Università degli Studi di Padova

Corso di Laurea in Ingegneria Informatica

Tecnologie per l'Integrazione e l'Elaborazione dell'Informazione

Laureando: Stefano Chinellato

Relatore: Prof. G.Clemente

Dipartimento di Ingegneria dell'Informazione

Anno Accademico 2012-1013

Indice

Introduzione

La rivoluzione dell'Informazione e l'era dei Social Network hanno portato ad una crescita esponenziale nella quantità di dati che vengono generati. Il tasso crescente con il quale viene prodotta nuova informazione, la facilità con la quale questi dati vengono duplicati e trasmessi su internet e l'incremento di canali accessibili attraverso i quali questa trasmissione avviene, ha portato ad un sovraccarico di dati incorrelati che spesso mancano di una struttura che ne riveli relazioni. Con il termine "Analisi di Dati" si vuole intendere il processo che, attraverso l'ispezione, la trasformazione e la modellazione dei dati, ha l'obiettivo di trovare ed evidenziare informazioni che possano essere di supporto nei processi decisionali.

L'idea di partenza di questo lavoro è la ricerca di uno strumento open source, che consenta di affrontare nella sua globalità un moderno processo di analisi di dati. La natura eterogenea delle possibili sorgenti di dati e dei formati, la necessità di strumenti che consentano la manipolazione, l'elaborazione e la visualizzazione dei dati e, infine la necessità di strumenti per la presentazione dei risultati dell'analisi, hanno spesso comportato l'utilizzo di un insieme di tecnologie distinte per poter portare a termine tutte le fasi del processo di analisi.

In questo lavoro si presenteranno un insieme di strumenti basati sul linguaggio di programmazione Python, che rappresentano un soluzione tecnologica omogenea alle problematiche presenti nelle varie fasi dell'elaborazione dell'informazione. Python è un linguaggio di programmazione general-purpose, dinamico ed interpretato che offre supporto per molteplici paradigmi di programmazione. Il recente sviluppo di librerie scientifiche, di manipolazione dei dati, la facilità di integrazione con linguaggi come C, C++ e FORTRAN e la sua natura general-purpose, lo rende una soluzione ideale per questo tipo di problematiche.

L'elaborato è suddiviso in due parti principali:

1. Introduzione al problema del processo di elaborazione dell'informazione. Caratteristice del Processo di analisi.
2. Introduzione e caratteristiche fondamentali delle tecnologie e delle librerie per l'elaborazione dei dati.

— Nel primo capitolo verranno introdotte alcune delle possibili tipologie di analisi di dati, le loro ipotesi di partenza,la natura eterogenea dei loro scopi(statistica, predittiva, confermativa etc), i loro eventuali modelli, con l'obiettivo di far emergere l'idea di quali possano essere gli strumenti, analitici e di visualizzazione dei dati, fondamentali per questa fase. Nella seconda parte del capitolo verrà introdotto un possibile processo di analisi, i concetti base e le problematiche relative ad ogni fase del processo. Nelle fasi che precedono quella di analisi vera e propria, si vedrà come, da un punto di vista del processo, la caratterizzazione iniziale dei dati in dati strutturati, semi-strutturati o non strutturati, dipenda fortemente dai modelli e dalle tecniche applicate nella fase di analisi e non sia solamente una caratteristica intrinseca del dato stesso. Nelle fasi successive, sempre propedeutiche all'analisi, verranno prese in considerazione altre operazioni di preparazione dei dati: il Data Cleaning, ovvero il processo che comporta la pulizia dei dati, consistente nell'individuazione e correzione di errori quali la presenza di ridondanze, dati duplicati, errori presenti nelle stringhe di testo o l'esistenza di rappresentazioni multiple che identificano le stesse entità; l'Analisi della Qualità dei dati, processo che comporterà la definizione di procedure da seguire in presenza di determinate situazioni quali possono essere la presenza di outliers, la presenza di campi privi di dati o con stringhe vuote. Tutte queste fasi, che comportano la ristrutturazione e la pulizia dei dati, renderanno necessario l'utilizzo di strumenti efficienti sia per il reperimento dei dati, sia per la loro efficace manipolazione/trasformazione. Nell'ultima parte del capitolo si sottolineerà come gli strumenti di visualizzazione, oltre ad essere strumento fondamentale in fase di analisi per metodologie come l'analisi esplorativa di dati(EDA), sono l'elemento fondamentale per la fase di presentazione dei risultati.

I capitoli successivi saranno dedicati all'analisi delle tecnologie che permettono di affrontare questi problemi di manipolazione, elaborazione e visualizzazione dei dati. In particolare:

— Nel secondo capitolo verrà introdotto l'ambiente IPython. Progetto nato nel 2001 per creare un ambiente python interattivo con molteplici obiettivi quali l'accesso alla shell di sistema e la navigazione del filesystem per quanto riguarda l'integrazione con l'ambiente sottostante, l'auto-completamento del codice, l'introspezione dinamica degli oggetti, un sistema di macro, un framework di persistenza, l'accesso al debugger e il supporto del profiler per quanto riguarda l'interattività. Si evidenzierà come l'attuale stato di sviluppo, oltre all'utilizzo nella modalità di semplice shell interattiva, ha portato al disaccoppiamento del ciclo dell'interprete in un modello a 2 processi realizzando un'architettura client-server. Questa soluzione, che comporta un processo client per l'interfaccia con l'utente e un processo

di elaborazione chiamato kernel,consente l'esecuzione dei due processi su sistemi separati e la connessione di più client allo stesso server, ma soprattutto ha consentito lo sviluppo di 2 strumenti: la Qt Console - shell grafica basata sulla libreria Qt che consente la possibilità di generare grafici inline, l'editing multi-riga del codice e l'evidenziazione della sintassi e il Notebook IPython. Il Notebook è una applicazione basata sul web che è in grado di soddisfare tutto il processo di elaborazione dati: sviluppo, documentazione ed esecuzione del codice, così come la capacità di organizzare e presentare graficamente i risultati. Infine si darà qualche cenno ad una architettura sofisticata e potente per l'elaborazione parallela e distribuita, fornita da IPython, che supporta molte tecniche di parallelizzazione (SPMD, MPMD, MPI etc) e soprattutto permette lo sviluppo, l'esecuzione ed il controllo in modo interattivo delle applicazioni parallele.

— Nel terzo capitolo si introdurrà la libreria Numpy. Numpy(Numerical Python) è la libreria fondamentale per la progettazione scientifica in python. Lo strumento principale che fornisce è l'oggetto ndarray, struttura dati vettoriale multidimensionale per dati omogenei, ottimizzata dal punto di vista prestazionale. E' una libreria ottimizzata per le operazioni vettoriali, fornisce strumenti di lettura e scrittura di array su disco, moduli per le operazioni di algebra lineare, trasformate di Fourier, generazioni di numeri casuali e strumenti per l'integrazione di codice C, C++, FORTRAN.
— Nel quarto capitolo si introdurrà la libreria grafica matplotlib. matplotlib è una libreria grafica che produce grafici 2D, si integra perfettamente con IPython fornendo un ambiente interattivo per la visualizzazione ed esplorazione dei dati. Inoltre genera grafici interattivi che possono essere ingranditi e navigati con gli strumenti di navigazioni forniti.
— Nel quinto capitolo verrà introdotta la libreria Pandas. Libreria costruita su Numpy che fornisce strutture dati e funzioni progettate per manipolare e analizzare dati strutturati in maniera semplice ed veloce. Lo strumento principale fornito è il DataFrame, struttura dati bidimensionale che combina la efficienza computazionale di calcolo su array di Numpy con la capacità di manipolazione proprie di un foglio di calcolo e di una base dati relazionale.

Capitolo 1

Caratterizzazione di Tecnologie per l'Elaborazione dell'Informazione

In questo capitolo si vuole fornire una panoramica introduttiva dei vari approcci e delle diverse possibili tipologie di analisi ed elaborazione di dati. Senza entrare nei dettagli si farà un elenco di alcune tipologie di analisi, cercando di evidenziare quali siano le caratteristiche particolari, le ipotesi e gli scopi di ognuna di esse. In particolare verranno esaminate brevemente:

— Statistica Descrittiva
— Analisi Esplorativa dei Dati (EDA)
— Analisi predittiva: predizione e classificazione
— Data mining
— Business intelligence

La vastità e la genericità dell'argomento, delle tecniche e delle metodologie, non consentono una specifica ed univoca formalizzazione del processo e una precisa demarcazione delle sue varie fasi, tuttavia nella seconda parte di questo capitolo, dopo aver analizzato i modelli di processo attualmente più utilizzati: CRISP-DM e SEMMA, si elencheranno ed analizzeranno una serie di fasi che, senza l'intento di voler essere ritenute generali, sono state ritenute fondamentali nel contesto considerato:

— Acquisizione ed integrazione dei dati
— Analisi iniziale e elaborazione di dati
— Presentazione dei risultati dell'analisi

La conoscenza delle fasi del processo e delle relative problematiche consentirà, nella parte finale del capitolo, di elencare ed organizzare una serie di requisiti necessari per una tecnologia in grado di elaborazione dei dati.

1.1. Tipologie di Analisi di Dati

Per analisi dei dati si intende il processo che, attraverso l'ispezione, la trasformazione e la modellazione dei dati, ha l'obiettivo di trovare ed evidenziare informazioni che possano essere di supporto nei processi decisionali[16]. In questa definizione si

possono identificare un insieme eterogeneo di tipologie di analisi, che spesso hanno molte caratteristiche comuni. Esaminiamone brevemente alcune:

1.1.1. Statistica Descrittiva - Caratterizzazione dei dati

La statistica descrittiva[19] è la disciplina che ha per oggetto la descrizione quantitativa e visiva delle principali caratteristiche di un insieme di dati. Al contrario della statistica inferenziale(o statistica induttiva), che ha per scopo lo conoscenza dell'universo di cui l'insieme di dati rappresenta un campione, la statistica descrittiva analizza e riassume le caratteristiche del campione. La sintesi può essere ottenuta attraverso due metodi:

— numerico: rappresentata da un insieme di dati statistici
— visivo: formata da grafici che rappresentano caratteristiche del campione

Nel caso di analisi univariata, riguardante quindi la descrizione della distribuzione di una singola variabile, le tre caratteristiche principali che possono essere studiate sono:

— Distribuzione
— Tendenza Centrale
— Dispersione

La Distribuzione è la sintesi della frequenza dei valori assunti dalla variabile, la rappresentazione più naturale di questo parametro è un istogramma.

La Tendenza Centrale è la stima del "centro" di una distribuzione di valori, in pratica fornisce degli indici che danno un'idea dell'ordine di grandezza dei valori della distribuzione. Le tre misure di tendenza centrale, facilmente rappresentabili sull'istogramma delle frequenze dei valori, sono:

— media: aritmetica, geometrica, armonica
— mediana: valore al centro della distribuzione di valori
— moda: valore della distribuzione caratterizzato dalla più alta frequenza

La Dispersione o Indice di Dispersione serve per descrivere la misura con la quale i valori di una distribuzione statistica sono distanti da un valore centrale (solitamente media o mediana). Indicatori di dispersione, rappresentabili in un box-plot, sono:

— intervallo di variazione: valori minimo e massimo della distribuzione
— deviazione standard: è una stima della variabilità di una popolazione di dati
— scarto interquartile(IQR o interquartile range): è la differenza tra il primo e il terzo quartile, cioè l'ampiezza della fascia di valori che contiene la metà centrale dei valori osservati. In una distribuzione ordinabile i quartili sono i valori che ripartiscono la popolazione in quattro parti con lo stesso numero di valori.

Nel caso di analisi multivariata, riguardante la descrizione di più variabili, la sintesi quantitativa può comportare misure di correlazione (di Pearson nel caso di variabili entrambe continue o di Spearman altrimenti) o di covarianza; la sintesi visiva può comportare grafici di dispersione(scatter plots), tabella di contingenza etc.

1.1.2. Analisi Esplorativa dei Dati

L'analisi esplorativa dei dati[20][10] - EDA acronimo di Exploratory Data Analysis - è una metodologia di analisi che ha come obiettivo quello di definire le caratteristiche principali di un insieme di dati, principalmente utilizzando strumenti visivi. La definizione di un modello statistico può essere data, ma l'utilizzo principale di questa tecnica è al fine di determinare l'informazione che può essere fornita dai dati, indipendentemente dai modelli o dalla formulazione di ipotesi. Gli obiettivi primari di questa tipologia di analisi sono:

— suggerire ipotesi riguardo le cause di un fenomeno osservato
— consentire di scoprire un'eventuale struttura nei dati
— evidenziare la presenza di outliers e anomalie
— essere di supporto nella determinazione di tecniche e strumenti statistici
— fornire una base di conoscenza che permetta la formulazione di nuove metodologie e nuovi esperimenti per il reperimento dei dati

La maggior parte di tecniche usate sono di tipo visivo, proprio per il fatto che la natura stessa di questa metodologia è di consentire l'esplorazione dei dati in modo che siano i dati stessi a fornire informazioni utili. Le tecniche utilizzate sono spesso combinazioni di grafici dei dati non elaborati, combinati con grafici di elaborazioni statistiche dei dati stessi.

1.1.3. Analisi predittiva

L'analisi predittiva[22] [11] è l'area dell'analisi di dati che comprende una varietà di tecniche statistiche, le quali, attraverso l'estrazione dell'informazione dai dati, hanno l'obiettivo di fare previsioni su trend e modelli di comportamento. La parte principale di questa analisi consiste nello stabilire relazioni tra variabili indipendenti e variabili dipendenti di eventi passati ed utilizzare queste relazioni per fare previsioni.

1.1.4. Data Mining

Il data mining, altrimenti chiamato knowledge discovery in databases (scoperta di conoscenza nelle basi di dati) è definito[1] come il processo di scoperta di modelli

e relazioni presenti in grandi volumi di dati. Questo campo di analisi strumenti di statistica e di intelligenza artificiale al fine di analizzare data sets composti da grandi collezioni di dati. La fase di analisi vera e propria riguarda 6 tipologie[3] di analisi:

1. Anomaly detection o Outlier detection (identificazione di anomalie): identificazione di oggetti o eventi di un dataset non conformi a modelli attesi.
2. Association rule learning: scoperta di relazioni interessanti tra variabili che caratterizzano i dati
3. Clustering: scoperta di gruppi o strutture che presentano un "similarità", senza l'utilizzo di strutture note presenti nei dati stessi.
4. Calssificazione: generalizzazione di strutture consociute presenti nei dati in modelli applicabili a nuovi dati
5. Regressione: ricerca e progettazione di modelli matematici che rappresentano i dati con la migliore approssimazione
6. Summarization: analisi e sintesi di un dataset in un rappresentazione compatta tramite visualizzazione grafica e generazione di resoconti

1.1.5. Business Intelligence

Per Business Intelligence[18] si intende l'insieme di teorie, metodologie, architetture e tecnologie che trasformano dati in informazione utile e significativa per fini aziendali/commerciali.

1.2. Modelli di Processo per l'Analisi dei Dati

Al fine di avere un'idea più precisa delle varie fasi e problematiche che può comportare il processo di analisi, si andranno ad analizzare ora i 2 modelli di processo più utilizzat per il data mining[6, 7, 8]:

1. CRISP-DM:
2. SEMMA

1.2.1. CRISP-DM

CRISP_DM[13], acronimo di Cross Industry Standard Process for Data Mining, è un modello di processo di data mining che descrive le più comuni metodologie utilizzate nell'analisi dei dati. Come si può evincere dal diagramma 1.1 questo standard suddivide il processo in 6 fasi principali: Business Understanding, Data Understanding, Data Preparation, Modelling, Evaluation e Deployment. Le frecce indicano le più importanti relazioni tra le fasi del processo mentre il cerchio esterno rappresenta

simbolicamente la natura ciclica del processo. Vediamo le fasi, elencando per ogni fase gli aspetti più interessanti da un punto di vista tecnologico:

1. Business Understanding: fase iniziale che consiste nella comprensione degli obiettivi e dei requisiti dell'analisi, da un punto di vista delle ambito di interesse, e nella definizione di un progetto preliminare che consenta di ottenere tali obiettivi.
2. Data Understanding: la fase di "comprensione dei dati" ha inizio con il reperimento iniziale dei dati e prosegue con attività che consentano la familiarizzazione con il dataset, l'identificazione di problemi nella qualità dei dati, l'intuizione sulla struttura dei dati e su eventuali sottoinsiemi che siano di aiuto nella formulazione di ipotesi sulla informazione contenuta nei dati.
   a) Reperimento iniziale dei dati:
   b) Descrizione dei dati: fase che può comportare la generazione di un resoconto che esamina i formati, la quantità dei dati, il numero di righe e campi di ogni tabella etc.
   c) Esplorazione dei dati
   d) Verifica della Qualità dei dati: si verifica la completezza dei dati, la mancanza di attributi o la presenza di campi vuoti
3. Data Preparation: fase che comprende tutte le attività che portano alla determinazione della struttura e alla costruzione dell'insieme finale di dati, che verrà analizzata nelle fasi successive. Compito di questa fase è selezionare tabelle, record, attributi etc., e di trasformare e pulire di dati per i modelli delle fasi di analisi.
   a) Selezione dei dati
   b) Pulizia dei dati
   c) Costruzione dei dati
   d) Integrazione dei dati
4. Modelling: in questa fase verranno selezionate e applicate varie tecniche di modellazione. Poichè ad ogni tipo di problema potranno essere applicate più tecniche ed potendo tali tecniche avere requisiti diversi per ciò che concerne la struttura dei dati, è possibile che si renda necessario un nuovo passaggio alla fase precendete di preparazione dei dati.
5. Evaluation: fase di valutazione del modello e di rivisitazione del processo che ha portato alla costruzione di tale modello, al fine di certificare che raggiunga gli obiettivi stabiliti dell'analisi. Punto fondamentale di questa fase è la determinazione di questioni del problema che possano essere state tracurate nel processo. La decisione riguardo all'uso dei risultati dell'analisi è il putno finale di questa fase.
6. Deployment: fase in cui i risultati del processo dovranno essere organizzati e presentati in modo da poter essere utilizzati da chi ha interessi. In funzione dei

requisiti del problema, questa fase potrà comportare da un semplice resoconto dei risultati fino alla presentazione delle fasi del processo dai dati iniziale ai risultati finali.

a) Produzione del resoconto finale

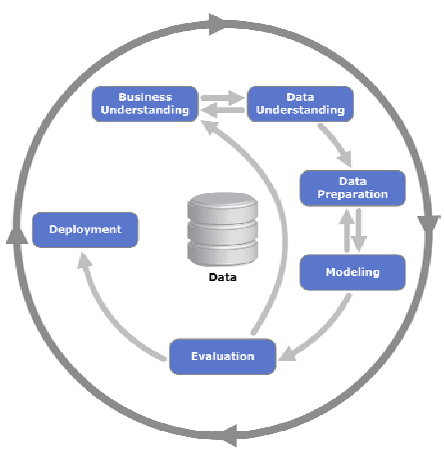


Figura 1.1. Diagramma di Processo CRISP-DM

1.2.2. SEMMA

SEMMA[17], acronimo di Sample, Explore, Modify, Model and Access, è un modello di processo di data mining sviluppato da SAS Institute Inc. Le cinque fasi del processo sono:

1. Sample: fase iniziale del processo che comporta il reperimento e la selezione del dataset per la modellazione. Il dataset, cioè l'insieme di dati, dovrebbe essere abbastanza grande da contenere informazioni e abbastanza piccolo da poter essere usato efficientemente senza che la riduzione comporti alterazione nell'informazione contenuta.
2. Explore: fase che prevede la comprensione dei dati attraverso la loro esplorazione con strumenti di tipo analitico e di tipo visivo. Questa fase porta a mettere in evidenza le relazioni tra le variabili, ma anche alla scoperta di relazioni tra le variabili non ipotizzate inizialmente e alla determinazione della presenza di anormalità o di outliers nell'insieme di dati.
3. Modify: fase che comprende tutti i passi per la selezione e trasformazione dei dati in strutture adatte alla fase di modellazione.
4. Model: fase che si focalizza sull'applicazione di metodi e tecniche sui dati strutturati al fine di creare modelli che consentano di ottenere gli obiettivi del processo di analisi
5. Assess: fase finale che prevede la stima dei risultati e la valutazione dei modelli create per l'analisi.

1.3. Caratteristiche Generiche del Processo di Analisi di Dati

L'Analisi dei Dati è il processo di ispezione, trasformazione e modellazione dei dati che ha l'obiettivo di evidenziare informazioni utili che siano di supporto nei processi decisionali. Come si vedrà, il processo di reperimento ed integrazione dei dati fornisce un accesso unificato ad un insieme di dati strutturati, che implica la loro conformità ad un modello predefinito, ma non la correttezza dei dati stessi. Di conseguenza, prima del processo di analisi vera e propria, è plausibile che siano necessarie una fase di pulizia dei dati ed una di analisi della loro qualità. Si andrà ora ad analizzare quelle fasi del processo che saranno determinanti nella scelta delle tecnologie utilizzate.

1.3.1. Acquisizione ed Integrazioni di dati

La fase propedeutica alla fase di analisi vera e propria è la fase di acquisizione di dati. Si può affermare che l'obiettivo di questa fase sia il reperimento di dati e la loro manipolazione allo scopo di ottenerne una forma strutturata compatibile con i modelli previsti dalla fase di analisi. L'eterogeneità delle sorgenti da cui i dati possono essere ottenuti incide in modo ragguardevole sulla struttura stessa. A tal proposito sia sufficiente considerare come alcune delle plausibili sorgenti possano essere:

— una base di dati SQL, che per la sua stessa natura fornisce dati fortemente strutturati
— un servizio che fornisce un api, che può fornire dati strutturati/semi-strutturati con particolari formati (JSON, XML etc)
— un crawler che recupera un insieme di pagine html secondo prefissate logiche.

I dati acquisiti, da punto di vista del processo di analisi, si possono classificare nelle seguenti 3 categorie :

1. Dati Strutturati - dati dotati di una struttura compatibile con i modelli propri della fase di analisi. Fondamentalmente questo tipo di dati è la conseguenza della definizione e creazione di un modello, in cui sono definiti i tipi dei dati (numerico, alfabetico, data etc), che stabilisce come questi dati saranno memorizzati e manipolati. A questa categoria appartengono dati provenienti da strutture quali una base di dati relazionale o fogli di calcolo.
2. Dati Semi-Strutturati - dati la cui struttura non è conforme ai modelli di dati richiesti dalla fase di analisi, ma che presentano strutture, tag o altri tipi di segnalatori che definiscono gerarchie all'interno dei dati stessi.
3. Dati Non Strutturati - dati che o non hanno un modello predefinito o non sono stati organizzati in maniera predefinita. Questa tipologia è caratteristica dell'informazione ricca di testo che solitamente contiene date, numeri e fatti. Esempi

di questo tipo di dati possono essere libri, giornali, documenti, il testo di una e-mail, una pagina Web etc. Alcune stime[14] portano a pensare che addirittura il 70-90% dell'informazione potenzialmente utilizzabile sia generata in forma non strutturata. La capacità di interpretare l'informazione contenuta in questi dati è lo scopo di molte tecniche quali il data mining, l'analisi dei testi etc.

Il processo di integrazione dei dati, sebbene non nettamente distinto dalla fase di acquisizione stessa, è definito come il processo che combina dati eterogenei provenienti da sorgenti differenti, fornendo una visuale unificata dei dati.

1.3.2. Pulizia dei Dati - Data Cleaning

Gli errori che rendono necessaria questa fase sono essenzialmente dovuti a problemi nelle modalità con cui i dati sono stati inseriti e memorizzati. Il processo di Data Cleaning ha l'obiettivo di prevenire ed eliminare tali errori. Alcuni esempi di problemi che devono essere risolti in tale fase possono essere:

— Record Matching: situazione in cui sia necessario fare un merge di dati che abbiano rappresentazioni diverse della stessa entità. Un esempio può essere fare il join di 2 tabelle in cui una nazione sia memorizzata col suo nome nella prima tabella e con l'abbreviazione del nome (o con un codice) nella seconda. Lo scopo in questa situazione è unificare le rappresentazioni in un'unica forma.
— De-duplicazione: processo di eliminazione di dati duplicati o ridondanti.
— Eliminazione di errori nel testo: utilizzo di correttori ortografici per minimizzare il numero di parole non corrette inserite.

1.3.3. Analisi della Qualità dei Dati

Altra fase è quella che comporta l'analisi della qualità dei dati. Questa comprende un insieme di processi fortemente dipendenti dal tipo di dati e di modelli che mira a trovare una soluzione a problemi tipo:

— dati mancanti: si stabilisce il comportamento da applicare quando in un campo di una struttura dati non è presente il dato stesso.
— outliers: comportamento in presenza di particolari dati che non sono comformi a modelli o valori attesi (e.g. abbiano una forte deviazione statistica)

1.4. Presentazione dei Risultati dell'Analisi

Fase finale in cui si riassumo numericamente e graficamente i risultati del processo.

La visualizzazione dei dati ha per oggetto la comunicazione della informazione in maniera chiara ed effettiva attraverso uno strumento grafico. Da un punto di vista del processo di analisi è possibile evidenziarne 2 aspetti:

— analisi grafica: come è già stato visto, consiste nello studio dei dati per mezzo di strumenti grafici. Lo scopo è la scoperta di "nuova" informazione riguardante l'insieme di dati; solitamente durante questo processo non è chiaro quale sia la domanda da porsi al riguardo dell'insieme di dati, ma la scoperta di questa è parte stessa del processo.
— grafica di presentazione: la grafica di presentazione riguarda invece la comunicazione di informazioni e risultati di un procedimento di analisi già avvenuto.

La distinzione tra questa due attività è doverosa in quanto esse richiedono tecniche differenti e forniscono diversi risultati. Durante la fase di analisi grafica la caratteristiche principali sono la facilità di utilizzo dello strumento, la possibilità di ridisegnare un grafico interattivamente, modificarne l'aspetto, ingrandirlo, applicare trasformazioni e cambiarne gli stili. In questo contesto ogni forma di decorazione(etichette, archi, simboli etc) non fornisce alcun vantaggio in quanto chi esegue l'analisi è a conoscenza del significato di ciò che il grafico rappresenta. Una situazione opposta si verifica nella fase della grafica di presentazione, in questa non è più necessario modificare e manipolare il grafico, ma, essendo i risultati dell'analisi già determinati, l'obiettivo sarà trovare un modo per comunicare questi ad altre persone. Di conseguenza assumeranno importanza strumenti che consentano di definire l'informazione contenuta nel grafico. Alcune caratteristiche che dovrebbero avere i grafici di presentazione sono:

— avere testo auto esplicativo, con l'informazione essenziale inserita direttamente nel grafico
— definire in modo chiaro con etichette, ciò che viene riportato sugli assi con relativa unità di misura
— avere un formato di output appropriato. Utilizzare per la stampa formati scalabili(PostScript, PDF,SVG) e non formati bitmap(GIF,JPG,PNG).

1.5. Valutazione di Tecnologie per l'Analisi dei Dati

Avendo un'idea delle fasi principali del processo di analisi, si andranno ora ad elencare ed organizzare le specifiche necessarie ad una tecnologia per affrontare un processo di analisi.

In letteratura sono presenti studi[2] che definiscono metodologie per la valutazione e la selezione di tecnologie di analisi, in questo paragrafo si cercherà di astrarre ed

organizzare un'insieme di caratteristiche, di una tecnologia di analisi, che sono ritenute fondamentali. Queste caratteristiche saranno poi gli elementi che forniranno i punti di vista fondamentali da cui si analizzeranno i software presentati nei capitoli successivi.

1.5.1. Architettura della Tecnologia

— Architettura della Tecnologia: la natura della tecnologia è stand-alone o client-server? è possibile scegliere tra queste 2 architetture?

1.5.2. Metodi di Accessibilità

— Accesso a tipologie eterogenee di dati: tipologie di sorgenti di dati a cui è possibile interfacciarsi
— Interoperabilità: capacità della tecnologia di interfacciarsi con altri strumenti

1.5.3. Funzionalità

— Varietà di algoritmi: l'insieme degli metodi forniti dallo strumento
— Flessibilità nel tipo di dati: determina se l'implementazione consente la gestione di molteplici tipi di dati
— Moficabilità degli algoritmi: consente all'utente di modificare e adattare algoritmi
— Generazioni di Numeri Casuali: lo strumento fornisce metodi per la generazione di numeri casuali
— Reporting: modalità con cui si possono fare resoconti dei risultati dell'analisi
— Esportazione dei modelli: è consentito esportare una metodologia sviluppata in modo che possa essere riutilizzata

1.5.4. Criteri di Usabilità

— Interfaccia utente: grado di difficoltà e navigazione dell'interfaccia utente. L'interfaccia è in grado di presentare risltati in maniera significativa
— Tipi di utenti: a che tipologia di utenti è rivolta la tecnologia
— Visualizzazione di dati: in che modo vengono presentati i dati
— Varietà dei domini di applicazione: lo strumento può essere utilizzato in tipologie differenti di analisi

1.5.5. Strumenti per la Manipolazione dei Dati

— Data Cleansing: capacità dello strumento di permettere all'utente di modificare dati alterati

— Sostituzione di valori:
— Filtraggio dei dati
— Derivazione Attributi
— Gestione dei dati Vuoti
— Manipolazione dei Metadati

Capitolo 2

IPython - Un ambiente di sviluppo interattivo

In questo capitolo verrà introdotto IPython[12], progetto nato nel 2001 con lo scopo di creare un ambiente python completo per l'elaborazione informatica interattiva ed esplorativa. Nel decennio successivo è evoluto in un ambiente che comprende una console GUI, che consente la rappresentazione inline di grafici, un formato "notebook" interattivo basato su una applicazione web ed un motore di elaborazione parallela.

IPython è composto da 3 componenti principali:

— una Shell di Python avanzata e interattiva
— una architettura basata su 2 processi disaccoppiati, che consente di avere un processo di elaborazione "computation kernel" e più processi client che vi si collegano
— un'architettura per l'elaborazione parallela interattiva

Nel capitolo precedente si è cercato di determinare quelle che possono essere le caratteristiche generiche comuni a un qualsiasi processo di analisi. In questo contesto siamo interessati ad alcuni di tali elementi e cercheremo di vedere come IPython e le tecnologie da esso derivate ci possano fornire strumenti per affrontare tali fasi. In particolare riguardo:

— l'interattività dello strumento che, in modo particolare nelle fasi iniziali del processo, deve consentire una rapida manipolazione dei dati ed una facile e veloce generazione di grafici per fini esplorativi
— la fase di presentazione, fase in cui devono essere generate relazioni contenenti un insieme di risultati numerici e grafici e corredati da una serie di informazioni esplicative
— la capacità di esportare il modello di processo in modo che possa essere replicato

IPython è per sua natura un ambiente interattivo, e ciò agevola enormemente il processo di analisi; tuttavia, per ciò che concerne la nostra sfera di interesse, sono 2 strumenti basati su di esso ad essere rilevanti, cioè la console grafica e il Notebook IPython. Questi strumenti, come analizzeremo successivamente, oltre ad avere le caratteristiche di interattività di IPython, hanno la caratteristica di consentire la generazione di grafici in linea. La capacità di manipolare i dati e di generare e modifi-

care i grafici inline interattivamente rappresenteranno uno strumento fondamentale nelle fasi esplorative dei dati.

La fase di presentazione, solitamente intesa come fase finale del processo ma che, come abbiamo visto, può riguardare anche fasi intermedie in cui sia necessario presentare risultati, ha spesso comportato la necessità di esportare dati e grafici in strumenti specificamente intesi per la presentazione. L'IPython Notebook è una applicazione web che consente di integrare codice python, eseguibile direttamente in un browser in particolari elementi chiamati "celle di codice", con grafici e documentazione. Questa connotazione grafica dello strumento consentirà di sviluppare resoconti "dinamici" che, oltre a documentare la logica del processo, i modelli utilizzati, i risultati numerici e grafici, consentirà la riesecuzione del codice direttamente in un browser. L'IPython Notebook salva i progetti in documenti chiamati Notebook, che sono file di testo in formato JSON, che sono facilmente caricabili nell'applicazione web e rieseguibili, consentendo quindi l'esportazione dei modelli e la loro facile replicabilità.

2.1. Shell Avanzata di Python

Verranno ora elencate le caratteristiche essenziali della shell di IPython

— Introspezione Dinamica degli Oggetti: Per introspezione dinamica si intende la possibilità di accedere ad una serie di informazioni riguardanti un oggetto direttamente dalla shell utilizzando il comando ? subito dopo il nome dell'oggetto. Le informazioni che si possono ottenere sono le docstring, la definizione di funzioni, dettagli sui costruttori di una classe etc.

— Auto Completamento con <Tab>: Uno dei comandi di maggiore utilità è certamente l'auto completamento del codice con il tasto <Tab>. Questo consente il completamento per keyword, moduli, metodi e variabili del namespace locale e anche per file della cartella di lavoro.

— Prompt Numerati: la shell presenta prompt numerati di input e output :In[num] e Out[num]. In questo lavoro il prompt numerato verrà rappresentato come:

```
In []:    comando di input

Out []:   output dell'interprete
```

— Gestione Storico: viene conservato lo storico di tutti i comandi eseguiti e di tutti i risultati prodotti. La navigazione dello storico può essere fatta semplicemente con i tasti freccia su e freccia giù, oppure utilizzando il fatto che gli input e gli output sono memorizzati in variabili chiamate "In" e "Out" il cui contenuto è accessibile con il relativo numero del prompt e considerando che gli ultimi 3 oggetti dello storico di output sono anche contenuti in variabili nominate "_","__"e"___". Infine il comando magico `%history` stampa lo storico di input della sessione.

— Comandi Magici: insieme di comandi col prefisso `%`, estendili dall'utente, che consentono:
  — di avere controllo su IPython e sulle directory
  — di avere informazioni sul namespace
  — di avere alias a comandi della shell di sistema
— Accesso Completo alla Shell di Sistema: le linee di comando che iniziano per ! vengono passate direttamente alla shell di sistema, il cui output può essere memorizzato in variabili di Python tramite il comando `var = !shell_cmd`, dove shell_cmd è il comando passato alla shell di sistema.
— Navigazione Filesystem
— Un Framework di Persistenza: consente, tramite il comando magico `%store`, di salvare le variabili python passate come argomento e di poterle recuperare con il comando `%store -r` in un'altra sessione.
— Sistema di Gestione di Macro: fornisce la possibilità di rieseguire linee di input precedenti con un singolo nome attraverso il comando magico `%macro`. Per definire una macro sarà necessario passare come argomento un nome e le righe dello storico(sia intervalli [riga iniziale-riga finale], che singole righe) che comporranno la macro. (e.g. %macro m1 23-35 17 3). Inoltre le macro potranno essere memorizzate persistentemente col comando %store già visto in precedenza.
— Logging di Sessione: possibilità di generare log di tutti gli input di sessione sia inizializzando IPython con "–logfile=nome_file_di_log.py" come argomento, sia attivando il logging durante una sessione di lavoro con il comando magico `%logstart`. I file di log potranno essere ricaricati ed eseguiti in modo da riprodurre una precedente sessione di lavoro.
— Accesso al Debugger: possibilità di impostare IPython in modo che invochi una versione avanzata(con completamento con <Tab> etc) del debugger di python ogni volta che viene tirata una eccezione.
— Accesso al Profiler: viene fornita la possibilità di eseguire codice sotto il controllo del profiler di python con il comando magico `%prun`.
— Funzioni di Temporizzazione: vengono fornite delle semplice funzioni di temporizzazione, tramite il comando %timeit, per misurare il tempo di esecuzione del codice. In determinate situazioni, quelle in cui il tempo di elaborazione è piccolissimo, vengono fatte ripetizioni dell'esecuzione e viene fornita una media dei tempi.

2.2. Modello 2 Processi Disaccoppiati

Il ciclo REPL(Read-Evaluate-Print Loop) proprio della shell normale di Python prevede le 3 fasi seguenti:

1. Read: funzione che accetta un'espressione dall'utente, ne fa il parsing in una determinata struttura dati in memoria
2. Evaluate: funzione che prende questa struttura dati e la valuta
3. Print: funzione che prende il risultato generato dalla funzione di Evaluate e lo stampa all'utente

In IPython oltre a questo tradizionale approccio su singolo processo(che è la modalità di esecuzione quando si esegue IPython senza sotto-comandi), è stato riprogettato il ciclo REPL portando al disaccoppiamento della fase di Evaluate in un suo proprio processo, trasformando il ciclo stesso in una architettura client-server, in cui il processo di valutazione, chiamato *kernel*, riceve da processi client istruzioni da eseguire e comunica loro i risultati dell'elaborazione. Questo disaccoppiamento non solo consente la connessione di più client allo stesso kernel ma permette a client e kernel di essere eseguiti su macchine separate. Conseguenza di questo modello, è stato lo sviluppo di client sempre più sofisticati come l'IPython Notebook, di cui parleremo più estesamente nel prossimo paragrafo, e della console Qt, di cui riportiamo uno screenshot in Figura 2.1, che è una console IPython eseguita in una GUI PyQt la quale consente funzionalità come la possibilità di generare grafici inline, di editare codice multi-linea con evidenziazione delle sintassi, di avere finestre di suggerimento etc.

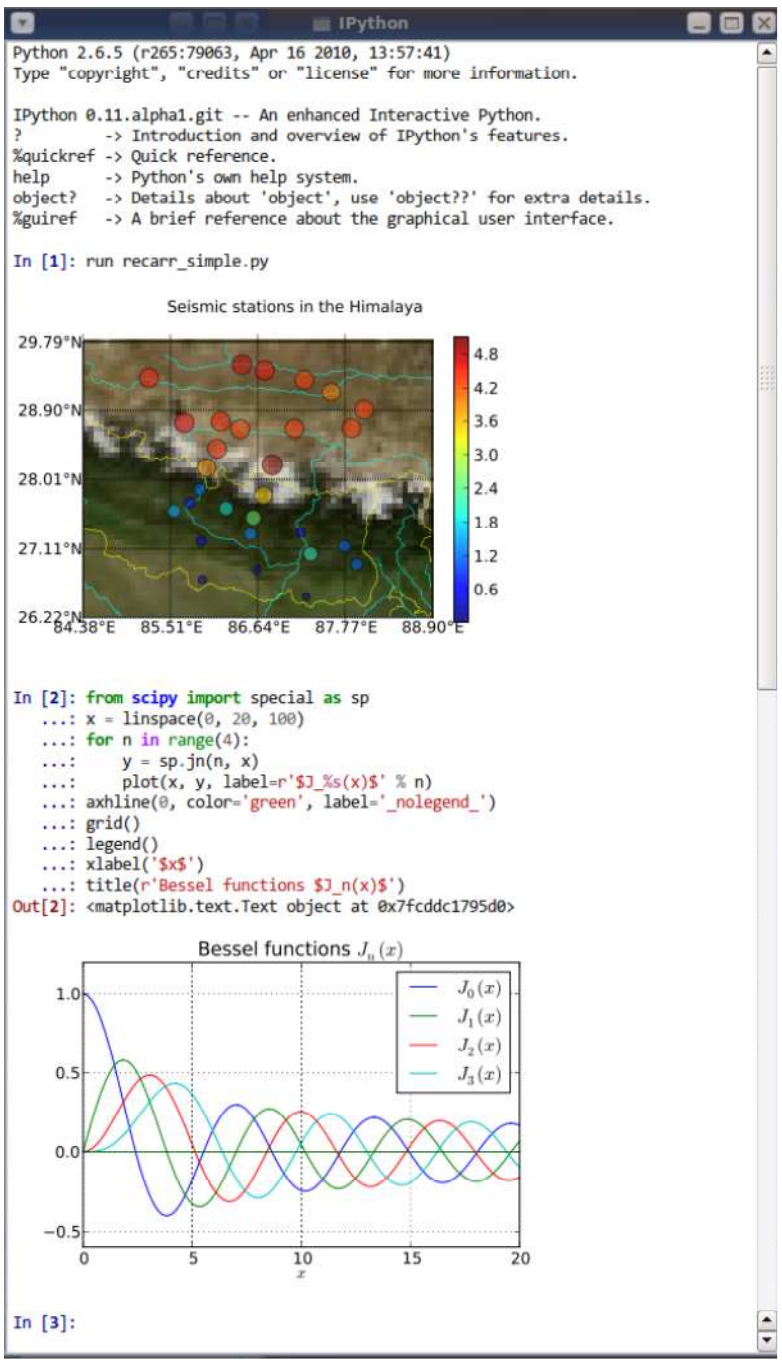


Figura 2.1. Screenshot della console Qt per IPython

2.2.1. IPython Notebook

IPython Notebook estende l'approccio alla elaborazione di dati interattiva basata su console grafica, fornendo una applicazione basata sul web in grado di soddisfare tutto il processo di elaborazione dati: sviluppo, documentazione ed esecuzione del codice, così come la capacità di organizzare e presentare graficamente i risultati. IPython Notebook combina 2 componenti:

1. Applicazione web: uno strumento basato su browser che consente la scrittura interattiva di documenti che combinano testi, formule matematiche, elaborazione e la rappresentazione dei risultati.
2. Notebook: documenti che rappresentano tutto il contenuto visibile nell'applicazione web, compresi input e output delle elaborazioni, testi, formule matematiche, immagini e grafici.

2.2.1.1. Caratteristiche dell'Applicazione Web di IPython Notebook

La caratteristica principale dell'applicazione Web di IPython Notebook,è di consentire:

— la scrittura di codice python direttamente nel browser con evidenziazione della sintassi e indentazione, integrando il processo con tutte le funzionalità avanzate fornite da IPython: completamento con <Tab>, introspezione etc.
— l'esecuzione del codice stesso nella finestra del browser e la presentazione dinamica dei risultati, attaccandoli al codice che li ha generati
— la rappresentazione dei risultati dell'elaborazione utilizzando strumenti come: *HTML*, *LaTeX*, *PNG*, *SVG* etc.
— la capacità di integrare testo utilizzando il linguaggio di markup *Markdown*

In figura 2.2 si vede uno screenshot di un IPython Notebook in cui si possono vedere un grafico generato inline e rappresentato nella parte di output di una cella di codice, una cella di markdown in cui è stato inserito del semplice codice LateX che viene renderizzato dinamicamente all'esecuzione della cella e la parte di Input di una cella di codice.

2.2.1.2. Caratteristiche dei Documenti *Notebook*

I documenti Notebook contengono tutti gli input e tutti gli output della sessione interattiva, che possono essere inframezzati da testo non eseguibile che accompagna il codice, formule matematiche, immagini etc. Questi documenti sono salvati in formato JSON con l'estensione .ipynb e possono essere esportati in vari formati 2.1 quali *HTML*, *LaTeX*, *PDF* e file di presentazione con il comando `ipython nbconvert --to FORMAT file_notebook.ipynb` dalla shell di sistema.

Comando	Note
`--to html`	Esporta in html
`--to latex`	Esporta in formato .tex Aggiungendo `--PDF` esporta in .pdf
`--to slides`	Esporta in uno slideshow Reveal.js servibile su server HTTP
`--to markdown`	Esporta in Markdown
`--to rst`	Esporta in un reStructuredText
`--to python`	Esporta il notebook in uno script di Python eseguibile

Tabella 2.1. Formati di esportazione supportati

2.2.1.3. Struttura dei documenti *Notebook*

Fondamentalmente un Notebook consiste in una sequenza di *celle*, dove per cella si intende un campo di input testuale multi-riga, il cui contenuto può essere mandato

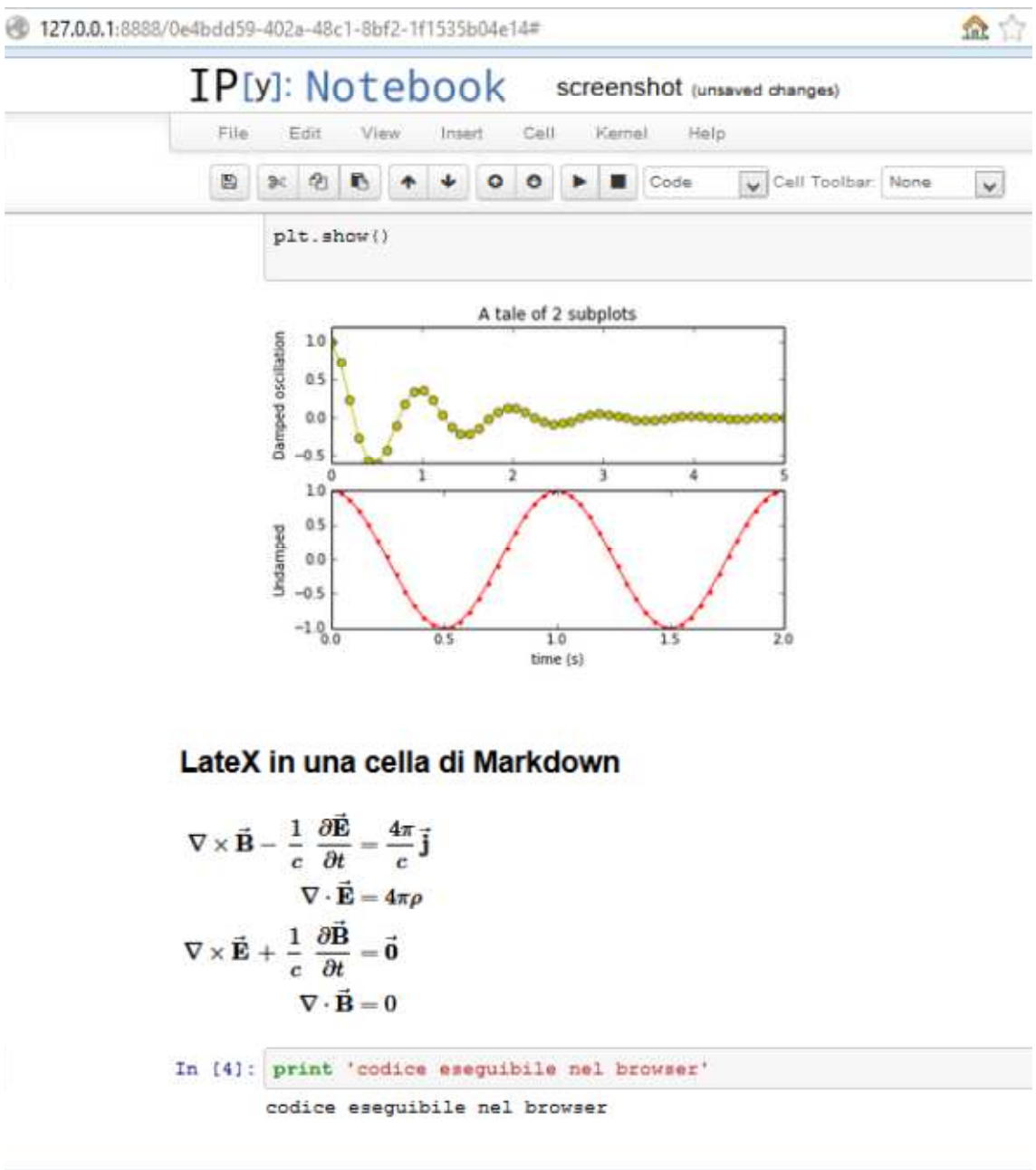


Figura 2.2. Screenshot IPython Notebook

in esecuzione. Esistono 4 tipologie di celle ed è questa tipologia che determina il tipo di esecuzione della cella. Vediamole:

1. Celle di Codice (Code cells): queste permettono la scrittura e la modifica del codice, con evidenziazione della sintassi e completamento con i <tab>. Quando la cella viene mandata in esecuzione, il codice in essa contenuto viene mandato al processo *kernel* associato con il Notebook. Il risultato di tale elaborazione viene rimandato alla cella che lo mostra nella sezione di output della cella stessa. Questo output non è limitato a semplice testo, ma possono essere file grafici (come vedremo analizzando la libreria grafica matplotlib) e tabelle html.
2. Celle di Markdown: forniscono la possibilità di alternare alle celle di codice, celle con testo strutturate con il linguaggio di markup Markdown. Quando la cella è eseguita il codice Markdown viene trasformato nel relativo formato. In queste celle è possibile utilizzare notazione LaTeX standard che, durante l'esecuzione verranno trasformate in HTML tramite la libreria Mathjax.

3. Raw cells: sono celle che consentono di scrivere output direttamente, che quindi non verrà mandato in esecuzione dal Notebook. Nel caso il notebook venga esportato, queste celle consentiranno la scrittura di LaTeX, che sarà interpretato e riprodotto da LaTeX durante l'esportazione.
4. Heading Cells: celle che permettono di definire una struttura concettuale al documento con 6 misure di intestazione.

2.2.1.4. Flusso di Lavoro

Il server di notebook si inizializza con il comando `ipython notebook` sulla shell di sistema. Questa operazione stamperà alcune informazioni sulla console di IPython ed aprirà in un browser l'url della applicazione web (solitamente http://127.0.0.1:8888). La pagina iniziale dell'applicazione, chiamata *dashboard*, mostra i notebook presenti nella cartella e consente di creare nuovi notebook. Un notebook "aperto" ha esattamente una sessione interattiva collegata ad un *kernel* di IPython, che sarà il processo a cui verrà mandato il codice e che comunicherà i risultati. Nel caso la finestra del browser venga chiusa, il kernel rimarrà attivo e, riaprendo il notebook dalla pagina dashboard, ricollegherà l'applicazione allo stesso kernel.

Il flusso di lavoro in un notebook è simile ad una sessione standard di IPython, con la differenza che le celle possono essere rimodificate in-loco e fatte rieseguire, finché non si ottengano i risultati desiderati.

2.2.1.5. Plotting

Una delle caratteristiche principali e più interessanti dei notebook è quella di poter generare e mostrare grafici nella sezione di output delle celle di codice. L'integrazione con la libreria grafica matplotlib(che sarà argomento dei prossimi capitoli) si ottiene con il comando magico %matplotlib. Questo comando invocato con l'opzione `inline`, utilizza un backend che consente l'integrazione dei grafici direttamente sotto la cella che li ha generati e la memorizzazione nel documento notebook.

2.3. Architettura interattiva per l'Elaborazione Parallela - Cenni

IPython fornisce una architettura per utilizzare, in modo semplice e interattivo, hardware parallelo come CPU mutlicore, cluster e super-computer. Le principali caratteristiche del sistema sono:

— la possibilità di parallelizzare codice Python direttamente da una sessione IPython interattiva.

— un'architettura che permette l'elaborazione interattiva consentendo l'utilizzo di differenti stili di parallelismo.
— interfacce bloccanti e totalmente asincrone.
— API di alto livello che consentono la parallelizzazione con poche righe di codice.
— la possibilità di scrivere codice parallelo che possa essere eseguito su qualsiasi tipo di architettura senza dover essere adattato.
— Completa integrazione con librerie MPI.

2.3.1. Architettura di IPython.parallel

Ad un alto livello i 3 componenti principali di IPython parallel sono:

1. Gli *Engine*: sono i processi remoti o distribuiti dove il codice viene effettivamente eseguito.
2. *Client*: l'interfaccia che consente di eseguire il codice sugli Engine.
3. *Controller*: l'insieme di processi che coordinano Engine e Client. Come si vedrà di seguito il Controller è composto da *Hub* e *Schedulers*

Questi componenti permettono di implementare un modello che consente di creare tutti gli Engine che si desiderano, su uno o più nodi computazionali, e di controllarli dai Client attraverso un oggetto Controller centrale, come mostrato in figura.

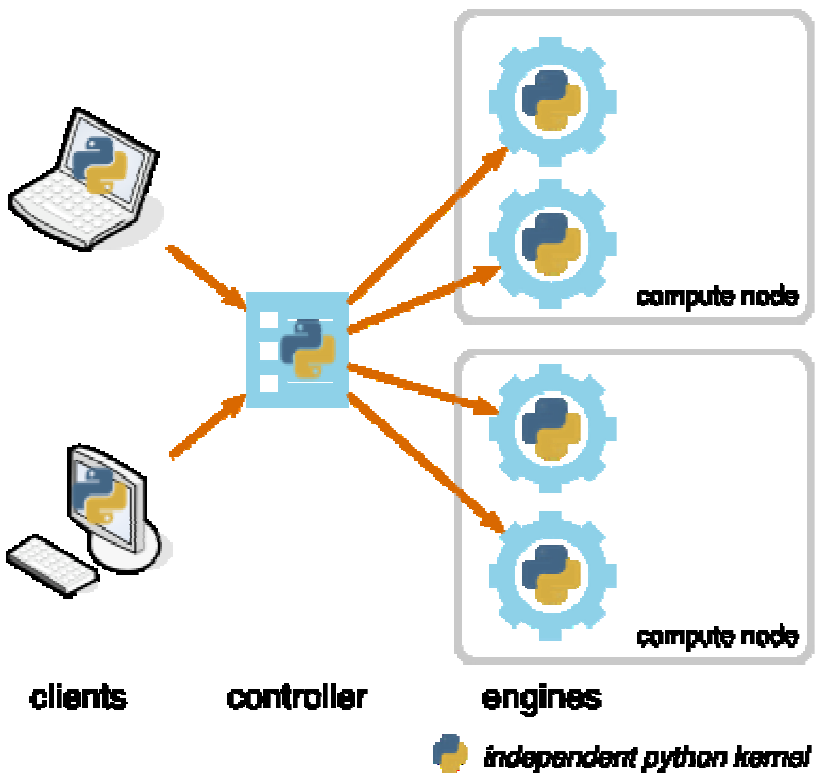


Figura 2.3. Modello Architettura Parallela IPython

L'architettura di IPython.parallel, schematizzata in figura 2.4, consiste di 4 componenti:

1. IPython Engine
2. IPython Hub
3. IPython Schedulers
4. Client

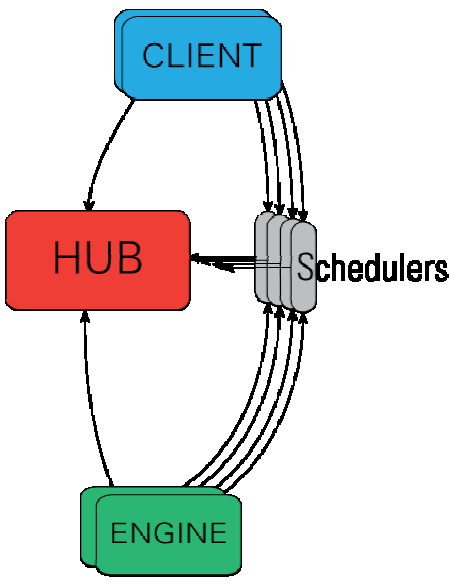


Figura 2.4. Architettura dei Componenti IPython.parallel

2.3.1.1. IPython Engine

L'IPython engine è un'istanza di Python che riceve comandi da una connessione di rete e che è in grado di gestire la ricezione e la trasmissione di oggetti di Python. Caratteristica importante dell'IPython engine è che è "bloccato" quando del codice vi è eseguito. L'IPython controller è l'elemento che, costituendo un *layer* tra client ed engine, consente all'architettura di presentare un'API asincrona.

2.3.1.2. IPython Controller

L'*IPython controller* è un'insieme di processi che fornisce un'interfaccia la quale consente di utilizzare l'insieme di *engine* che fanno parte dell'architettura. Da un punto di vista generale, il controller è un'insieme di processi a cui gli engine e i *client* possono collegarsi. Il controller è composto da un *Hub* e una collezione di *Scheduler*. Gli scheduler sono processi che possono essere eseguiti sulla stessa macchina dell'Hub, ma anche su macchine remote. Il controller fornisce un singolo punto di accesso agli utenti che vogliano utilizzare gli engine connessi al controller stesso. Le due principali modalità di interazioni con gli engine sono:

— l'interfaccia Direct: consente l'esplicito utilizzo di un engine. L'idea di base è che le capacità di ogni engine siano direttamente ed esplicitamente esposte all'utente. In questo contesto ad ogni engine viene assegnato un id ed è questo identificatore che consente di assegnargli lavoro direttamente.

— l'interfaccia LoadBalanced: lo scheduler assegna lavoro agli engine. Questa interfaccia presenta gli engine come un sistema di "processi lavoratori", fault tolerant (tollerante ai guasti) e con bilanciamento dinamico dei carichi di lavoro.

2.3.1.3. Hub

Questo processo è l'elemento centrale di un cluster IPython; esso tiene traccia delle connessioni degli engine, degli scheduler, dei client e di tutte le richieste e i risultati dell'elaborazione. La funzione principale è di facilitare le interrogazioni sulla stato del cluster e di minimizzare l'informazione necessaria per stabilire connessioni tra client ed engine.

2.3.1.4. Scheduler

Tutte le elaborazioni eseguite da un engine sono controllate da uno scheduler. Questo, mentre il codice è eseguito da un engine, fornisce l'interfaccia totalmente asincrona all'insieme di engine.

Capitolo 3

NumPy

In questo capitolo si introdurranno le caratteristiche principali di *NumPy*, libreria fondamentale per quanto riguarda l'elaborazione scientifica in Python. Un comprensione di NumPy è necessaria in questo contesto, essendo la libreria su cui si basano i moduli presentati nei capitoli successivi ed inoltre poiché consente l'elaborazione di tipologie di dati. numerici. Alcuni degli strumenti che fornisce sono:

— *ndarray*: struttura vettoriale multidimensionale per dati omogenei.
— funzioni vettorizzate: una serie di routine ottimizzate per operazioni sull'array senza la necessità di dover scrivere cicli, che comprendono funzioni matematiche, logiche, di ordinamento
— strumenti di I/O che consentono la lettura/scrittura di dati vettoriali da e su disco
— funzioni di algebra lineare, di statistica base, di generazione di numeri casuali e di calcolo di trasformate di Fourier
— strumenti che consentono l'integrazione con codice C, C++, FORTRAN

Oltre alle caratteristiche fondamentali del vettore ndarray e alle modalità di accesso ai suoi elementi, si introdurranno due caratteristiche di NumPy:

— vettorizzazione: questa è la possibilità di applicare operazioni su vettori, elemento per elemento, senza la necessità di esplicitare cicli sulla struttura dati, quindi senza dover esplicitamente avere accesso agli elementi del vettore
— broadcasting: insieme di regole che consentono l'applicazione di funzioni a vettori di dimensioni diverse(e.g. prodotti tra vettori e scalari)

Infine, si sottolineeranno alcuni aspetti della libreria particolarmente interessanti dal punto di vista dell'elaborazione dati:

— logica condizionale
— funzioni matematiche e statistiche
— metodi per array booleani
— logica su insiemi

Nell'analisi di processo fatta nel primo capitolo, abbiamo visto come l'elaborazione numerica dei dati sia la fase principale del processo di analisi vero e proprio, e che

può essere elemento importante anche nelle fasi di analisi iniziali dei dati in cui la capacità di generare dati statistici è in particolar modo fondamentale. NumPy fornisce una struttura e funzioni ottimizzate per dati omogenei; questo può rappresentare una limitazione per una analisi reale in cui difficilmente ci si ritrova in questa situazione, tuttavia, nei capitoli successivi analizzeremo delle librerie costruite su NumPy che forniscono strutture di dati di tipo eterogeneo alle quali sarà possibile applicare la maggior parte di strumenti presenti nei moduli di NumPy.

3.1. La Struttura Dati Multidimensionale ndarray

Lo strumento fondamentale di NumPy è l'oggetto ndarray, struttura vettoriale multidimensionale per dati omogenei, le cui caratteristiche principali sono:

— dimensione (solitamente) prefissata, determinata in fase di creazione. Una modifica della dimensione di un ndarray richiederà la creazione di un nuovo array e la cancellazione del vettore originale
— tutti gli elementi del ndarray devono essere dello stesso tipo di dato. Questo, nella maggior parte dei casi, comporterà che ogni elemento occupi la stessa quantità di memoria; un caso in cui ciò può non avvenire è il caso in cui gli elementi dell'ndarray siano essi stessi degli oggetti.

Il numero delle dimensioni e degli elementi di un ndarray è definito dalla sua proprietà "shape", che è una tupla di N numeri positivi che specificano ogni dimensione; il tipo di dato degli elementi del vettore è specificato in un oggetto "data-type" associato all'ndarray.

3.1.1. La Creazione di Oggetti ndarray

I meccanismi generali per creare gli array sono 5:

1. Conversione da una qualche struttura di Python (e.g. liste, tuple)
2. Utilizzare strumenti intrinseci di NumPy per la creazione di array (e.g. arange, ones, zeros, etc.)
3. Leggere dati da file, sia in formato standard che in formati particolari
4. Creare array da byte attraverso l'uso di stringhe o buffer
5. Utilizzare funzioni di librerie speciali(e.g. random)

```
# Cinque modi per creare un vettore
import numpy as np
# Da una lista di Python
vec1 = np.array( [ 0., 1., 2., 3., 4. ] )
vec1_ = np.array([[1,2.0],[0,0],(1+1j,3.)])
# arange( start inclusive, stop exclusive, step size )
```

```
np.arange(2, 10, dtype=np.float)
# linspace( start inclusive, stop inclusive, number of elements )
vec3 = np.linspace(1., 4., 6)
# zeros(n) returns a vector filled with n zeros
vec4 = np.zeros( 5 )
for i in range( 5 ):
    vec4[i] = i
# leggere da un file di testo, un numero per riga
vec5 = np.loadtxt( "data" )
```

3.1.2. I Data Type degli ndarray

Gli oggetti *dtype* contengono l'informazione necessaria affinché l'ndarray possa interpretare blocchi di memoria come un tipo particolare di dati:

```
>>>arr1 = np.array([10, 20, 30], dtype=np.float64)
>>>arr1.dtype
dtype('float64')
>>>arr2 = np.array([10, 20, 30], dtype=np.int8)
>>>arr2.dtype
dtype('int8')
```

Nella maggior parte dei casi i tipi di dati vengono mappati direttamente sulla rappresentazione macchina sottostante, questo permette la connessione a codice scritto in linguaggi di più basso livello come C e Fortran. Tutti i tipi sono denominati nello stesso modo: il nome del tipo concatenato al numero di bit da cui è composto l'elemento; in tabella sono riportati i tipi di dati supportati da NumPy.

Tipo	Codice del Tipo	Descrizione
int8, uint8	i1, u1	Interi, signed e unsigned a 8-bit
int16, uint16	i2, u2	Interi, signed e unsigned a 16-bit
int32, uint32	i4, u4	Interi, signed e unsigned a 32-bit
int64, uint64	i8, u8	Interi, signed e unsigned a 64-bit
float16	f2	Virgola mobile a 16-bit
float32	f4	Virgola mobile a 32-bit
float64	f8	Virgola mobile a 32-bit. Compatibile con float di Python
complex64	c8	2 float a 32-bit
complex128	c16	2 float a 64-bit
bool		tipo booleano
oggetto	O	oggetto di Python
string_	S	Tipo stringa a lunghezza fissa(1 byte per carattere)

Tabella 3.1. Tipi di Dati di Numpy

Il metodo `astype` consente di fare cast espliciti:

```
>>>arr = np.arange(10)
>>>arr.dtype
```

```
dtype('int32')
>>>arr1 = arr.astype(np.float64)
>>>arr1.dtype
dtype('float64')
```

Un caso interessante di cast è quando si ha un array di stringhe che rappresentano numeri e li si voglia trasformare in valori numeri. Vediamone un esempio:

```
>>>str_arr = np.array(['3.456', '365'])
>>>str_arr
array(['3.456', '365'],        dtype='|S5')
>>>arr = str_arr.astype(np.float64)
>>>arr
array([   3.456,  365.   ])
>>>arr.dtype
dtype('float64')
```

3.1.3. Indexing - Modalità di Accesso agli Elementi del Vettore

Con il termine Indexing si intendono le operazioni che consentono l'accesso agli elementi dell'array tramite l'udo delle parentesi quadre []. Di seguito vedremo le modalità fornite da NumPy:

3.1.3.1. Indexing Base

L'indexing base per un'array 1-dimensionale funziona esattamente come per tutte le sequenze presenti nello standard Python. Le sequenze hanno indici che vanno dallo 0 a (lunghezza della sequenza - 1) e si può accedere agli elementi sia con indici positivi che con indici negativi. Nel caso di ndarray, NumPy supporta l'indexing multidimensionale per gli array n-dimensionali. Ciò significa che è possibile passare tra parentesi un lista indici separati da virgole per selezionare gli elementi. Bisogna sottolineare che se l'insieme di indici è minore delle dimensioni dell'ndarray, si otterrà un array e non un semplice elemento. Vediamo un breve esempio:

```
>>>v = np.arange(20).reshape(4,5)
>>>v
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
# indexing multidimensionale dell'ndarray 2-dimensionale
# accesso seconda riga, ultimo elemento
>>>v[1,-1]
9
# indexing con meno indici della dimensione dell'ndarray
>>>v[2]
array([10, 11, 12, 13, 14])
```

Dall'ultima riga si può notare come vi sia equivalenza tra le 2 forme `v[i][k]` e `v[i,k]`. Poiché l'indexing base ritorna una copia dell'array, la prima soluzione sarà molto più inefficiente perché comporterà la creazione in memoria di un array temporaneo.

Un'altra modalità di accesso al vettore è lo *slicing*, con l'operatore `[:]`, che analogamente alla funzione standard di python, ritorna viste e non copie del vettore. Si tenga presente che una modifica ad una vista corrisponde una modificato al vettore originario. Vediamo qualche esempio

```
# considerando il vettore v precedente
# lo slicing comprende il primo indice, mentre esclude il secondo
# otteniamo una "vista" sulla seconda e terza riga
>>>v[1:3]
array([[ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
# otteniamo una "vista" su terza e quarta colonna
>>>v[:,2:4]
array([[ 2,  3],
       [ 7,  8],
       [12, 13],
       [17, 18]])
```

3.1.3.2. Indexing di Array con Array

Un tipo particolare di indexing consentito da NumPy è quello con gli array(o con altri oggetti di tipo sequenza, escluse le tuple di python). Questo consente di selezionare, in qualsiasi ordine e quante volte si vuole, elementi dell'ndarray. In questo caso gli indici dell'array passato indicano quali sono gli elementi dell'ndarray che si vogliono e in quale ordine si vogliono. Ciò che viene ritornato è sempre una copia dell'array di base. Vediamo un esempio:

```
# si consideri il vettore v precedente
# passiamo la lista [2,1]. Ciò significa che, in questo preciso ordine,
# stiamo selezionando la terza riga e la seconda
>>>v[[2,1]]
array([[10, 11, 12, 13, 14],
       [ 5,  6,  7,  8,  9]])
# selezione quinta, terza e quinta(di nuovo) colonna
>>>v[:,[4,2,4]]
array([[ 4,  2,  4],
       [ 9,  7,  9],
       [14, 12, 14],
       [19, 17, 19]])
```

Una comportamento particolare si ha quando l'indexing viene fatto con più di un array.

```
# indexing con array n-dimensionali
>>>v[[2,3],[4,1]]
```

```
# questo ritorna l'array composto da v[2,4] e v[4,1]
array([14, 16])
```

3.1.3.3. Indexing con Vettori Booleani (o con Maschere Booleane)

In questo situazione l'array booleano deve essere o dello stesso shape dell'ndarray indicizzato o della stessa dimensione dell'asse indicizzato.

```
# caso di array booleano della stesso shape
# l'operatore > è vettorizzato
>>>v_bool = (v>8)
array([[False, False, False, False, False],
       [False, False, False, False,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]], dtype=bool)
>>>v[v_bool]
array([0, 1, 2, 3, 4, 5, 6, 7])
# selezione su righe
>>>v_rows = np.array([False, True, False, True])
>>>v[v_rows]
array([[ 5,  6,  7,  8,  9],
       [15, 16, 17, 18, 19]])
#selezione su righe
>>>v_cols = np.array([False, True, False, True, True])
>>>v[:,v_cols]
array([[ 1,  3,  4],
       [ 6,  8,  9],
       [11, 13, 14],
       [16, 18, 19]])
```

3.2. Le Funzioni Universali - Universal Functions

Le *funzioni universali*, o *ufuncs*, sono funzioni ottimizzate per operare su ndarray elemento per elemento. Sono sia funzioni unarie che funzioni binarie, nelle tabelle 3.2 sono riportate le principali. Vediamo qualche applicazione:

```
# generiamo il vettore di interi 0-11, in shape 3x4
>>>vr = np.arange(12).reshape(3,4)
# applichiamo la ufunc np.sin che calcola il seno element-wise
>>>v_sin = np.sin(vr)
>>>v_sin
array([[ 0.        ,  0.84147098,  0.90929743,  0.14112001],
       [-0.7568025 , -0.95892427, -0.2794155 ,  0.6569866 ],
       [ 0.98935825,  0.41211849, -0.54402111, -0.99999021]])
```

E un esempio di ufunc binaria:

```
# ufunc binaria np.subtract
>>>v1 = np.arange(16).reshape(4,4)
# la somma di uno scalare è effettuata elemento per elemento
```

```
>>>v1+=7
>>>v1
array([[ 7,  8,  9, 10],
       [11, 12, 13, 14],
       [15, 16, 17, 18],
       [19, 20, 21, 22]])
>>>v2 = np.arange(16).reshape(4,4)
>>>v2
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
>>>np.subtract(v1,v2)
array([[7, 7, 7, 7],
       [7, 7, 7, 7],
       [7, 7, 7, 7],
       [7, 7, 7, 7]])
```

Funzione	Descrizione
abs.	valore assoluto di interi, float e complessi
sqrt	radice quadrata
square	calcola quadrato
log, log10, log2	calcola logaritmi
sign	calcola segno: 1 positivo, -1 negativo e 0
ceil	calcola il più piccolo intero >= elemento
floor	calcola il più grande intero <= elemento
isnan	Restituisce un array booleano indicante se l'elemento è NaN)
cos, cosh, sin, sinh, tan, tanh	Funzioni trigonometriche

Tabella 3.2. Ufuncs unarie di NumPy

Funzione	Descrizione
add	somma elemento per elemento
subtract	sottrazione elemento per elemento
multiply	moltiplicazione elemento per elemento
divide	divisione elemento per elemento
power	eleva elementi del primo array alla potenza presente nel secondo
maximum	massimo, ignora casi di NaN
minimum	minimo, ignora casi di NaN
mod	calcola modulo elemento per elemento

Tabella 3.3. Ufuncs binarie di NumPy

3.2.1. Broadcasting

Il broadcasting definisce delle regole che sono applicate ad input che non condividono esattamente le stesse dimensioni (non hanno lo stesso shape), in modo che possano essere applicate operazioni aritmetiche e ufuncs. Vediamo le 4 regole:

1. a tutti gli array in input che hanno un numero di dimensioni minore dell'ndarray di dimensione maggiore vengono inseriti "1" a inizio shape per eguagliare la dimensione massima
2. la misura in ogni dimensione della shape del vettore di output è uguale al massimo di tutte le misure di input in quella direzione
3. qualsiasi input può essere usato nei calcoli se la sua misura in una determinata direzione o ugualia la misura del vettore in output in quella direzione, o è uguale a 1
4. se la misura in una direzione è 1, il dato in quella direzione sarà usata su tutta la dimensione

Andiamo a vedere qualche esempio. Il caso più semplice è un'operazione aritmetica di un vettore con uno scalare; lo scalare ha shape (1) che viene "allargata" alla dimensione del vettore a cui si somma:

```
>>>vr = np.arange(10).reshape(2,5)
>>>vr
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>>vr + 3
array([[ 3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12]])
```

Il broadcasting ci consente di moltiplicare un vettore (2x5) per un vettore (2x1) nel seguente modo:

```
>>>v_column = np.arange(2).reshape(2,1)
>>>v_column
array([[0],
       [1]])
>>>vr * v_column
array([[0, 0, 0, 0, 0],
       [5, 6, 7, 8, 9]])
```

E anche di moltiplicare un vettore di dimensione 2x5 per uno di dimensione 1x5:

```
>>>v_row = np.arange(5)
>>>v_row
array([0, 1, 2, 3, 4])
>>>vr * v_row
array([[ 0,  1,  4,  9, 16],
       [ 0,  6, 14, 24, 36]])
```

3.3. Elaborazione Dati su Array

Verranno ora prese in considerazione alcune funzionalità fornite da NumPy che possono essere utili per l'elaborazione di dati su array.

3.3.1. Logica Condizionale

Alcune funzioni permettono di esprimere logica condizionale come operazioni su vettori; *numpy.where* è una forma "vettorizzata" dell'espressione condizionale "x if condition else y". Si consideri la situazione in cui vi siano 3 vettori di uguale dimensione: uno booleano e 2 di valori:

```
x =  np.array([11,12,13,14,15])
y =  np.array([21,22,23,24,25])
condition = np.array([True, True, False, False, True])
```

Volendo prendere il valore da x nel caso in cui il relativo valore in condition sia True e da y altrimenti, la funzione *where* ci permette di scrivere la condizione in modo conciso:

```
np.where(condition,x,y)
```

Aspetto interessante è che il secondo e terzo parametro di *np.where* possono essere numeri scalari; un utilizzo che se ne può fare è per generare un nuovo vettore di valori in funzione di un altro vettore. Ad esempio si consideri un array bidimensionale di valori casuali e da questo si voglia ricavare un vettore in cui si voglia sostituire 7 se l'elemento è positivo, 3 altrimenti, sarà sufficiente:

```
# creo vettore 4x4 di valori casuali
arr = np.random.randn(4,4)
# genero un nuovo array in base ai valroi di arr
np.where(arr>0, 7, 3)
```

3.3.2. Funzioni Matematiche e Statistiche

Numpy dispone di una serie di funzioni matematiche, di cui riportiamo le principali in tabella 3.4, che consentono il calcolo di statistiche su tutto l'ndarray o anche lungo un'asse particolare dell'array stesso. Le funzioni di "aggregazione", cioè funzioni che calcolano un valore scalare associato all'array, possono essere chiamate come metodo d'istanza o come funzione della libreria Numpy. Vediamo qualche esempio:

```
In []:  arr = np.arange(20).reshape(5,4)
        print arr
        # media su tutti gli elementi di un vettore 5x4
        print arr.mean()
        # somma degli elementi
        print arr.sum()
        # passiamo un asse come argomento
        # somma sulle righe
        print arr.sum(axis=1)
```

```
Out []: [[ 0  1  2  3]
         [ 4  5  6  7]
         [ 8  9 10 11]
         [12 13 14 15]
         [16 17 18 19]]
        9.5
        190
        [ 6 22 38 54 70]
```

Le funzioni non aggregate producono array di risultati intermedi, per esempio sempre considerando l'array arr appena generato:

```
In []:  # somma cumulativa sulle colonne
        print arr.cumsum(0)
        # prodotto cumulativo su righe
        print arr.cumprod(1)
```

```
Out []: [[ 0  1  2  3]
         [ 4  6  8 10]
         [12 15 18 21]
         [24 28 32 36]
         [40 45 50 55]]
        [[    0     0     0     0]
         [    4    20   120   840]
         [    8    72   720  7920]
         [   12   156  2184 32760]
         [   16   272  4896 93024]]
```

Tabella 3.4. Funzioni matematiche e statistiche

Metodo	Descrizione
sum	Somma tuttil gli elementi del vettore o lungo l'asse specificato
mean	Media aritmetica lungo un asse specificato
median	Mediana lungo un asse specificato
std, var	Deviazione standard e varianza
min, max	Minimo e massimo
cumsum, cumprod	Somma e prodotto cumulativo
histogram	Istogramma dei dati

3.3.3. Metodi su Array Booleani

Nei metodi matematici e statistici appena visti, gli array di valori booleani sono interpretati come numeri:

— True = 1
— False = 0

Dunque per calcolare il numero di elementi True (oppure False) in un array è sufficiente usare la funzione *sum*:

```
        arr = np.random.randn(100)
In []:  # genero il vettore booleano ponendo a True i valori di arr>0
        print (arr > 0).sum()
```

Altri due metodi che possono risultare utili sono *any* e *all*, che verificano, rispettivamente, se almeno 1 valore e se tutti i valori dell'array sono True. In particolare questi ultimi due metodi funzionano su vettori di tipi non booleani, considerando True ogni valori diverso da 0.

3.3.4. Ordinamento

Gli array multidimensionali possono essere ordinati "in loco" con il metodo *sort*, passando l'asse lungo il quale si vuole ordinare gli elementi:

```
        arr = np.random.randn(3,4)
In []:  # passo l'asse 1 => ordinamento su righe
        arr.sort(1)
        print arr
```

Esiste anche la possibilità di utilizzare la funzione della libreria stessa, np.sort, tenendo presente che in questo caso il sorting non avviene in loco, ma la funzione ritorna una copia dell'array ordinato.

3.3.5. Logica su Insiemi

NumPy fornisce anche una serie di funzioni di logica sugli insiemi, di cui si riportano i principali in tabella 3.5, da applicare a vettori 1-dimensionali, mentre i vettori di dimensioni superiori a 1 verranno "appiattiti" ad 1 dimensione. La funzione *np.unique*, ad esempio, restituisce un vettore ordinato di valori unici:

```
In []:  arr = np.array(['c','b','c','a','a','a','c'])
        np.unique(arr)
Out []: array(['a', 'b', 'c'],
              dtype='|S1')
```

Altri esempi di funzioni interessanti sono *np.in1d*, che testa l'appartenenza di elementi di un vettore in un altro, e *np.setdiff1d(x, y)*, che restituisce l'insieme di elementi di x che non appartengono ad y:

```
In []:  tmp_arr = ['c', 'm', 'z']
        # appartenenza degli elementi del vettore arr a tmp_arr
        print np.in1d(arr, tmp_arr)
        # differenza tra l'insieme del vettore arr e tmp_arr
        print np.setdiff1d(arr, tmp_arr)

Out []: [ True False  True False False False  True]
        ['a' 'b']
```

Tabella 3.5. Operazioni sugli insiemi

Funzione	Descrizione
`unique(x)`	vettore ordinato degli elementi unici di x
`intersect1d(x, y)`	vettore ordinato dell'intersezione degli elementi di x e y
`union1d(x, y)`	vettore ordinato dell'unione degli elementi di x e y
`in1d(x, y)`	vettore booleano che indica, per ogni elemento di x, l'appartenenza ad y
`setdiff1d(x, y)`	vettore differenza tra l'insieme degli elementi di x e quello di y
`setxor1d(x, y)`	vettore degli elementi che sono in x o y, ma non in entrambi

3.4. Altre Funzionalità di NumPy

NumPy fornisce molte altre funzionalità, che, sebbene non essenziali da un punto di vista dell'elaborazione dati su array, sono elementi fondamentali per una libreria matematica di natura vettoriale. Le 2 principali sono:

3.4.1. Algebra Lineare

Precedentemente abbiamo visto come l'operatore * sia un operatore "vettorizzato" che applicato a 2 array, non calcola il prodotto matriciale, bensì il prodotto elemento per elemento dei 2 vettori. NumPy fornisce sia l'operatore np.dot, come funzione di libreria, sia il metodo *dot* per eseguire prodotto matriciale:

```
In []:  # prodotto matriciale np.dot()
        x = np.arange(8).reshape(2,4)
        y = np.arange(8).reshape(4,2)
        np.dot(x,y)

Out []: array([[28, 34],
               [76, 98]])
```

Tutte le altre funzioni standard di algebra lineare sono fornite nel modulo numpy.linalg. Le funzioni di questo modulo sono implementate utilizzando le stesse librerie standard in Fortran utilizzate in tutto i pacchetti di algebra lineare in commercio. In tabella 3.6 riportiamo alcune delle funzioni di questo modulo. Vediamo qualche esempio:

```
In []:  # matrice inversa
        x = np.arange(4).reshape(2,2)
        x_inv = np.linalg.inv(x)
        # verifico che sia l'inversa
        np.dot(x, x_inv)

Out []: [[ 1.  0.]
         [ 0.  1.]]
```

Vediamo un esempio di soluzione di un sistema lineare:

```
In []:  # soluzione sistema lineare Ax=b
        A = np.array([[1, -1,  1],
                      [2,  1, -1],
                      [1, -1, -1]
                     ])
        b = np.array([6, -3, 0])
        x = np.linalg.solve(A, b)
        # verifichiamo
        np.dot(A,x)

Out []: array([ 6., -3.,  0.])
```

Tabella 3.6. Funzioni di algebra lineare di numpy.linalg

Funzione	Descrizione
`dot`	Moltiplicazione matriciale
`det`	Determinante della matrice
`eig`	Autovalori e autovettori di una matrice quadrata
`inv`	Inversa di una matrice quadrata
`solve`	Soluzione del sistema lineare Ax=b, con A matrice quadrata

3.4.2. Numeri Casuali

Elemento fondamentale nelle simulazioni è la generazione di numeri casuali, NumPy per questo aspetto fornisce una libreria ottimizzata per la generazione di vettori di numeri casuali generati da diversi tipi di distribuzioni di probabilità. In tabella 3.7 ne vediamo le principali. Qualche esempio:

```
In []:  # vettore di numeri casuali 2x3 da distribuzione uniforme
        rnd_arr = np.random.rand(2,3) rnd_arr

Out []: array([[ 0.00765954,  0.01770951,  0.40293959],
               [ 0.53409849,  0.80731357,  0.90559858]])

In []:  # permutazione dei primi 8 numeri naturali
        np.random.permutation(8)

Out []: array([0, 4, 7, 6, 2, 5, 3, 1])
```

Tabella 3.7. Funzioni di numpy.random

Funzione	Descrizione
`rand(d0,d1,..,dn)`	Array con shape (d0,d1,..,dn) da distribuzione uniforme [0,1)
`randn(d0,d1,..,dn)`	Array con shape (d0,d1,..,dn) da distribuzione normale standard
`randint`	Numeri interi casuali da intervallo [low, high)
`beta`	Campioni da distribuzione Beta su [0, 1]
`normal`	Campioni da distribuzione normale (Gaussian)
`uniform`	Campioni da distribuzione uniforme

Capitolo 4

Matplotlib

Come si è avuto modo di vedere precedentemente, oltre alle necessità di manipolazione dei dati e di elaborazione numerica, uno degli strumenti essenziali nel processo di analisi è una tecnologia per la visualizzazione grafica. Durante la fase di analisi grafica, le caratteristiche principali sono la facilità di utilizzo dello strumento, la possibilità di ridisegnare grafici interattivamente, confrontarli, modificarne l'aspetto, ingrandirli, applicare trasformazioni e cambiarne gli stili. Nelle fasi di presentazione, che possono riguardare un resoconto finale così come un resoconto iniziale sull'analisi della qualità dei dati o su una sintesi statistica descrittiva dei dati, assumeranno, invece, importanza strumenti che consentano di definire l'informazione contenuta nel grafico. I grafici in questa situazione dovranno avere testo auto-esplicativo, con l'informazione essenziale inserita direttamente nel grafico.

In questo capitolo verranno introdotte le caratteristiche principali del modulo matplotlib[4] [5], libreria in Python che fornisce gli strumenti per la rappresentazione grafica 2D, mostrando alcuni degli strumenti che fornisce per la manipolazione dei grafici. In seguito si farà qualche esempio per dimostrare come IPython Notebook, con matplotlib inizializzato in modalità inline, consenta la generazione di grafici interattivamente, col grafico riportato nella sezione di output della relativa cella che lo ha generato. La possibilità di modificare il codice di input della cella, rieseguirlo e riottenere un grafico modificato sarà uno strumento essenziale per ogni tipo di analisi esplorativa dei dati. Infine si vedrà come matplotlib fornisca dei backend per l'esportazione dei grafici in vari formati, principalmente PNG, PDF, SVG, PS, con una sola riga di comando, facilitando la gestione di immagini per chi voglia utilizzare strumenti alternativi per la fase di presentazione.

4.1. Analisi Grafica

La fase di analisi grafica, elemento fondamentale di varie tipologie di analisi, in particolare dell'EDA, è una situazione in cui l'esplorazione dei dati in modo interattivo permette di ottenerne informazioni utili riguardo alla natura e struttura degli stessi.

In questo contesto la facilità d'uso e la capacità di generare interattivamente grafici sono le caratteristiche fondamentali che si cercano nello strumento. Si vedrà ora come matplotlib può essere utilizzato in tale situazione. Nel seguito di questo capitolo faremo sempre l'ipotesi di essere all'interno di un IPython Notebook con la modalità grafica inline attivata, funzionalità che si ottiene digitando il seguente comando nel Notebook stesso.

```
In []:  %matplotlib inline
```

Inoltre useremo le seguenti convenzioni per importare le librerie

```
In []:  import matplotlib.pyplot as plt
        import numpy as np
```

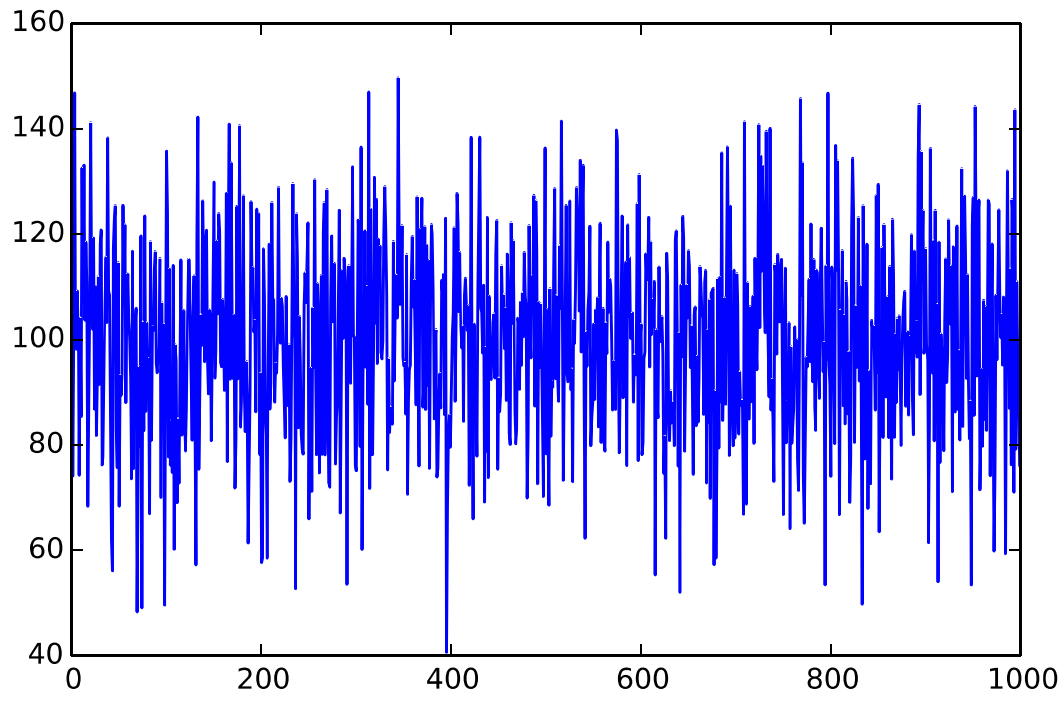
Si prenda in considerazione una semplice situazione in cui si abbia una lista di dati numerici di cui non si conosca la natura e si voglia analizzarli graficamente. Si è visto precedentemente come NumPy fornisca delle librerie ottimizzate per la generazione di vettori di numeri casuali, ci serviremo di questa, nel seguente esempio, per generare una lista di numeri da una distribuzione Normale (Gaussiana) [21] con media = 100 e deviazione standard = 18:

```
        # generazione 1000 campioni casuali
In []:  mu, sigma = 100, 18 # media e deviazione standard
        samples = np.random.normal(mu, sigma, 1000)
```

Interattivamente si provi a fare una semplice rappresentazione in un piano cartesiano. Questa si ottiene semplicemente passando al comando `plot` della libreria la lista con i numeri che si vogliono analizzare. Il comando plot in questa situazione genererà la linea che collega i punti che hanno rispettivamente, come ascissa numeri interi da 0 a (lunghezza dei dati in input - 1) e come ordinata il relativo valore del vettore dato in ingresso. In questa situazione il grafico fornisce poche informazioni: l'intervallo indicativo di massimo e minimo dell'oscillazione (tra 40 e 180) e il numero di campioni:

```
In []:
          # plot del campione di dati
          plt.plot(samples)
```

```
Out []:
```

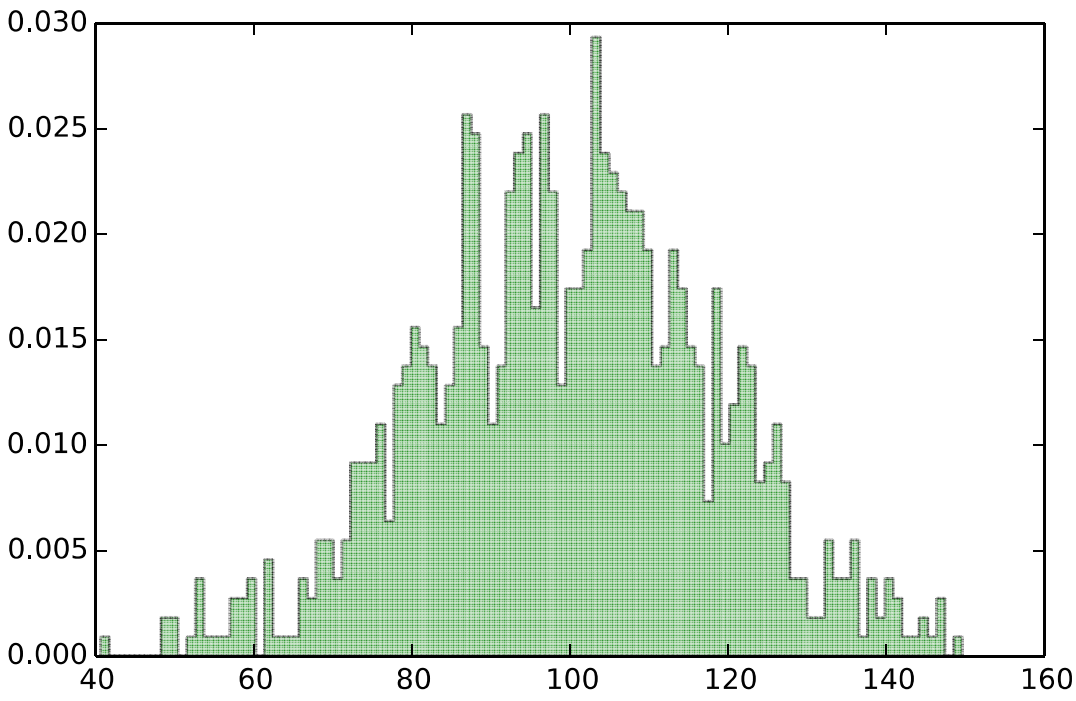


Si può provare a visualizzare la densità del campione attraverso un istogramma, al fine di cercare se segua una densità predefinita. Nel caso in considerazione si inizierà ad intravvedere la funzione di densità di una variabile con distribuzione normale. L'istogramma di un campione si visualizza in matplotlib con la semplice funzione `hist` e può essere configurato con numerosi parametri, in questo caso: suddividiamo l'intervallo di valori in 100 (bins) intervalli di uguale dimensione, passiamo come `histtype` il valore `'step'` in modo da avere la funzione a gradino.

In []:

```
# numero di intervalli su cui fare l'istogramma
num_bins = 100
n, bins, patches = plt.hist(samples, num_bins, normed=True, facecolor='green',
                            alpha=0.3, histtype='step')
```

Out []:

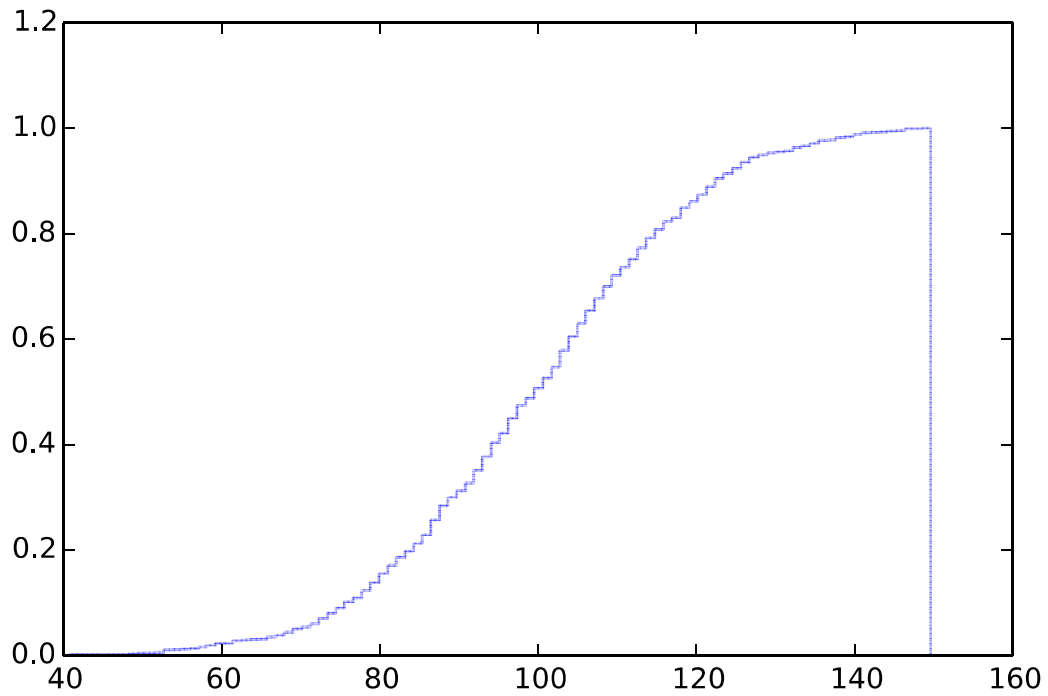


Sempre con la funzione `hist`, passando il parametro `cumulative=True`, si andrà a rappresentare la funzione cumulativa della funzione di densità che, essendone l'integrale, genererà la funzione di distribuzione della variabile. Nel caso in esame si può verificare dal grafico, come effettivamente la funzione cumulativa sia la funzione di distribuzione di una variabile Gaussiana:

In []:

```
# rappresentazione grafica del istogramma cumulativo
n, bins, patches = plt.hist(samples, num_bins, normed=True, facecolor='green',
                            alpha=0.3, histtype='step', cumulative=True)
```

Out []:

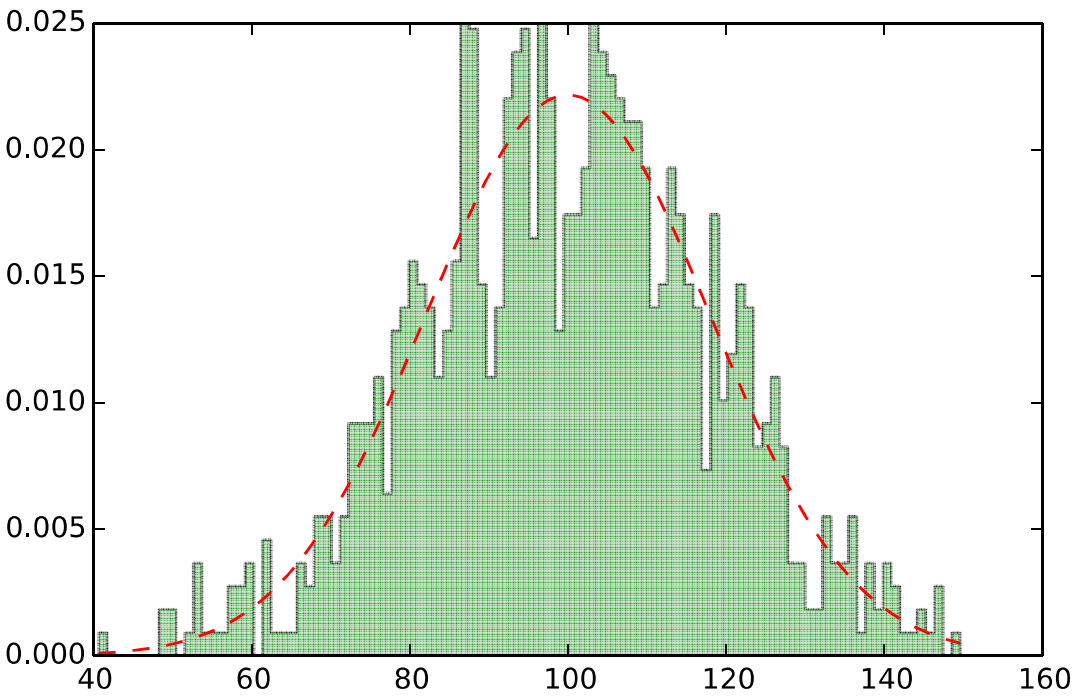


L'intuizione può successivamente essere in qualche modo confermata sovrapponendo al precedente istogramma, il grafico della funzione con media = 100 e deviazione standard = 18:

In []:

```
import matplotlib.mlab as mlab
plt.hist(samples, num_bins, normed=True, facecolor='green', alpha=0.3,
         histtype='stepfilled')
y = mlab.normpdf(bins, mu, sigma)
plt.plot(bins, y, 'r--')
```

Out []:



Da questo breve esempio si può intuire come sia facile interagire e manipolare grafici con matplotlib e IPython Notebook.

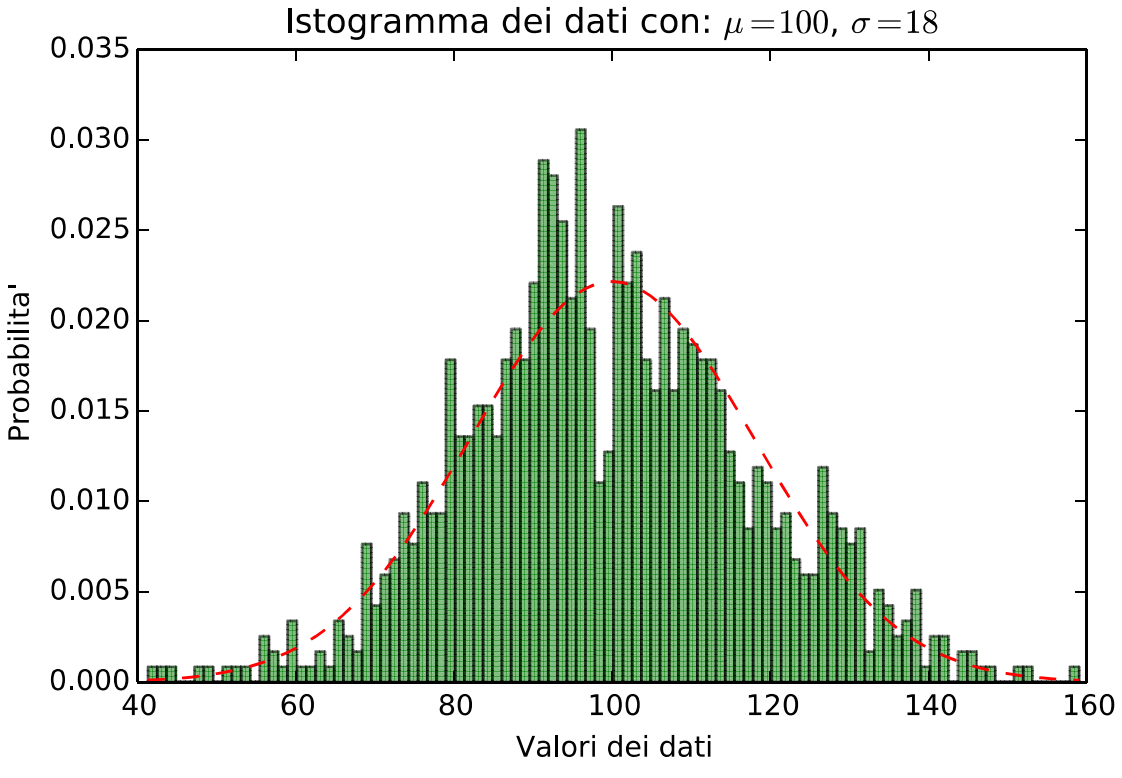
4.2. Grafica di Presentazione

Si andrà ora a vedere qualche esempio che consente di mostrare le funzionalità che fornisce la libreria per inserire informazioni all'interno dei grafici stessi.

Si consideri il grafico precedente e si vogliano mettere etichette sugli assi e definire un titolo

```
In []:
        mu = 100
        sigma = 18
        x = mu + sigma * np.random.randn(1000)
        num_bins = 100
        n, bins, patches = plt.hist(x, num_bins, normed=1, facecolor='green', alpha=0.5)
        y = mlab.normpdf(bins, mu, sigma)
        plt.plot(bins, y, 'r--')
        plt.xlabel('Valori dei dati')
        plt.ylabel('Probabilita\'') plt.title(r'Istogramma dei dati con: $\mu=100$, $\sigma=18$')
        plt.show()
```

```
Out []:
```



4.3. Esportazione dei Grafici

Un aspetto non meno importante è la capacità di poter esportare i grafici in modo che possano essere integrati in vari documenti, a fini documentativi. Sarà dunque essenziale avere uno strumento che, a seconda delle necessità, consenta di generare grafici in vari formati. Da questo punto di vista matplotlib facilità enormemente le cose, fornendo la possibilità di esportare in modo interattivo i grafici in vari formati. In particolare è possibile esportare grafici nei formati: PNG, SVG, PDF, PS. Tutti i grafici di questo lavoro sono stati esportati in formato PDF, direttamente da un IPython Notebook, con il comando `savefig`:

```
In []:  # plt è il plot del grafico
        plt.savefig("nome.pdf", format="pdf")
```

Capitolo 5

Pandas

Pandas[9] è una libreria di Python che fornisce strutture dati di alto livello e strumenti ottimizzati, flessibili ed espressivi, per la manipolazione e la elaborazione di dati di tipo relazionale e tabellare; è un elemento essenziale per fare analisi di dati in Python.

Le tipologie di dati per cui la libreria è ottimizzata sono:

— Dati in forma tabellare con colonne di tipi eterogenei, quali una tabella SQL o fogli di calcolo
— Serie temporali ordinate e non, con frequenze variabili
— dati matriciali arbitrari, di tipo omogeneo o eterogeneo, con etichette su righe e colonne
— qualsiasi insieme di dati statistici

Le due strutture dati principali, Series (dati di tipo 1-dimensionale) e DataFrame (dati di tipo 2-dimensionale), forniscono gli strumenti per la maggior parte delle situazioni che si incontrano nelle analisi di tipo finanziario, statistico, ingegneristico.

Nel seguito del capitolo si cercherà di porre in evidenza come Pandas possa essere efficace nella soluzione di molti punti delle fasi del processo di analisi. In particolare le strutture di alto livello che mette a disposizione e gli strumenti che fornisce, e che consentono manipolazione e trasfomazioni dei dati nelle strutture, saranno essenziali sia nella fase iniziale del processo, durante la quale i dati dovranno essere ristrutturati secondo i modelli della fase di elaborazione, sia nella fase di elaborazione vere a propria. Anche la fase del processo che si occupa del reperimento dei dati verrà presa in considerazione in queste pagine; Pandas offre alcune funzionalità per la lettura e scrittura su file in formato testo e per la connessione e il reperimento con database relazionali. Tuttavia, pur ricadendo il reperimento dell'informazione in 3 categorie principali:

1. lettura/scrittura da file su disco
2. caricamento dati da database
3. sorgenti di rete

vedremo come con Python ed alcune librerie si potrà avere accesso ad un insieme estremamente eterogeneo di sorgenti, di cui qui riportiamo un elenco:

— file di testo, csv, hdf5, fogli di calcolo(Excel)
— basi di dati relazionali e No-Sql
— api di server web

5.1. Strutture Dati

5.1.1. Series

Una struttura dati *Series* è un oggetto 1-dimensionale che contiene un array di dati (che possono essere di qualsiasi tipo tra i data type di NumPy) ed un array di etichette di dati, chiamato *index*, associati ai dati. Il metodo più semplice per costruire l'oggetto è passando un array di dati alla Series, in tal caso verrà generato automaticamente un index composto di interi da 0 a N-1, dove N è la lunghezza dell'array di dati, oppure passando anche un array associato all'index nel costruttore:

```
In []:  labels = ['a', 'b', 'c', 'd', 'e']
        s = Series([3, 7, 9, 12, -3], index=labels)
        s
Out []: a     3
        b     7
        c     9
        d    12
        e    -3 dtype: int64
```

Per selezionare degli elementi si possono usare 1 o più elementi dell'index:

```
In []:  # si può selezionare un singolo elemento con s['a']
        s[['c','d','a']]
Out []: c     9
        d    12
        a     3
        dtype: int64
```

L'indexing booleano, le operazioni di Numpy e le operazioni aritmetiche manterranno inalterata l'associazione index-valore:

In []:	`# elementi > 7` `s[s>7]`	In []:	`#import numpy as np` `np.sin(s)`	In []:	`s**2`
Out []:	c 9 d 12 dtype: int64	Out []:	a 0.141120 b 0.656987 c 0.412118 d -0.536573 e -0.141120 dtype: float64	Out []:	a 9 b 49 c 81 d 144 e 9 dtype: int64

Una Series può essere costruita passando un dizionario di Python come argomento; in questa situazione le chiavi formeranno l'index e gli elementi del dizionario l'array di dati. Se oltre al dizionario si passa un array come index e questo array contiene elementi non presenti tra le chiavi del dizionario, in automatico la Series presenterà il valore NaN(Not a Number), associato alle chiavi mancanti, che in pandas rappresenta i valori "mancanti" (missing):

```
In []:  # si costruisce il dizionario
        s_dati = {'a': 53, 'b': 13, 'c': 4, 'd': 9}
        # l'index contiene chiavi non presenti nel dizionario
        s_index = ['a','b','c','d','e']
        s = Series(s_dati, index=s_index)
        s

Out []: e   NaN
        a    53
        b    13
        c     4
        d     9
        dtype: float64
```

Una funzionalità interessante della Series è l'allineamento sulle chiavi nelle operazioni a aritmetiche:

```
In []:  # si consideri la Series s precedente
        # si costruisce una Series s_1 con sole 3 chiavi comuni: b,c,d
        s_1 = Series({'f': 3, 'b': 2, 'c': 4, 'd': 1},
                     index=['b','c','d','f'])
        # si sommano le Series, le chiavi non corrispondenti saranno NaN
        s + s_1

Out []: a   NaN
        b    15
        c     8
        d    10
        e   NaN
        f   NaN
        dtype: float64
```

5.1.2. DataFrame

Un *DataFrame* rappresenta una struttura dati tabellare (e.g. foglio di calcolo) che consta di una collezione ordinata di colonne, ognuna delle quali può essere di un tipo di dati diverso. Il DataFrame possiede sia un index per le righe che un index per le colonne e può essere pensato come un dizionario di Series che, a loro volta, condividono uno stesso indice.

Un DataFrame può essere costruito passando un dizionario di liste di uguale lunghezza, in tale situazione, il vettore index verrà generato automaticamente, come avviene per le Series, e le colonne saranno disposte in modo ordinato. Nel caso si vogliano avere le colonne ordinate in un preciso ordine sarà sufficiente passare i nomi delle

colonne nell'ordine desiderato, in una lista, al parametro `columns` del costruttore; mentre per definire un index sarà necessario passare una lista di indici al parametro `index`. Vediamone un esempio in cui generiamo numeri da 0 a 15 e passiamo una lista a index 1 a columns :

```
In []:  df = DataFrame(np.arange(16).reshape((4, 4)),
                       index=['A', 'B', 'C', 'D'],
                       columns=['1', '2', '3', '4']) df
```

Out []:

	1	2	3	4
A	0	1	2	3
B	4	5	6	7
C	8	9	10	11
D	12	13	14	15

Una qualsiasi colonna può essere recuperata come una Series o via attributo del DataFrame (df.attr) o con notazione da dizionario (df['attr'])

```
In []:  df['2']
        A     1
        B     5
Out []: C     9
        D    13
        Name: 2, dtype: int32
```

Le righe invece vengono ottenute col metodo indicizzatore `ix` nel seguente modo. Per recuperare la riga 'C', applichiamo al DataFrame il metodo ix e , tra parentesi quadre passiamo l'indice della riga che vogliamo:

```
In []:  df.ix['C']
        1     8
        2     9
Out []: 3    10
        4    11
        Name: C, dtype: int32
```

L'assegnamento di valori ad una colonna può avvenire in 2 modi: con uno scalare come nell'esempio seguente. Si noti che nel caso la colonna non sia presente verrà creata fase di assegnamento:

```
In []:  df['5'] = 200
        df
```

Out []:

	1	2	3	4	5
A	0	1	2	3	200
B	4	5	6	7	200
C	8	9	10	11	200
D	12	13	14	15	200

Altro modo è assegnando alla colonna un lista di uguale lunghezza della colonna. In questo caso generiamo una lista di 4 numeri che a partire da 200, distano 7 uno dall'altro:

```
In []:   df['5'] = np.arange(200, 200 + 7 * 4, 7) df
```

Out []:

	1	2	3	4	5
A	0	1	2	3	200
B	4	5	6	7	207
C	8	9	10	11	214
D	12	13	14	15	221

L'eliminazione di righe o colonne è altrettanto semplice, sarà necessario passare la metodo `drop` l'indice della riga che si voglia eliminare, mentre nel caso delle colonne, oltre all'indice, sarà necessario passare il parametro `axis=1`:

```
In []:   # eliminazione colonna
         df.drop('1', axis=1)
```

Out []:

	2	3	4	5
A	1	2	3	200
B	5	6	7	207
C	9	10	11	214
D	13	14	15	221

Per terminare queste breve introduzione sui DataFrame andremo a vedere come, nello operazioni aritmetiche tra tabelle, l'allineamento sugli indici sia gestito in modo automatico. Si considerino i 2 seguenti DataFrame:

```
In []:   df_1 = DataFrame(np.arange(16).reshape((4, 4)),
                          index=['A', 'B', 'C', 'D'],
                          columns=['1', '2', '4', '5'])
         df_1
```

Out []:

	1	2	4	5
A	0	1	2	3
B	4	5	6	7
C	8	9	10	11
D	12	13	14	15

```
In []:   df_2 = DataFrame(np.arange(16).reshape((4, 4)),
                          index=['E', 'B', 'C', 'D'],
                          columns=['1', '2', '3', '5'])
         df_2
```

Out []:

	1	2	3	5
E	0	1	2	3
B	4	5	6	7
C	8	9	10	11
D	12	13	14	15

Facendo una somma dei 2 DataFrame si avrà un DataFrame con indici dati dall'unione degli indici delle 2 tabelle:

In []:
```
df_1 + df_2
```

Out []:

	1	2	3	4	5
A	NaN	NaN	NaN	NaN	NaN
B	8	10	NaN	NaN	14
C	16	18	NaN	NaN	22
D	24	26	NaN	NaN	30
E	NaN	NaN	NaN	NaN	NaN

Come si può notare, nelle posizioni non comuni, cioè nelle posizioni in cui almeno uno dei 2 DataFrame non ha un dato disponibile, il risultato della somma è NaN. Si può ovviare a questo problema utilizzando uno dei metodi aritmetici del DataFrame e passando in parametro fill_value per riempire le celle non coincidenti. In questo caso andremo a imporre il valore 0 nelle celle non coincidenti, mentre le celle non presenti in entrambi i DataFrame rimarranno NaN

In []:
```
df_1.add(df_2, fill_value=0)
```

Out []:

	1	2	3	4	5
A	0	1	NaN	2	3
B	8	10	6	6	14
C	16	18	10	10	22
D	24	26	14	14	30
E	0	1	2	NaN	3

5.2. Funzionalità di Pandas

Di seguito descriveremo brevemente le funzionalità principali e più interessanti della libreria, mentre per una dettagliata e completa documentazione si rimanda alla documentazione contenuta nel sito

5.2.1. Strumenti di I/O

L'utilizzo di strumenti che possa facilitare l'importazione e l'esportazione dei dati nelle strutture è una caratteristica necessaria di un sistema d'analisi che sia completo. Le principali categorie in cui queste operazioni possono ricadere sono:

— lettura e scrittura di file di testo ed altri formati di memorizzazione
— caricamento dati da una base di dati relazionale
— reperimento dati da una sorgente di rete quale può essere un API web

5.2.1.1. Importazione Dati in Formato Testo e altri Formati

Pandas fornisce un insieme di funzioni che consentono di importare dati tabellari direttamente in un oggetto DataFrame. In tabella 5.1 sono riportati i principali. Le opzioni che queste funzioni permettono, di cui la tabella 5.2 riporta le principali, riguardano:

— indicizzazione: la possibilità di formare un DataFrame con un sottoinsieme delle colonne lette, di ottenere i nomi delle colonne dal file, di impostare i nomi col parametro `names`, di definire l'indice passando il nome della colonna al parametro `index_col` etc, nel caso venga passata una lista di colonne verrà creato un indice multi-livello, detto indice gerarchico, che vedremo dopo nei particolari. Nel caso di file che non abbiano un delimitatore fisso ma che usino spazi o altri pattern per separare i campi è possibile definire un'espressione regolare e passarla al metodo nel parametro `sep`. Nel caso sia necessarie non importare determinate righe è sufficiente passare il numero della riga, tenendo presente che la numerazione parte da 0, nel parametro `skiprows`.

— inferenza sui dati: possibilità di definire tipologie di conversione e capacità di passare liste di valori mancanti. Questo consente all'utente di non doversi preoccupare di specificare quali colonne siano numeriche, intere, booleane, string etc. La gestione di celle mancanti consente di definire il comportamento in caso di valori mancanti o di determinati tipi di sentinelle che indicano il dato mancante. Di default Pandas considera sentinelle le stringhe `NA`, `-1.#IND` e `NULL`. Nel caso si vogliano definire specifiche sentinelle per dati mancanti è sufficiente passare la stringa che rappresenta queste sentinelle nel parametro `na_values`. Nel caso particolare vi siano sentinelle diverse per ogni colonna è possibile creare un dizionario che avrà, come chiavi i nomi delle colonne e come elementi del dizionario una lista delle stringhe sentinelle; questo dizionario sarà poi passato al parametro na_values come prima.

— riconoscimento delle date: possibilità di combinare varie colonne, con date e ore, in una singola colonna

— iterazione: possibilità di avere un iteratore per gestire file di grandi dimensioni. Passando al metodo il numero di righe da leggere per volta al parametro `chunksize`, verrà ritornato un oggetto `TextParser` che consentirà di iterare sul file.

`read_csv`	Importa dati delimitati da virgola, da oggetti di tipo File o URL
`read_table`	Importa dati delimitati da tab ('\t'), da oggetti di tipo File o URL
`read_fwf`	Importa formato di larghezza fissa di colonna, da File o URL
`read_clipboard`	Importa dati dal clipboard

Tabella 5.1. Funzioni di Pandas per l'importazione da Formati Testuali

Argomento	Descrizione delle Funzionalità
path	Stringa indicante path filesystem o URL
sep or delimiter	Sequenza di caratteri o espressione regolare per separare celle delle righe
header	Numero di riga da utilizzare come nome delle colonne. Default = 0
index_col	Numero delle colonne o nomi da usare come indice nel DataFrame
names	Lista di nomi da assegnare alle colonne
skiprows	Righe da saltare a inizio file o lista di righe da saltare
na_values	Sequenza di valori da sostituire con NA
parse_dates	Tenta di interpretare date. False di default.
nrows	Numero di righe da leggere da inizio file
iterator	Ritorna un oggetto TextParser per leggere file a blocchi
chunksize	Dimensione dei blocchi nella letture iterativa
skip_footer	Numero di righe da ignorare a fine file

Tabella 5.2. Argomenti unzioni Lettura/Scrittura file testuali

Pandas ovviamente fornisce metodi per la scrittura su file in formato testuale. Il metodo `to_csv` dei DataFrame consente di scrivere file in formato CSV; anche altri separatori sono consentiti, sarà sufficiente passarli come stringa di testo, al metodo come argomento del parametro `sep`. I valori mancanti del DataFrame saranno esportati come stringhe vuote, nel caso si voglio denotare questi campi con una sentinella particolare, sarà sufficiente passarla come stringa al paramero `na_rep`. Anche i nomi di righe e colonne saranno esportati, nel caso si voglia disabilitare l'esportazione dei nomi delle righe si passerà il parametro `index=False`, mentre per le colonne si passerà `header=False`. Un altro parametro utile è quello che permette di esportare solo un sottoinsieme delle colonne, per esempio per esportare solo le colonne n1,n2,n4, si passerà cols=['n1', 'n2', 'n4'].

5.2.1.2. Altri Formati

Vediamo in qualche dettaglio come sia possibile l'importazione di altri tipi di dati.

— HDF5 - Hierarchical Data Format è un formato che consente di salvare in un file dataset multipli e relativi metadati. HDF5 è una libreria C che consente questo tipo di file e fornisce interfacce per Python

— Excel - Pandas consente la lettura di dati tabulari da file Excel 2003 utilizzando la classe ExcelFile. Questa a sua volta richiede due pacchetti siamo installati xlrd e openpyxl.

— JSON - JavaScript Object Notation è il formato standard per mandare a richieste HTTP. Python ha una libreria standard `json`, per la manipolazione di questo formato

— DataBase - Pandas fornisce un modulo pandas.io.sql che permette l'importazione di dati da basi di dati relazionali.

5.2.2. Applicazione di Funzioni agli Elementi(Mapping)

L'elaborazione dei valori degli elementi di un DataFrame è l'elemento centrale del processo di analisi. Nelle strutture dati di Pandas si possono utilizzare le *ufuncs* di NumPy, vediamo un esempio in cui si genera un DataFrame di valori casuali e di seguito si applica la funzione di numPy che calcola il valore assoluto di un numero:

```
In []:  # ufuncs di numpy
        df = DataFrame(np.random.randn(4, 3), columns=list('abc'),
                       index=['uno', 'due', 'tre', 'quattro'])
        df
```

Out []:

	a	b	c
uno	-0.562711	1.558955	0.307610
due	-0.039113	-0.330573	0.688095
tre	1.429720	0.252972	1.585372
quattro	0.536453	-0.397521	-0.841003

Applichiamo la funzione di numPy alla tabella

```
In []:  # ufuncs di numpy
        np.abs(df)
```

Out []:

	a	b	c
uno	0.562711	1.558955	0.307610
due	0.039113	0.330573	0.688095
tre	1.429720	0.252972	1.585372
quattro	0.536453	0.397521	0.841003

5.2.2.1. Il Metodo `apply`

Vediamo ora come si può applicare un funzione, che prende in input un vettore 1-dimensionale, alle righe o alle colonne. La funzione di Pandas che ci consente di fare questa operazione è il metodo `apply` del DataFrame, a cui si potrà passare la funzione da applicare e l'asse lunga la quale applicarla. Di default queste funzioni vengono applicate sulle colonne, per far si che venga applicata sulle righe sarà sufficiente passare il parametro `axis=1`. Nel seguente esempio andremo a definire una lambda function di Python, che calcolerà la media tra massimo e minimo di un vettore 1-dimensionale.

```
In []:  f = lambda x: (x.max() - x.min()) / 2
```

Senza specificare l'asse, la lambda function verrà applicata da `apply` alle colonne ed il risultato sarà ovviamente un Series:

```
In []:    df.apply(f)

          a    0.996216
          b    0.978238
Out []:
          c    1.213188
          dtype: float64
```

Andando invece a specificare axis=1 avremo la funzione applicata alle righe:

```
In []:    df.apply(f, axis=1)

          uno        1.060833
          due        0.509334
Out []:   tre        0.666200
          quattro    0.688728
          dtype: float64
```

5.2.2.2. Il Metodo `applymap`

Il metodo applymap consente, esattamente come una ufunc, di applicare una funzione elemento per elemento al DataFrame. Supponiamo di voler formattare gli elementi del DataFrame precedente in modo che abbiano 4 cifre decimali. Andremo a definire una funzione che prende un numero e ritorna una stringa con 4 numero decimali. Di seguito applicheremo, elemento per elemento, la funzione al DataFrame.

```
          #formattazione numeri a 4 cifre decimali
In []:    def f(num):
              return '%.4f' % num
```

Applichiamola al DataFrame, ottenendo un altro DataFrame:

```
In []:    df.applymap(f)
```

Out []:

	a	b	c
uno	-0.5627	1.5590	0.3076
due	-0.0391	-0.3306	0.6881
tre	1.4297	0.2530	1.5854
quattro	0.5365	-0.3975	-0.8410

5.2.3. Ordinamento - Sorting

In Pandas le 2 strutture dati principali, Series e DataFrame, possono essere ordinate sia in funzione degli indici, con il metodo `sort_index`,che in funzione degli elementi della struttura con il metodo `order` per le Series e con il metodo `sort_index` con parametri speciali per i DataFrame.

5.2.3.1. Sorting di Series

Costruiamo una semplice Series passando valori e l'index al costruttore:

```
In []:   s = Series([2, 7, 22, -4], index=['d', 'c', 'a', 'b'])
```

Andiamo ora a ordinarla lessicograficamente, in modo inverso, secondo l'index; questo si otterrà passando al costruttore il parametro `ascending=False`:

```
In []:   # la ordino sugli index
         s.sort_index(ascending=False)

         d     2
         c     7
Out []:  b    -4
         a    22
         dtype: int64
```

Infine andremo ad ordinare la Series sui valori col metodo `order`:

```
In []:   s.order()

         b    -4
         d     2
Out []:  c     7
         a    22
         dtype: int64
```

5.2.3.2. Sorting di DataFrame

Il discorso si complica leggermente per i DataFrame in quanto l'ordinamento può avvenire sia su uno qualsiasi degli indici, sia secondo una o più colonne. Generiamo un DataFrame e vediamo qualche esempio esplicativo.

```
In []:   df = DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1], 'c': [3, 22, 2, 45]},
                        columns=('b', 'a', 'c'), index=('j', 'p', 'f', 'x'))
```

Out []:

	b	a	c
j	4	0	3
p	7	1	22
f	-3	0	2
x	2	1	45

Andando a ordinarlo sull'index si otterrà un DataFrame con righe ordinate secondo (f,j,p,x):

```
In []:   df.sort_index()
```

Out []:

	b	a	c
f	-3	0	2
j	4	0	3
p	7	1	22
x	2	1	45

L'ordinamento secondo i valori in questo caso non avviene con la funzione order, come per le Series, ma sempre con la funzione sort_`index` a cui si passano, nel parametro

`by`, gli indici di 1 più colonne secondo cui si vuole ordinare la struttura. Se si volesse ordinare secondo la colonna 'a' e 'c', sarebbe sufficiente passare by=['a','c'] al metodo:

```
In []:   df.sort_index(by=['a','c'])
```

Out []:

	b	a	c
f	-3	0	2
j	4	0	3
p	7	1	22
x	2	1	45

5.2.4. Indexing Gerarchico

L'*Indexing Gerarchico* è una funzionalità che consente di avere più livelli di indicizzazione su di un asse, fornendo la possibilità di manipolare strutture dati n-dimensionali in una forma 1/2-dimensionale.

Si consideri un semplice esempio di una Series a cui, nel costruttore, si passi una lista di liste di uguali misure come parametro `index` e si veda come l'indice gerarchico venga generato con le coppie di elementi presi dalle liste passate in index:

In []:
```
d_h = Series(np.arange(10),
             index=[['a', 'a', 'a', 'b', 'b', 'b', 'c', 'c', 'd', 'd'],
             [1, 2, 3, 1, 2, 3, 1, 2, 2, 3]])
d_h
```

Out []:
```
a  1    0
   2    1
   3    2
b  1    3
   2    4
   3    5
c  1    6
   2    7
d  2    8
   3    9
dtype: int32
```

Questa tipologia di struttura permette funzioni come l'indicizzazione parziale (partial indexing) che consente di estrarre dalla struttura elementi in funzione di uno solo degli indici. Ad esempio si consideri il caso seguente in cui vengono estratti, dalla Series, i blocchi di elementi 'c' ed 'a':

```
In []:
# partial indexing
d_h[['c', 'a']]
```

```
Out []:
a  1    0
   2    1
   3    2
c  1    6
   2    7
dtype: int32
```

Vi è anche la possibilità di estrarre elementi dai livelli interni. Nel seguente esempio si vedrà come estrarre dal livello interno, gli elementi corrispondenti al valore 3:

```
In []:
d_h[:,3]
```

```
Out []:
a    2
b    5
d    9
dtype: int32
```

La Series precedente, composta da un index di 2 livelli, si può pensare come un DataFrame in cui i 2 livelli di indici sono ridistribuiti sui 2 assi. Andiamo a vedere come Pandas ci permetta di ristrutturare la Series in un DataFrame con il metodo `unstack`:

```
In []:
d_h.unstack()
```

Out []:

	1	2	3
a	0	1	2
b	3	4	5
c	6	7	NaN
d	NaN	8	9

Nel caso si voglia ristrutturare un DataFrame che abbia indici di un solo livello, in una Series con indice multi-livello, sarà sufficiente applicare il metodo `stack` al DataFrame.

Analogamente al caso appena visto, anche nel caso dei DataFrame è possibile creare indici gerarchici sugli assi. In questo caso sarà necessario passare al parametro columns e/o al parametro index liste di liste di uguale dimensione, i cui elementi, presi a coppie, andranno a formare l'indice multi-livello. Vediamo un semplice esempio:

```
In []:  df_h = DataFrame(np.arange(12).reshape((4, 3)),
                         index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
                         columns=[['x', 'x', 'y'],
                                  ['1', '0', '1']])
        df_h
```

Out []:

		x		y
		1	0	1
a	1	**0**	**1**	**2**
	2	**3**	**4**	**5**
b	1	**6**	**7**	**8**
	2	**9**	**10**	**11**

Vediamo il funzionamento dell'indexing parziale per la selezione di blocchi di colonne:

```
In []:  df_h['x']
```

Out []:

		1	0
a	1	**0**	**1**
	2	**3**	**4**
b	1	**6**	**7**
	2	**9**	**10**

E l'indexing parziale per le righe col metodo `ix` del DataFrame:

```
In []:  df_h.ix['a']
```

Out []:

	x		y
	1	0	1
1	**0**	**1**	**2**
2	**3**	**4**	**5**

Manipolazioni di Strutture con Indexing Gerarchico

Si vedranno ora alcune tipologie di manipolazione consentite sulle strutture con indice gerarchico. La prima che prenderemo in considerazione è l'inversione dei livelli all'interno dell'indice sull'asse delle righe; gli indici nelle strutture sono numerati dall'esterno verso l'interno con numeri che vanno da 0 a (numero di livelli -1). Il metodo che ci consente di fare l'operazione è `swaplevel` a cui vengono passati i livelli da scambiare:

```
In []:
df_h.swaplevel(0,1)
```

```
Out []:
```

		x		y
		1	0	1
1	a	0	1	2
2	a	3	4	5
1	b	6	7	8
2	b	9	10	11

Un'altra operazione che spesso può risultare utile è lo scambio degli assi. Il metodo che consente l'operazione è `swapaxes`, a cui vanno passati gli assi interessati:

```
In []:
df_h.swaplevel(0,1)
```

```
Out []:
```

		a		b	
		1	2	1	2
x	1	0	3	6	9
	0	1	4	7	10
y	1	2	5	8	11

Infine guardiamo la funzione che consente di ordinare i dati secondo un determinato livello. Il metodo è sortlevel a cui si passa come argomento il livello dell'incide sulle righe con cui si vuole ordinare il DataFrame. Nel caso seguente si scambieranno i livelli nell'indice delle righe e si ordinerà la struttura secondo il livello esterno:

```
In []:
df_h.swaplevel(0,1).sortlevel(0)
```

```
Out []:
```

		x	y	
		1	0	1
1	a	0	1	2
	b	6	7	8
2	a	3	4	5
	b	9	10	11

5.2.5. Merging di Data Sets

Le tipologie di dati presenti nei DataFrame di Pandas possono essere combinati in una varietà di modi:

— facendo il merge di 2 tabelle e connettendo le righe dei DataFrame in base a una o più chiavi, in pratica replicando l'operazione di join di una base di dati relazionale.
— concatenando oggi lungo un determinato asse

5.2.5.1. Merge di DataFrame

Questa operazione consiste nel combinare righe di di 2 tabelle utilizzando 1 o più chiavi. Vediamo un esempio.

Generiamo 2 DataFrame

In []:
```
df_1 = DataFrame({'k': [ 'a', 'a', 'a','b', 'b', 'c', 'c'],
                  'data_1': range(7)})
df_1
```

Out []:

	data_1	k
0	0	a
1	1	a
2	2	a
3	3	b
4	4	b
5	5	c
6	6	c

In []:
```
df_2 = DataFrame({'k': ['a', 'b', 'd'],
                  'data_2': range(3)})
df_2
```

Out []:

	data_2	k
0	0	a
1	1	b
2	2	d

Andiamo ad applicare la funzione `merge` di Pandas. In questo caso, non avendo specificato quali siano le chiavi per fare il merge, Pandas andrà a considerare tutte le colonne che hanno nome comune, cioè k in questo caso.

In []:
```
pd.merge(df_1, df_2)
```

Out []:

	data_1	k	data_2
0	0	a	0
1	1	a	0
2	2	a	0
3	3	b	1
4	4	b	1

Nella situazione in cui le colonne su cui fare il merge abbiano nomi diversi, è possibile definire i nomi delle colonne direttamente nel metodo merge nei parametri `left_on` e `right_on`. Ridefiniamo il secondo DataFrame e rinominiamo la colonna 'j' e facciamo il merge, notando come vengano riportate tutte e due le chiavi:

```
In []: df_2 = DataFrame({'j': ['a', 'b', 'd'],
                        'data_2': range(3)})
       pd.merge(df_1, df_2, left_on='k', right_on='j')
```

Out []:

	data_1	k	data_2	j
0	0	a	0	a
1	1	a	0	a
2	2	a	0	a
3	3	b	1	b
4	4	b	1	b

Come si può notare dai precedenti esempi, i valori delle chiavi 'c' e 'd' ed i relativi valori non sono stati riportati nei risultati. Questo accade perché di default la funzione merge fa quello che, nella teoria delle basi di dati, viene chiamato *inner join*; di conseguenza le chiavi della tabella risultante sono l'intersezione delle chiavi delle due tabelle. Altre modalità di join sono *left*, *right* e *outer*. Queste comportano che le chiavi della tabella risultante siano rispettivamente, le chiavi della tabella di sinistra, le chiavi della tabella di destra e l'unione delle chiavi di entrambe le tabelle.

```
In []: pd.merge(df_1, df_2, left_on='k', right_on='j', how='outer')
```

Out []:

	data_1	k	data_2	j
0	0	a	0	a
1	1	a	0	a
2	2	a	0	a
3	3	b	1	b
4	4	b	1	b
5	5	c	NaN	NaN
6	6	c	NaN	NaN
7	NaN	NaN	2	d

Nel caso si voglia fare il join su più colonne sarà sufficiente passare la lista dei nomi delle colonne che saranno chiavi al parametro `on`.

Può presentarsi il caso in cui le chiavi su cui fare il merge siano parte dell'index del DataFrame, in questo caso sarà sufficiente passare il parametro left_index=True e/o right_index=True per indicare che l'index sia usato come chiave per il merge. Vediamo un esempio nel quale generiamo un DataFrame con un index che poi andremo a utilizzare per fare il merge:

In []:
```
df1 = DataFrame({'k': ['a', 'd', 'a', 'c'],
                 'v1': range(4,8)})
df1
```

Out []:

	k	v1
0	a	4
1	d	5
2	a	6
3	c	7

In []:
```
df2 = DataFrame({'v2': [9, 14]}, index=['a', 'b']) df2
```

Out []:

	v2
a	9
b	14

Facendo il merge, specificando la chiave k nella tabella sinistra e l'index nella tabella a destra e tenendo in considerazione che di default è un *inner* join, si ottiene:

In []:
```
pd.merge(df1, df2, left_on='k', right_index=True)
```

Out []:

	k	v1	v2
0	a	4	9
2	a	6	9

5.3. Visualizzazione di Dati da Strutture di Pandas

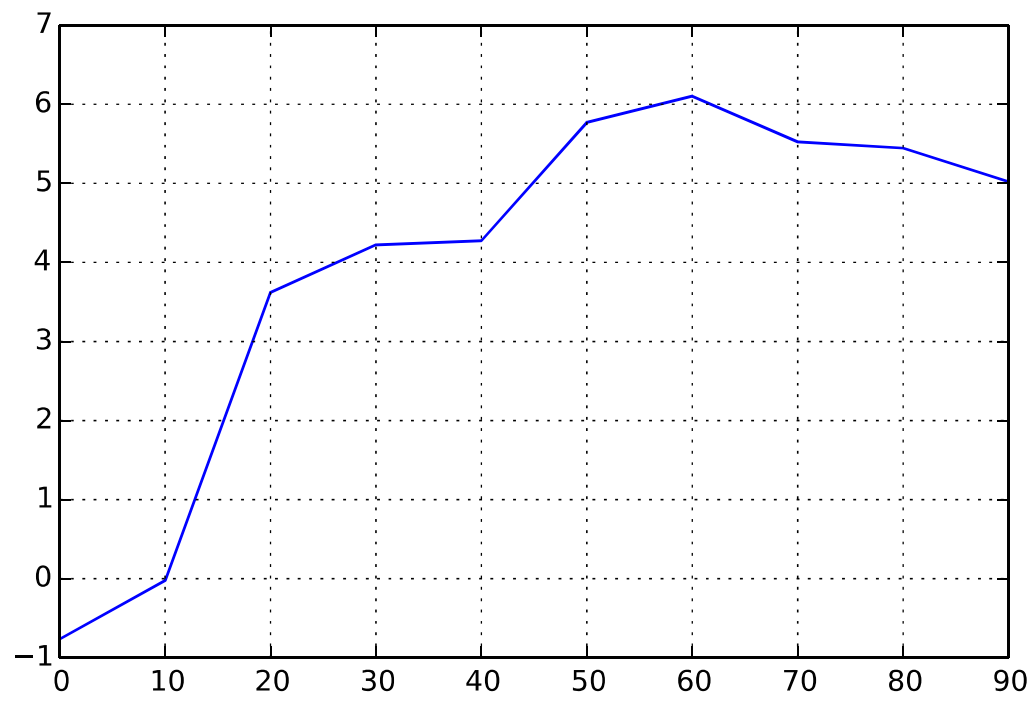
Pandas fornisce un insieme di metodi grafici costruiti su matplotlib, per creare grafici sfruttando l'organizzazione dei dati nelle Series e nei DataFrame. Sia le Series che i DataFrame hanno un metodo `plot` che consente di disegnare alcune tipologie di grafici.

Facendo l'ipotesi di essere in IPython Notebook in modalità matplotlib inline, vediamo qualche esempio:

In []:

```
s = Series(np.random.randn(10).cumsum(), index=np.arange(0,100,10))
s.plot()
```

Out []:



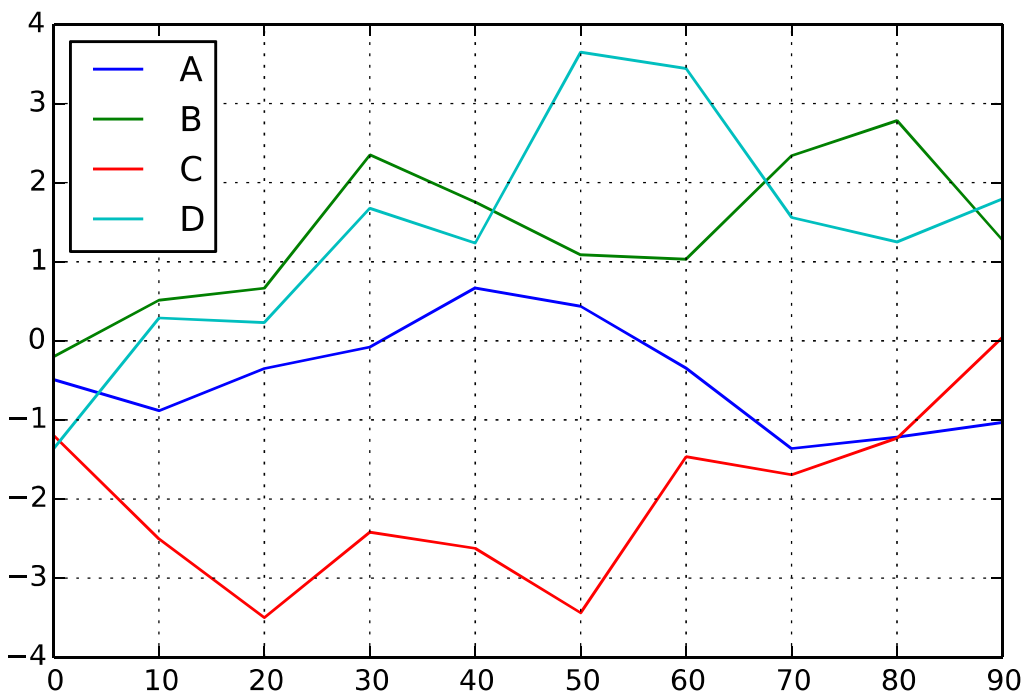
come si può vedere dal grafico l'index della Series viene passato a matplotlib che lo rende l'asse delle x, funzionalità che può essere disabilitata passando il parametro `use_index=False`. Passando come una stringa al parametro `label` possiamo impostare una legenda al grafico.

Vediamo ora come funziona il metodo `plot` di un DataFrame. In questa situazione il metodo mapperà ognuna delle colonne su una linea diversa, utilizzando l'index, comune a tutte le colonne, come asse x, mentre i nomi delle colonne verrano automaticamente inseriti in una legenda.

In []:

```
df = DataFrame(np.random.randn(10, 4).cumsum(0),
               columns=['A', 'B', 'C', 'D'],
               index=np.arange(0, 100, 10))
df.plot()
```

Out []:



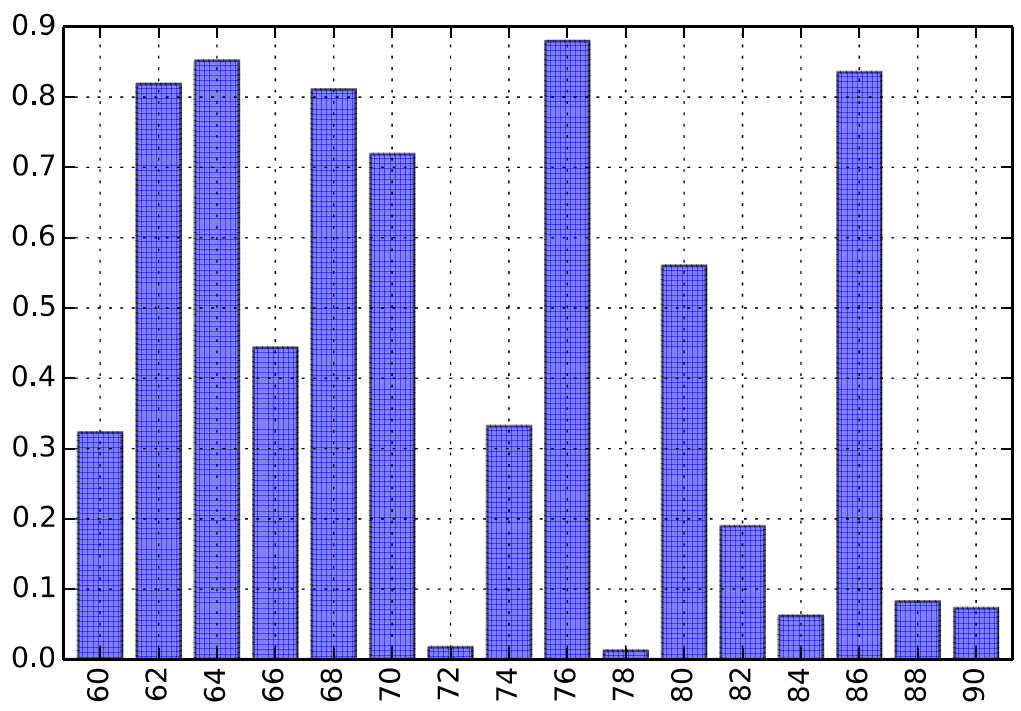
Al metodo `plot` del DataFrame possono essere passati diversi parametri per la configurazione del grafico ad esempio,nel caso si voglia avere una scala logaritmica sulle y si passerà logy=True; è possibile non utilizzare l'index delle righe del DataFrame come indice dell'asse x, passando il parametro `use_index=False`. Si può generare un titolo del grafico passando una stringa col titolo al parametro `title`. La legenda può essere disabilitata passando `legend=False`. Il parametro `kind` definisce il tipo di grafico che sarà generato: 'line,'bar','barh','kde'.

Vediamo ora come sia semplice generare un diagramma a barre per le Series:

In []:

```
data = Series(np.random.rand(16),
              index=np.arange(60,60+2*16,2))
data.plot(kind='bar', color='b', alpha=0.5)
```

Out []:

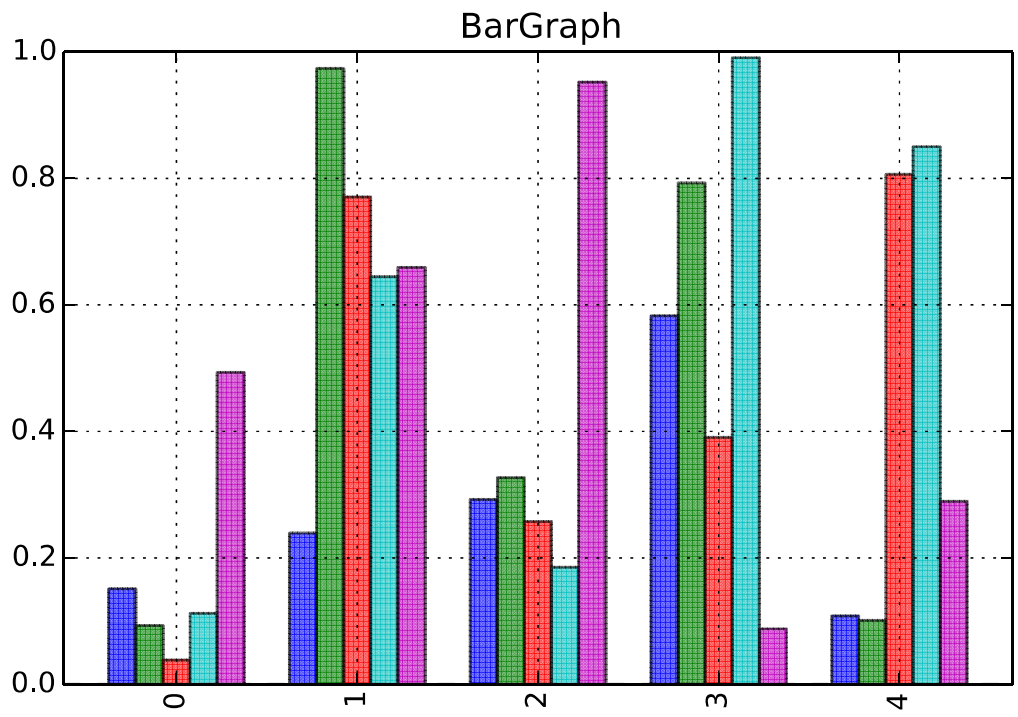


Nel caso di DataFrame i grafici a barre rappresentano le righe come gruppi di barre contigue. Vediamo un esempio:

In []:

```
df = DataFrame(np.random.rand(5, 5),
               columns=['A', 'B', 'C', 'D', 'E'])
df.plot(kind='bar', alpha=0.6, title='BarGraph', legend=False)
```

Out []:

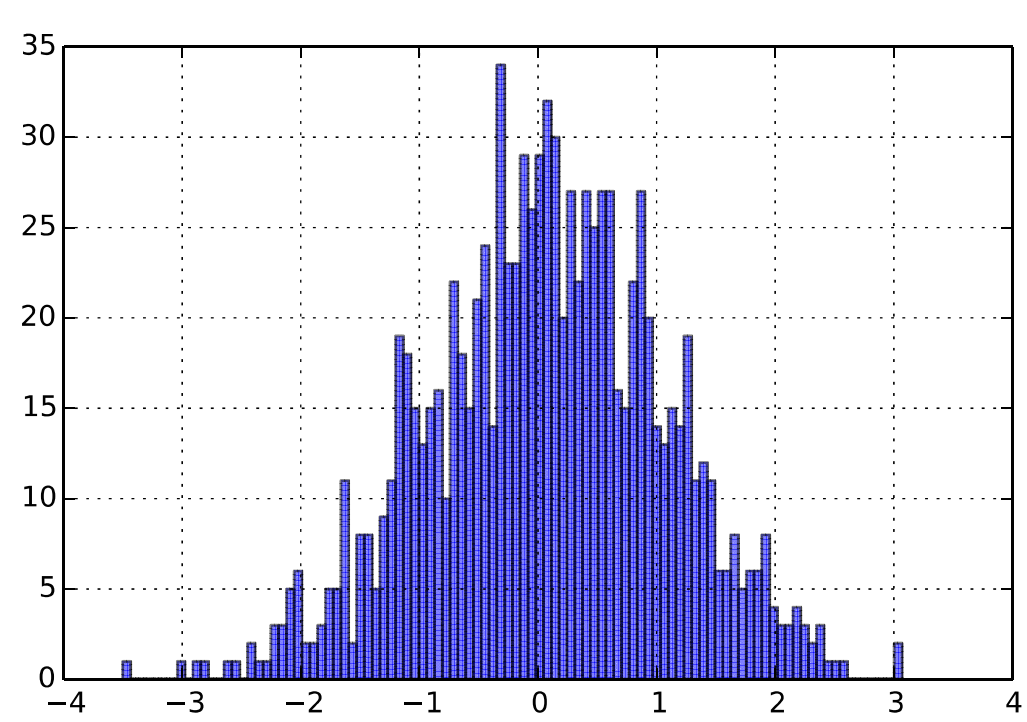


Il metodo hist delle Series consente di generare un istogramma:

In []:

```
s = Series(np.random.randn(1000))
s.hist(bins=100, color='b', alpha=0.5)
```

Out []:



Capitolo 6

Conclusioni

In questo lavoro si è voluto presentare una tecnologia open source, basata su librerie e moduli Python, che potesse rappresentare una soluzione in qualche modo omogenea per affrontare tutte le fasi di un generico processo di analisi di dati. L'analisi di alcuni standard di processo ha consentito di astrarre una serie di fasi che caratterizzano ogni processo di analisi dei dati, in particolare:

— reperimento dei dati
— analisi iniziale
— elaborazione numerica dei dati
— presentazione dei risultati

Nella presentazione delle varie tecnologie si è cercato di porre in evidenza come esse possano essere utilizzate per risolvere tutti compiti da eseguire in ciascuna delle fasi. Abbiamo quindi visto come la libreria Pandas, assieme a Python e ad altre librerie, consenta di affrontare il problema di connessione con l'eterogeneo insieme di possibili sorgenti di dati: file di testo, file binari, fogli di calcolo, basi di dati relazionali e non e, infine, con tecnologie di rete. Sia Numpy che Pandas stessa, fornendo le strutture dati di alto livello e i metodi per manipolarle, rispondono alle problematiche che si devono affrontare sia nella fase iniziale che nella fase di elaborazione numerica dei dati, e che riguardano compiti come la pulizia dei dati, la sostituzione di valori, il filtraggio, la derivazione di attributi, la gestione dei dati vuoti e, ovviamente, la stessa elaborazione numerica dei dati. La libreria grafica matplotlib, generando grafici e rendendo possibile la loro esportazione in vari formati, ci consente infine di risolvere i problemi relativi alla visualizzazione dei dati, sia nella fase esplorativa attuata nell'analisi iniziale, sia nella fase di presentazione, . L'elemento che in qualche modo integra tutte queste tecnologie è l'ambiente IPython. Questo ambiente ci fornisce degli strumenti essenziali per l'elaborazione interattiva e, nella sua versione di IPython Notebook, rende disponibile una applicazione web che, consentendo l'integrazione di codice eseguibile, documentazione e grafici, fornisce uno strumento di presentazione estremamente innovativo. La capacità di salvare i Notebook in formato JSON consente inoltre l'esportazione di tutti i modelli, consentendo a chiunque di reimportare

i file e di replicare tutto il processo di analisi. Infine IPython, rendendo disponibile un motore di elaborazione parallelo, fornisce la possibilità di progettare modelli di analisi distribuiti che ben si adattano alle moli di dati attualmente prodotte. La trasversalità dello strumento presentato, il fatto di poter essere eseguito in qualsiasi browser moderno indipendentemente dalle tecnologie che caratterizzano lo strumento su cui viene eseguito, e la possibilità di delegare l'effettiva esecuzione a risorse distribuite, rende questa tecnologia uno strumento ideale per le moderne necessità dell'analisi dei dati.

Bibliografia

[1] Christopher Clifton. Encyclopaedia britannica: Definition of data mining. *http://www.britannica.com/EBchecked/topic/1056150/data-mining*, 2010. [Online; accessed 3-April-2014].

[2] Ken Collier, Bernard Carey, Donald Sautter, Curt Marjaniemi. A methodology for evaluating and selecting data mining software. *Systems Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on*, strony 11–pp. IEEE, 1999.

[3] Usama Fayyad, Gregory Piatetsky-shapiro, Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

[4] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[5] John D. Hunter. Matplotlib: a python 2d plotting library. *http://matplotlib.org/*.

[6] KdNuggets.Com. Data mining methodology. *http://www.kdnuggets.com/polls/2002/methodology.htm*, 2002.

[7] KdNuggets.Com. Data mining methodology. *http://www.kdnuggets.com/polls/2004/data_mining_methodology.htm*, 2004.

[8] KdNuggets.Com. Data mining methodology. *http://www.kdnuggets.com/polls/2007/data_mining_methodology.htm*, 2007.

[9] Wes McKinney. Pandas: Python data analysis library. *http://pandas.pydata.org/*.

[10] NIST/SEMATECH. e-handbook of statistical methods. *http://www.itl.nist.gov/div898/handbook/eda/eda.htm*. [Online; accessed 31-March-2014].

[11] Charles Nyce. Predictive analytics white paper. *American Institute for Chartered Property Casualty Underwriters/Insurance Institute of America*, 2007.

[12] Fernando Pérez, Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, Maj 2007.

[13] Colin Shearer. The crisp-dm model: The new blueprint for data mining. *Journal of Data Warehousing*, 5, 2000.

[14] Christopher C. Shilakes, Julie Tylman. Enterprise information portals. *Merrill Lynch.*, 1998.

[15] Wikipedia. Beta distribution — wikipedia the free encyclopedia. *http://en.wikipedia.org/w/index.php?title=Beta_distribution&oldid=602929322*. [Online; accessed 7-April-2014].

[16] Wikipedia. Data analysis — wikipedia the free encyclopedia. *http://en.wikipedia.org/wiki/Data_analysis*. [Online; accessed 29-March-2014].

[17] Wikipedia. Semma — wikipedia the free encyclopedia. *http://en.wikipedia.*

org/w/index.php?title=SEMMA&oldid=584339278, 2013. [Online; accessed 31-March-2014].

[18] Wikipedia. Business intelligence — wikipedia the free encyclopedia. *http://en.wikipedia.org/w/index.php?title=Business_intelligence&oldid=602473828*", 2014. [Online; accessed 3-April-2014].

[19] Wikipedia. Descriptive statistics — wikipedia the free encyclopedia. *http://en.wikipedia.org/w/index.php?title=Descriptive_statistics&oldid=601461306*, 2014. [Online; accessed 31-March-2014].

[20] Wikipedia. Exploratory data analysis — wikipedia the free encyclopedia. *http://en.wikipedia.org/w/index.php?title=Exploratory_data_analysis&oldid=595294588*, 2014. [Online; accessed 31-March-2014].

[21] Wikipedia. Normal distribution — wikipedia the free encyclopedia. *http://en.wikipedia.org/w/index.php?title=Normal_distribution&oldid=602961562*, 2014. [Online; accessed 7-April-2014].

[22] Wikipedia. Predictive analytics — wikipedia the free encyclopedia. *http://en.wikipedia.org/w/index.php?title=Predictive_analytics&oldid=601087176*, 2014. [Online; accessed 2-April-2014].