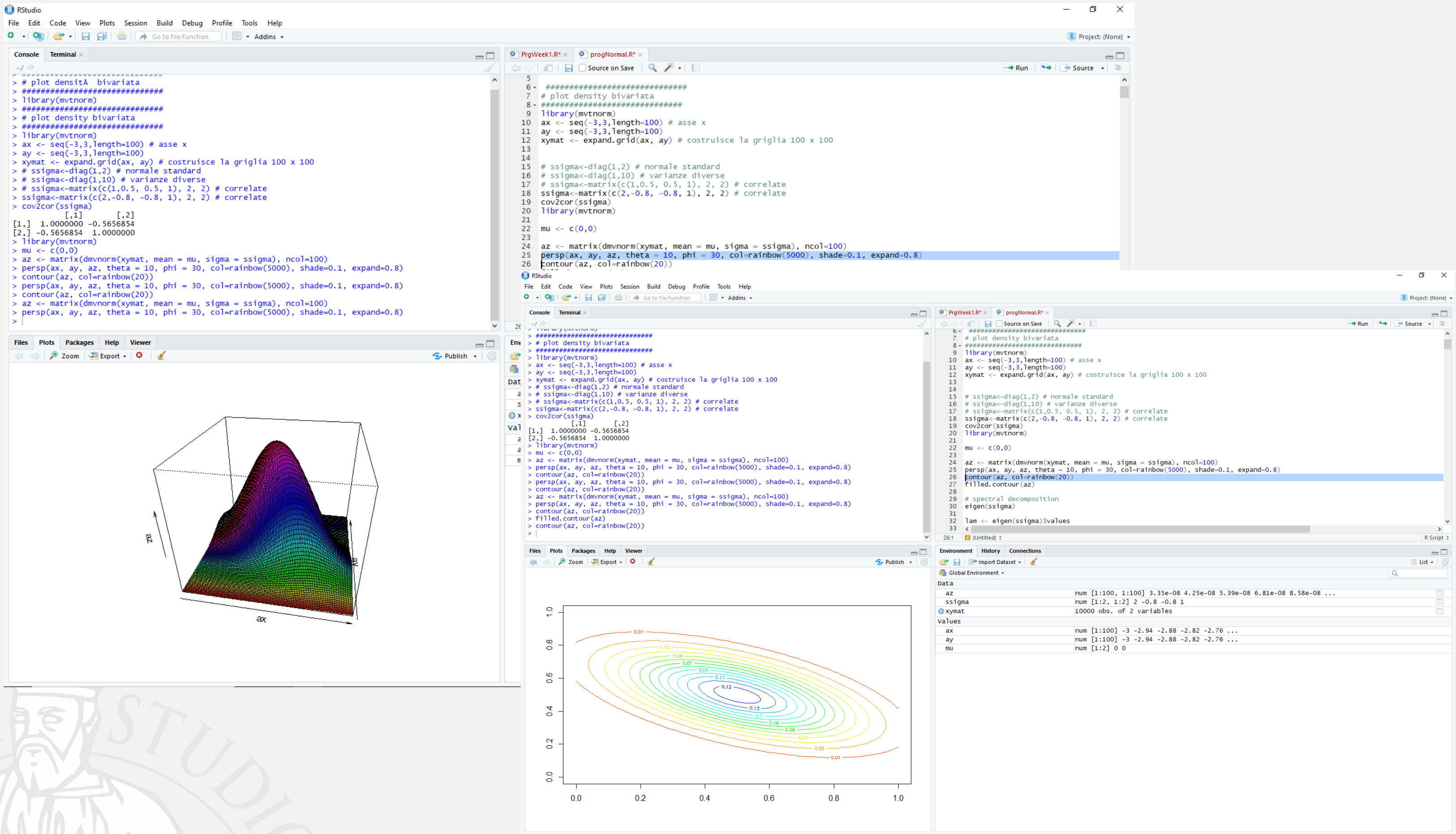
➢Scatterplot: pairwise relationships



Setting panel=panel.smooth helps to understand the form of the relationship

An introduction to R: *powerful calculator*

- Perspective Plot and Contour Plot



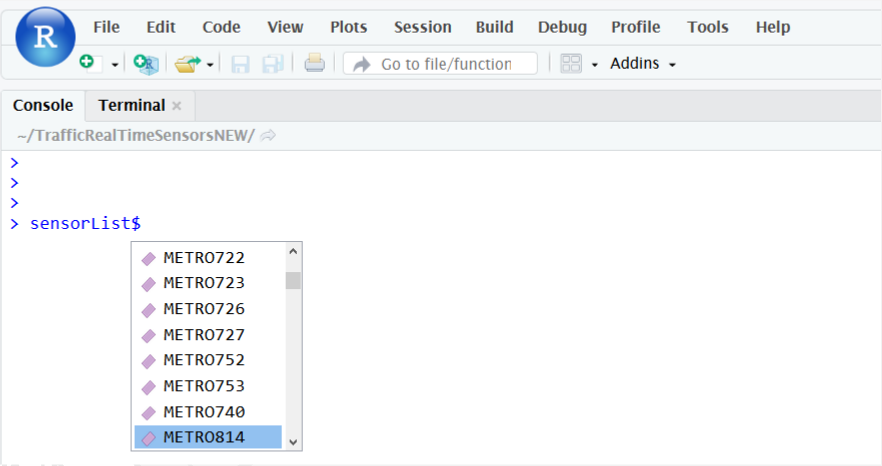
An introduction to R: *List*

- An R list is an object consisting of an ordered collection of objects known as its components.
- There is no particular need for the components to be of the same mode or type, and, for example, a list could consist of a numeric vector, a logical value, a matrix, a complex vector, a character array, a function, and so on.
- Components are always numbered and may always be referred to as such.

An introduction to R: *List*

- Components of lists may also be named, and in this case the component may be referred to either by giving the component name as a character string in place of the number in double square brackets, or, more conveniently, by giving an expression of the form

  > name$component_name



R interface to…

reticulate: R interface to Python

JJ Allaire

2018-03-26

Categories: Packages R Markdown Tags: Packages R Markdown Python reticulate

We are pleased to announce the **reticulate** package, a comprehensive set of tools for interoperability between Python and R. The package includes facilities for:

- Calling Python from R in a variety of ways including R Markdown, sourcing Python scripts, importing Python modules, and using Python interactively within an R session.
- Translation between R and Python objects (for example, between R and Pandas data frames, or between R matrices and NumPy arrays).
- Flexible binding to different versions of Python including virtual environments and Conda environments.

R interface to…

rJava - Low-level R to Java interface

RForge.net

rJava

About rJava
GIT access
Download/Files
News
Check results
Package R docs

About *rJava*

What is rJava?

rJava is a simple R-to-Java interface. It is comparable to the `.C`/`.Call` C interface. rJava provides a low-level bridge between R and Java (via JNI). It allows to create objects, call methods and access fields of Java objects from R.

rJava release versions can be obtained from **CRAN** - usually `install.packages("rJava")` in R will do the trick. The current development version can be downloaded from the **files section**.

In a sense the inverse of rJava is JRI (Java/R Interface) which provides the opposite direction - calling R from Java. **JRI** is now shipped as a part of the **rJava** package, although it still can be used as a separate entity (especially for development). Currently rJava is used as a part of **JGR**, **iPlots** and **JavaGD** software/packages.

Please report any bugs or wishes related to rJava or JRI using **Issues on GitHub**.

What's new?

rJava source repository is now on **GitHub** and that is also the place to **report bugs**. The main page and builds are still on RForge.net.

2016/01/06 - rJava 0.9-8 released. Mostly work-arounds for bugs in Oracle Java on OS X - see **NEWS** for details.

2015/07/27 - rJava 0.9-7 released. Mostly various bugfixes - see **NEWS** for details.

2012/12/23 - rJava 0.9-6 released. Fixes Java parameter issue introduced in 0.9-5 on systems with headless mode (e.g. OS X).

2011/06/22 - rJava 0.9-0 released. This is a major upgrade that changes behavior of array references in low-level calls back to early 0.8 state as intedended by the original design. It should be more consistent now. We have had rJava 0.9-0 in RC state for a long time so hopefully all developers relying on rJava have checked their packages. For the full list of fixes and changes see **NEWS**.

2009/10/27 - rJava 0.8-0 released. Many new features mostly thanks to Romain Francois -- check the **NEWS** file for details.

2009/08/22 - rJava 0.7-0 released. Recommended update (includes bugfixes to fields handling, new support for with(), method/field auto-completion and more forgiving `$` operator), includes JRI 0.5-0.

2008/09/22 - rJava 0.6-0 released. Adds support for Java serialization and R-side cache which enables Java objects to become persistent across sessions. See `?.jcache` in R (or the **online help**) for details.

2007/11/05 - rJava 0.5-1 released. Fixes issues with Windows; and minor updates (see NEWS).

2007/08/22 - rJava 0.5-0 released. This is a major update featuring many new features and bugfixes. It sports a new custom class loader, much improved (and faster) field support, integration of all native Java types, automatic fall-back to static methods, infrastructure for writing Java packages easily (see `.jpackage`), support for custom convertors and call-backs. Please read the **NEWS** file for details.

2007/02/27 - rJava 0.4-14 (update is recommended to all users due to memory leak fixes), please use CRAN to get the latest release. The current *development* version is **rJava 0.5-0** (available from here - see SVN access and download on the left). It is under heavy construction right now with many new features, so feel free to test-drive it if you want to be on the bleeding edge (it is a bit chatty as some debugging output is still enabled). Some of the highlights are memory profiler, intelligent class loader, easy Java package integration and callback support.

R interface to…

Advanced R by Hadley Wickham

Table of contents

Want a physical copy of this material? Buy a book from Amazon!

Contents

Calling C functions from R
C data structures
Creating and modifying vectors
Pairlists
Input validation
Finding the C source code for a function

How to contribute

Edit this page

R's C interface

Reading R's source code is an extremely powerful technique for improving your programming skills. However, many base R functions, and many functions in older packages, are written in C. It's useful to be able to figure out how those functions work, so this chapter will introduce you to R's C API. You'll need some basic C knowledge, which you can get from a standard C text (e.g., *The C Programming Language* by Kernigan and Ritchie), or from Rcpp. You'll need a little patience, but it is possible to read R's C source code, and you will learn a lot doing it.

The contents of this chapter draw heavily from Section 5 ("System and foreign language interfaces") of Writing R extensions, but focus on best practices and modern tools. This means it does not cover the old `.C` interface, the old API defined in `Rdefines.h`, or rarely used language features. To see R's complete C API, look at the header file `Rinternals.h`. It's easiest to find and display this file from within R:

```
rinternals <- file.path(R.home("include"), "Rinternals.h")
file.show(rinternals)
```

All functions are defined with either the prefix `Rf_` or `R_` but are exported without it (unless `#define R_NO_REMAP` has been used).

I do not recommend using C for writing new high-performance code. Instead write C++ with Rcpp. The Rcpp API protects you from many of the historical idiosyncracies of the R API, takes care of memory management for you, and provides many useful helper methods.

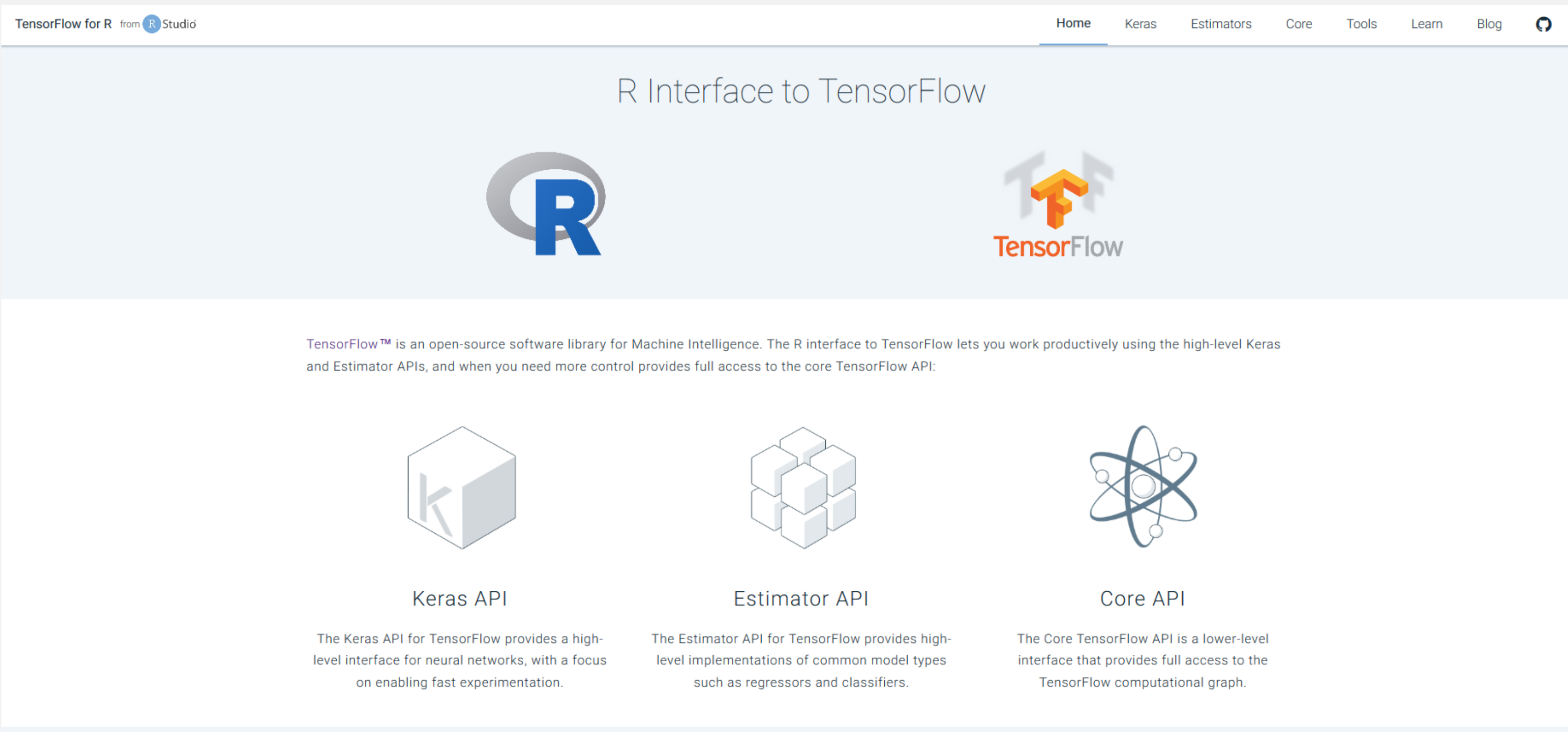
Outline

- Calling C shows the basics of creating and calling C functions with the inline package.
- C data structures shows how to translate data structure names from R to C.
- Creating and modifying vectors teaches you how to create, modify, and coerce vectors in C.
- Pairlists shows you how to work with pairlists. You need to know this because the distinction between pairlists and list is more important in C than R.
- Input validation talks about the importance of input validation so that your C function doesn't crash R.
- Finding the C source for a function concludes the chapter by showing you how to find the C source code for internal and primitive R functions.

Prerequisites

To understand existing C code, it's useful to generate simple examples of your own that you can experiment with. To that end, all examples in this chapter use the `inline` package, which makes it extremely easy to compile and link C code to your current R session. Get it by running `install.packages("inline")`. To easily find the C code associated with internal and primitive functions, you'll need a function from pryr. Get the package with `install.packages("pryr")`.

R interface to…



An introduction to TensorFlow

- TensorFlow is an open source software library for numerical computation using data flow graphs.
- Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them.
- The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.

An introduction to TensorFlow

- TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.
- Tensorflow is one of the widely used libraries for implementing algorithms involving large number of mathematical operations.
- A detailed guide can be found in www.tensorflow.org/guide

DataFlow Graph

- TensorFlow uses a **dataflow graph** to represent your computation in terms of the dependencies between individual operations. Then it is necessary to create a TensorFlow **session** to run parts of the graph across a set of local and remote devices.
- DataFlow is a common programming model for parallel computing. In a dataflow graph, the nodes represent units of computation, and the edges represent the data consumed or produced by a computation.

DataFlow Graph

- Dataflow has several advantages that TensorFlow leverages when executing your programs:
  - **Parallelism.** By using explicit edges to represent dependencies between operations, it is easy for the system to identify operations that can execute in parallel.
  - **Distributed execution.** By using explicit edges to represent the values that flow between operations, it is possible for TensorFlow to partition your program across multiple devices (CPUs, GPUs, and TPUs) attached to different machines. TensorFlow inserts the necessary communication and coordination between devices.

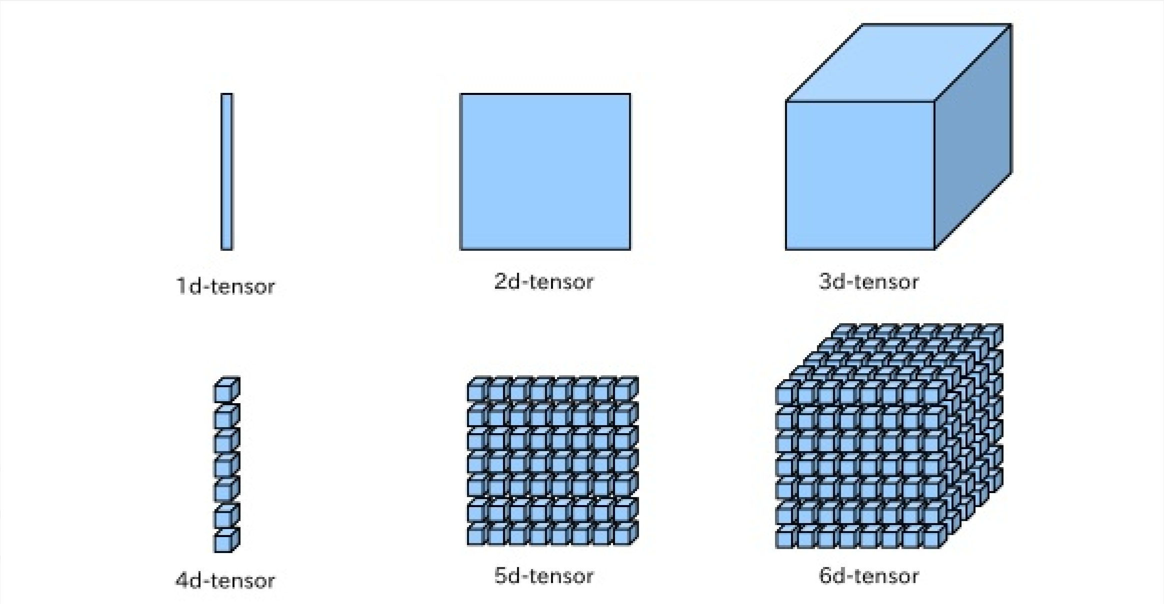
Tensors

- TensorFlow, as the name indicates, is a framework to define and run computations involving tensors. A **tensor** is a generalization of vectors and matrices to potentially higher dimensions. Internally, TensorFlow represents tensors as n-dimensional arrays of base datatypes.

Rank	Math entity
0	Scalar (magnitude only)
1	Vector (magnitude and direction)
2	Matrix (table of numbers)
3	3-Tensor (cube of numbers)
n	n-Tensor (you get the idea)

Tensors

- In particular, Tensor is a general name of multi-way array data. For example, 1d-tensor is a vector, 2d-tensor is a matrix and 3d-is a cube. We can image 4d-tensor as a vector of cubes, 5d-tensor is a matrix of cubes and 6d-tensor is a cube of cubes



Tensors

- When writing a TensorFlow program, the main object you manipulate and pass around is the *tf.Tensor* (in R *. -> $*).
- A *tf.Tensor* has (necessarily) the following properties:
  - a data type (float32, int32, ecc.)
  - a shape
- TensorFlow programs work by first building a graph of *tf.Tensor* detailing how each tensor is computed based on the other available tensors and then by running parts of this graph to achieve the desired results.
- Then, a *tf.Tensor* object represents a partially defined computation that will eventually produce a value.

Tensors

- Some types of tensors are special, the main are:
  - *tf.Variable*
  - *tf.constant*
  - *tf.placeholder*
- In general, with the exception of *tf.Variable*, the value of a tensor is immutable, which means that in the context of a single execution tensors only have a single value.
- Each element in the Tensor has the same data type, and the data type is always known. The shape (that is, the number of dimensions it has and the size of each dimension) might be only partially known. Most operations produce tensors of fully-known shapes if the shapes of their inputs are also fully known.

Variables

- A TensorFlow **variable** is the best way to manipulate your program.
- Before you can use a variable in a *session run*, it must be initialized and they have to be explicitly initialized.
- To initialize all trainable variables in one go, before training starts, call *tf$global_variables_initializer()*
- Just like any other TensorFlow operation, you can place variables on particular devices (in the case of more than one Graphic Devices)

Variables

- For example in the following case it is possible to set the session run on the GPU:0 by means *tf$device('/gpu:0')*

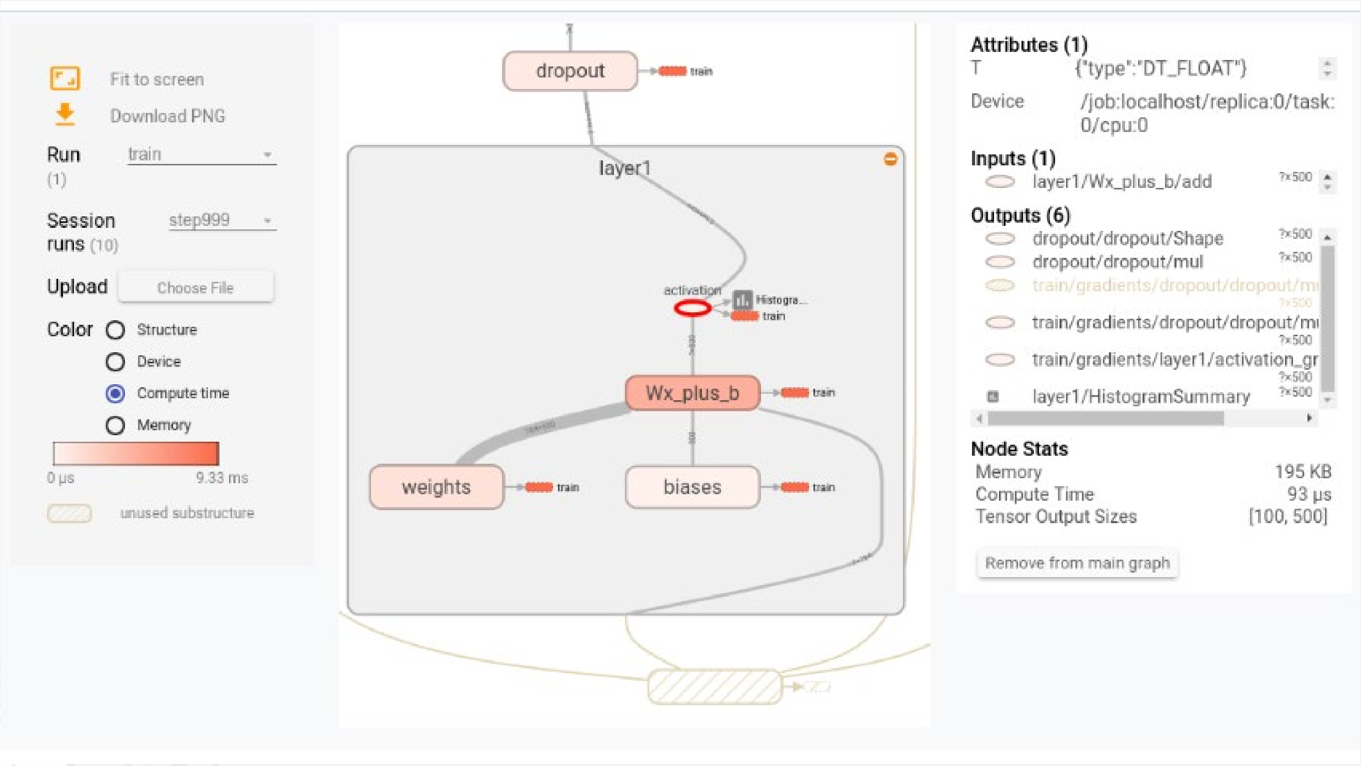
```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 375.26                 Driver Version: 375.26                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Graphics Device     Off  | 0000:02:00.0     Off |                  N/A |
| 39%   63C    P2   122W / 250W |   4449MiB / 12189MiB |     30%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Quadro K620         Off  | 0000:07:00.0     Off |                  N/A |
| 38%   50C    P8     1W /  30W |    228MiB /  1997MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

Session Run

- TensorFlow uses the *tf$Session* class to represent a connection between the client program (typically a Python program, although a similar interface is available in other languages, as R in this case) and the C++ runtime.
- A *tf$Session* object provides access to devices in the local machine, and remote devices using the distributed TensorFlow runtime.
- After the runtime, a *tf$Session* object have to be closed to free the resoures.

TensorBoard: Graph Visualization

- Often, TensorFlow computation graphs are powerful but complicated. The graph visualization can help you understand and debug them.



Welcome to the Traffic Flow Reconstruction

Overview

We aim to improve the urban mobility through a **general** and **self-adaptive** model for a **low-cost** traffic reconstruction at **real-time** in **every position** of the city.

We propose to use a **fluid-dynamic traffic model** on road networks getting the road infrastructure and traffic restrictions from the **Open Street Map** and the traffic sensors specifications and detections from the **publicly available Open Data**.

Features

Low-cost
It uses stationary sensors that were already deployed in the city.

Unobtrusive
It does not require users to take any action (install app, submit data…).

Visual
Traffic flows are displayed on a street map through colored lines.

Open
Methods and software are made available under an open license.

Real-Time
The reconstruction is updated after every new traffic sensor detection.

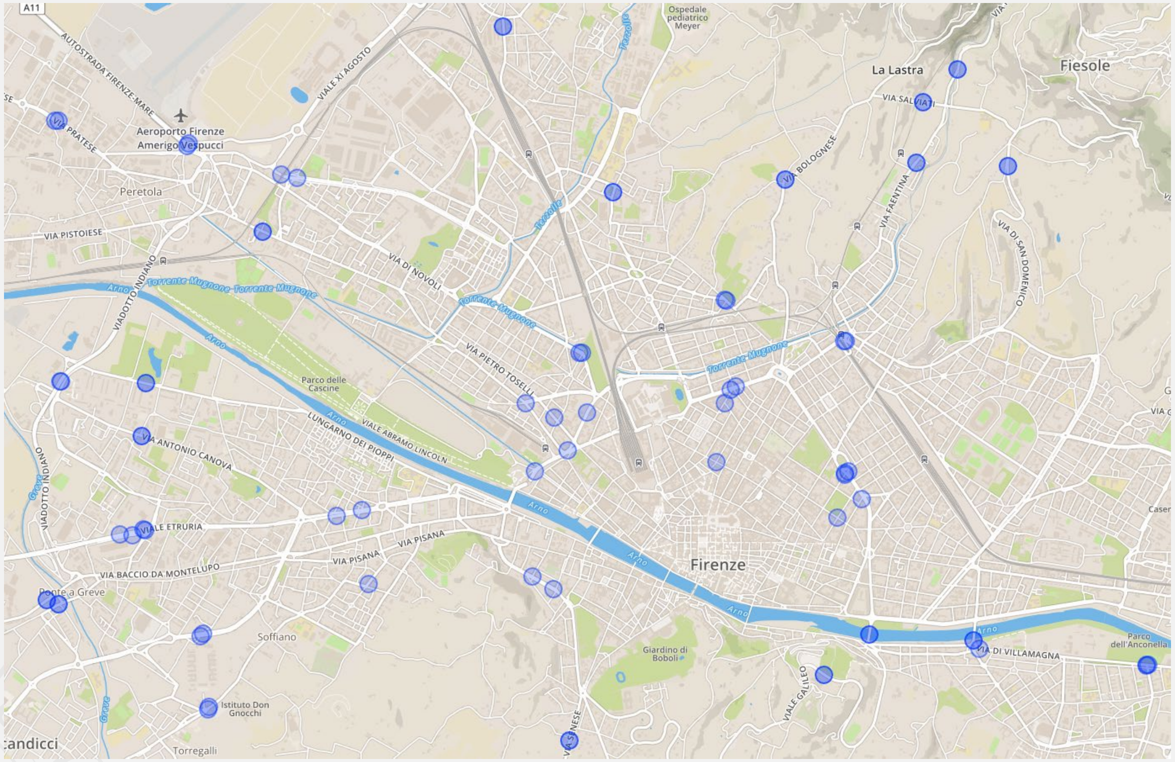
General
No simplistic assumption is made about the street graph.

Dense
The reconstruction is made at every location in the area of interest.

Verified
The accuracy of the reconstruction has been rigorously verified.

Target

- Starting from fixed traffic sensors scattered in the city, our scope is the prediction/reconstruction of the real-time vehicular traffic density in the whole urban network.



Ingestion process

- Given a generic geographic area of observation, an ingestion process is needed in the *Km4city* Knowledge Base to import both the detailed real street paths and the sensors data placed in the selected area

- The traffic data from stationary sensors in a municipality (spire road sensors and cameras) give the state of the traffic counting the number of vehicles which pass through the supervised area and the data are simultaneously uploaded 10 minutes.

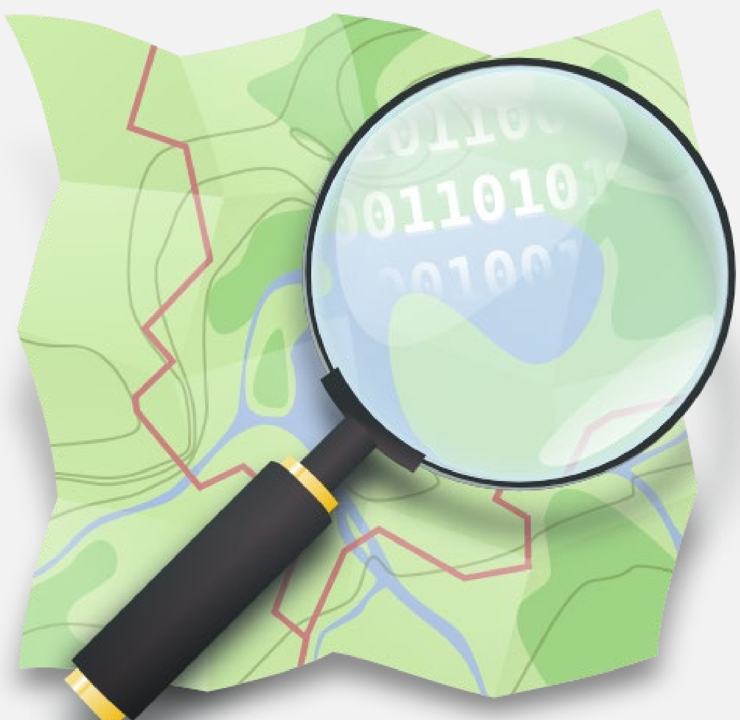
Km4City – A Knowledge Model for Smart Cities

An **Open Urban Platform for a Sentient Smart City**, aimed at:

- Implementing the city vision;
- Monitoring the city evolution;
- Providing new services for improving the quality of life of the citizens;
- Supporting the economic grow of the city;
- Promoting virtuous behaviours.

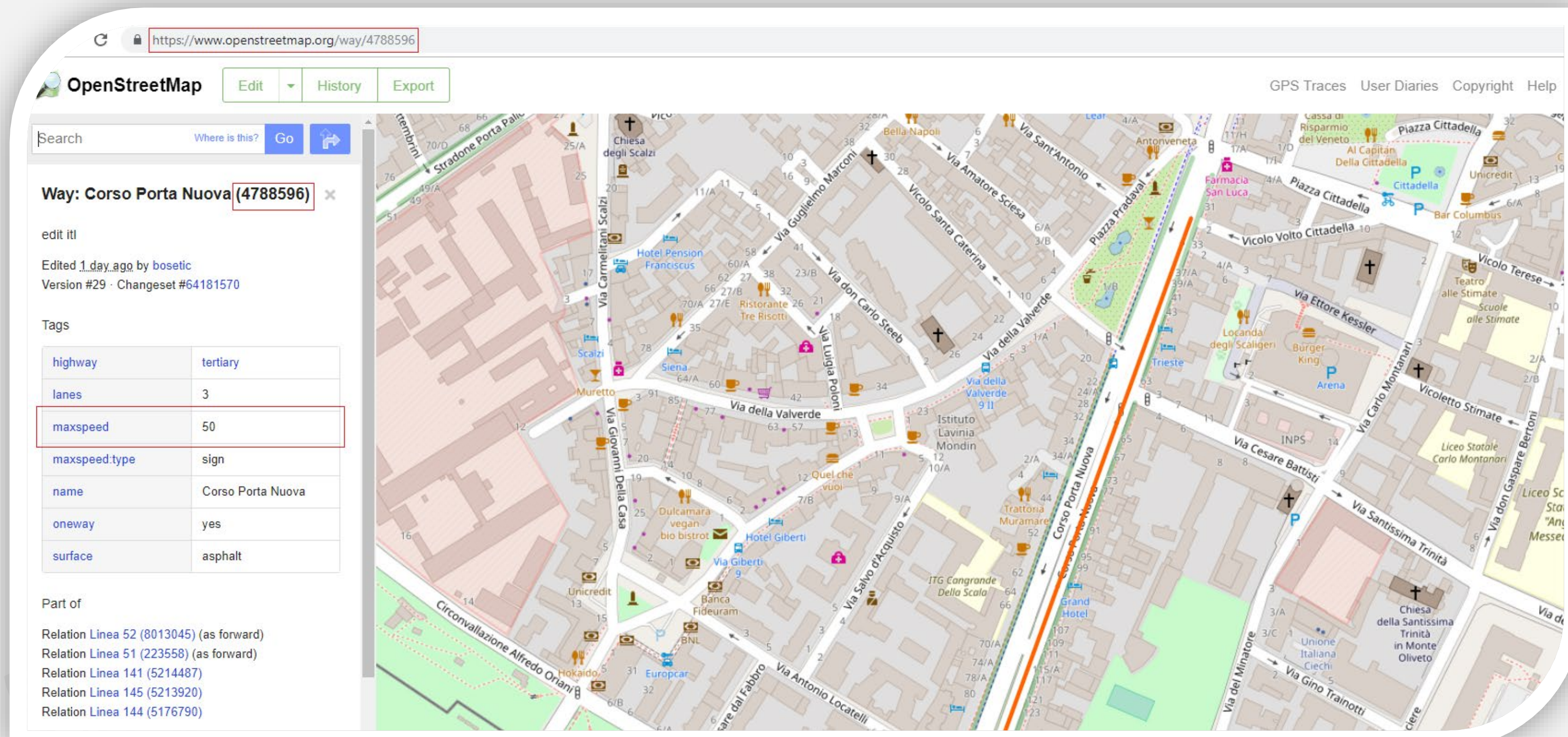
Briefly, we aim to support **cities that produce** with happy and proud **citizens** and with crowds of enthusiastic **tourists** and **investors**.

Open Street map



- OpenStreetMap powers map data on **thousands** of web sites, mobile apps, and HW devices.
- It is built by a **community** of mappers that contribute and maintain data about roads, trails, cafés, railway stations, and much more, all over the world.
- It provides **open data**: you are free to use it for any purpose as long as you credit OpenStreetMap and its contributors.
- **OSM data is stored in a RDB, and then transformed and stored in a triplestore, based on a mapping of the OSM data model to the Km4City Ontology street graph modelling.**

Data from OSM



Sensors and detections

- Traffic Sensors static information (identifier, geolocation, street address, technical specifications…) and the traffic flow detections (sensor, timestamp, detected traffic flow, estimated speed…) all come from publicly available Open Data.
- They are managed through ETL processes, and stored in a No-SQL database.
- The traffic reconstruction model implementation accesses those data through dedicated APIs. Traffic flows are read every 10 minutes, the refresh frequency of the traffic sensors.

Mathematical model

- The traffic sensor detections are interpreted as *sources of traffic* leading into the outcoming roads of the nodes where sensors are located.
- We consider a mathematical model for fluid dynamic flows on networks which is based on conservation laws.
- Road network is studied as a directed graph composed by arcs that meet at some nodes, corresponding to junctions.

Mathematical model

- We deal with the fluid-dynamic models which can be seen as a macroscopic model which allows to observe the network in the time evolution through waves formation.
- In a single road, this nonlinear model is based on the conservation of cars described by the following scalar hyperbolic conservation law (1)

$$\frac{\partial \rho(t,x)}{\partial t} + \frac{\partial f\big(\rho(t,x)\big)}{\partial x} = 0$$

$$\rho(t,a) = \rho_a(t) \text{ and } \rho(t,b) = \rho_b(t)$$

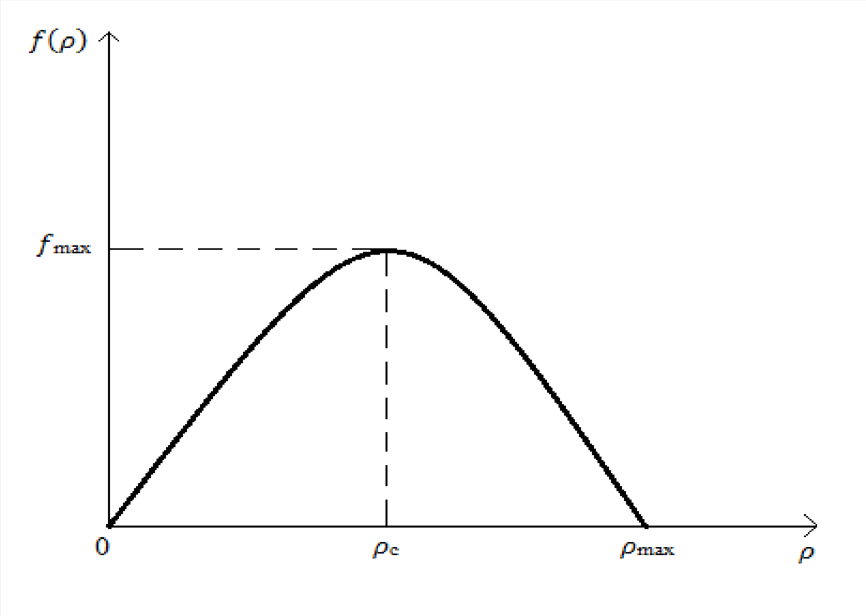
$$\rho(0,x) = \rho_0(x).$$

Mathematical model

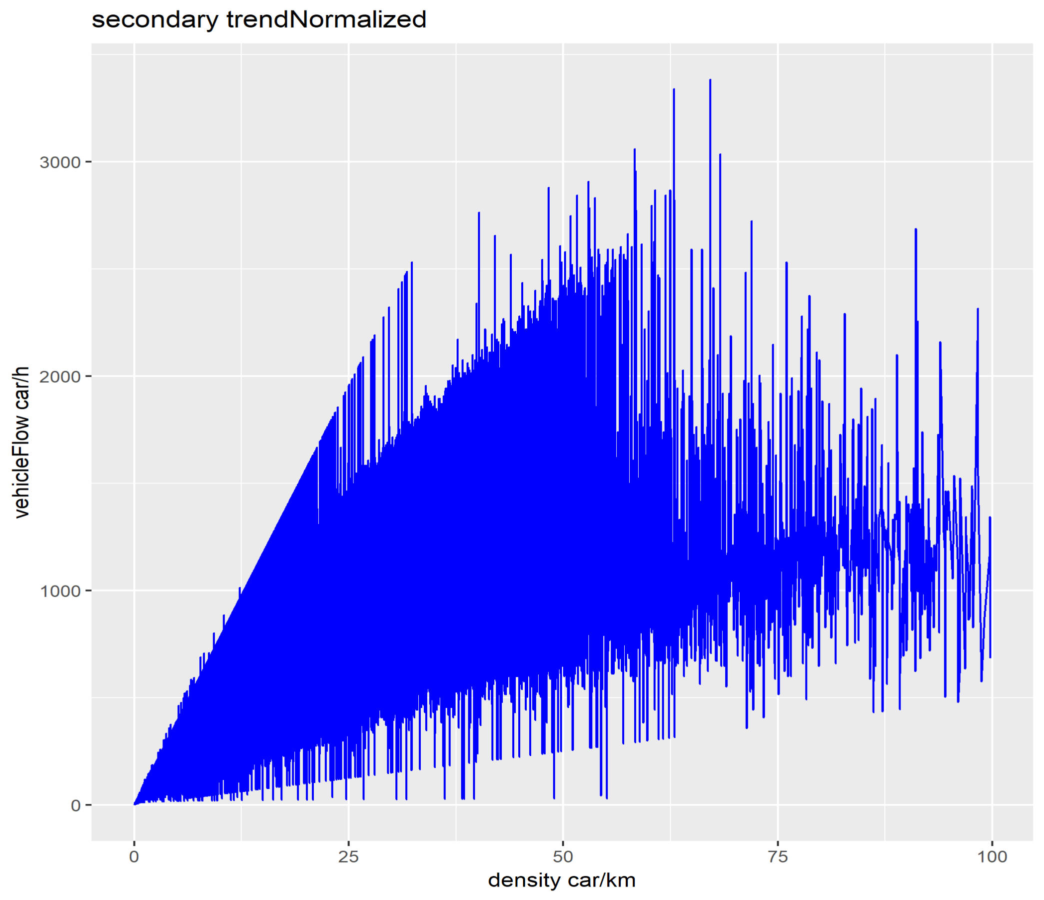
- $\rho(t, x)$ denotes the car density which admits values from 0 to $\rho_{\max}$, where $\rho_{\max} > 0$ is the maximal vehicular density on the road.
- The function $f(\rho(t, x))$ is the vehicular flux which is defined as the product $\rho(t, x)v(t, x)$, where $v(t, x)$ is the local speed of the cars.
- As usually it happens, in the case of first order approximation, if we assume that $v(t, x)$ is a decreasing function, only depending on the density, then the corresponding flux is a concave function

Fundamental diagram

- We consider the local cars' speed as $v(\rho) = v_{max}(1 - \frac{\rho}{\rho_{max}})$ obtaining that $f(\rho) = v_{max}\left(1 - \frac{\rho}{\rho_{max}}\right)\rho$, where v_{max} is the limit speed



Fundamental diagram: data analysis



Discretization scheme

The following discretization and simplification of the model is operated:

- Each road is partitioned in segments Δ*x* long.
- The time is partitioned in intervals Δ*t* long.
- Denote (h,m) a bounded time-space region (cell) of duration h and length m. Let $u_m^h = u(t_h, x_m) = u(h\Delta t, m\Delta x)$ be a continuous function defined on (h,m). Denote *F* the numerical flux. Then, the vehicular density results from:

$$u_m^{h+1} = u_m^h - \frac{\Delta t}{\Delta x}\left(F\left(u_m^h, u_{m+1}^h\right) - F\left(u_{m-1}^h, u_m^h\right)\right)$$

Sensors' measurements

- The measured data sensor is interpreted as the source of traffic leading into the outcoming roads of the considered junction.
- suppose to assign a condition at the incoming boundary for $x = 0$ as $\rho(t,0) = \rho_b^{inc}(t)$.
- we proceed by inserting an "incoming ghost cell" and the discretization becomes

$$u_0^{h+1} = u_0^h - \frac{\Delta t}{\Delta x}\left(F\left(u_0^h, u_1^h\right) - F\left(v_{(inc)}^h, u_0^h\right)\right)$$

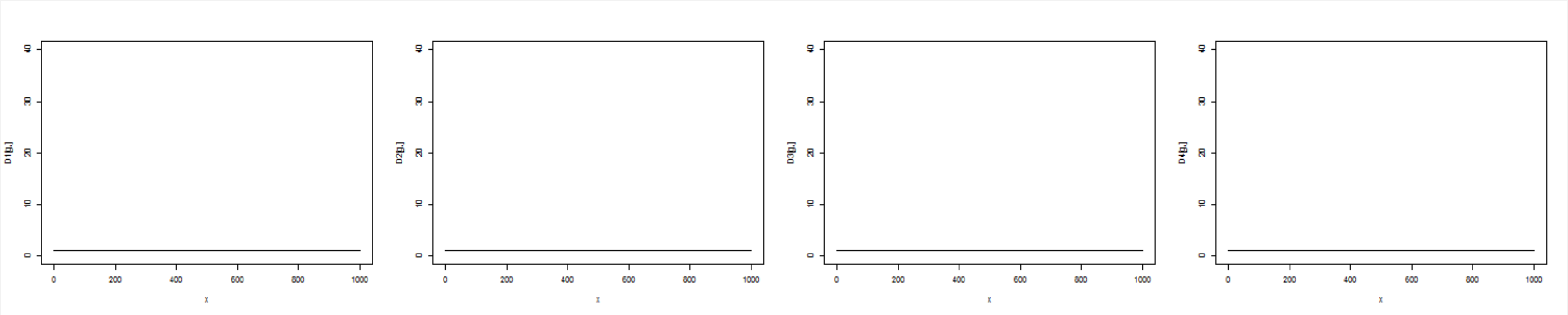
where

$$v_{(inc)}^h = \frac{1}{\Delta t}\int_{t_h}^{t_h+1} \rho_b^{inc}(t)\,dt$$

- replaces the "ghost value" u_{-1}^h

Fluid Dynamics Approach

Phisical Principle of Narrowing in roads



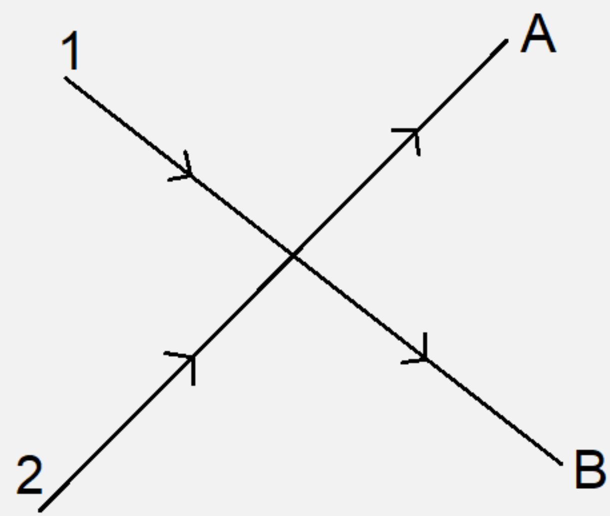
Numerical meaning

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]	[,11]
[1,]	9.343250	8.938677	8.563212	8.201665	7.848900	7.502591	7.161543	11.50547	19.58907	20.61252	20.74469
[2,]	9.346269	8.943543	8.569777	8.209852	7.858653	7.513860	7.174282	12.09272	19.73380	20.63228	20.75818
[3,]	9.349261	8.948364	8.576283	8.217966	7.868318	7.525029	7.186909	12.70006	19.86189	20.65113	20.77141
[4,]	9.352225	8.953141	8.582730	8.226007	7.877897	7.536099	7.199425	13.32546	19.97522	20.66917	20.78439
[5,]	9.355162	8.957875	8.589120	8.233976	7.887391	7.547071	7.234340	13.94458	20.07550	20.68647	20.79712
[6,]	9.358073	8.962567	8.595452	8.241873	7.896801	7.557947	7.301140	14.54628	20.16430	20.70311	20.80961
[7,]	9.360957	8.967216	8.601727	8.249701	7.906128	7.568729	7.404075	15.12479	20.24300	20.71915	20.82187
[8,]	9.363816	8.971824	8.607947	8.257460	7.915373	7.579416	7.546541	15.67538	20.31287	20.73464	20.83391
[9,]	9.366649	8.976391	8.614111	8.265150	7.924538	7.590011	7.730996	16.19439	20.37501	20.74963	20.84572
[10,]	9.369457	8.980917	8.620222	8.272773	7.933623	7.600514	7.958933	16.67929	20.43040	20.76415	20.85733
[11,]	9.372240	8.985404	8.626279	8.280330	7.942629	7.610928	8.230920	17.12855	20.47992	20.77825	20.86873
[12,]	9.374999	8.989851	8.632282	8.287821	7.951557	7.621252	8.546680	17.54164	20.52431	20.79196	20.87993
[13,]	9.377733	8.994260	8.638234	8.295247	7.960409	7.631488	8.905205	17.91883	20.56425	20.80531	20.89094
[14,]	9.380444	8.998630	8.644134	8.302610	7.969186	7.641638	9.304891	18.26110	20.60031	20.81832	20.90176
[15,]	9.383131	9.002962	8.649984	8.309909	7.977887	7.651701	9.743678	18.56995	20.63300	20.83100	20.91240
[16,]	9.385794	9.007257	8.655783	8.317146	7.986515	7.661680	10.219185	18.84726	20.66275	20.84339	20.92286
[17,]	9.388435	9.011515	8.661533	8.324322	7.995070	7.671576	10.728829	19.09520	20.68994	20.85550	20.93315
[18,]	9.391053	9.015736	8.667233	8.331437	8.003553	7.681388	11.269937	19.31607	20.71491	20.86734	20.94328
[19,]	9.393649	9.019922	8.672886	8.338492	8.011965	7.691119	11.839829	19.51221	20.73794	20.87894	20.95324
[20,]	9.396222	9.024072	8.678491	8.345487	8.020306	7.700770	12.435887	19.68596	20.75928	20.89029	20.96304
[21,]	9.398774	9.028188	8.684048	8.352424	8.028579	7.714870	13.051077	19.83957	20.77913	20.90142	20.97269
[22,]	9.401305	9.032268	8.689560	8.359304	8.036783	7.752828	13.663633	19.97518	20.79769	20.91233	20.98219
[23,]	9.403814	9.036315	8.695025	8.366126	8.044919	7.819804	14.266206	20.09478	20.81511	20.92304	20.99155
[24,]	9.406302	9.040327	8.700445	8.372892	8.052989	7.920452	14.852134	20.20021	20.83154	20.93355	21.00076
[25,]	9.408769	9.044307	8.705820	8.379603	8.060992	8.058718	15.415642	20.29316	20.84708	20.94387	21.00984
[26,]	9.411216	9.048254	8.711151	8.386258	8.068931	8.237704	15.951981	20.37513	20.86184	20.95401	21.01878
[27,]	9.413643	9.052168	8.716438	8.392859	8.076805	8.459583	16.457502	20.44748	20.87592	20.96397	21.02758
[28,]	9.416049	9.056050	8.721682	8.399407	8.084615	8.725588	16.929655	20.51143	20.88938	20.97376	21.03626
[29,]	9.418436	9.059900	8.726883	8.405902	8.092363	9.036048	17.366946	20.56805	20.90229	20.98339	21.04482
[30,]	9.420804	9.063719	8.732042	8.412344	8.100048	9.390478	17.768828	20.61828	20.91472	20.99286	21.05325
[31,]	9.423152	9.067507	8.737160	8.418734	8.107672	9.787692	18.135581	20.66297	20.92670	21.00218	21.06157
[32,]	9.425481	9.071265	8.742237	8.425074	8.115236	10.225939	18.468159	20.70282	20.93829	21.01135	21.06976
[33,]	9.427792	9.074992	8.747273	8.431363	8.122740	10.703041	18.768044	20.73849	20.94951	21.02038	21.07785

Application of the model

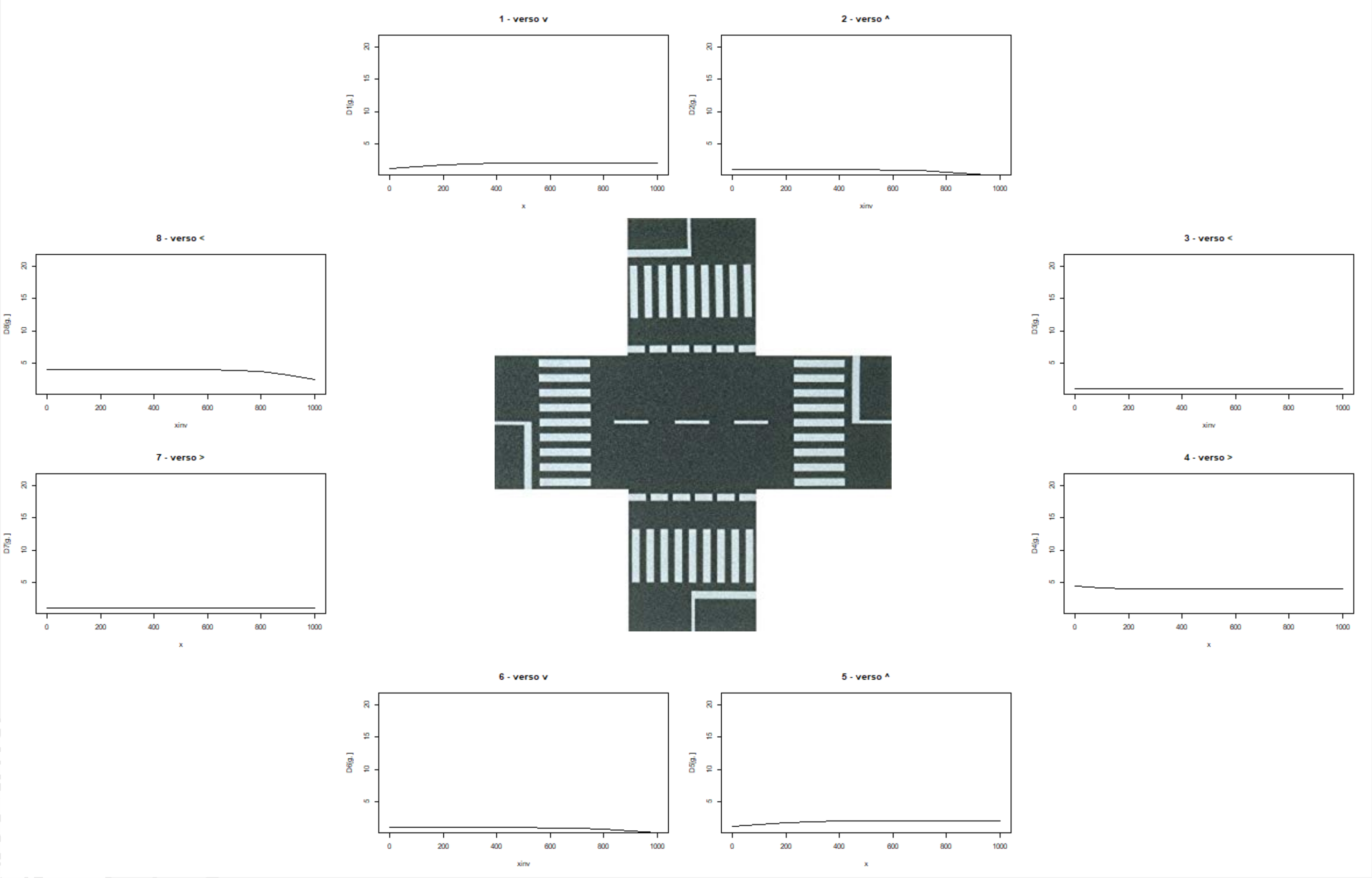
- For each time slot *t*, each traffic sensor detection is interpreted as a **source of traffic** that leads into the segments of road that origin from the node where the sensor is located that has produced the data.
- The vehicular traffic flow is propagated in the network according to the fluid dynamic model in (1).
- The distribution of the traffic at crossroads is governed by a **Traffic Distribution Matrix** whose coefficients are based on the ***weights*** of the segments of roads that make the crossroad.

Traffic distribution on a junction



$$\begin{pmatrix} d_{1A} & d_{2A} \\ d_{1B} & d_{2B} \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} f_A \\ f_B \end{pmatrix}$$

Fluid Dynamics of a junction

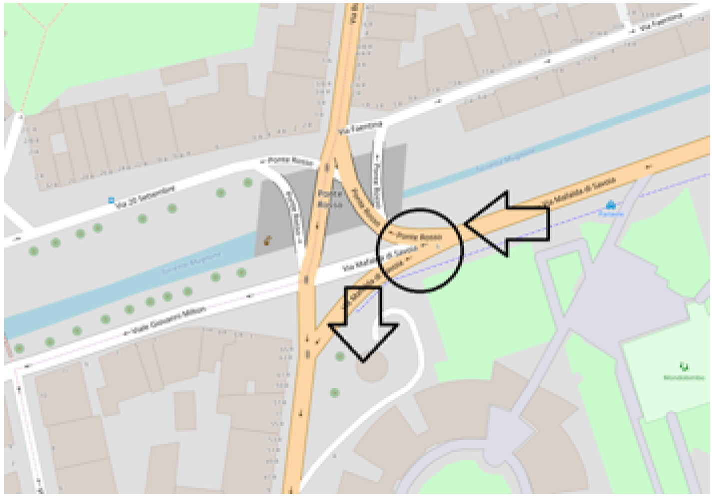


Junction distribution learning

The fork of via Mafalda di Savoia (East), in via Mafalda di Savoia (South), Viale Giovanni Milton (West) and Via del Ponte Rosso (North), in Florence.

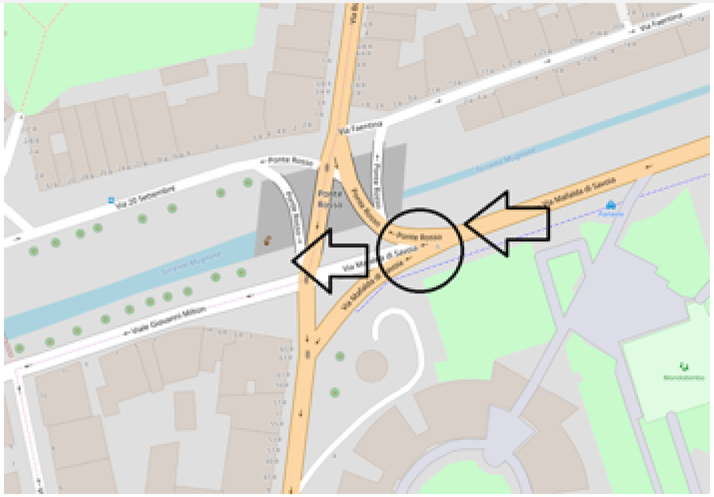
Junction distribution learning

Road Type: primary
Lanes: 2
Designated Lanes: 0
Restrictions: none
Learning Factor: 61
Elem. Type: T.O.C.
Length: 63
Direction: positive
…
Weight: 31.122%



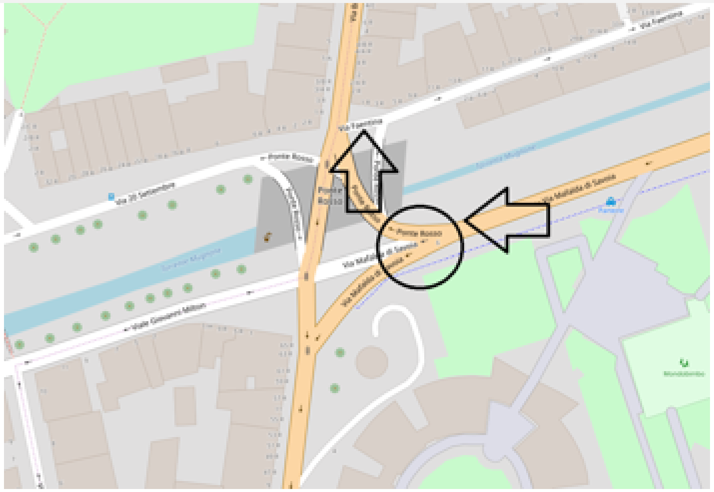
Junction distribution learning

Road Type: tertiary
Lanes: 1
Designated Lanes: 0
Restrictions: none
Learning Factor: 24
Elem. Type: T.O.C.
Length: 51
Direction: positive
…
Weight: 12.245%



Junction distribution learning

Road Type: primary
Lanes: 2
Designated Lanes: 0
Restrictions: none
Learning Factor: 111
Elem. Type: T.O.C.
Length: 60
Direction: positive
…
Weight: 56.633%



Weight initialization

Weights are **initialized** based on the following:

- **Road type**: motorway, trunk, primary, secondary, tertiary, unclassified, residential, service;
- **Lanes**: how many lanes are drawn on the asphalt, also considering possible restrictions (e.g. lanes reserved to public transport);
- **Traffic restrictions**: examples are mandatory/forbidden directions at crossroads, speed limits, limited traffic zones.

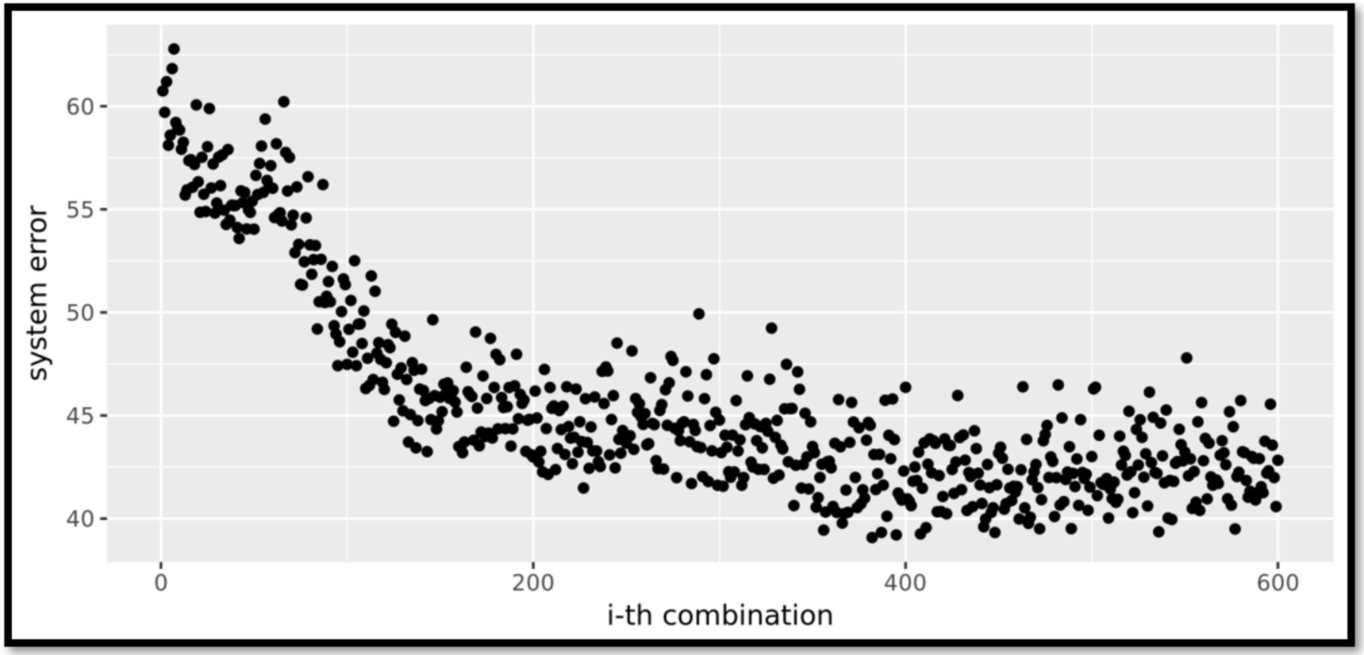
Stochastic learning

It has been observed that:

- The way how vehicles distribute at crossroads varies depending of the day of the week, and of the time of the day;
- A random variation of some weights is very likely to lead to an improved accuracy;
- If no improvements are achieved after *n* attempts, it is reasonable to move anyway to the best of the last *n* configs.

An offline process is run, based on the above, that leads to time-based weight adjustments, aimed at an improved accuracy.

Stochastic learning



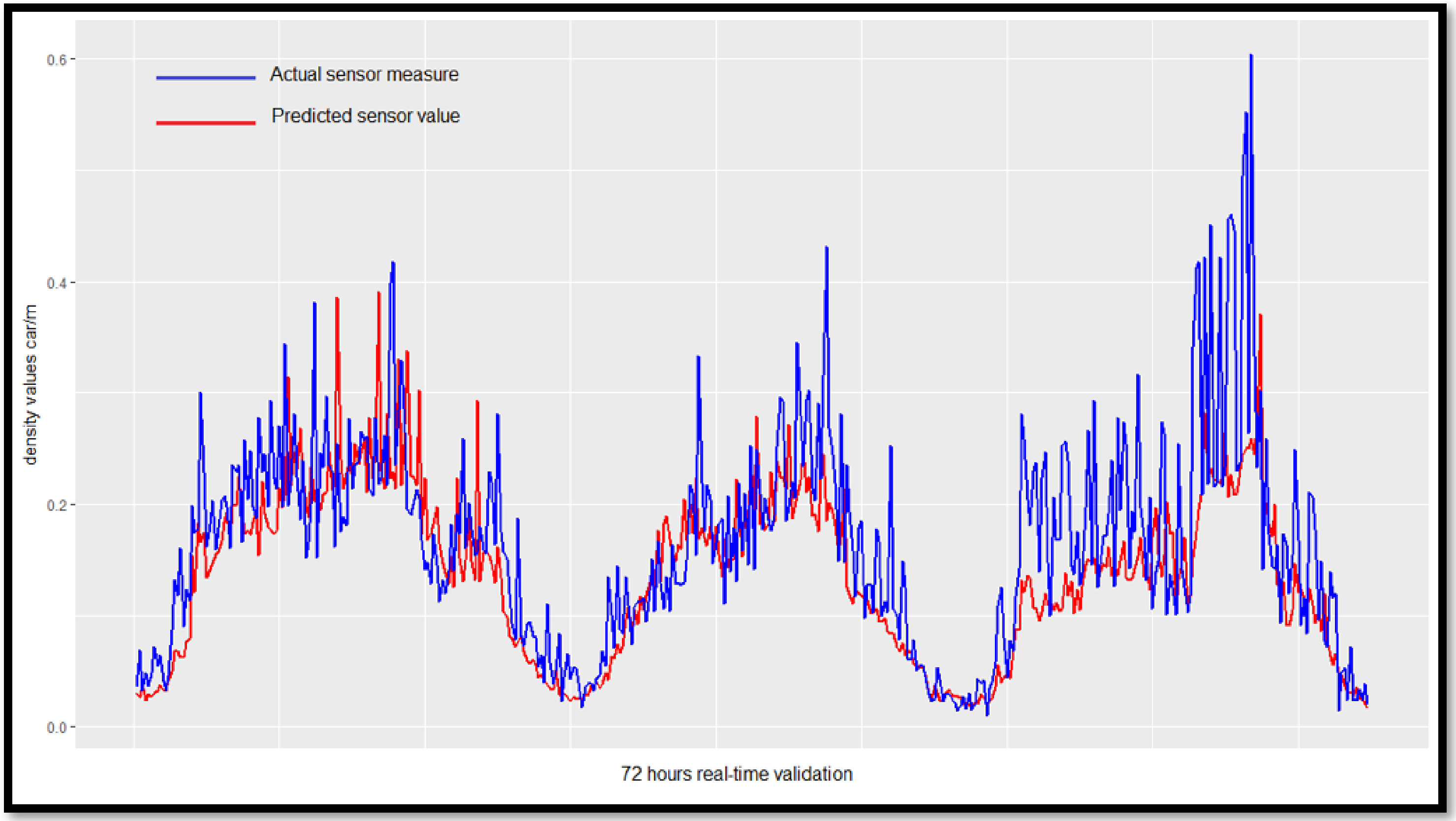
In the x axis, the number of the learning iterations. In the y axis, the (decreasing) system error.

Validation approach

- Let the **error at a sensor at a given time *t*** be the percentage error computed removing a *given sensor* from the inputs and comparing the traffic flow *reconstructed* at the sensor with the traffic flow *detected* by the sensor, at the given time.
- Let the **system error over a time period *T*** be the average of the system errors computed over all the traffic sensors and all the times $t \in T$.

The system error has been computed to be the **30%** about.

Validation approach



The diagram refers to one in particular of the sensors, and it displays the predicted vs actual values over the time in the 72 hours validation.

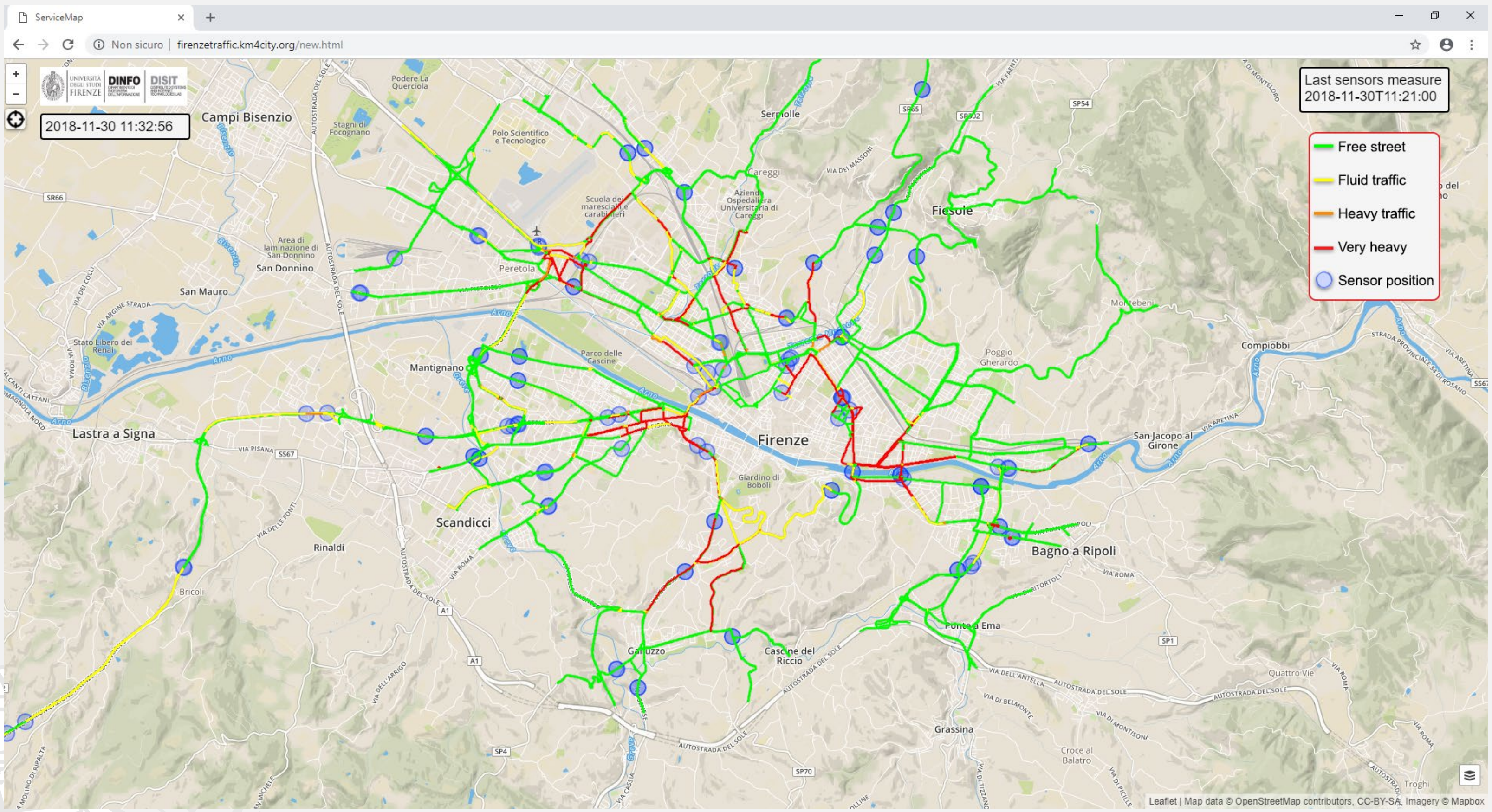
Displaying results

- Segments of road are categorized based on the road type and the number of lanes.
- Segments of each category that have one at least of the extremities that coincide with a traffic sensor, are used for determining the range of the traffic flows that can be observed on the specific category of segments.
- For each segment category, the range is partitioned into four subranges, that correspond to the four colors that you can find on the map.
- The reconstruction is presented to users through colored lines traced over the road paths on the city map.
- The date and time when the most up-to-date values from the sensors have been acquired can also be seen at the top-right corner of the map.

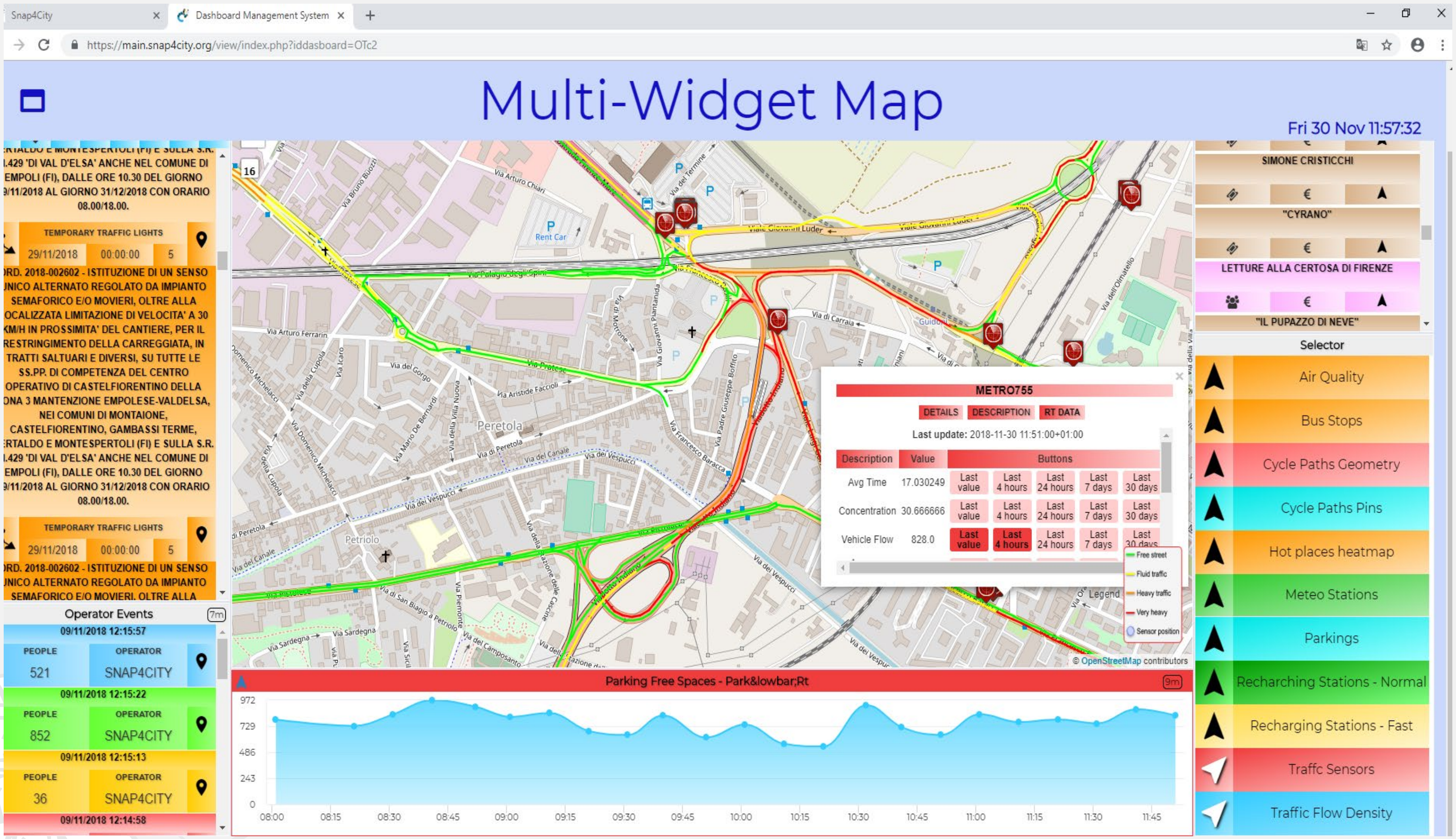
Displaying results

- Segments of road are categorized based on the road type and the number of lanes.
- Segments of each category that have one at least of the extremities that coincide with a traffic sensor, are used for determining the range of the traffic flows that can be observed on the specific category of segments.
- For each segment category, the range is partitioned into four subranges, that correspond to the four colors that you can find on the map.
- The reconstruction is presented to users through colored lines traced over the road paths on the city map.
- The date and time when the most up-to-date values from the sensors have been acquired can also be seen at the top-right corner of the map.

Displaying results



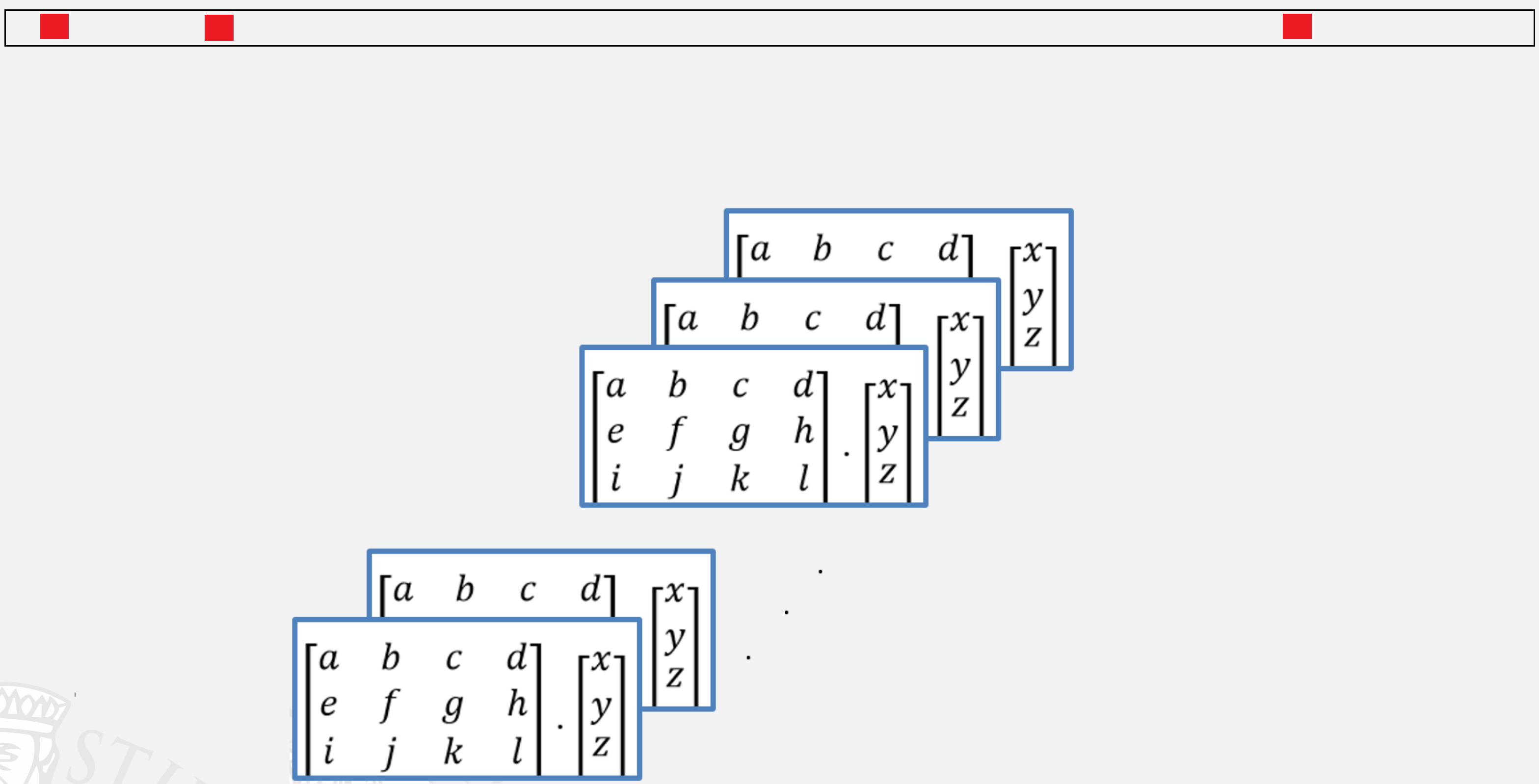
Displaying results: on dashboard



Data structure in real-time computing

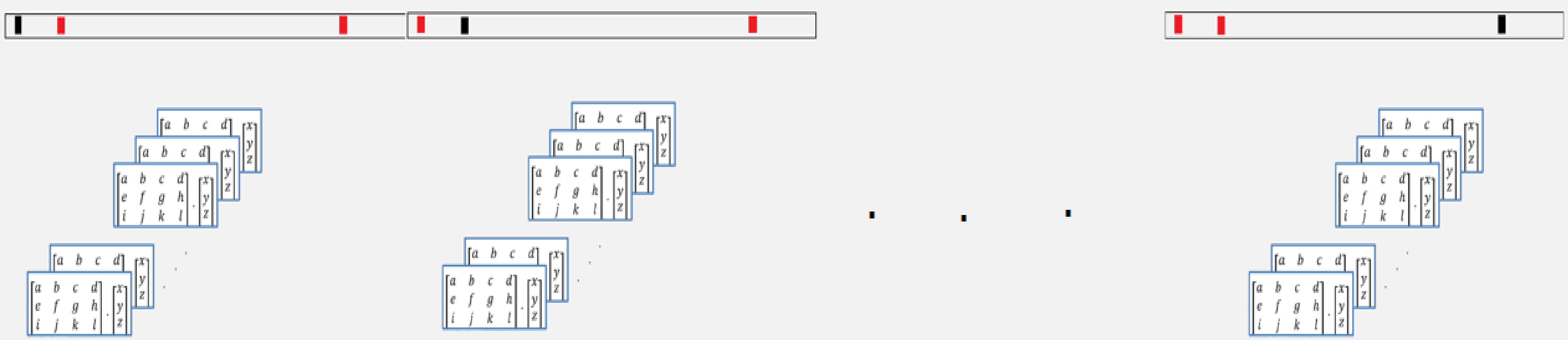
- A network area of Florence consisting of 173 data sensors (■), 1532 junctions and 1377 road-segments giving the estimation of the vehicular density in 31217 road-units having length 20/30 meters is considered to test the model
- Parallel computing of road-units by 1d-tensor (complex indexing)
- Parallel computing of junctions by 3d-tensor
- Each update of the sensors is every 10 minutes.

Graphical idea of real-time computing



Data structure in Stochastic Learning

- The Graph structure is repeated for about 60 removed sensors obtaing a concatenated data structure having about 2000000 road-units computed in parallel.



Thanks for your attention

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research