Analisi dei dati con Python Pandas, un caso d'uso: dalla contabilità di OpenERP alla produzione di report

Marco Pattaro
<mpattaro@gmail.com>
<marco.pattaro@servabit.it>

ERLUG

Data Analisys

- Gli ambienti più diffusi per statistica e data analisys
  - SAS
  - S+
  - Stata
  - Matlab
  - R (GNU GPLv3)
- Questi sono tutti linguaggi domain specific
  - Ottimi per analisti dei dati, molto limitati per implementare applicazioni.
  - Spesso si pone il problema di integrare procedure implementate con questi linguaggi in applicazioni scritte con linguaggi più generici (C++, Java, .NET, Python, etc).
  - Non sarebbe meglio poter scrivere tutto con lo stesso linguaggio senza perdere di usabilità nel contesto “Data analisys” e espressività nel contesto “Sviluppo di applicazioni”?
  - Obiettivo ultimo: ridurre il gap culturale tra analisti e sviluppatori.

Python

- Linguaggio multi-paradigma:
  - si presta alla scrittura di brevi procedure come di grandi sistemi software
- Interpretato: si presta ad un utilizzo interattivo
- Performante:
  - non sempre
  - ma facilmente estendibile tramite linguaggi di più basso livello (C/C++, Fortran)
- Comunity molto attiva, esistono moltissimi framework e librerie
- Relativamente semplice anche per non informatici
- Open Source (BSD License)

Python (2)

- Python per l'analisi scientifica:
  - Calcolo numerico e operazioni su matrici (SciPy, NumPy)
    - http://www.scipy.org
    - http://numpy.scipy.org/
  - Produzione di grafici (matplotlib)
    - http://matplotlib.org/
  - ORM (SqlAlchemy)
    - http://www.sqlalchemy.org/
  - Un ambiente di ricerca interattivo (ipython)
    - http://ipython.org/
- Mancava un linguaggio specifico per la Data Analisys

Pandas!

http://pandas.pydata.org/

Pandas

- Etimologia: **pan**el **da**ta **s**tructures
- Nasce nel 2008 (rilasciato sotto licenza BSD nel 2009)
- Obiettivi:
  - Fornire delle strutture dati semplici per lavorare su insiemi di dati strutturati.
  - Creare una base intuitiva che consenta, sia ad utenti che sviluppatori, l'implementazione di modelli statistici.
  - Fornire un insieme di strumenti per le operazioni di analisi più comuni.
- Ispirato ad R per molti versi, si pone l'obiettivo di perfezionarne alcuni aspetti (es. data alignment)
- Idea di fondo: `numpy.ndarray` con etichette sugli assi e... un sacco di metodi!

numpy.ndarray

- Array multidimensionale (N Dimensional Array)
  - Omogeneo
  - Contiene oggetti di dimensione fissa
  - Associato ad un oggetto (`dtype`) che descrive il formato degli elementi:
    - Int, float, bool, object
    - Byte order
    - Size
- Molti metodi
  - Operazioni condizionali su serie di elementi: all(), any() ...
  - Operazioni di aggregazione: sum(), mean() ...
  - Operazioni vettoriali: dot(), transpose() ...

DataFrame

- Il DataFrame è la struttura portante di Pandas
  - Ispirato al `data.frame` di R e al dataset di SAS
  - Ha le funzionalità di calcolo vettoriale di numpy.ndarray
  - Lo si può pensare come un dizionario di liste (per quanto riguarda l'interfaccia di base)
- Funzionalità aggiuntive
  - Consente di associare etichette agli assi
  - Consente tipi di dati eterogenei
  - Indici multi-livello
  - Missing data handling
- Operazioni
  - Slicing
  - Reshape
  - Merge (database join)
  - Raggruppamento (group by) e aggregazione

DataFrame (Un primo sguardo)

- In prima approssimazione un DataFrame è una tabella bidimensionale contenente tipi eterogenei
- Le colonne e le righe (indici) possono essere identificate da etichette

```
>>> df = DataFrame.from_csv('data.csv')
   YEAR  COUNTRY      AMOUNT
0  2009      IND    0.002000
1  2008      FRA  283.221985
2  2010      SVK    9.337000
3  2011      SVK   57.012001
4  2010      SVK   21.473000
```

DataFrame selezione

- Selezionare colonne è molto semplice

```
>>> df[['YEAR', 'X']]
   YEAR           X
0  2009    0.002000
1  2008  283.221985
2  2010    9.337000
3  2011   57.012001
4  2010   21.473000
```

- Una selezione condizionale per righe (analoga ad uno statement `where` in SQL)

```
>>> df[(df['YEAR'] == 2011) and (df['COUNTRY']== 'CZE')]
   YEAR COUNTRY     AMOUNT
3  2011     CZE  57.012001
```

DataFrame indexing e slicing

- È possibile accedere facilmente alle righe (indici) di un DataFrame tramite il “magic attribure” `ix` e la usuale sintassi Python
  - Ad un singolo elemento

```
>>> df.ix[4]
   YEAR COUNTRY      AMOUNT
4  2010     DEU  21.473000
```

  - Oppure ad una slice

```
>>> df.ix[3:7]
   YEAR COUNTRY      AMOUNT
3  2011     CZE  57.012001
4  2010     DEU  21.473000
5  2011     DEU  12.076000
6  1998     HKG   0.015047
```

DataFrame indici

- Finora si è usato un indice progressivo per identificare i record, perché non utilizzare una colonna?

```
>>> df.set_index(['COUNTRY', 'YEAR'])
                    AMOUNT
COUNTRY YEAR
    AUS 2009     0.002000
CZE 2008   283.221985
    2010     9.337000
    2011    57.012001
DEU 2010    21.473000
    2011    12.076000
```

DataFrame Indici

- Abbiamo creato un indice a due livelli che può essere trattato esattamente come un array

```
>>> df.ix['ITA'][2011]
                X
YEAR
2011   0.009524
```

- Più chiaro in presenza di più di 3 dimensioni
- Può essere visto come una rappresentazione bidimensionale di dati n-dimensionali
- Particolarmente utile nel rendere efficienti le operazioni di join e aggregazione

DataFrame pivot

- Immaginiamo ora di volere organizzare i dati per anno e paese:

```
>>> piv = df.pivot('YEAR', 'COUNTRY', 'X')
        AUS   CZE   DEU   HKG   IND   ITA   MEX
1990   2.45  3.56  7.53  9.15  8.19  4.23  5.15
1991   4.26  2.34  1.45  2.45  6.72  8.19  8.93
1992   6.12  9.70  4.65  7.87  3.54  6.42  9.88
```

- Come con gli indici possiamo accedere ai dati come se fosse un array

```
>>> piv['MEX'][1992]
9.88
```

- Aumentando il numero di dimensioni il meccanismo crea indici gerarchici ma non perde di efficienza (anche se il risultato appare visivamente meno chiaro)

DataFrame aggregazioni

- Spesso è utile calcolare una qualche funzione che aggreghi i dati. Analogamente all'SQL

```
> select country, sum(amount)
  from mytable
  group by country
```

- É possibile esprimere la medesima query ad un DataFrame

```
>>> df.groupby('country').sum('amount')
```

- `groupby()` crea un oggetto iterabile DataFrameGroupBy

```
>>> grouped = df.groupby('country')
>>> for name, group in grouped:
        do_some_cool_stuff()
```

- Dove ogni `group` è a sua volta un DataFrame

Merge, join e concat

- Pandas fornisce vari metodi per combinare tra loro i dati
  - Concat: Concatena oggetti diversi lungo i diversi assi

```
concat(objs, axis=0, join='outer', join_axes=None,
       ignore_index=False, keys=None, levels=None,
       names=None, verify_integrity=False)
```

  - Merge: può verificare la corrispondenza su colonne qualsiasi (indice o no), chiaramente un join su indici è più efficiente.

```
merge(left, right, how='left', on=None, left_on=None,
      right_on=None, left_index=False,
      right_index=False, sort=True,
      suffixes=('.x', '.y'), copy=True)
```

  - Join : è una shortcut per indicare un merge() indice-a-indice o indice-a-colonna

OpenERP

- ERP Open Source (AGPLv3) modulare ed estendibile
- Classico MVC scritto in Python
- Implementa un ORM costruito su PostgreSQL
- Comunità molto attiva
- Gran quantità di moduli (principalmente per la gestione di cicli di produzione/approvvigionamento)
- Modulo di contabilità con localizzazione italiana in continua evoluzione

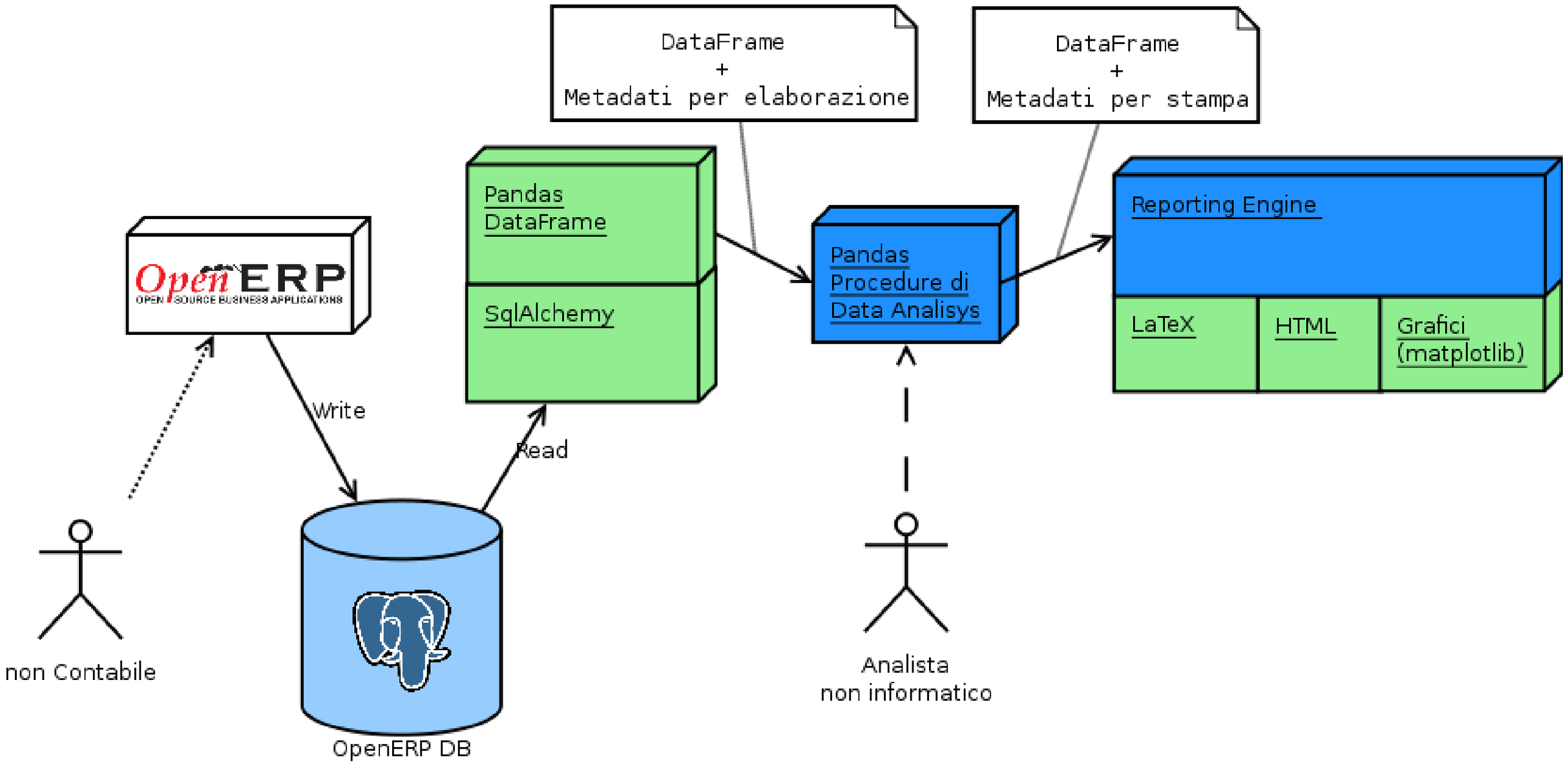
OpenERP: Pro e contro

- Pro
  - Semplice da estendere per un informatico
  - Semplice da usare per l'utente finale
- Contro
  - Difficile da personalizzare per un utente privo di conoscenze informatiche. Il sistema fornisce un frontend grafico per la personalizzazione, ma richiede comunque una comprensione piuttosto profonda dei meccanismi che realizzano il sistema
- Nello sviluppo di un software “produttivo” la difficoltà maggiore sta nel recepire le esigenze dell'utente...
- ... e se l'utente potesse contribuire non solo in fase di analisi, ma fosse in grado di aggiungere funzionalità in autonomia?

OpenERP Goal e Pandas

- In Servabit abbiamo creato una personalizzazione della contabilità di OpenERP, chiamata GOAL, con questi obiettivi:
  - Togliere la complessità delle scritture in partita doppia, queste vengono effettuate, ma l'utente non ha (quasi) mai necessità di vederle.
  - Creare una serie di report legali
  - Creare una base su cui poter fare controllo di gestione
- La parte difficile è stata creare i report!
  - Difficoltà di comunicazione tra contabili e informatici
- Pandas non è uno strumento tanto diverso da un foglio elettronico (come operazioni che consente di fare sui dati), con la differenza che è molto più scalabile, efficiente e integrabile in altro codice Python.

Architettura



Risultati

- Utenti con competenze da analisti economici e contabili sono stati in grado di implementare procedure per elaborare i dati che popolano i seguenti report
  - Libro giornale
  - Registri IVA
  - Bilancio d'esercizio
  - Flussi di cassa
  - Scadenziario
- Simili procedure, in un contesto più ampio, si stanno sviluppando per:
  - Analisi di bilancio
  - Flussi di commercio internazionale
- La comunicazione è notevolmente semplificata dal momento che l'informatico diventa una guida, più che un interprete, il cui lavoro consiste nel “limare” procedure definite da chi è maggiormente addentro alle materie specifiche.

GRAZIE PER L'ATTENZIONE

Le slides e le riprese audio/video dell'intervento saranno disponibili su:

http://erlug.linux.it/linuxday/2012/