Il pacchetto bupaR: Business Process Analytics in R

Elaborato di Statistica Applicata e Analisi dei Dati

Emanuele Lena, Riccardo Belliato

13/02/2021

Introduzione

Il seguente elaborato tratta la tematica dell'analisi di dati relativi a processi (process mining) tramite il linguaggio R; in particolare, si presenterà il pacchetto `bupaR` e le sue librerie.

Nel primo capitolo si farà una panoramica generale sulla suite `bupaR`, sulle sue librerie e sul tipo di funzionalità che vengono offerte. Dopo di che, nel secondo si introdurrà il concetto di log di processo, si presenterà la struttura attesa per questo genere di dati e si illustreranno le informazioni principali solitamente contenute.

Nel terzo capitolo si tratterà la tematica dell'EDA nel contesto dell'analisi dei processi; si presenteranno quindi alcuni strumenti per l'estrazione di informazioni da un log di processo e per la generazione di grafici. Il tutto con il supporto di diversi esempi e con alcuni richiami ai classici strumenti della statistica.

Nel quarto capito si presenteranno alcuni esempi di rappresentazioni grafiche tipiche dell'analisi dei processi, come ad esempio le mappe di processo e le matrici di precedenza. Nel sesto capitolo si mostrerà come tali visualizzazioni sono integrabili all'interno di dashboard dinamiche accessibili tramite web app.

Nel quinto capitolo infine si tratterà brevemente la tematica del pre-processamento, cioè dell'attività di pulizia, subsetting e verifica della qualità dei dati.

Si noti che tale elaborato è solo un'introduzione (e anche molto "pratica") alla vasta tematica del process mining. La suite `bupaR` presenta molte funzionalità avanzate che in questo elaborato non verranno trattate. Ad esempio, non si approfondirà molto tutto ciò che riguarda la pulizia e strutturazione dei dati, in quanto sono tematiche più correlate all'ambito della data science rispetto che alla statistica. Allo stesso modo non si approfondiranno nemmeno alcuni particolari algoritmi avanzati di esplorazione dei processi e di ricostruzione della loro struttura.

I processi, l'analisi di processi e il process mining

Il process mining è un'attività che si colloca a metà tra il data mining e l'analisi e la modellazione di processi. L'analisi e la modellazione di processi è le disciplina che si occupa dello studio e del miglioramento delle attività svolte all'interno delle organizzazioni; tale attività viene tipicamente svolta da manager e ingegneri gestionali. Il process mining è una tecnica a supporto dell'analisi dei processi.

L'idea alla base del process mining è che all'interno dei sistemi informativi moderni di un'azienda capita spesso di raccogliere informazioni relative ai processi. Tali informazioni si concretizzano nella registrazione (in file di log, oppure anche in basi di dati classiche) di eventi, attività che vengono svolte, casi che vengono trattati, esecuzione di procedure strutturate e altri aspetti legati ai processi. A partire da tali dati si può svolgere un'attività di process mining, cioè di estrazione di informazioni rilevanti riguardo la struttura, i volumi, ecc.

Se si desidera approfondire tale tematica, si suggerisce la lettura di questo articolo (Aalst et al. 2011).

Contents

1 La suite bupaR: caratteristiche e funzionalità

`bupaR` (Business Process Analytics in R) è una suite integrata di pacchetti R per la manipolazione e l'analisi dei dati provenienti dai processi aziendali.

La suite bupaR è sviluppata dal Business Informatics research group dell'università di Hasselt, in Belgio, ed è open-source (il codice è consultabile al seguente link: https://github.com/bupaverse). Esiste inoltre un sito ufficiale del progetto: https://bupar.net/. Le informazioni che si riportano in questo elaborato provvengono dalla documentazione ufficiale (bupar.net, n.d.).

L'intera suite di pacchetti è disponibile su CRAN, ed è quindi installabile tramite il comando `install.packages`

```
install.packages("bupaR")
install.packages("eventdataR")
install.packages("xesreadR")
install.packages("edeaR")
install.packages("processmapR")
install.packages("processmonitR")

library(bupaR) # Carica tutte le librerie della suite nella sessione corrente
```

La suite bupaR è stata sviluppata nell'ambito della data science, per cui dipende da alcuni pacchetti R tipicamente utilizzati in questo contesto, in particolare i pacchetti `dplyr`, `tidyr` e `magrittr`, provenienti dalla suite tidyverse. I primi due pacchetti servono a manipolare datasets e data frame di grandi dimensioni in modo efficiente, il terzo invece introduce il costrutto sintattico `%>%`, che serve a concatenare le funzioni (nello specifico l'argomento a sinistra è il primo parametro dell'argomento a destra). Qui sotto è possibile visualizzare un esempio di utilizzo di questo costrutto:

```
patients %>% activity_frequency(level = "activity") %>% plot()
```

Questa scrittura è equivalente al seguente codice R

```
plot(
  activity_frequency(patients, level = "activity")
)
```

Si introducono ora brevemente i principali pacchetti che compongono la suite, il loro utilizzo verrà approfondito nei successivi capitoli di questo elaborato.

1.1 bupaR

`bupar` è la libreria principale della suite, da cui dipendono tutte le altre. Il suo scopo principale è quello di definire la classe di oggetti `eventlog`, insieme a una serie di funzioni per l'ottenimento di informazioni di base su questi ultimi.

1.2 eventdataR

Si tratta di un repository di oggetti `eventlog` di esempio:

- `patients` : log contenente alcune registrazioni fittizie di prestazioni sanitarie (raggi X, esami del sangue, visite mediche, ...). La struttura del dataset è molto semplice (contiene infatti solo le sei colonne richieste da `eventlog`).
- `hospital` : log contenente 150291 eventi registrati tra il 2005 e il 2008 in ospedale olandese. Questo dataset è stato creato per il Business Process Intelligence Contest (BPIC) del 2011.
- `hospital_billings` : log ottenuto registrando gli eventi relativi ai pagamenti di alcune prestazioni sanitarie in un ospedale.

- `sepsis` : in questo dataset sono memorizzati 1050 casi di pazienti colpiti da setticemia all'interno di un ospedale. Di ogni paziente è stato registrato il percorso compiuto all'interno dell'ospedale, in termini di prestazioni ospedaliere effettuate.
- `traffic_fines` : log relativo all'emissione e al pagamento di 10000 multe stradali.

Tutti i dataset, tranne `patients`, sono stati creati a partire da log di eventi reali, opportunamente modificati per garantire l'anonimato dei soggetti coinvolti.

1.3 xesreadR

Pacchetto R per la lettura e la scrittura degli oggetti `eventlog` da e su file in formato XES, uno schema XML standard (IEEE 1849-2016).

1.4 edeaR

In questo pacchetto sono contenute tutte le funzioni per il subsetting e l'analisi esplorativa e descrittiva dei log di processo.

1.5 processmapR

Questo pacchetto fornisce una serie di funzioni per visualizzare graficamente la struttura dei processi. Un esempio di queste visualizzazione è la `process_map`, ossia il grafo che rappresenta il flusso delle varie attività di un processo in rapporto ai casi analizzati.

1.6 processmonitR

Crea delle dashboard interattive (utilizzando il pacchetto `shiny`) combinando metriche e visualizzazioni provenienti dagli altri pacchetti di bupaR. Queste dashboard possono essere utilizzate standalone o integrate in altre applicazioni.

1.7 PM4Py, Process mining e Process Discovery

Attraverso il pacchetto `pm4py`, bupaR è in grado di interfacciarsi all'omonima libreria scritta in Python per eseguire operazioni di Process Mining.

Il Process Mining è un insieme di algoritmi e di tecniche che servono a ricostruire la struttura di un processo sconosciuto a partire dal log degli eventi di quest'ultimo.

Questa parte della suite, tuttavia, non sarà approfondita nel proseguo dell'elaborato, sia per mancanza di competenze da parte degli autori sia perchè l'argomento è vasto ed esulerebbe dagli scopi prefissati.

2 Il Log di processo

Un log di processo è un insieme di dati contenente la registrazione di accadimenti (o eventi) riconducibili ad una certa attività, un certo sistema o una certa organizzazione. Il processo è costituito dall'insieme di tutti gli eventi registrati. In `bupaR` il log di processo è un oggetto di tipo `eventlog`, che può essere considerato come un dataset caratterizzato da una certa struttura particolare che si analizzerà qui di seguito.

2.1 Eventi e attività

2.1.1 Gli eventi

L'evento è l'unità elementare di log di processo; la sequenza temporale degli eventi è ciò che concretamente costituisce il processo.

In un oggetto `eventlog` ogni evento è rappresentato da una riga del dataset (un'osservazione). Tale riga contiene un timestamp (cioè un marcatore temporale che indica quando l'evento è avvenuto) e una serie di informazioni allegate. Alcune di queste informazioni, sono variabili "libere" e specifiche del contesto (e.g., dei dati associati all'evento), altre invece sono generalizzabili per buona parte dei file di log e possono essere usate per ricostruire la struttura del processo. Per `bupaR`, alcune di tali informazioni sono obbligatorie, altre opzionali.

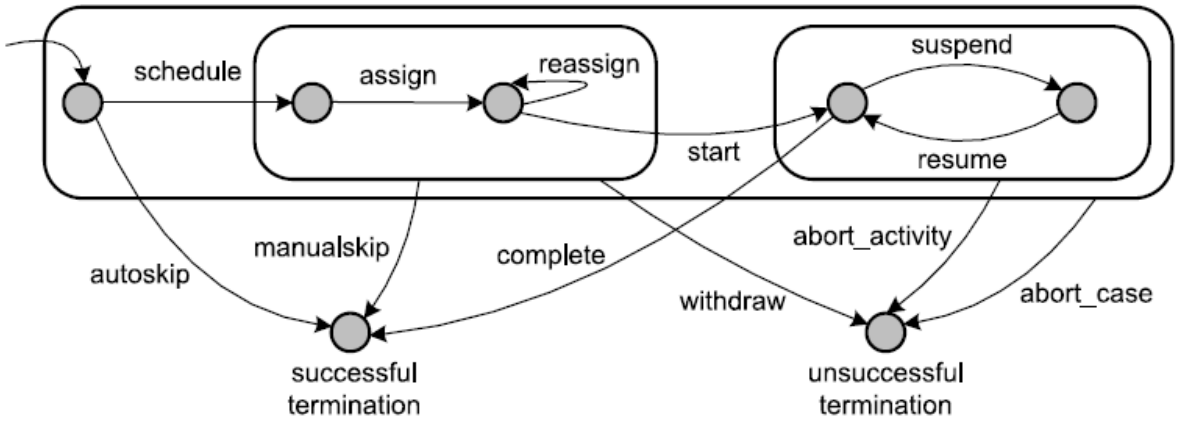
2.1.2 Le attività e il loro ciclo di vita

Il primo elemento che costituisce la struttura di un processo è l'attività. Un'attività è un insieme di eventi aggregabili in un unico "passo" del processo. Se gli eventi sono singole registrazioni specifiche di "accadimenti," le attività sono considerabili più come le vere e proprie "fasi" del processo.

In `bupaR`, l'evento è visto come una fase del ciclo di vita di un'attività. Il ciclo di vita segue una struttura rappresentabile da una sorta di macchina a stati finiti, dove l'evento è la transizione da uno stato al successivo: si parte da un dall'inizio e si termina con una conclusione (o con l'annullamento dell'attività). Un esempio di ciclo di vita (quello di default) è illustrato nell'immagine sottostante; se necessario, tale struttura può essere personalizzata in base alle proprie esigenze (tematica che non verrà approfondita in questo elaborato).

In un oggetto `eventlog`, l'attività viene identificata con un'etichetta univoca associata ad ogni evento (solitamente si usa un numero intero incrementale), mentre la fase del ciclo di vita è un valore testuale (variabile categoriale).

La tripla $< Timestamp, IdentificatoreAttività, FaseCicloDiVita >$ può essere usata per identificare univocamente un evento all'interno di un log di processo.



2.2 Tipi di attività, casi e tracce

Nel contesto reale di un processo, ci si aspetta che esso venga svolto più volte. Ogni esecuzione del processo verrà registrata nel file di log come una sequenza di eventi, ovvero come sequenza di attività. Da questa idea si parte per definire altri tre concetti essenziali: il caso, il tipo di attività e la traccia.

2.2.1 Il caso

In un log di processo, ci si aspetta ci siano più esecuzioni distinte dello stesso processo; tali esecuzioni sono chiamate "casi." Nel contesto di un'organizzazione, il caso potrebbe essere utilizzato per identificare un certo cliente, un certo progetto, ecc.

Negli oggetti `eventlog`, il caso è indicato con una variabile, che può essere un numero intero oppure anche una stringa (in ogni caso, il caso si deve intendere come una variabile categoriale). Il caso è una delle informazioni obbligatorie da associare all'evento.

2.2.2 Il tipo di attività

Partendo dal concetto di caso, si introduce il concetto di tipo di attività. Da quanto visto fin ora, ogni (esecuzione di un') attività è identificata da un valore univoco. Detto ciò, in un contesto reale ci si aspetta esistano "tipi di attività."

Negli oggetti `eventlog`, il tipo di attività è rappresentato da una variabile categoriale, che ad ogni evento (e quindi ad ogni istanza di attività) associa un "tipo" (spesso rappresentato da un nome testuale). Tale dato viene utilizzato per modellare in modo astratto e generico il "flusso" del processo; ciò consente di estrarre molte informazioni interessanti, come ad esempio il grafo che modella le sequenze osservate di attività.

2.2.3 La traccia

Dati i concetti di caso e di tipo di attività, si può definire il concetto di traccia. Una traccia non è altro che la sequenza di (tipi di) attività che viene eseguita per un certo caso. A diversi casi può corrispondere la stessa traccia, per questo motivo è possibile studiare quali sono le tracce tipiche, con che frequenza vengono osservate, *etc.*

2.2.4 Le risorse

Un'ultima (e non sempre presente) informazione rilevante in un oggetto `eventlog` è la risorsa. La risorsa è una variabile categoriale (opzionale) che indica "che asset viene impiegato" per eseguire un'istanza di attività.

Nel contesto di un'organizzazione, una risorsa può essere un dipendente, una stanza, un macchinario, un reparto, *etc.* In certi tipi di processo può essere interessante studiare quali risorse vengono usate in quali tipi di attività, ricavando ed analizzando diverse frequenze congiunte.

2.3 Costruzione di un file di log

Per costruire un oggetto `eventlog`, innanzi tutto è necessario avere un dataset contenente tutti i dati necessari e strutturato in un formato appropriato. Su tali dati, si può usare la funzione `eventlog` per creare un oggetto trattabile da `bupaR`. Tale funzione richiede come parametri:

- il dataframe di partenza (primo parametro);
- il nome delle colonne che contengono ciascuna delle informazioni necessarie.

L'oggetto in output rimarrà trattabile come fosse un normale dataframe, ma ora sarà anche considerato un'istanza di `eventlog` e sarà utilizzabile nelle diverse funzioni delle librerie di `bupaR`.

```
library(bupaR)

# (in questo esempio si finge che il log hospital non sia già un oggetto eventlog)
dummuy_df <- as.data.frame(hospital) %>%
```

```
  dplyr::select(timestamp, case_id, activity, activity_instance_id, lifecycle, group) %>%
  dplyr::rename(patient=case_id, status=lifecycle)

# con la funzione eventlog, si costruisce l'oggetto
# (indicando le colonne contenti le informazioni rilevanti)
dummuy_df %>% eventlog(
        case_id = "patient",
        activity_id = "activity",
        activity_instance_id = "activity_instance_id",
        lifecycle_id = "status",
        timestamp = "timestamp",
        resource_id = "group"
    )
```

```
## Log of 150291 events consisting of:
## 981 traces
## 1143 cases
## 150291 instances of 624 activities
## 43 resources
## Events occurred from 2005-01-03 until 2008-03-20
##
## Variables were mapped as follows:
## Case identifier:     patient
## Activity identifier:     activity
## Resource identifier:     group
## Activity instance identifier:    activity_instance_id
## Timestamp:           timestamp
## Lifecycle transition:        status
##
## # A tibble: 150,291 x 7
##    timestamp           patient  activity   activity_instan~ status group  .order
##    <dttm>              <chr>    <fct>      <chr>            <fct>  <fct>   <int>
##  1 2005-01-03 00:00:00 00000000 1e consul~ 1                compl~ Radio~      1
##  2 2005-01-03 00:00:00 00000000 administr~ 2                compl~ Radio~      2
##  3 2005-01-05 00:00:00 00000000 verlosk.-~ 3                compl~ Nursi~      3
##  4 2005-01-05 00:00:00 00000000 echografi~ 4                compl~ Obste~      4
##  5 2005-01-05 00:00:00 00000000 1e consul~ 5                compl~ Nursi~      5
##  6 2005-01-05 00:00:00 00000000 administr~ 6                compl~ Nursi~      6
##  7 2005-01-24 00:00:00 00000000 simulator~ 7                compl~ Radio~      7
##  8 2005-01-31 00:00:00 00000000 behandelt~ 8                compl~ Radio~      8
##  9 2005-01-31 00:00:00 00000000 telethera~ 9                compl~ Radio~      9
## 10 2005-02-15 00:00:00 00000000 aanname l~ 10               compl~ Gener~     10
## # ... with 150,281 more rows
```

2.3.1 La pulizia dei dati

Come si può intuire, avere un dataframe la cui struttura sia esattamente quella richiesta non è scontato. Per questo motivo, a meno che non si abbia controllo sul processo di raccolta dati, probabilmente ci si troverà a svolgere una serie di operazioni di pulizia, formattazione e integrazione delle informazioni mancanti.

In questo elaborato non si approfondiranno tali aspetti (che sono più inerenti all'ambito della data science), ma basti sapere che:

- è possibile manipolare un dataset in maniera semplice ed intuitivaattraverso le funzioni delle librerie `dplyr` e `tidyr`;

- sul sito ufficiale di `bupaR` si trovano alcuni esempi di tipiche trasformazioni che si effettuano prima di generare un log (https://bupar.net/creating_eventlogs.html#Common_transformations).

2.4 Salvataggio di un file di log (e il formato XES)

Per salvare un log di processo è possibile utilizzare il formato XES (o eXtensible Event Stream). XES è un formato standard riconosciuto e usato a livello internazionale, che all'atto pratico, è una serializzazione in XML (la cui struttura qui non verrà approfondita). Per maggiori informazioni, è possibile consultare il sito https://www.tf-pm.org/resources/xes-standard/about-xes.

Per salvare e leggere file in formato XES è possibile utilizzare le funzioni della libreria `xesreadR`.

```
# Lettura di un file in formato XES
xesfile <- xesreadR::read_xes("https://bupar.net/eventdata/repairExample.xes")

# salvataggio di un file in formato XES
xesreadR::write_xes(xesfile, "esempioLogProcesso.xes")
```

In alternativa al formato XES, si può salvare il log come fosse un comune dataset (ad esempio in formato CSV).

3 Analisi esplorativa

L'analisi esplorativa di un log di processo è l'insieme di tutte le operazioni che permettono di "esplorare e descrivere" il processo sotto diverse prospettive e con strumenti abbastanza tipici dell'EDA (tabelle di contingenza, frequenze aggregate, barplot, ecc.)

Durante l'analisi esplorativa, si possono prendere in considerazione tre macro-prospettive:

- la prospettiva temporale;
- la prospettiva dell'organizzazione;
- la prospettiva strutturale (del processo).

Per ciascuna di queste prospettive, la libreria `edeaR` offre diverse funzioni per estrarre diverse informazioni.

3.1 Le funzioni dell'analisi esplorativa: la struttura

Le funzioni della libreria `edeaR` servono ad estrarre informazioni da un oggetto `eventlog`. La libreria include un vasto numero di funzioni, anche abbastanza diverse tra di loro. Tuttavia, ci sono delle caratteristiche comuni che è bene comprendere prima di iniziare ad esplorare gli strumenti disponibili. Purtroppo, la documentazione ufficiale è abbastanza povera da questo punto di vista.

3.1.1 L'output

Ciascuna funzione ha lo scopo di estrarre informazioni riguardo ad uno specifico aspetto del log di processo. Tali informazioni possono essere frequenze osservate, sintesi statistiche oppure metriche particolari.

Nel caso più semplice, l'output consiste in un insieme di sintesi di una variabile numerica continua (e.g., valore minimo, massimo, medio, quantili, varianza, ecc.). Spesso però l'output è un dataframe più complesso, dove si associa ad ogni "oggetto" una frequenza osservata, oppure di nuovo delle sintesi di una variabile numerica. Il dataframe viene ritornato come oggetto di tipo `tibble`.[1]

L'output delle funzioni - spesso - può essere dato in pasto alla funzione `plot` per produrre automaticamente un grafico appropriato. Nel caso di dataframe contententi frequenze aggregate, spesso viene prodotto un grafico a barre; per le sintesi di variabili numeriche invece spesso l'output consiste in uno o più boxplot. Per l'output di certe funzioni vengono prodotti grafici "ad hoc."

Per concludere, quando l'output è un dataframe, nulla vieta di ricavare manualmente ulteriori sintesi statistiche (e.g., calcolo media, varianza, ecc. sui valori), fare considerazioni sulla distribuzione, produrre i propri grafici e più in generale usare i tipici strumenti della statistica descrittiva (e inferenziale).

3.1.2 L'input e il concetto di livello

Ciascuna funzione, solitamente, è caratterizzata da due o tre input. I due input principali (praticamente sempre presenti) sono il dataset e il livello.

Il concetto di livello è particolarmente importante, in quanto è un parametro che viene richiesto da quasi tutte le funzioni per indicare "rispetto a cosa" si vuole estrarre una certa informazione. Indicando il livello si dice alla funzione di "ricavare delle sintesi (oppure un valore singolo - e.g., frequenza osservata)" per ogni:

- `log` (ricava un unico output per tutto il file di log);
- `trace` (un dato per ogni tipo di traccia);
- `case` (un dato per ogni caso);
- `activity` (un dato per ogni tipo di attività);
- `resource` (un dato per ogni risorsa);
- `resource-activity` (un dato per ogni combinazione di attività e risorsa impiegata).

[1] Formato utilizzato nelle librerie di `tidyverse` per estendere le funzionalità dei classici dataframe; per l'uso di `bupaR`, basti sapere che tale tipo di oggetto può essere trattato allo stesso modo di un normale dataframe.

A seconda del contesto l'output sarà un insieme di sintesi oppure un dataframe.

Si osservi che ciascuna delle funzioni accetta solo un sotto-insieme dei livelli elencati e il significato attribuito alla richiesta dipende molto dal contesto.

Per vedere quali livelli specifici possono essere usati su quali funzioni, si consiglia di leggere documentazione ufficiale (https://bupar.net/exploring.html). Purtroppo, essa non riporta molto sul significato dei vari livelli e sull'output previsto per ogni funzione; per cui bisogna andare un po' per tentativi.

3.2 Panoramica generale del dataset

Per tutti gli esempi successivi, si utilizzerà il dataset `sepsis`, già presentato nel capitolo 2.

```
library(bupaR)

# estrazione delle variabili essenziali
# (solo quelle necessarie a rappresentare gli aspetti fondamentali del processo)
sepsis2 <- eventdataR::sepsis %>% select()

summary(sepsis2)
```

```
## Number of events:  15214
## Number of cases:  1050
## Number of traces:  846
## Number of distinct activities:  16
## Average trace length:  14.48952
##
## Start eventlog:  2013-11-07 08:18:29
## End eventlog:  2015-06-05 12:25:11
```

```
##    case_id                     activity    activity_instance_id
##  Length:15214       Leucocytes     :3383   Length:15214
##  Class :character   CRP            :3262   Class :character
##  Mode  :character   LacticAcid     :1466   Mode  :character
##                     Admission NC   :1182
##                     ER Triage      :1053
##                     ER Registration:1050
##                     (Other)        :3818
##    timestamp                       resource       lifecycle         .order
##  Min.   :2013-11-07 08:18:29   B      :8111   complete:15214   Min.   :    1
##  1st Qu.:2014-03-30 21:23:15   A      :3462                    1st Qu.: 3804
##  Median :2014-06-30 19:31:10   C      :1053                    Median : 7608
##  Mean   :2014-07-04 13:08:45   E      : 782                    Mean   : 7608
##  3rd Qu.:2014-10-12 01:51:47   ?      : 294                    3rd Qu.:11411
##  Max.   :2015-06-05 12:25:11   F      : 216                    Max.   :15214
##                                (Other):1296
```

Chiamando `summary` su un oggetto `eventlog` si può farsi un'idea generale sulla struttura e sulle dimensioni del dataframe. Si osserva che si ottiene:

- un conteggio degli eventi (cioè delle righe), dei casi, delle tracce e delle attività;
- la lunghezza media delle tracce;
- l'intervallo temporale degli eventi;
- una panoramica rapida su ciascuna variabile.

In merito a tale dataframe, si osserva una discreta dimensione (anche abbastanza contenuta per un file di log), un numero di casi e tracce abbastanza alto e un numero di tipi di attività contenuto (vista la specificità del processo, è normale aspettarsi un numero non troppo grande di attività). Si è scelto di usare questo

dataset proprio per quest'ultima caratteristica, in quanto i log con un grande numero di attività tendono a rappresentare processi piuttosto complessi, con tantissime tracce diverse e quindi una struttura piuttosto elaborata.

Si osserva inoltre che in questo dataset non si registra ciclo di vita delle attività, in quanto tutti gli eventi sono associati ad un'etichetta `complete` per la variabile `lifecycle`. Ciascun evento coincide quindi con un'attività.

3.3 L'analisi nella prospettiva temporale

L'analisi nella prospettiva temporale ha come scopo lo studio di tutto ciò che ha a che fare con la durata dei processi e delle attività. Tale studio si può effettuare considerando tre aspetti:

- il **throughput time**: il tempo trascorso tra l'inizio o la fine di qualcosa (e.g., la durata di un caso specifico, le sintesi calcolate sulla durata di un certo tipo di traccia);
- il **processing time**: il tempo durante il quale un processo è stato effettivamente portato avanti (e.g, il tempo per il quale un certo caso è stato effettivamente trattato, per quanto tempo una risorsa è stata utilizzata, le statistiche sulle durate effettive delle attività, le statistiche sulla durata di una certa attività impiegando una certa risorsa);
- l'**idle time**: il tempo intermedio tra ciascuna attività relativa a qualcosa (e.g., tempi morti per un caso o per un certo tipo di traccia, il tempo in cui una risorsa non è stata impiegata, ecc.)

Per ciascuno di questi tre aspetti, esiste una funzione del pacchetto `edeaR` - rispettivamente: `throughput_time`, `processing_time`, `idle_time` - che permette l'estrazione di informazioni rilevanti a diversi livelli.

Si vuole far notare che uno studio approfondito della prospettiva temporale, specie per quanto riguarda processing time e idle time delle attività, richiederebbe un log di processo che includa la registrazione dei singoli eventi del ciclo di vita (quando si inizia un'attività, quando si conclude, ecc.); purtroppo nella maggior parte dei dataset non si riporta un tale livello di dettaglio. È abbastanza tipico invece registrare direttamente le attività come eventi, etichettando tutte le righe come `completed`.

Figure 1: Le tre prospettive temporali.

3.3.1 Esempio: qualche statistica sulla durata dei casi

Immaginiamo di voler effettuare un'analisi preliminare sulla durata dei casi.

Partiamo estraendo qualche sintesi globale sulla durata dei casi (in giorni, ore e minuti):

```
sepsis2 %>% throughput_time(level="log", units="days")
```

```
##          min           q1       median         mean           q3          max
## 1.412037e-03 6.434635e-01 5.343385e+00 2.846934e+01 1.799657e+01 4.223239e+02
##       st_dev          iqr
## 6.053737e+01 1.735310e+01
```

```
## attr(,"units")
## [1] "days"
```

```
sepsis2 %>% throughput_time(level="log", units="hours")
```

```
##          min           q1       median         mean           q3          max
## 3.388889e-02 1.544313e+01 1.282413e+02 6.832642e+02 4.319176e+02 1.013577e+04
##       st_dev          iqr
## 1.452897e+03 4.164745e+02
## attr(,"units")
## [1] "hours"
```

```
sepsis2 %>% throughput_time(level="log", units="mins")
```

```
##          min           q1       median         mean           q3          max
## 2.033333e+00 9.265875e+02 7.694475e+03 4.099585e+04 2.591506e+04 6.081465e+05
##       st_dev          iqr
## 8.717381e+04 2.498847e+04
## attr(,"units")
## [1] "mins"
```

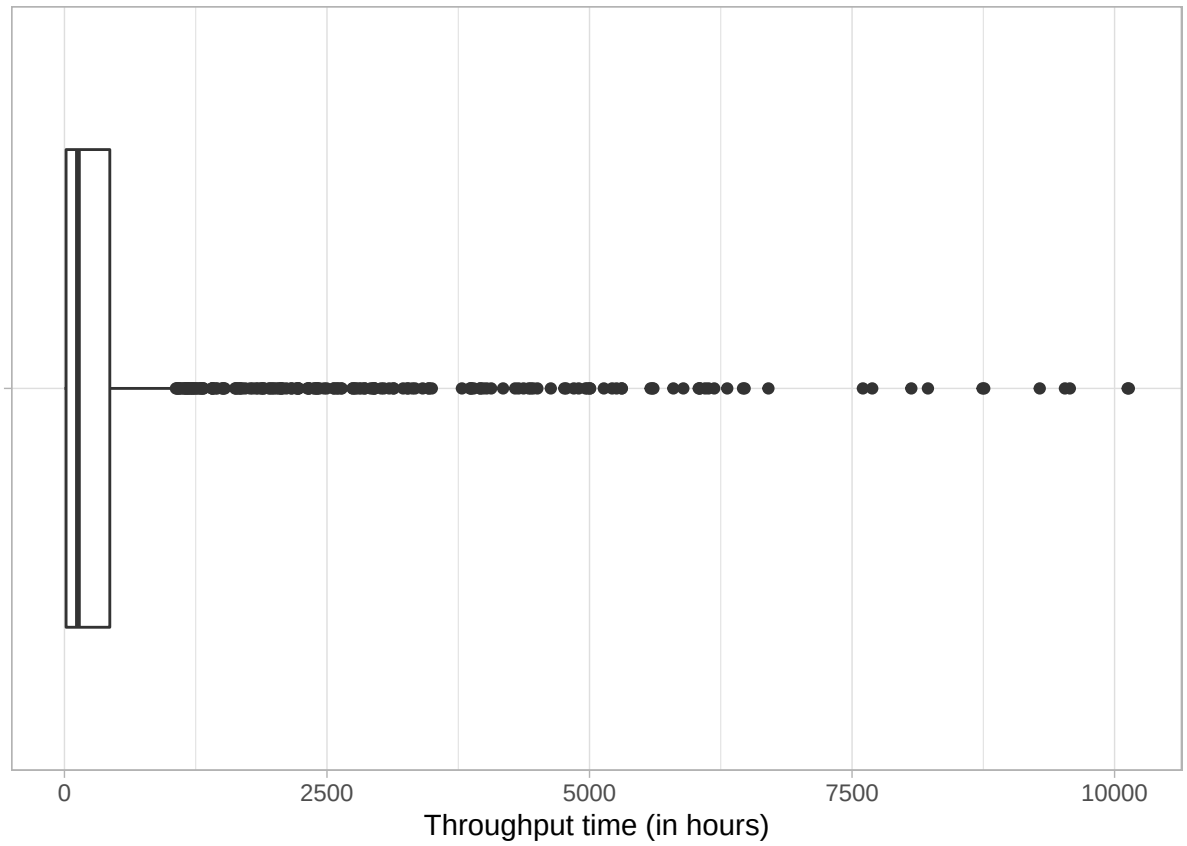
Estraiamo ora la durata di ogni singolo caso:

```
sepsis2 %>% throughput_time(level="case", units="hours") %>% summary
```

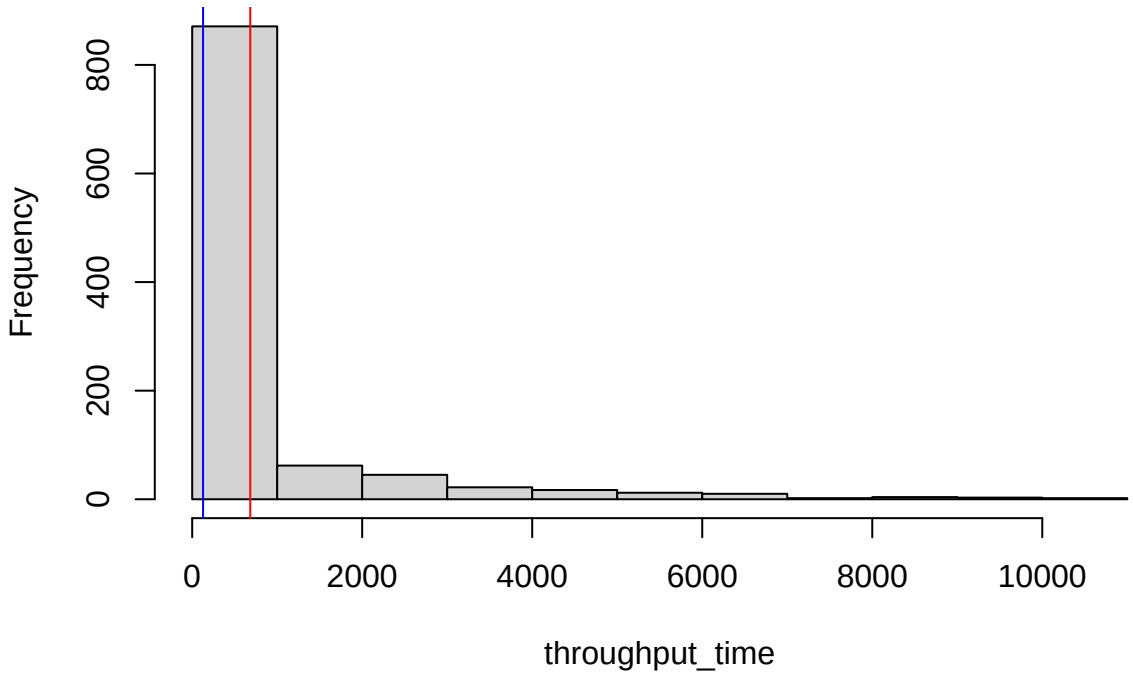
```
##    case_id          throughput_time
##  Length:1050        Min.   :    0.034
##  Class :character   1st Qu.:   15.443
##  Mode  :character   Median :  128.241
##                     Mean   :  683.264
##                     3rd Qu.:  431.918
##                     Max.   :10135.775
```

Proviamo a produrre qualche grafico per farci un'idea sulla distribuzione della durata:

```
# Boxplot (prodotto automaticamente chiamando plot sull'output con livello log)
sepsis2 %>% throughput_time(level="log", units="hours") %>% plot
```



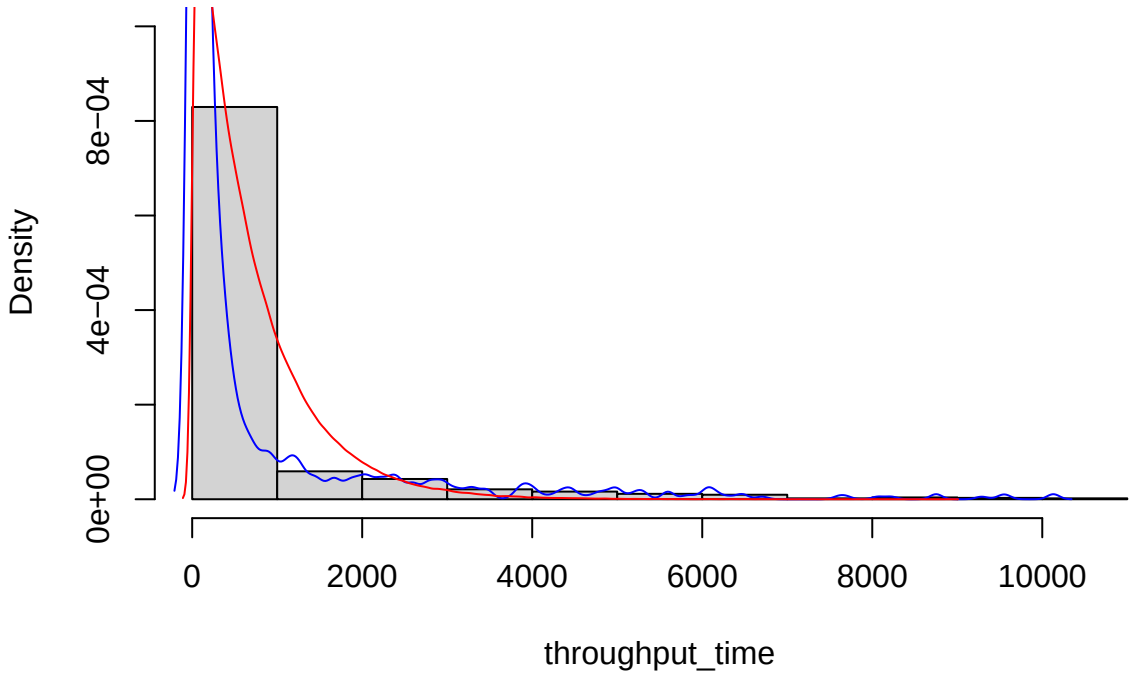

```
# Estrazione delle singole osservazioni
case_duration<- throughput_time(sepsis2, level="case", units="hours")$throughput_time

# Istogramma generato sulle singole osservazioni
hist(case_duration, main="Durata dei casi", xlab = "throughput_time")

# sovrapponiamo come riferimento la media e la mediana
abline(v=mean(case_duration), col="red")
abline(v=median(case_duration), col="blue")
```



Da entrambi i grafici, si osserva una distribuzione piuttosto asimmetrica, con la maggior parte delle osservazioni concetrate a sinistra e un numero non indifferente di valori anche molto elevati (non considerabili come outlyer). Forse le osservazioni potrebbero essere riconducibili ad un fenomeno di tipo esponenziale.

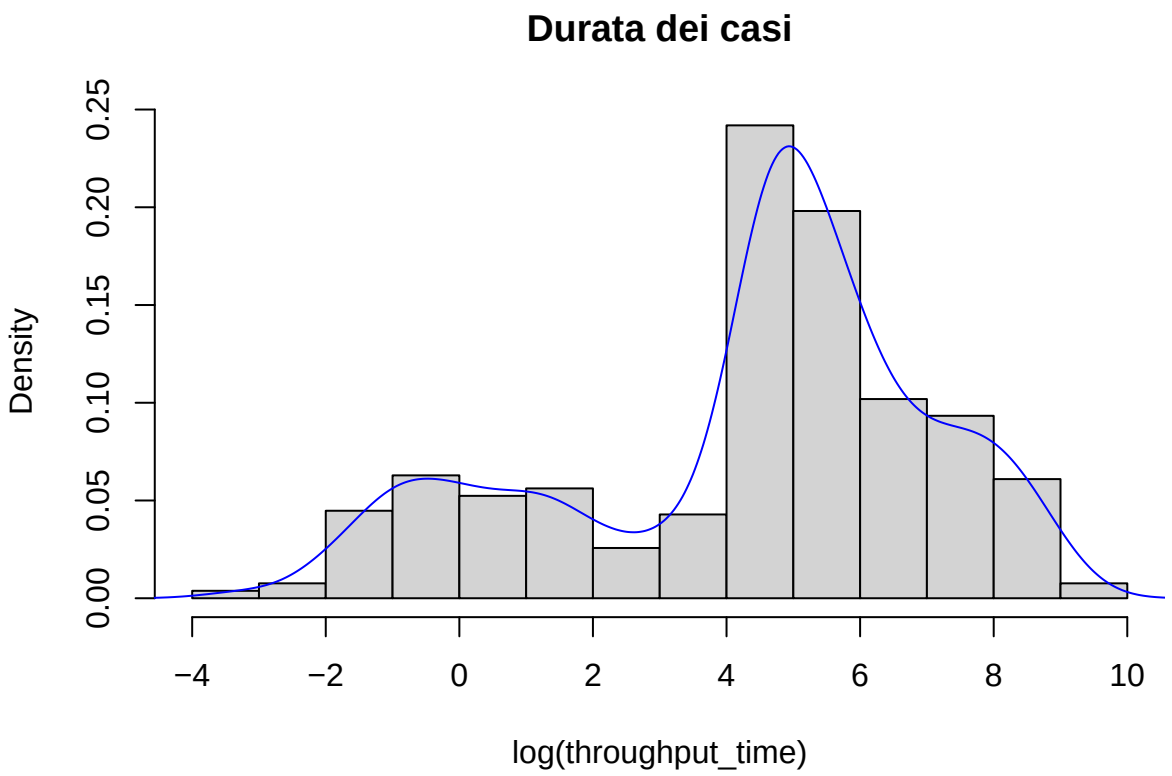
Mettiamo alla prova tale ipotesi sovrapponendo all'istogramma la curva di densità e un modello esponenziale. Per costruire tale modello esponenziale usiamo il reciproco della media campionaria come stima del parametro $\hat{\lambda} = 1/\bar{Y}$.

```
# Istogramma della distribuzione di probabilità, sovrapposto a
# curve di densità e curva di un modello esponenziale
hist(case_duration, probability=T, ylim=c(0,0.001),
     main="Durata dei casi", xlab="throughput_time")
lines(density(case_duration), col="blue")

# costruiamo un campione casuale di osservazioni di un modello esponenziale
expsample <- rexp(500000, 1/mean(case_duration))

# sovrapponiamo la curva di densità del modello
lines(density(expsample), col="red")
```



Riportando una curva esponenziale a fianco alla curva di densità ci si rende conto che, seppur c'è una vaga somiglianza, la distribuzione sembra essere più complessa. Si riporta quindi la variabile su scala logaritmica e si produce un istogramma e una curva di densità.

```
# Istogramma in scala logaritmica (sempre con curva di densità)
hist(log(case_duration), probability=T, # ylim=c(0,0.001),
     main="Durata dei casi", xlab="log(throughput_time)")
lines(density(log(case_duration)), col="blue")
```



Da tale istogramma si può cogliere la reale complessità della distribuzione della durata dei casi. Si osservano almeno due gruppi distinti di osservazioni, che probabilmente in un'analisi formale dovrebbero essere considerati diversamente (e.g., casi brevi e casi lunghi).

3.4 L'analisi nella prospettiva dell'organizzazione

La prospettiva dell'organizzazione consiste nello studio di come vengono impiegate le risorse all'interno dei processi. Le funzioni appartenenti a questa categoria intendono studiare (da diverse prospettive) l'utilizzo e l'allocazione delle risorse nei casi, nelle attività, ecc..

Di seguito, si riportano le tre funzioni per l'analisi nella prospettiva dell'organizzazione.

- `resource_frequency` <- ricava la frequenza di utilizzo delle risorse (a diversi livelli);
- `resource_involvement` <- ricava il coinvolgimento delle risorse nei casi;
- `resource_specialisation` <- rileva la specializzazione delle risorse (quanto certe risorse sono specializzate per certi casi).

3.4.1 Esempio: specializzazione delle risorse per lo svolgimento di certe attività

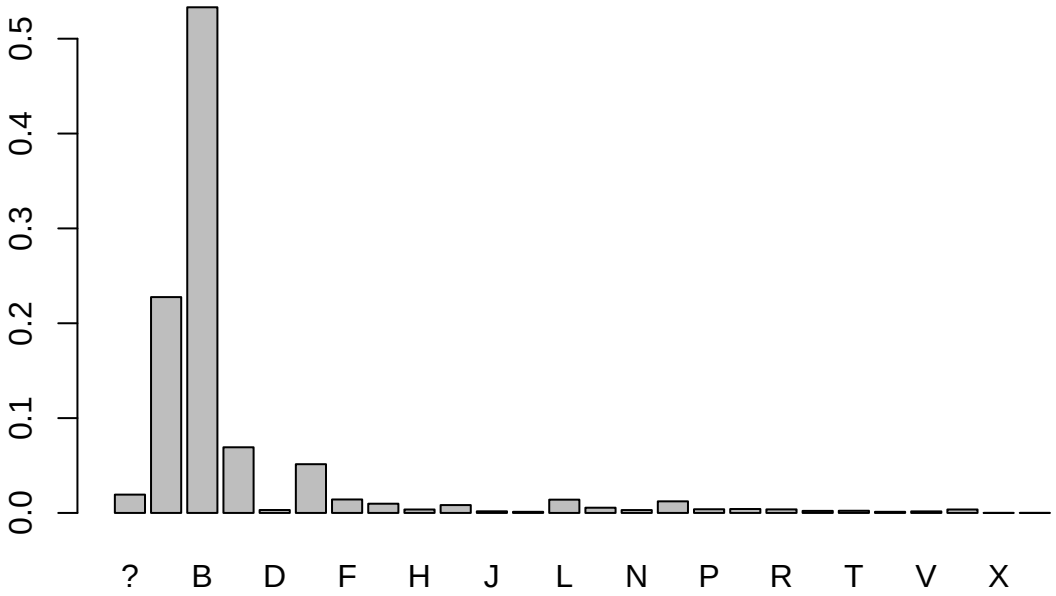
Un fenomeno interessante da analizzare nei processi è la specializzazione delle risorse nello svolgimento di certi tipi particolari di attività.[2]

```
summary(sepsis2$resource)
```

```
##    ?    A    B    C    D    E    F    G    H    I    J    K    L    M    N    O
##  294 3462 8111 1053   47  782  216  148   55  126   26   18  213   84   46  186
##    P    Q    R    S    T    U    V    W    X    Y
##   59   63   57   33   35   18   25   55    1    1
```

[2]Un'analisi analoga si può svolgere anche per i casi, ad esempio, per analizzare come le risorse di un'organizzazione sono allocate ai vari progetti, clienti, ecc.

```
barplot(table(sepsis2$resource)/nrow(sepsis2))
```



In questo dataset, la risorsa è indicata nella colonna `resource` e i livelli della variabile categoriale sono non meglio precisate lettere dell'alfabeto (pertanto è difficile dare un significato).

```
sum(is.na(sepsis2$resource))
```

```
## [1] 0
```

```
sum(sepsis2$resource=="?")
```

```
## [1] 294
```

Si osserva inoltre che non sono presenti valori nulli, ma per alcune osservazioni la variabile assume un valore "?" (che si presuma sia un modo per indicare che il dato non era disponibile).

Ricaviamo le frequenze congiunte di risorse e attività (escludendo le righe con valore "?").

```
resource_activity <- sepsis2 %>%
  filter(resource!="?") %>%
  resource_frequency("resource-activity")

resource_activity %>% head()
```

```
## # A tibble: 6 x 5
##   resource activity         absolute relative_resource relative_activity
##   <fct>    <fct>               <int>             <dbl>             <dbl>
## 1 B        Leucocytes           3383             0.417             1
## 2 B        CRP                  3262             0.402             1
## 3 B        LacticAcid           1466             0.181             1
## 4 C        ER Triage            1053             1                 1
## 5 A        ER Registration       985             0.285             0.938
## 6 A        ER Sepsis Triage      984             0.284             0.938
```

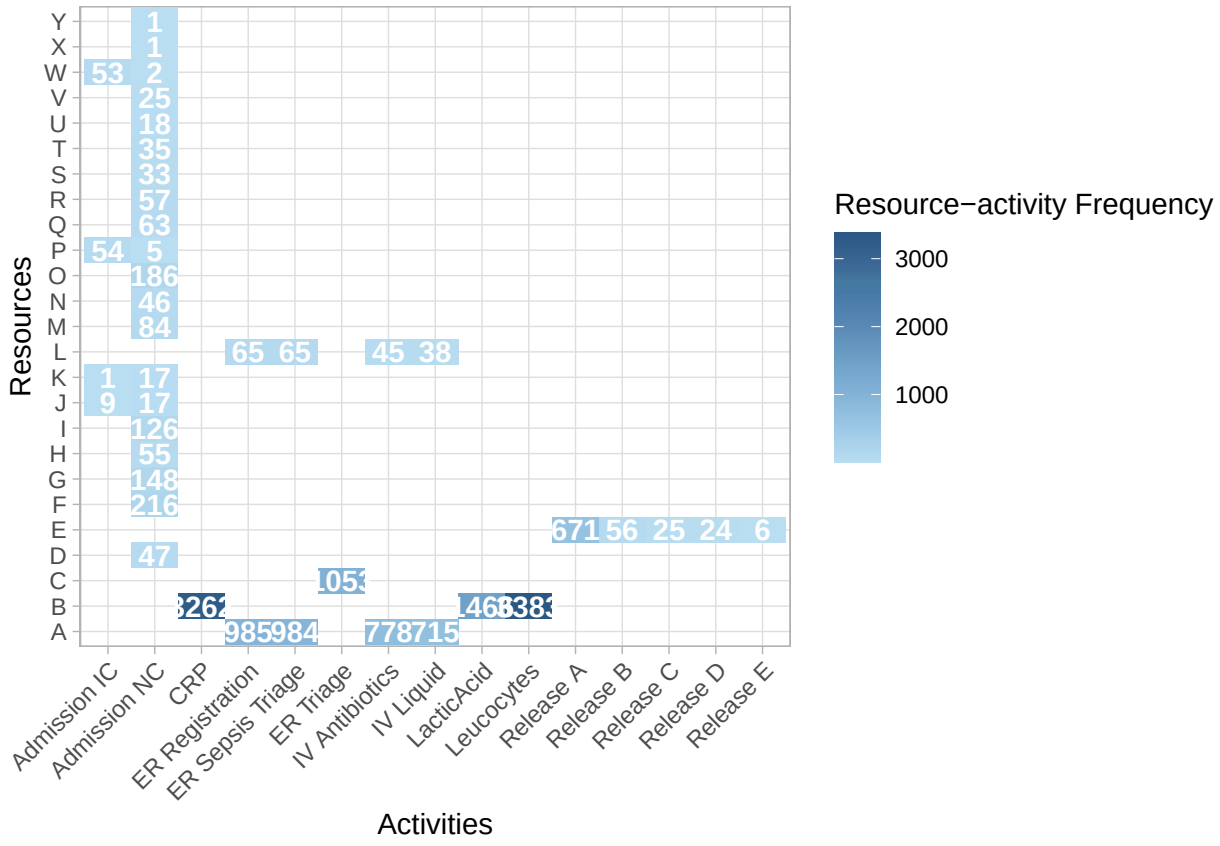
```
# summary(resource_activity)
```

Si osserva che in questo modo si ottiene un dataframe con cinque variabili: la coppia risorsa attività, la frequenza assoluta osservata per la coppia e le frequenze marginali relative (`relative_resource` rappresenta

la percentuale di volte in cui la risorsa viene impiegata in una certa attività, `relative_activity` invece rappresenta la percentuale di volte in cui per questa attività viene impiegata una certa risorsa).

Stampando il plot automatico, viene prodotta un'interessante griglia che illustra le frequenze di ogni combinazione risorsa-attività, con un valore numerico per ongi cella e una colorazione in scala per evidenziare subito le coppie più frequenti.

```
resource_activity %>% plot
```



Si osserva che, su un tale tipo di dato potrebbe essere interessante effettuare un test chi-quadro come misura della specializzazione (se le variabili risultano fortemente dipendenti, vuol dire che c'è una grande specializzazione). Proviamo a manipolare la struttura dei dati (con l'ausilio di alcune funzioni del pacchetto `tidyverse`) per costruire una tabella di contingenza ed effettuare un test di indipendenza (vista la natura sparsa della matrice, si opta per un test esatto di Fisher).

```
# Ricreiamo una tabella di contingenza (utilizzando alcune funzioni di tidyverse)
resource_activity_contingency_table <- resource_activity %>%
  # seleziona solo le righe risorsa, attività e f. assoluta
  select(resource, activity, absolute) %>%

  # genera a partire dalle coppie di osservazioni una griglia, dove si associa
  # una frequenza per ogni combinazione di riga/colonna
  tidyr::pivot_wider(names_from=activity, values_from=absolute, values_fill = 0) %>%

  # trasforma la colonna "resource" in un indice
  tibble::column_to_rownames(var="resource") %>%

  # converti il dataframe in matrice
  as.matrix()
```

```r
# Effettuiamo il test d'indipendenza
fisher.test(resource_activity_contingency_table, simulate.p.value = T)
```

```
##
##  Fisher's Exact Test for Count Data with simulated p-value (based on
##  2000 replicates)
##
## data:  resource_activity_contingency_table
## p-value = 0.0004998
## alternative hypothesis: two.sided
```

Dato il p-value basso, si rifiuta l'ipotesi di indipendenza e si conferma che esiste una relazione significativa tra le risorse e il tipo di attività, quindi una specializzazione abbastanza forte.

3.5 L'analisi nella prospettiva strutturale

La prospettiva strutturale infine include una più vasta gamma di funzioni legate alle attività e alle tracce. Si misurano ad esempio:

- la presenza delle attività nei casi (percentuale di casi in cui un'attività è presente);
- la più frequenza delle attività (a diversi livelli);
- la rilevazione delle attività di inizio e conclusione (di un caso);
- la frequenza di casi coperti da ciascuna traccia;
- la lunghezza delle tracce.

La prospettiva strutturale infine include una vasta gamma di funzioni legate allo studio delle attività e delle tracce. Le attività e le tracce sono ciò che costituisce la vera e propria struttura del processo.

Le funzionalità spaziano da diversi conteggi di frequenze (e.g., ricava la frequenza con cui una certa attività viene svolta nei casi) a misure più specifiche (e.g., la lunghezza delle tracce). Di seguito, si descrivono brevemente le principali funzioni per l'analisi nella prospettiva strutturale.

- `activity_presence`: misura la percentuale di casi in cui ogni attività è presente.
- `activity_frequency`: strumento più ampio e generale per misurare la frequenza di apparizione di ogni tipo di attività a diversi livelli (e.g., nelle tracce, nell'intero log, ecc.).
- `start_activities`, `end_activities`: rileva quali attività costituiscono l'inizio e la fine delle tracce (e con che frequenza ciò accade).
- `trace_coverage`: particolare strumento che permette di ricavare un'indicazione su che percentuale di tracce copre che percentuale di casi; è uno strumento utile per farsi un'idea del tipo di distribuzione presente (le diverse possibili tracce sono distribuite uniformemente? oppure si ha un basso numero di tracce che copre la maggior parte dei casi?).
- `trace_length`: ricava le lunghezze delle tracce.

3.5.1 Esempio: panoramica generale sulle tracce

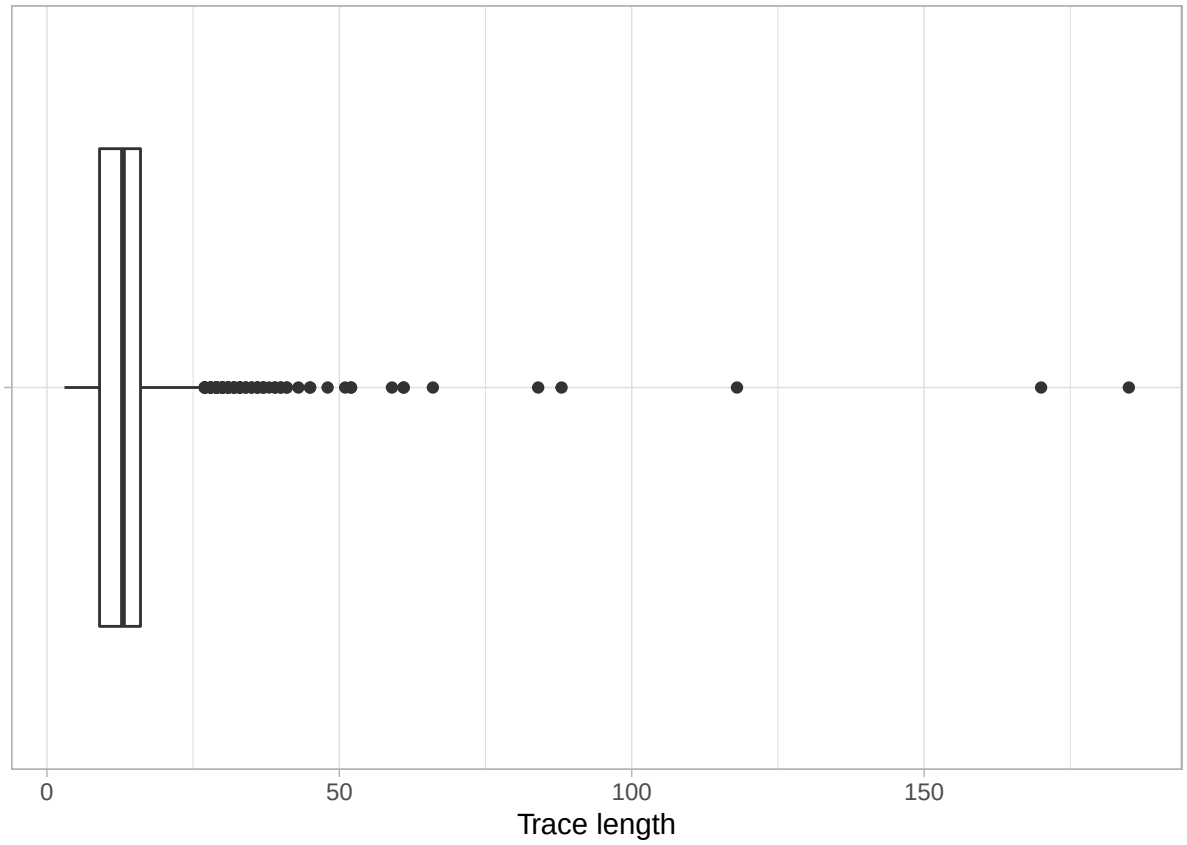
Supponiamo di voler modellare il processo con un diagramma (e.g., BPMN, un generico grafo, ecc.). Per modellare il processo, vogliamo prima farci un'idea su quali come sono fatte le tracce e come queste sono distribuite per i vari casi.

Eseguendo il `summary` sul dataset, si osserva che questo log di processo contiene 1050 casi e 846 distinte tracce, composte da diverse combinazioni di 16 attività. Già da questi numeri si dovrebbe intuire la complessità del problema. Spesso i file di log presentano un grandissimo numero di tracce differenti, anche a fronte di

relativamente poche attività e pochi casi. Per questo motivo, modellare con precisione la struttura di un certo processo (o almeno la "struttura tipica") è un problema non banale.

Partiamo con una rapida panoramica sulla lunghezza delle tracce.

```
sepsis2 %>% trace_length(level="log") %>% plot
```




```
sepsis2 %>% trace_length(level="log")
```

```
##       min        q1    median      mean        q3       max    st_dev       iqr
##   3.00000   9.00000  13.00000  14.48952  16.00000 185.00000  11.47594   7.00000
```

Come visto in precedenza per la durata dei casi, osserviamo che la maggior parte delle tracce è di lunghezza abbastanza ridotta (circa ≤ 25, il 75% delle osservazioni è addirittura situato prima di $q_3 = 16$). Detto ciò, c'è comunque un alto numero di tracce molto lunghe, che probabilmente non possono essere ignorate.

Invece di focalizzarci sulla lunghezza delle tracce, proviamo a studiarne la copertura, cioè vedere quali tracce coprono che percentuali di casi.

```
trc_cov <- sepsis2 %>% trace_coverage(level="trace")
summary(trc_cov)
```

```
##     trace              absolute         relative            cum_sum
##  Length:846         Min.   : 1.000   Min.   :0.0009524   Min.   :0.03333
##  Class :character   1st Qu.: 1.000   1st Qu.:0.0009524   1st Qu.:0.39643
##  Mode  :character   Median : 1.000   Median :0.0009524   Median :0.59762
##                     Mean   : 1.241   Mean   :0.0011820   Mean   :0.59471
##                     3rd Qu.: 1.000   3rd Qu.:0.0009524   3rd Qu.:0.79881
##                     Max.   :35.000   Max.   :0.0333333   Max.   :1.00000
```

```
trc_cov %>% head()
```

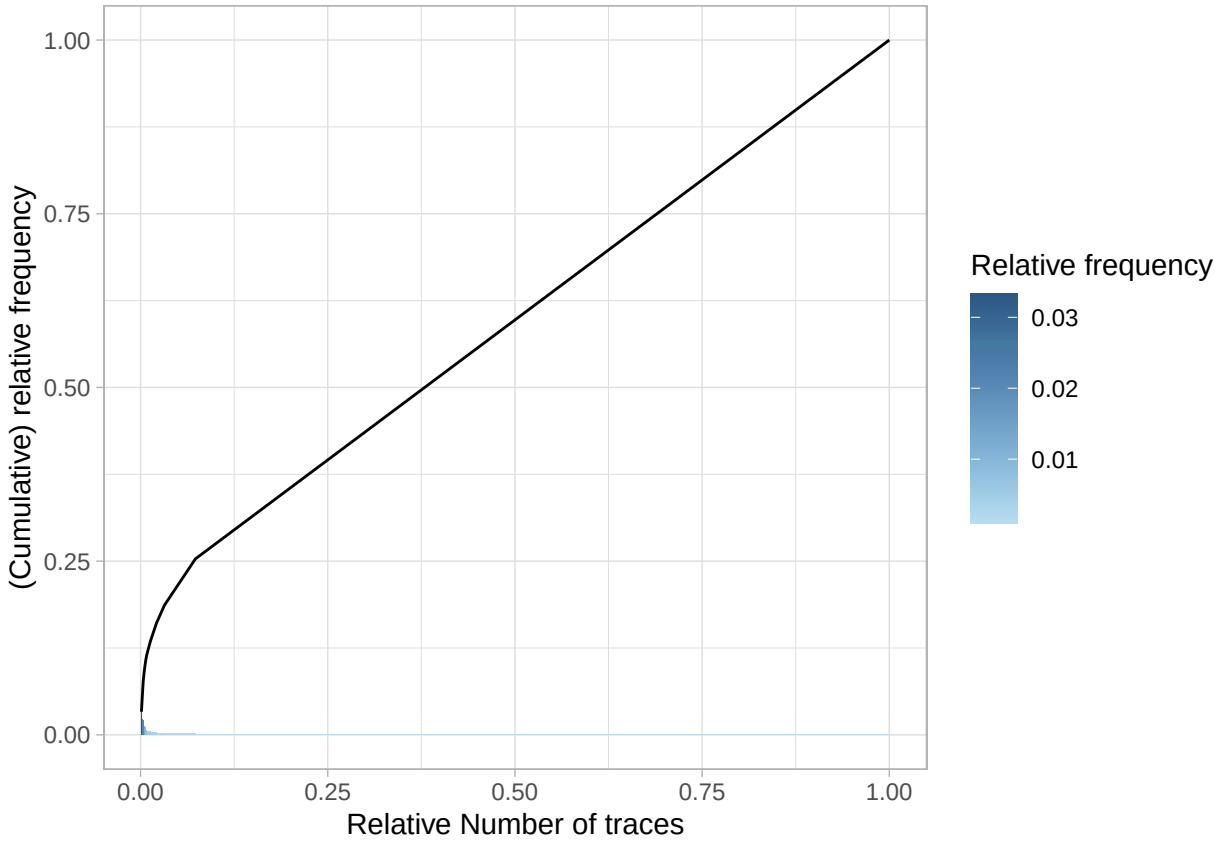
```
## # A tibble: 6 x 4
##   trace                                                absolute relative cum_sum
##   <chr>                                                   <int>    <dbl>   <dbl>
## 1 ER Registration,ER Triage,ER Sepsis Triage                 35  0.0333   0.0333
## 2 ER Registration,ER Triage,ER Sepsis Triage,Leucocyt~       24  0.0229   0.0562
## 3 ER Registration,ER Triage,ER Sepsis Triage,CRP,Leuc~       22  0.0210   0.0771
## 4 ER Registration,ER Triage,ER Sepsis Triage,CRP,Lact~       13  0.0124   0.0895
## 5 ER Registration,ER Triage,ER Sepsis Triage,Leucocyt~       11  0.0105   0.1
## 6 ER Registration,ER Triage,ER Sepsis Triage,Leucocyt~        9  0.00857  0.109
```

Si osserva che l'output di tale funzione è un dataframe - in formato `tibble` - che associa ad ogni traccia il numero di casi coperti. Ogni traccia è rappresentata da un elenco di attività. Oltre al numero di casi coperti, si riporta anche la frequenza relativa (percentuale di casi coperti) e una frequenza cumulativa.

Dalle sintesi si può già intuire qualcosa sulla distribuzione: si ha difatto un altissimo numero di tracce che copre soltanto 1 caso (almeno il 75%) e poche tracce che coprono alcuni "casi tipici" (comunque un numero ristretto).

Applicando `plot` sull'output della funzione si ottiene un grafico particolare, non troppo utile in questo caso ma che comunque merita menzione.

```
trc_cov %>% plot
```



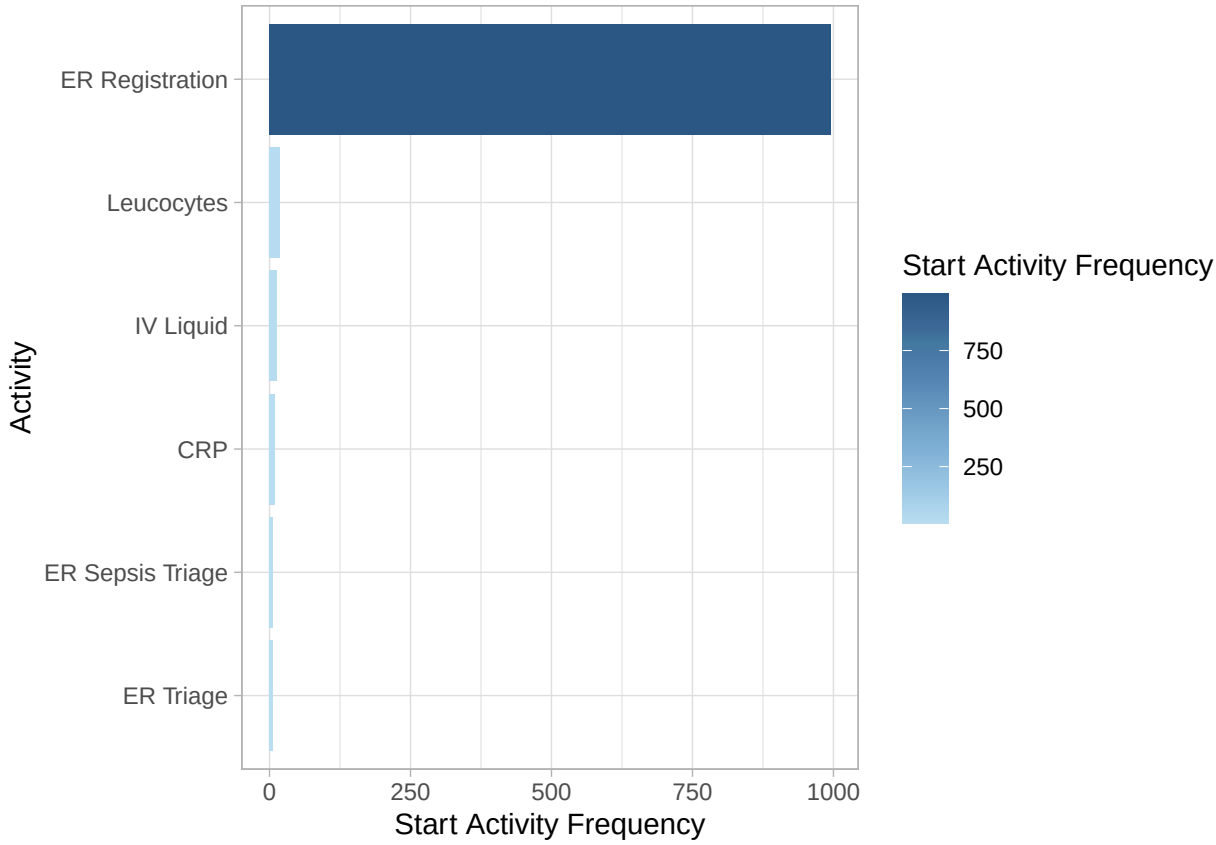
Il grafico si può interpretare nella seguente maniera:

- anche se non sono molto visibili, in basso si possono osservare tante piccole barre colorate (una per ogni traccia) ordinate per percentuale di casi coperti (in ordine decrescente);

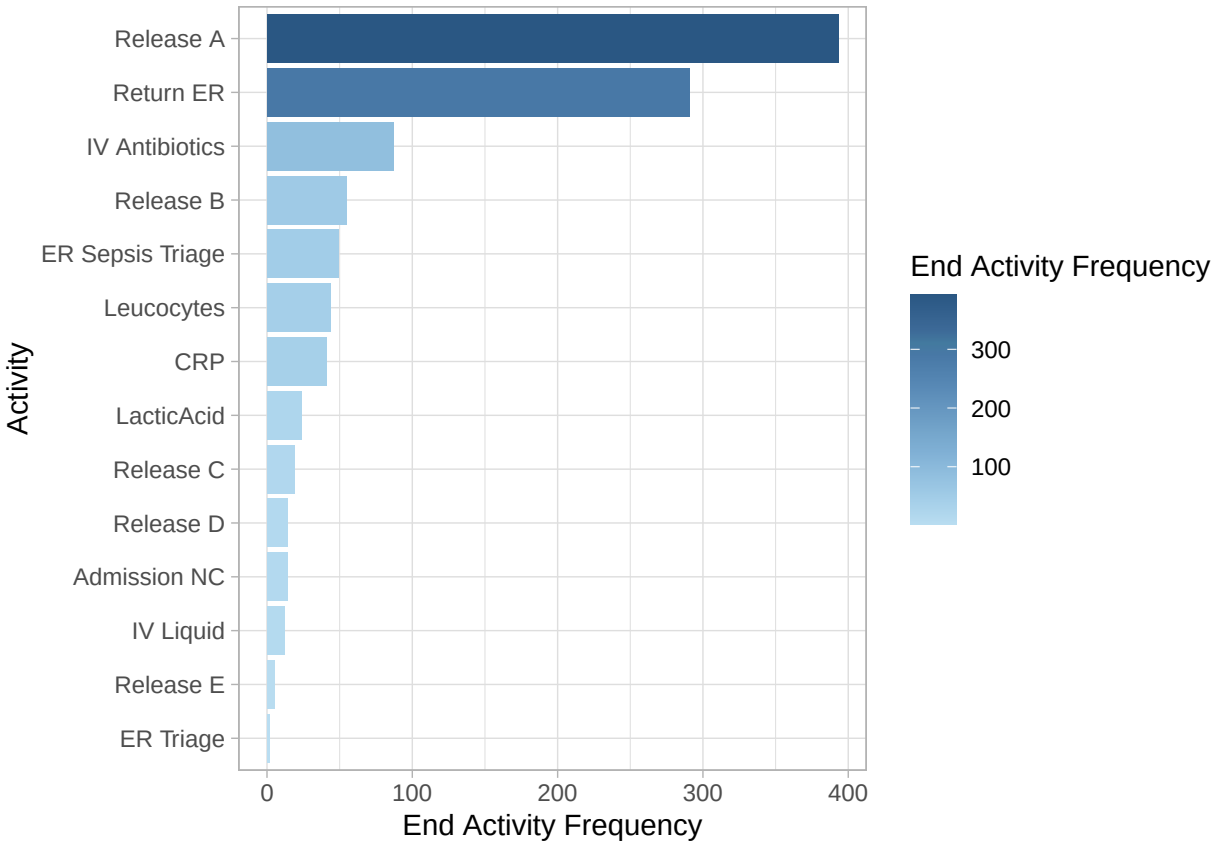
- la curva nera invece rappresenta la frequenza relativa cumulativa (che, visto l'ordinamento decrescente, sarà inevitabilmente concava).

Per concludere, si utilizzano le funzioni `start_activities` e `end_activities` per provare ad individuare quali sono le tipiche attività di inizio e di fine traccia più tipiche.

```
sepsis2 %>% start_activities(level="activity") %>% plot
```




```
sepsis2 %>% end_activities(level="activity") %>% plot
```



L'analisi effettuata potrebbe essere seguita da uno studio della mappa del processo utilizzando grafici ad hoc (che verranno esposti nel capitolo successivo), oppure anche utilizzando algoritmi di process discovery (https://bupar.net/processdiscovery.html) attraverso le librerie `pm4py`, `petrinetR` e `heuristicsmineR`.

4 Grafici per la visualizzazione dei processi

4.1 Visualizzare i processi

Nell'ambito della business process analysis, oltre ai classici grafici sui dati contenuti negli eventi (come visto nel cap. 3 è utile ottenere delle rappresentazioni grafiche del processo in analisi nel suo complesso, dal punto di vista del flusso delle attività o delle sue risorse.

Il pacchetto `processmapR` di bupaR fornisce le funzioni per visualizzare i processi sottoforma di grafi, matrici di precedenza, *trace explorer* e *dotted chart*.

4.2 Mappe di processo

Il modo più immediato di rappresentare un processo è utilizzare dei grafi in cui ogni nodo rappresenta un elemento di interesse.

`processmapR` fornisce due tipi di grafi: `process_map` e `resource_map`. Il primo inserisce nei nodi le attività, il secondo le risorse. Ogni nodo è etichettato con un numero.

Dai nodi escono tanti archi quante sono le possibili attività/risorse da eseguire/utilizzare al passo successivo del processo. Anche le frecce sono etichettate con un numero.

Sia lo spessore delle frecce che la sfumatura di colore dei nodi è proporzionale all'etichetta.

Concludono il grafo i nodi Start e End (a indicare l'inizio e la fine del processo). Ogni possibile cammino tra Start e End all'interno del grafo è una traccia nel log.

All'interno di nodi e archi possono essere inserite di misure di tre tipi (parametro `type`):

- misure di frequenza (`type=frequency(value)`): riferito al numero di casi che hanno eseguito/utilizzato una certa attività/risorsa. Possono essere in numero assoluto (`value = "absolute", value = "absolute-case"`) oppure in percentuale relativa rispetto ai casi totali (`value = "relative-case"`) o alle istanze (`value = "relative"`);
- misure di performance (`type=performance(FUN, units, flow_time)`): riferito al tempo impiegato ad eseguire/utilizzare una certa attività/risorsa. Sugli archi può essere indicato il tempo impiegato tra la fine dell'attività e la successiva (`flow_time="idle_time"`) oppure la differenza tra i timestamp di inizio di due attività consecutive (`flow_time="inter_start_time"`). I numeri indicati negli archi e nei nodi sono ottenuti applicando la funzione indicata nel parametro `FUN` (`mean, max, min, median,` *etc.*) ai valori calcolati e riportati nell'unità di misura indicata in `units`;
- altre misure definite dall'utente (`type=custom(FUN, units, attribute)`) in base ad altre variabili presenti nel dataset.

4.2.1 Esempi di process map con le tracce meno frequenti in sepsis

Costruiamo un sottoinsieme del dataset `sepsis` filtrando i casi sulla base delle tracce meno frequenti del 10% sui casi totali.

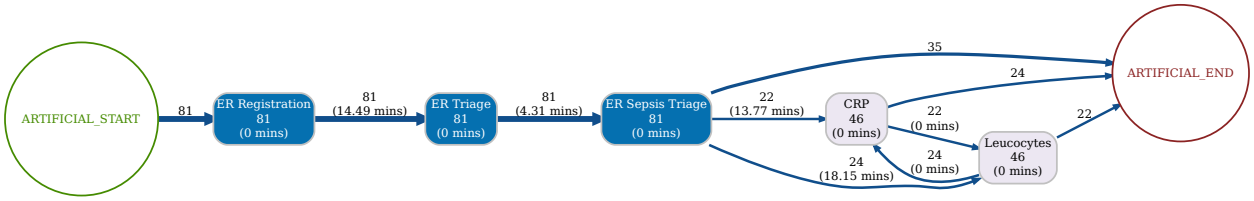
```
library(bupaR)
sepsis.trace_less_frequent <- sepsis %>% filter_trace_frequency(percentage = 0.1)
```

Costruiamo una process map da questo dataset limitandoci ai casi con max 5 attività (per rendere il grafo più leggibile).

Questa process map contiene una sia misura di frequenza (numero assoluto di casi) che una di performance tra parentesi (l'inter start time medio tra le attività).

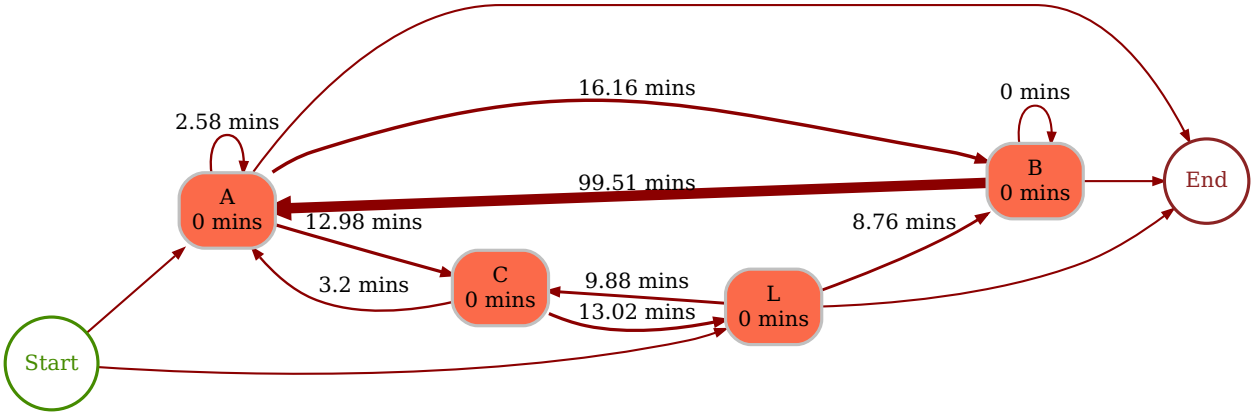
```
sepsis.trace_less_frequent %>%
  filter_trace_length(interval = c(1,5)) %>%
  process_map(
```

```
    type = frequency("absolute"),
    sec = performance(flow_time = "inter_start_time", units = "mins")
    )
```



In questa resource map, invece sono mostrati i tempi di inutilizzo (idle time) tra l'utilizzo di una risorsa e quella successiva.

```
sepsis.trace_less_frequent %>%
  resource_map(
    type = performance(units = "mins", flow_time = "idle_time")
    )
```



4.3 Matrici di precedenza

Un modo sicuramente meno immediato per visualizzare i processi rispetto ai grafi, ma più compatto. Il funzionamento è sostanzialmente quello delle matrici di adiacenza: sulla riga leggo un'attività/risorsa, sulle colonne l'attività/risorsa successiva da eseguire. All'interno della cella leggo il valore dell'arco corrispondente nel grafo di processo (quello ottenuto con `process_map`/`resource_map`).

Per costruire una matrice di processo si utilizzano le funzioni `precedence_matrix`/`resource_matrix`.

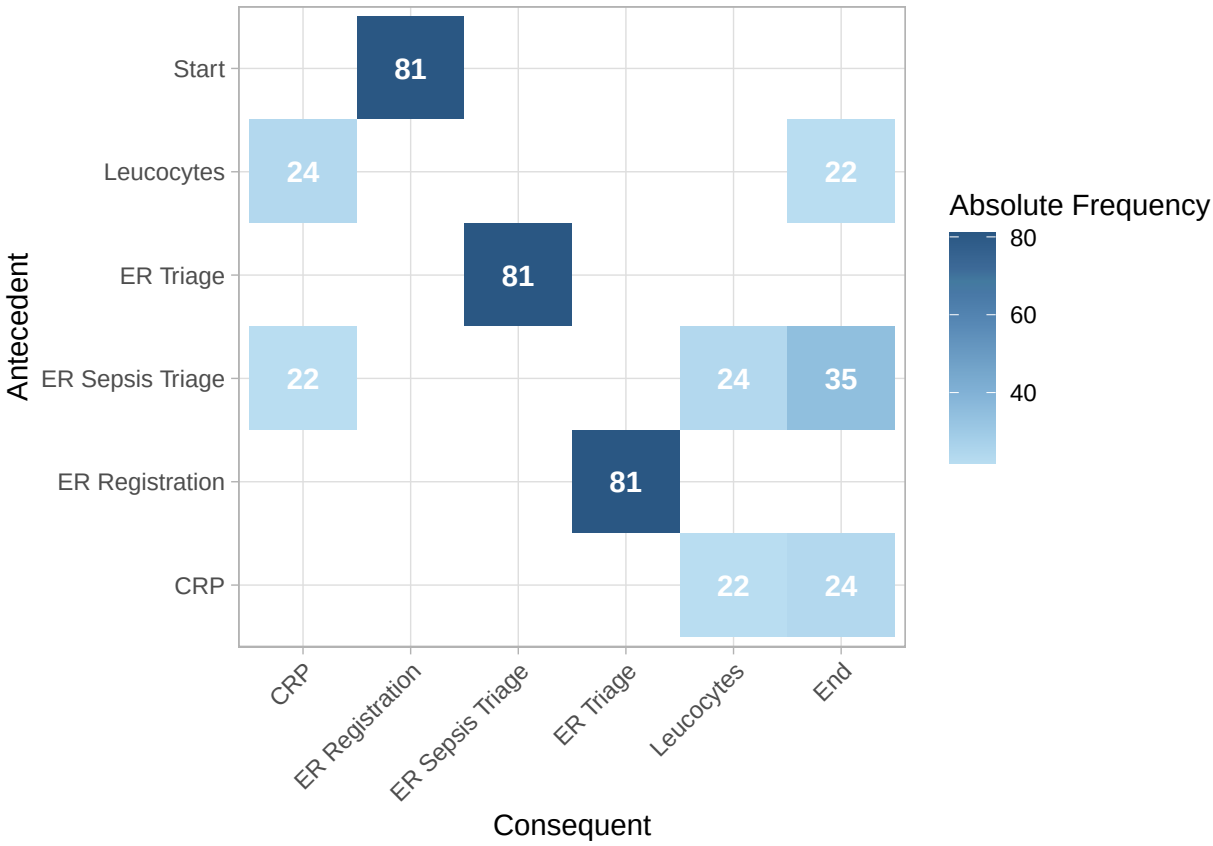
I valori all'interno delle celle possono essere di quattro tipi (parametro `type`), tutti riferiti al numero di casi (come per le misure di frequenza nei grafi):

- valori numerici assoluti (`type="absolute"`);
- valori numerici relativi rispetto alle istanze (`type="relative"`);
- per ogni riga, la percentuale relativa di casi che passano a una certa attività/risorsa (`type="relative-antecedent"`)
- per ogni colonna, la percentuale relativa di casi che provengono da una certa attività/risorsa (`type="relative-consequent"`)

4.3.1 Esempio di matrice di precedenza

Il questo esempio è stata riprodotta la stessa process map mostrata nella sezione precedente sotto forma di matrice di precedenza.

```
sepsis.trace_less_frequent %>%
  filter_trace_length(interval = c(1,5)) %>%
  precedence_matrix() %>%
  plot()
```



4.4 Trace explorer

Il *trace_explorer*, come suggerito dal nome, è una visualizzazione delle tracce (o di un sottoinsieme di queste) all'interno di un log di processo.

Si presenta come una tabella in cui ogni riga rappresenta una traccia e sulle colonne il numero di attività della traccia. In ogni cella viene inserita un'attività (contraddistinta da un colore riportato in legenda).

Nelle celle grigie a lato delle tracce vengono inserite delle misure di frequenza (parametro `coverage_labels`) che possono essere di tre tipi:

- numero assoluto di casi che hanno seguito la traccia (`coverage_labels="absolute"`);
- percentuale relativa dei casi che hanno seguito la traccia sui casi totali (`coverage_labels="relative"`);
- percentuale cumulativa dei casi (`coverage_labels="cumulative"`).

Se questo parametro non indicato vengono inserite tutte e tre le statistiche.

L'ordine in cui vengono mostrate le tracce è in base alla frequenza relativa della traccia e viene impostato attraverso il parametro `type`. Se impostato a `"frequent"` vengono messe in alto le tracce più frequenti, se impostato a `"infrequent"` vengono messe in alto le tracce meno frequenti.

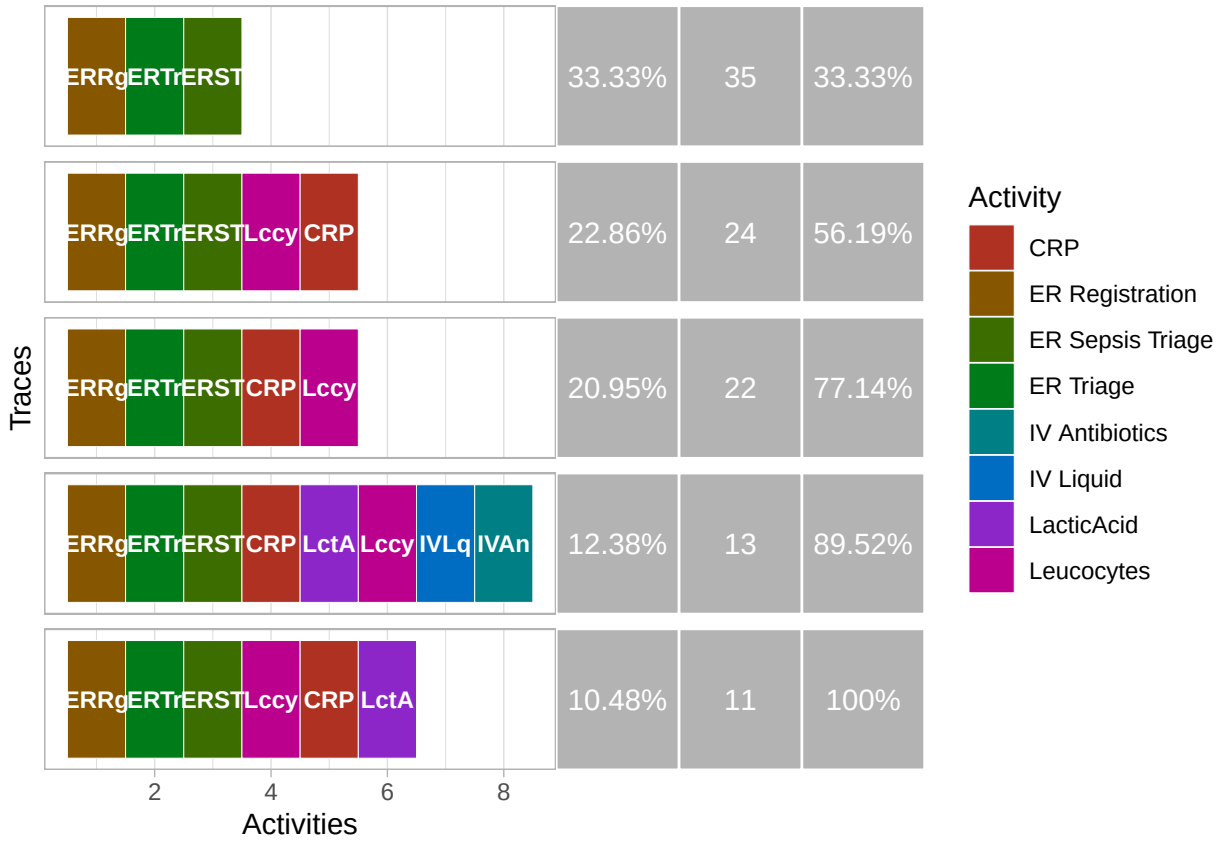
Il sottoinsieme di tracce da visualizzare si può impostare con due parametri: `n_traces` (si indica il numero di tracce da visualizzare in base al valore di `type`), oppure con `coverage` (indicando una soglia di frequenza

relativa).

4.4.1 Esempio di trace explorer

In questo trace explorer sono visibili tutte le possibili tracce presenti all'interno del dataset ridotto con le relative frequenze relative, assolute e cumulative.

```
sepsis.trace_less_frequent %>% trace_explorer(coverage = 1)
```



4.5 Dotted chart

Il *dotted_chart* mette in relazione i casi all'interno di un log rispetto alla loro durata.

Si presenta come una sorta di grafico a dispersione: sull'asse delle x viene messo il tempo assoluto (`x="absolute"`) o relativo rispetto al primo evento in ordine cronologico nel log (`x="relative"`), sulle y invece sono presenti i casi ordinati per inizio, fine o per durata.

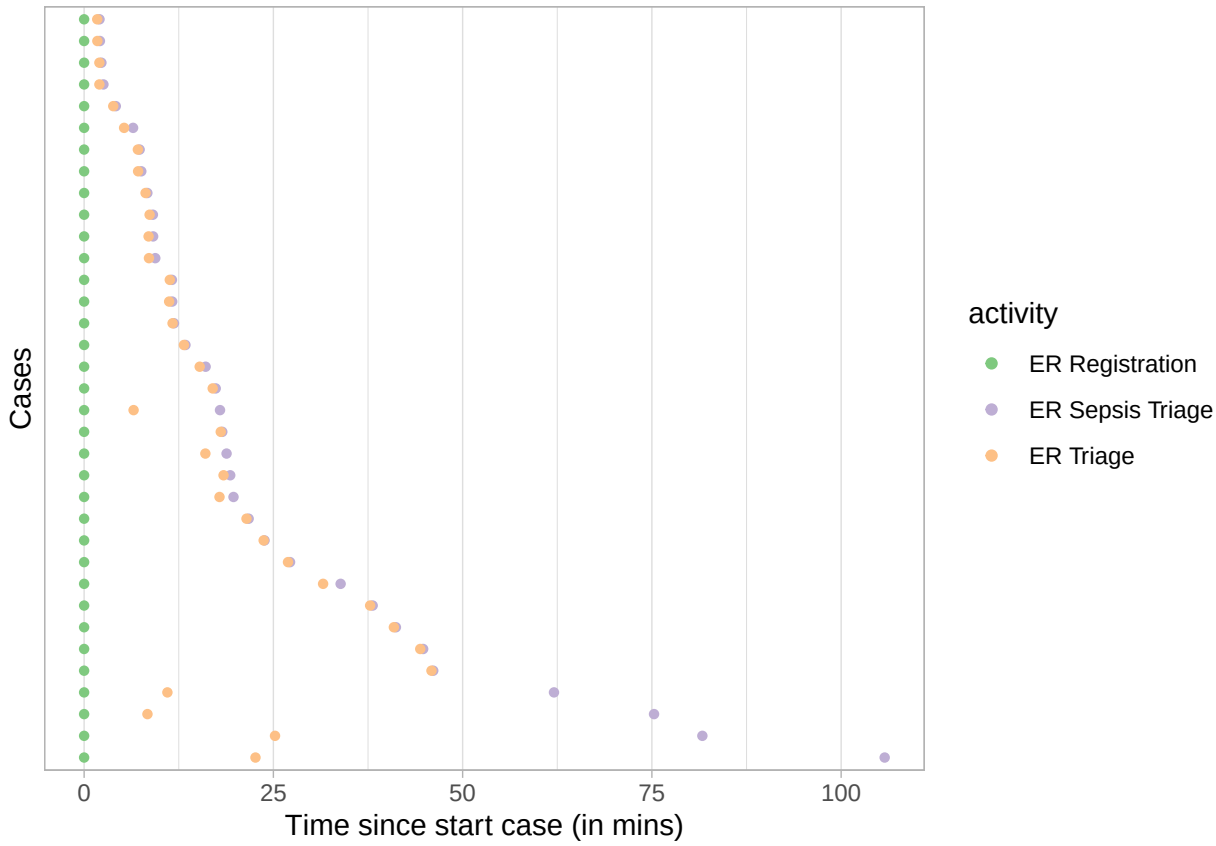
Per ogni attività del caso viene inserito un punto colorato in base alla fine dell'attività.

In questo modo si può visualizzare e confrontare in maniera più immediata la durata di uno o più casi all'interno del log di processo.

4.5.1 Esempio di dotted chart

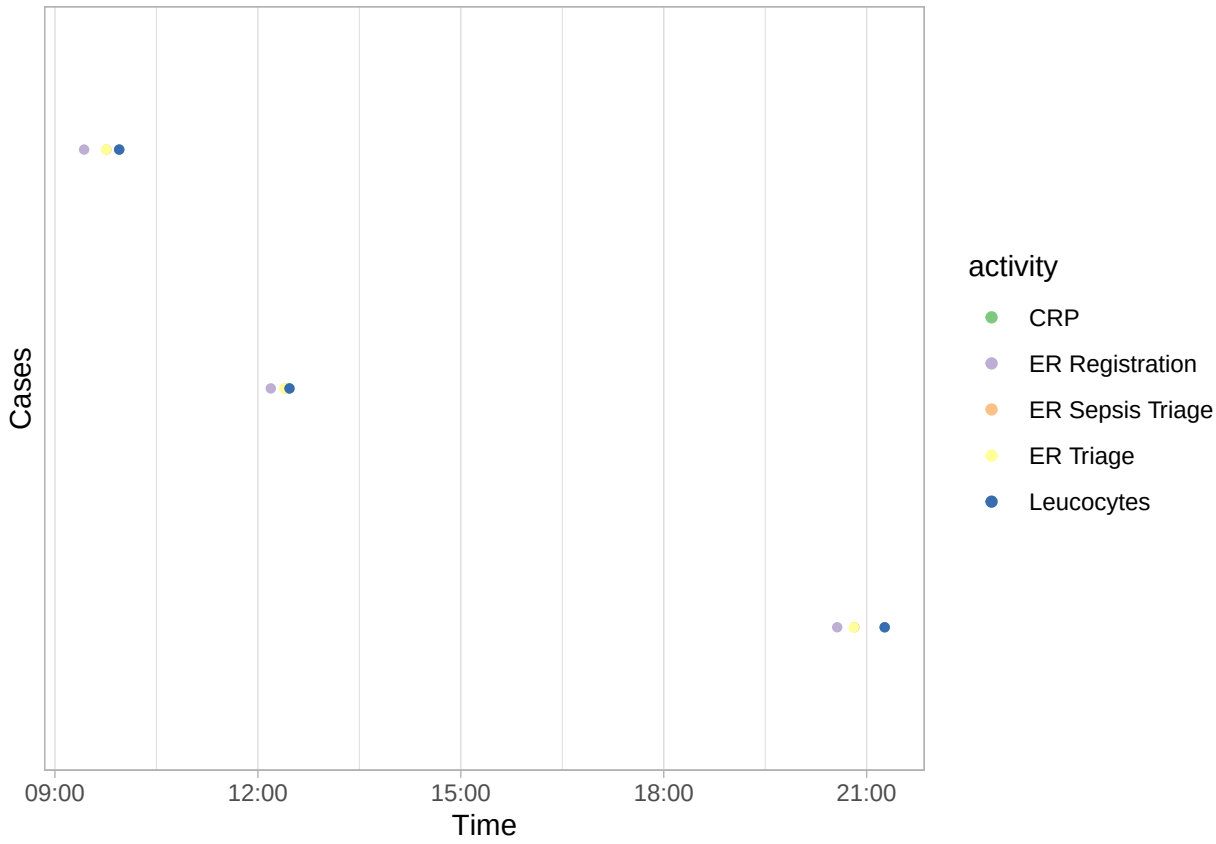
In questo dotted chart è visibile la durata (in minuti) dei casi con traccia *ER Registration -> ER Triage -> ER Sepsis Triage*.

```
sepsis.trace_less_frequent %>%
  filter_trace_length(interval = c(1,3)) %>%
  dotted_chart(x = "relative", units = "mins", sort = "duration")
```



In questo esempio, invece sono indagati i casi avvenuti dal 20 (compreso) al 21 (non compreso) novembre 2014.

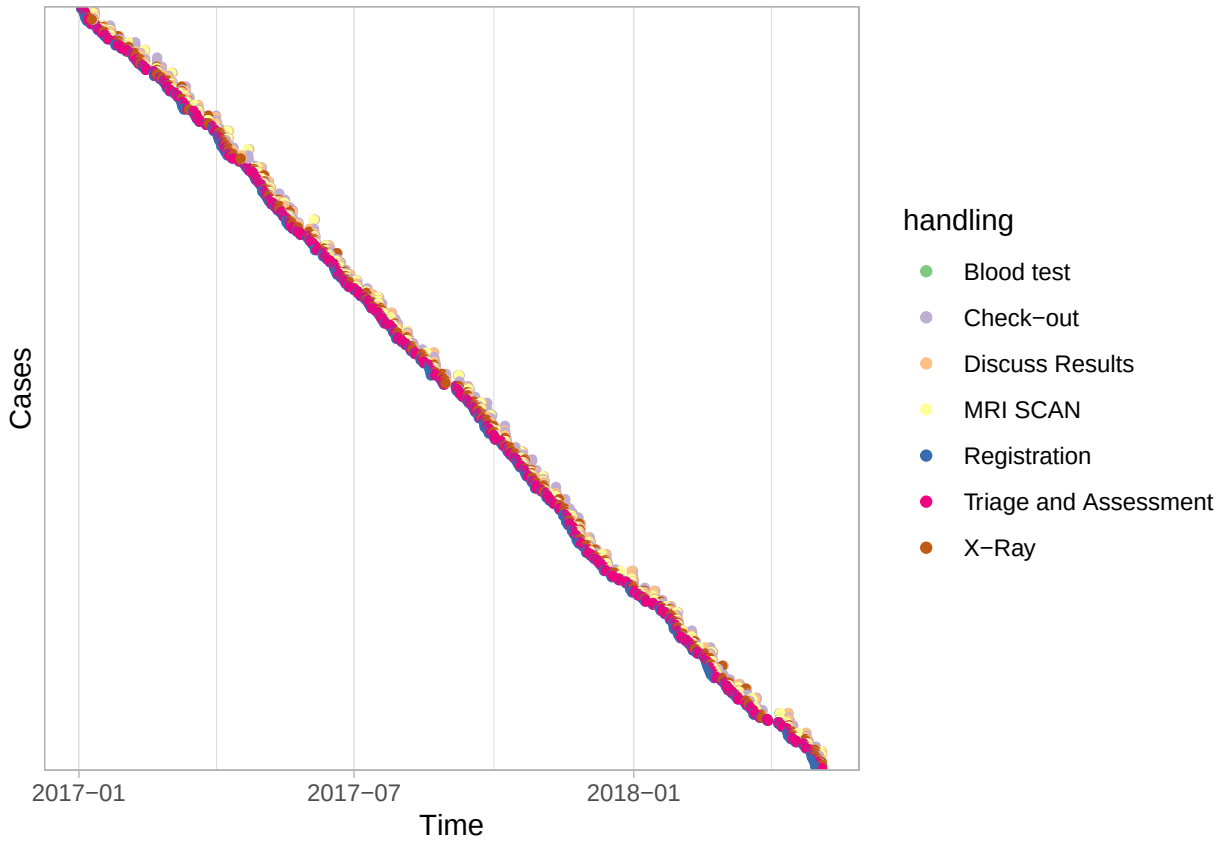
```
sepsis.trace_less_frequent %>%
  filter_time_period(interval = c(as.Date("2014-11-20"), as.Date("2014-11-21"))) %>%
  dotted_chart(x = "absolute", sort = "start")
```



4.6 Il problema di visualizzare dataset di grandi dimensioni

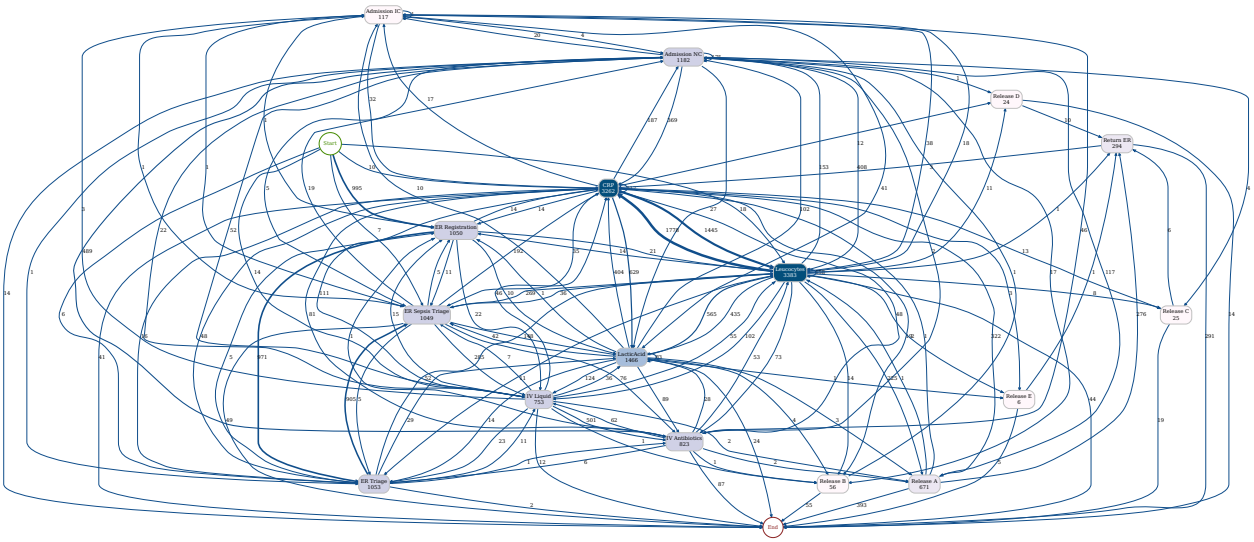
Uno dei problemi di tutte queste visualizzazioni qui presentate è la dimensione del dataset di partenza: più queste aumentano più si rischia di ottenere grafici complessi e di difficile comprensione. Si guardi ad esempio il dotted chart di `patients`:

```
patients %>% dotted_chart()
```



o la process map di `sepsis`:

```
sepsis %>% process_map(rankdir = "TB")
```



In questi casi è evidente come sia difficile (per non dire impossibile) ricavare informazioni utili sul processo in analisi, motivo per cui questo tipo di grafici è adatto solamente a visualizzare dei piccoli sottoinsiemi di casi, tracce o istanze molto specifici, motivo per cui la fase di subsetting e preprocessing dei dati acquisisce una particolare rilevanza.

5 Preprocessamento, subsetting e qualità dei dati

5.1 Le operazioni elementari su un file di log

Prima di affrontare il problema del pre-processameto di un file di log, si riporta un'introduzione sulle operazioni elementari svolgibili su un oggetto di tipo eventlog. Come anticipato nell'introduzione, il pacchetto bupar è pensato per essere utilizzata assieme ad altre librerie del contesto tidyverse. Per questo motivo, bupar ha ri-implementato alcune funzioni tipiche di dplyr per gli oggetti eventlog.

Di seguito, una rapida panoramica sulle funzioni disponibili e sul loro significato (tutte queste funzioni sono implementate dalla libreria `edeaR`).

- `select` <- consente di selezionare soltanto alcune variabili del dataset (in modo simile alla clausola "SELECT ..." del linguaggio SQL).
- `filter` <- consente di selezionare soltanto alcune osservazioni, in base ad una o più espressioni booleane (in modo simile alla clausola "WHERE ..." del linguaggio SQL).
- `mutate` <- consente di applicare delle modifiche di vario genere ad una o più variabili, dichiarando come parametri una serie assegnazioni (che possono sfruttare qualsiasi altra funzione di R, ad esempio operazioni aritmetiche, logiche, manipolazione di stringhe, aggregazioni statistiche, ecc.).
- `arrange` <- ordina le osservazioni del dataset in base ad un certo criterio (prevalentemente l'ordine crescente o decrescente di una o più variabili). Assomiglia alla clausola "ORDER BY ... ASC|DESC" del linguaggio SQL).
- `group_by` <- consente di svolgere tutte le successive operazioni considerando il dataset come "partizionato" in base al valore osservato per una o più variabili categoriali. È utile se combinato con funzioni come mutate (per effettuare un certo calcolo per che tiene conto della "categoria di appartenenza" - e.g., voglio associare ad ongi osservazione la differenza tra un valore osservato ed un valore medio di categoria) oppure summarize di dplyr (per ricavare aggregazioni di vario genere). Per certi versi ricorda il "GROUP BY" del linguaggio SQL. Olte alla semplice group_by, vengono forniti anche una serie di funzioni specifiche predefinite per raggruppare per caso, attività, tipo di attività, ecc.
- `slice` (e `slice_activities` e `slice_events`) <- per estrarre sotto-insiemi di casi, attività ed eventi. Ad esempio, slice permette di estrarre "i casi dal 5° al 15°," "gli eventi dal primo al decimo," ecc.
- `first_n` e `last_n` <- shortcut per ottenere le prime (o le ultime) n attività di un dataset.
- `sample_n` <- estai un campione casuale di n eventi.

Per comprendere meglio queste funzioni, la loro logica e il modo in cui vengono utilizzate, si suggerisce di approfondire prima la libreria dplyr e il pacchetto tidyverse.

5.2 Il problema dei dataset corposi

Quando si vuole fare attività di process mining, può capitare di trovarsi ad analizzare un log particolarmente "corposo." Un dataset di log può essere corposo sotto diversi aspetti.

Nella situazione migliore, potrei avere semplicemente un grande numero di casi ed eventi, e quindi tanti dati da analizzare. Solitamente però quando si ha un grande volume di dati (e quindi eventi) si ha anche una generale "eterogeneità" del dataset, specie dal punto di vista delle attività e delle tracce. Avere tanti tipi di attività possibili può complicare parecchio l'analisi. Una traccia è la sequenza di attività che viene svolta per un certo caso. Avere un gran numero di livelli per il tipo di attività, probabilmente implica che ci saranno anche tanti, molto diversi, tipi di "percorso" (traccia) che un certo caso può compiere. Questo compromette ad esempio la visualizzazione grafica, l'analisi delle frequenze delle attività/delle tracce, la comparazione di casi molto differenti e la ricerca di un "caso tipico" e più in generale la comprensione del processo.

```
library(bupaR)
hospital %>%
```

```
  select(timestamp, case_id, activity_instance_id, activity, lifecycle, group) %>%
  summary()
```

```
## Number of events:  150291
## Number of cases:  1143
## Number of traces:  981
## Number of distinct activities:  624
## Average trace length:  131.4882
##
## Start eventlog:  2005-01-03
## End eventlog:  2008-03-20
```

```
##    timestamp                      case_id          activity_instance_id
##  Min.   :2005-01-03 00:00:00   Length:150291      Length:150291
##  1st Qu.:2006-01-07 00:00:00   Class :character   Class :character
##  Median :2006-09-18 01:00:00   Mode  :character   Mode  :character
##  Mean   :2006-09-17 15:55:44
##  3rd Qu.:2007-06-28 01:00:00
##  Max.   :2008-03-20 00:00:00
##
##                                      activity        lifecycle
##  aanname laboratoriumonderzoek           :15353   complete:150291
##  ligdagen - alle spec.beh.kinderg.-reval.:10897
##  190205 klasse 3b        a205            : 9351
##  ordertarief                             : 9008
##  190101 bovenreg.toesl.  a101            : 6241
##  vervolgconsult poliklinisch             : 5239
##  (Other)                                 :94202
##                               group           .order
##  General Lab Clinical Chemistry :94917   Min.   :     1
##  Nursing ward                   :31066   1st Qu.: 37574
##  Obstetrics & Gynaecology clinic: 7065   Median : 75146
##  Medical Microbiology           : 4170   Mean   : 75146
##  Radiology                      : 3171   3rd Qu.:112718
##  (Other)                        : 9886   Max.   :150291
##  NA's                           :   16
```

Nel dataset in questione, ad esempio, si osserva un grande numero di eventi, casi, attività e tracce. Si osserva inoltre una lunghezza media della traccia piuttosto considerevole e un orizzonte temporale abbastanza ampio. Come si illustra nelle fasi successive, un dataset del genere risulta difficile da trattare.

5.2.1 Subsetting: Estrazione di porzioni di un dataset

Il metodo principale per il ridimensionamento di un log particolarmente corposo e complesso è il subsetting. Il subsetting consiste nell'estrazione di porzioni piccole del dataset, cioè di sotto-insiemi di tutti gli eventi che siano più facilmente approcciabili per un'analisi.

La selezione degli eventi può avvenire in due macro-modi: filtrando sugli eventi (in base ad un certo criterio) oppure filtrando sui casi (scartando quindi tutti gli eventi correlati a casi che non rispecchiano certe caratteristiche).

5.3 Subsetting sugli eventi

Il subsetting sugli eventi consiste nell'ottenere una copia del dataset, contenente solo le righe che rispettano certe condizioni legate ad eventi, attività e risorse. Di seguito, una panoramica dei tipi di subsetting sugli eventi, associati alla rispettiva funzione `edeaR`.

- `filter_activity` <- estrai solo gli eventi associati ad una o più attività.
- `filter_activity_frequency` <- estrai gli eventi associati ad attività che:
  - si riferiscono ad attività che appartengono ad un certo percentile delle attività più/meno frequenti;
  - si riferiscono ad attività di frequenza appartentente ad un certo intervallo.
- `filter_resource` <- estrai gli eventi associati ad una o più risorse. Analogamente alla funzione per le attività, esiste anche la variante filter_resource_frequency, che presenta opzioni simili.
- `filter_trim` <- questa particolare funzione consente di selezionare solo eventi "compresi" tra due attività. Essendo le tracce sequenze di attività, posso selezionare solo eventi che vengono dopo un certo tipo di attività e prima di un'altro. In questo modo, ad esempio, posso focalizzarmi solo su un pezzo particolare del processo ipotizzato.
- `filter_time_period` <- seleziona solo eventi inclusi in un certo range temporale.

5.3.1 Esempio: estrazione di un subset di `sepsis` con gli eventi avventuti tra aprile e maggio 2015

```
sepsis %>%
  filter_time_period(
    interval = c(as.Date("2015-4-1", "%Y-%m-%d"), as.Date("2015-5-31", "%Y-%m-%d")),
    filter_method = "trim"
  ) %>% select() %>% summary
```

```
## Number of events:  14
## Number of cases:  14
## Number of traces:  1
## Number of distinct activities:  1
## Average trace length:  1
##
## Start eventlog:  2015-04-01 14:51:36
## End eventlog:  2015-05-30 11:14:55
```

```
##    case_id                     activity  activity_instance_id
##  Length:14          Return ER       :14   Length:14
##  Class :character   Admission IC    : 0   Class :character
##  Mode  :character   Admission NC    : 0   Mode  :character
##                     CRP             : 0
##                     ER Registration : 0
##                     ER Sepsis Triage: 0
##                     (Other)         : 0
##    timestamp                      resource     lifecycle      .order
##  Min.   :2015-04-01 14:51:36   ?      :14   complete:14   Min.   : 1.00
##  1st Qu.:2015-04-07 12:48:13   A      : 0                 1st Qu.: 4.25
##  Median :2015-04-26 07:01:28   B      : 0                 Median : 7.50
##  Mean   :2015-04-24 12:02:26   C      : 0                 Mean   : 7.50
##  3rd Qu.:2015-05-07 06:44:50   D      : 0                 3rd Qu.:10.75
##  Max.   :2015-05-30 11:14:55   E      : 0                 Max.   :14.00
##                                (Other): 0
```

5.3.2 Esempio: estrazione di un subset solo con le attività sufficientemente frequenti

```
sepsis %>% filter_activity_frequency(percentage = 0.9) %>% activities
```

```
## # A tibble: 9 x 3
##   activity         absolute_frequency relative_frequency
##   <fct>                         <int>              <dbl>
## 1 Leucocytes                     3383             0.241
## 2 CRP                            3262             0.233
## 3 LacticAcid                     1466             0.105
## 4 Admission NC                   1182             0.0843
## 5 ER Triage                      1053             0.0751
## 6 ER Registration                1050             0.0749
## 7 ER Sepsis Triage               1049             0.0748
## 8 IV Antibiotics                  823             0.0587
## 9 IV Liquid                       753             0.0537
```

5.3.3 Esempio: estrazione di eventi che NON cominciano con "ER Registration" e le relative tracce

```
sepsis %>% filter_trim(start_activities = "ER Registration", reverse = T) %>% traces
```

```
## # A tibble: 12 x 3
##    trace                                    absolute_frequency relative_frequen~
##    <chr>                                                 <int>             <dbl>
##  1 IV Liquid                                                14            0.255
##  2 Leucocytes,CRP                                           13            0.236
##  3 CRP,Leucocytes                                            7            0.127
##  4 ER Triage                                                 5            0.0909
##  5 ER Sepsis Triage                                          5            0.0909
##  6 Leucocytes                                                5            0.0909
##  7 ER Sepsis Triage,IV Antibiotics,IV Liqu~                  1            0.0182
##  8 CRP                                                       1            0.0182
##  9 ER Sepsis Triage,IV Liquid,IV Antibioti~                  1            0.0182
## 10 CRP,Leucocytes,LacticAcid                                 1            0.0182
## 11 CRP,LacticAcid,Leucocytes                                 1            0.0182
## 12 ER Triage,Leucocytes,CRP,LacticAcid,Leu~                  1            0.0182
```

5.4 Subsetting sui casi

Il subsetting sui casi serve a filtrare direttamente i casi. Di seguito, una panoramica dei tipi di subsetting sugli eventi, associati alla rispettiva funzione `edeaR`.

- `filter_activity_presence` <- permette di filtrare i casi in base alla presenza o meno di un gruppo di attività nella traccia seguita. Esistono tre varianti (indicate con il parametro `method`):
  - `"all"`: tutte le attività devono essere presenti nella traccia,
  - `"none"`: nella traccia non è presente nessuna delle attività,
  - `"one_of"`: almeno una attività deve essere presente nella traccia.
- `filter_endpoints` <- filtra i casi in base all'attività iniziale e/o finale
- `filter_processing_time, filter_throughput_time, filter_time_period` <- filtra i casi in base a un intervallo temporale (relativo per filter_processing_time e filter_throughput_time, assoluto per filter_time_period).
- `filter_trace_frequency, filter_trace_length` <- filtra i casi in base alla frequenza della traccia o alla sua lunghezza.

5.4.1 Esempio: casi di sepsis tale che i pazienti hanno svolto le attività IV Antibiotics e IV Liquid

```
sepsis %>% filter_activity_presence(
  activities = c("IV Antibiotics", "IV Liquid"),
  method = "all"
) %>% cases %>% head
```

```
## # A tibble: 6 x 10
##   case_id trace_length number_of_activi~ start_timestamp     complete_timestamp
##   <chr>          <int>             <int> <dttm>              <dttm>
## 1 A                 22                10 2014-10-22 11:15:41 2014-11-02 15:15:00
## 2 AA                 8                 8 2014-12-03 09:06:44 2014-12-03 14:28:01
## 3 AAA               11                11 2014-11-19 03:16:21 2014-11-28 16:15:17
## 4 AB                 8                 8 2014-02-16 09:55:43 2014-02-16 13:52:06
## 5 ABA               17                10 2014-10-12 11:22:24 2014-10-18 16:15:00
## 6 AC                13                11 2014-09-24 15:39:13 2014-12-02 18:47:17
## # ... with 5 more variables: trace <chr>, trace_id <dbl>,
## #   duration_in_days <dbl>, first_activity <fct>, last_activity <fct>
```

5.4.2 Esempio: casi di sepsis con processing time $<$ 1 settimana

```
sepsis %>% filter_processing_time(
  interval = c(NA, 1),
  units = "weeks"
) %>% cases %>% head
```

```
## # A tibble: 6 x 10
##   case_id trace_length number_of_activi~ start_timestamp     complete_timestamp
##   <chr>          <int>             <int> <dttm>              <dttm>
## 1 A                 22                10 2014-10-22 11:15:41 2014-11-02 15:15:00
## 2 AA                 8                 8 2014-12-03 09:06:44 2014-12-03 14:28:01
## 3 AAA               11                11 2014-11-19 03:16:21 2014-11-28 16:15:17
## 4 AB                 8                 8 2014-02-16 09:55:43 2014-02-16 13:52:06
## 5 ABA               17                10 2014-10-12 11:22:24 2014-10-18 16:15:00
## 6 AC                13                11 2014-09-24 15:39:13 2014-12-02 18:47:17
## # ... with 5 more variables: trace <chr>, trace_id <dbl>,
## #   duration_in_days <dbl>, first_activity <fct>, last_activity <fct>
```

5.5 Verifiche di qualità

Quando si ha a che fare con dati reali, oltre che i problemi del formato, della dimensione e dell'eterogeneità può capitare anche di dover verificare la qualità e la "correttezza" del dataset. In un file di log reale difatti possono esserci diversi tipi di anomalie e violazioni (e.g., anomalie temporali, attività ripetute troppe volte per lo stesso caso, nomi di attività errati, valori fuori range, ecc.).

Con il pacchetto `daqapo` è possibile effettuare diversi tipi di verifiche di qualità, che non verranno trattate in questo elaborato.[3]

[3] documentazione ufficiale: https://bupar.net/daqapo.html)

6 Dashboard

Il pacchetto `processmonitoR` di bupaR fornisce delle dashboard interattive sottoforma di web app create con `shiny` (una suite di pacchetti per generare pagine web a partire da codice R) e, quindi, facilmente integrabili in applicazioni di terze parti.

All'interno di queste dashboard vengono mostrati alcuni grafici classici ottenuti con le funzioni del pacchetto `edeaR` viste nel capitolo 3 sul dataset passato come parametro.

Una dashboard è composta da una o più schede, in ognuna viene visualizzato un tipo grafico. Nel caso in cui siano previsti livelli di dettaglio diversi, è possibile selezionare il livello desiderato dalla colonna a sinistra dell'immagine.

6.1 Activity dashboard

Questa dashboard contiene due schede:

- Nella prima si riporta il risultato dell'espressione

```
eventlog %>% activity_frequency(level="activity") %>% plot
```

```
Mostrando la frequenza delle attività nel log in valore assoluto;
```

- Nella seconda il grafico è ottenuto con

```
eventlog %>% activity_presence() %>% plot
```

```
Mostrando la frequenza relativa delle attività rispetto ai casi totali.
```

6.2 Performance dashboard

Questa dashboard si concentra sull'analisi della prospettiva temporale, per cui esiste una scheda per ognuno dei tre aspetti: throughput time, processing time, idle time.

Inoltre, è possibile specificare l'unità di misura per l'asse delle ascisse dal menù a tendina in basso a sinistra.

I tre grafici vengono generati dalle seguenti espressioni:

```
eventlog %>%
  idle_time(
    level = "scelto dall'interfaccia",
    units = "scelto dall'intefaccia"
  ) %>% plot

eventlog %>%
  processing_time(
    level = "scelto dall'interfaccia",
    units = "scelto dall'intefaccia"
  ) %>% plot

eventlog %>%
  throughput_time(
    level = "scelto dall'interfaccia",
    units = "scelto dall'intefaccia"
  ) %>% plot
```

6.3 Rework dashboard

Questa dashboard è composta da due schede:

- Nella prima scheda vengono messi in relazione le risorse utilizzate con il numero di self-loop di tipo "redo" (ossia quando un'attività viene eseguita almeno due volte consecutivamente da due risorse diverse). Alla casella sulla riga i e sulla colonna j è riportato il numero di redo self-loop iniziati utilizzando la risorsa j e terminati con la risorsa i.
- Nella seconda scheda, invece, sono riportati il numero di ripetizioni di tipo "redo" (quando un'attività viene ripetuta almeno due volte nello stesso caso da due risorse diverse non consecutivamente) in relazione alle risorse utilizzate. Alla casella sulla riga i e sulla colonna j è riportato il numero di ripetizioni redo iniziati utilizzando la risorsa j e terminati con la risorsa i.

6.4 Resource dashboard

Questa dashboard invece fornisce delle misure sull'utilizzo delle risorse all'interno dei processi.

Sono presenti tre schede:

- Resource frequency: rappresenta il grafico ottenuto dall'espressione

```
eventlog %>%
  resource_frequency(
    level = "scelto dall'interfaccia"
  ) %>% plot
```

I grafici ottenuti rappresentano la frequenza assoluta di utilizzo delle risorse nel log (il grafico ottenuto dipende dal livello scelto).

- Resource involvement: rappresenta il grafico ottenuto dall'espressione

```
eventlog %>%
  resource_involvement(
    level = "scelto dall'interfaccia"
  ) %>% plot
```

Il resource involvement è una misura dell'utilizzo delle risorse in rapporto ai casi in un log di processo (quante/quali risorse vengono utilizzate per un certo caso)

- Resource specialisation rappresenta il grafico ottenuto dall'espressione

```
eventlog %>%
  resource_specialisation(
    level = "scelto dall'interfaccia"
  ) %>% plot
```

Il resource specialization misura quanto una specifica risorsa venga usata da quali attività.

6.5 Esempio di dashboard

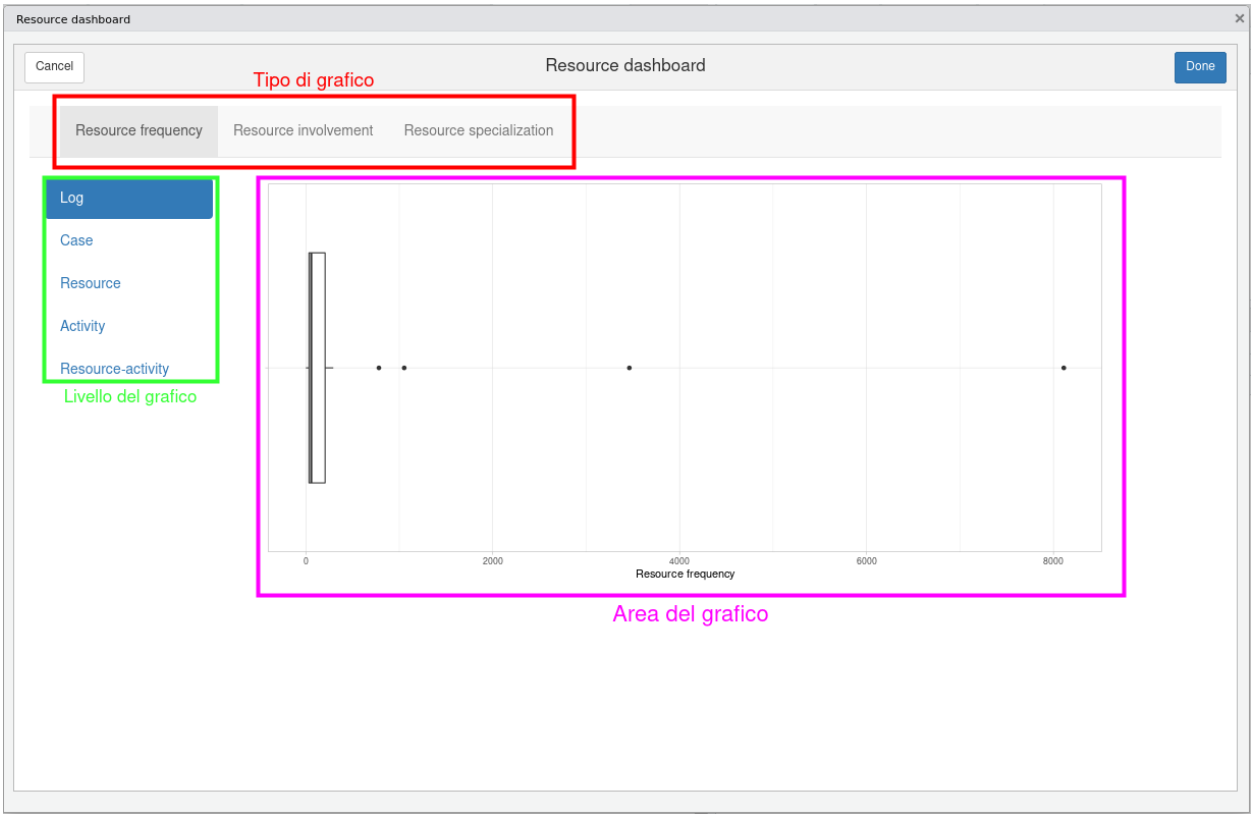


Figure 2: Resource dashboard di `sepsis` con le aree di interesse evidenziate

Bibliografia

Aalst, Wil van der, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, et al. 2011. “Process Mining Manifesto.” In *International Conference on Business Process Management*, 169–94. Springer; "https://link.springer.com/content/pdf/10.1007/978-3-642-28108-2_19.pdf".

bupar.net. n.d. “bupaR Documentation - Getting Started.” *Getting Started*. https://bupar.net/getting_started.html.