# Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica e dell'Automazione



**Tesi di Laurea**

## Realizzazione di una campagna di data analytics per l'individuazione di anomalie da dati di produzione di un'azienda manifatturiera

## Implementation of a data analytics campaign for the detection of anomalies from production data of a manufacturing company


Relatore

Prof. Domenico Ursino

Candidata

Beatrice Moliterno



**Anno accademico 2019-2020**

# Indice

# Elenco delle figure

# Introduzione

L'espressione "Industria 4.0" nacque in Germania nel 2011, anno della presentazione del piano industriale che si concretizza nel 2013. Il progetto prevedeva investimenti per ammodernare il sistema industriale tedesco promuovendo l'informatizzazione del processo di produzione come parte della strategia nazionale ad alta tecnologia. Da allora, il termine è diventato sempre più popolare in tutto il settore industriale ed è stato promosso come la giusta intuizione circa il futuro della produzione, tanto dagli esperti del settore e dai fornitori di tecnologia, quanto dal mondo accademico. L'impresa digitale è il futuro della produzione e gli appassionati di questa rivoluzione tecnologica direbbero che è l'unico mezzo attraverso il quale un'azienda possa sperare di mantenere una posizione rilevante e competitiva nel sempre più complesso ed esigente mercato globale di domani. Dunque, il concetto di "Industria 4.0" non si limita ad essere un neologismo per identificare la quarta rivoluzione industriale ma un "simbolo" di cambiamento profondo, tecnologico e visionario.

Le nuove tecniche e i nuovi strumenti di Business Intelligence, Data Visualization, Simulation e Forecasting e Data Analytics devono elaborare i Big Data, intesi nell'accezione più "classica" di grandi moli di dati, provenienti da Produzione e Supply Chain, quindi da sistemi IoT o dall'interconnessione dei sistemi IT con quelli OT (Operational Technology). Tali sistemi sono atti a "far emergere" la conoscenza celata nei dati.

I dati provengono da sistemi di rilevamento automatico, da sistemi automatici, sensori e strumentazione scientifica, oppure vengono prodotti quotidianamente e inconsciamente attraverso, ad esempio, un acquisto o l'utilizzo di social network. Essi sono flussi di byte che, a prima vista, non hanno un vero e proprio significato. L'informazione è, in realtà, il risultato di un'elaborazione che, tenendo conto di un certo insieme di dati, estrae delle conclusioni che possono essere utilizzate in vari modi. Questo processo di estrazione delle informazioni dai dati grezzi è chiamato Data Analytics.

La Data Analytics è un campo le cui funzioni sono altamente interconnesse con quelle della Data Science. Quest'ultimo campo ha come obiettivo principale quello di porre domande e individuare potenziali vie di studio, con meno attenzione a risposte specifiche e maggiore enfasi posta sulla ricerca della giusta domanda da porre. I data scientist utilizzano diverse tecniche per ottenere risposte, incorporando l'informatica, l'analisi predittiva, le statistiche e l'apprendimento automatico per

analizzare grandi set di dati nel tentativo di stabilire soluzioni a problemi che non sono stati ancora posti. Il campo della Data Science ha acquisito un'importanza sempre maggiore e i vantaggi che ha portato sono molteplici; tra questi ricordiamo una migliore gestione delle risorse aziendali, un monitoraggio e una manutenzione più efficace, un'ottimizzazione e un controllo della produzione, un aumento della sicurezza negli impianti, nonché un miglioramento nell'esperienza di acquisto.

La presente tesi si colloca proprio in tale contesto e si propone di realizzare un'analisi di un dataset dell'azienda Elica, un'impresa marchigiana attiva principalmente nel mercato delle cappe da cucina. Questo dataset ha come protagonista un innovativo dispositivo chiamato *Snap "Air Quality Balancer"*, un sistema intelligente dotato di tre sensori che rilevano in maniera continuativa la qualità, l'umidità e la temperatura dell'aria nell'ambiente in cui il dispositivo è stato installato. Questa raccolta di dati semi-strutturati appartiene alla categoria dei Record Data, ovvero dataset organizzati in formato tabellare; in essa sono raccolte informazioni riguardo le rilevazioni effettuate dal 25 Gennaio 2020 al 16 Giugno 2020 dai di 1.311 Snap. Il nostro obiettivo sarà quello di studiare, attraverso l'analisi di questi dati, il comportamento dei dispositivi, individuando informazioni che potrebbero risultare utili per l'azienda. In particolare, ci concentreremo nell'individuazione di anomalie nel comportamento di questi sistemi.

Nella fase preliminare del nostro progetto approfondiremo la Data Science e parleremo del linguaggio di programmazione Python, ovvero lo strumento che utilizzeremo per la realizzazione del nostro progetto. Presenteremo Snap e descriveremo dettagliamene il nostro dataset per comprendere appieno il dispositivo e per porci domande e ipotesi riguardo il suo funzionamento. Sulla base delle conoscenze acquisite, effettueremo una fase di ETL (Extract, Trasform and Load) per filtrare i dati per noi più utili e progetteremo un'analisi esplorativa basata su grafici semplici ed intuitivi.

La nostra campagna sperimentale si dividerà in due fasi sviluppate tramite la divisione del dataset in Snap senza errori e Snap con errori. Durante la prima fase studieremo gli Snap senza errori e sceglieremo, tra questi, un dispositivo che sarà sottoposto ad analisi approfondite riguardo la temperatura, l'umidità e il livello di VOcs rilevati nell'ambiente. Durante la seconda fase studieremo gli Snap con errori e sceglieremo tra questi quattro dispositivi, ciascuno con uno o più errori differenti, che saranno sottoposti ad analisi per la valutazione degli effetti che ciascun errore ha sul corrispondente dispositivo. In entrambe le fasi verrà eseguita l'anomaly detection, una tecnica che ha lo scopo di individuare anomalie nel comportamento dei sensori. Al termine di tali analisi trarremo le nostre conclusioni valutando i punti di forza e debolezza di questo dispositivo; inoltre, descriveremo le lezioni apprese da tale progetto e le eventuali analisi future che potrebbero essere svolte.

La presente tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 verrà introdotta la Data Science, approfondendo la storia, l'importanza e le sue fasi di studio descritte nella gerarchia dei bisogni. In seguito verrà presentato il linguaggio di programmazione Python come suo potente strumento e, in particolare, verranno analizzate le librerie di Python più comuni in questo contesto.

- Nel Capitolo 2 verrà introdotto il concetto di dataset, soffermandoci sulle diverse tipologie e caratteristiche, per collocare e comprendere al meglio la nostra raccolta di dati che, successivamente, verrà descritta in ogni suo campo.
- Nel Capitolo 3 verranno illustrate le fasi di ETL (Extract, Trasform and Load), l'analisi esplorativa e l'anomaly detection. In particolare, verrà dapprima fornita una descrizione generale e successivamente ci si concentrerà sui dati relativi alla nostra campagna sperimentale.
- Nel Capitolo 4 verrà implementato il nostro progetto e seguiranno due analisi; la prima si concentrerà sui dispositivi senza errore, e verrà studiato lo Snap 2.257, mentre, la seconda fase si concentrerà sui dispositivi con errore, e verranno studiati gli Snap 1.347, 1.838, 1.320 e 1.872. In entrambe le fasi presteremo particolare attenzione agli outlier.
- Nel Capitolo 5 descriveremo i principali risultati ottenuti da questo progetto, soffermandoci sulle lezioni apprese durante questa campagna sperimentale.
- Nel Capitolo 6 verranno tratte delle conclusioni in merito al lavoro svolto e si analizzeranno alcuni possibili sviluppi futuri.

# 1

# L'Ecosistema Python per la Data Science

*Nell'attuale contesto di trasformazione digitale e passaggio in quella che molti analisti chiamano "quarta rivoluzione industriale", il data scientist sta emergendo come la figura professionale più importante per dare valore ai dati aziendali, ricoprendo, così, un ruolo essenziale nella strategia di sviluppo del business di ogni impresa e organizzazione. In questo capitolo parleremo della Data Science e di come ha rivoluzionato diversi aspetti del nostro mondo, nonché dell'ecosistema Python, come suo potente strumento.*

## 1.1 La Data Science

La Data Science è un campo interdisciplinare che utilizza metodi e processi scientifici per trarre intuizioni dai dati. Allo scopo di far maggior chiarezza su questa scienza, Hugh Conway, nel 2010, creò un diagramma di Venn composto da tre grandi aree: matematica e statistica, competenza del soggetto (conoscenza del dominio da astrarre e calcolare), e abilità di hacking (Figura : 1.1).


**Figura 1.1.** Data Science Venn Diagram - Hugh Conway

Ognuna di queste competenze è di per sé molto preziosa, ma la loro intersezione è necessaria affinché si parli di Data Science. Vediamole più approfonditamente.

- Innanzitutto i data scientist devono gestire una grande quantità di dati, e per questa ragione, hanno la necessità di utilizzare strumenti molto sofisticati. I linguaggi fondamentali utilizzati in questo campo includono SQL, Python, R e SAS. Ma non si tratta solo di conoscere i fondamenti del linguaggio: quando si parla di *hacking* in questo caso, ci si riferisce all'abilità di trovare soluzioni intelligenti, rapide e creative a problemi complessi di codifica per la manipolazione, la raccolta dei dati e la costruzione di un modello di previsione di dati futuri. Sono, dunque, fondamentali le capacità di modificare file di testo a riga di comando, capire le operazioni vettoriali, pensare algoritmicamente.
- Una volta acquisiti e puliti i dati, il passo successivo è quello di estrarre effettivamente delle intuizioni da essi. Per fare questo, è necessario applicare metodi matematici e statistici appropriati, il che richiede almeno una familiarità di base con questi strumenti. La statistica, infatti, è un insieme di metodi e strumenti matematici che ci consentono di rispondere a domande importanti sui dati. Si divide in due categorie: *statistica descrittiva* e *statistica inferenziale*. La prima offre metodi per riassumere i dati trasformando le osservazioni grezze in informazioni significative facili da interpretare e condividere. La seconda offre metodi per studiare esperimenti condotti su piccoli campioni di dati e trarre le inferenze sull'intera popolazione (intero dominio).
  In particolare, le analisi statistiche più semplici che possiamo incontrare nella Data Science sono:
  - *A/B testing*: è una tecnica di ottimizzazione e sperimentazione semplice utilizzata nel marketing. Consiste nell'avere due versioni di uno stesso elemento: una è la versione originale, chiamata la *versione di controllo*, e l'altra è la versione modificata, chiamata il *trattamento*. Queste vengono mostrate ad un gruppo di utenti nello stesso arco temporale e, al termine del test, la versione migliore, cioè quella che ha ottenuto il rendimento migliore rispetto agli obiettivi che ci si era prefissati, viene scelta e utilizzata come versione definitiva. Dunque l'A/B tuttavia può essere considerato come una forma di test di verifica d'ipotesi o "test di ipotesi a 2 campioni" nel campo della statistica.
  - *Correlazione*: sono delle tecniche che permettono di valutare la presenza o meno del legame tra due variabili, che non sia necessariamente causale, ma tale per cui, se c'è un cambiamento in una di esse, ci sarà sicuramente un cambiamento anche nell'altra. Il grado di associazione tra le due variabili è misurato dal coefficiente di correlazione, indicato con $r$:

$$r = \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \overline{x})^2 \sum_{i=1}^{n} (y_i - \overline{y})^2}}$$

    $r$ è un valore che varia da 1 a -1. Parliamo di *correlazione diretta*, se la correlazione fra due variabili tende a 1 e, di conseguenza, al crescere di una variabile cresce anche l'altra. Mentre, parliamo di *correlazione inversa* se la

correlazione fra due variabili tende a -1 e, di conseguenza, l'aumento di una grandezza provoca la diminuzione dell'altra. Se la correlazione è nulla, vuol dire che non abbiamo correlazione.

– *Regressione*: sono delle tecniche che identificano una legge che lega una o più variabili indipendenti ad una variabile dipendente. Parliamo, in questo caso, di una relazione di causa/effetto, per la quale è possibile stabilire un delta che permette di valutare l'aumento/diminuzione di una variabile rispetto all'aumento/diminuzione dell'altra. Il caso più semplice di regressione è la *regressione lineare* nel qual caso le grandezze in gioco sono soltanto 2. Il suo grafico rappresentativo è una retta e ha come funzione tipica $Y = \alpha + \beta X$. Per determinare $\alpha$ e $\beta$ esistono diversi metodi, il più comune è quello dei minimi quadrati:

$$\beta = \frac{\sum_{i=1}^{s} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{s} (x_i - \overline{x})^2}$$

$$\alpha = \overline{y} - \beta \overline{x}$$

I casi più complessi di regressione riguardano la regressione quadratica, logaritmica ed esponenziale.

- La conoscenza del dominio è la conoscenza specifica del campo in cui si sta applicando la Data Science. Senza di essa potremmo trarre conclusioni sbagliate riguardo alle intuizioni sui dati. Nella conoscenza del dominio è, dunque, necessario avere ben chiaro l'obiettivo, i metodi e anche i vincoli del campo prima di implementare un modello di Data Science o di Machine Learning. In sostanza, ci aiuta sia per formulare le domande giuste sia per contestualizzare le loro risposte.

### 1.1.1 La storia

Dall'inizio del 21° secolo, i volumi di dati sono aumentati in modo rapido ed esponenziale, in gran parte grazie ai progressi di elaborazione e archiviazione su scala. Ma il "dare un senso ai" dati ha una lunga storia, ed è stato discusso da scienziati, statistici, bibliotecari informatici per diversi anni.

Potremmo far risalire le sue radici al Teorema di Bayes del 1740, che ha creato le basi per il calcolo delle probabilità, ed è lo stesso che oggi alimenta l'IA (Intelligenza Artificiale). Ma la vera prima pietra miliare nella storia di questa scienza è globalmente riconosciuta con il brillante matematico americano John Tukey che, scrisse nel 1962, *The Future of Data Analysis*. Nel 1977 lo stesso autore pubblica *Exploratory Data Analysis*, nel quale afferma *"Today, exploratory and confirmatory can–and should–proceed side by side"*, riuscendo così a dare una nuova chiave di lettura per l'analisi dei dati. Nel 1989, Gregory Piatetsky-Shapiro organizza e presiede il primo workshop *Knowledge Discovery in Databases (KDD)*, che ancora oggi, organizza conferenze annuali per promuovere la rapida crescita nel campo della conoscenza dei dati e del Data-Mining (Figura 1.2).

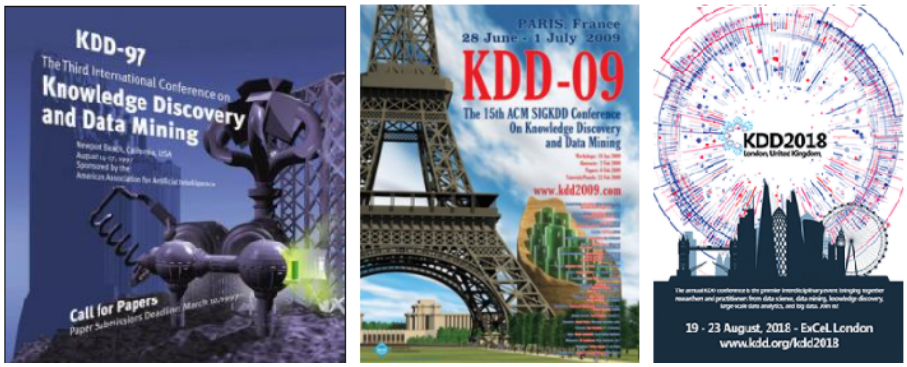


**Figura 1.2.** Alcuni poster della conferenza KDD

Finalmente si porta a compimento l'inizio dell'era dei Big Data, quando, nel 1994, *BusinessWeek* pubblica una storia di copertina di *"Database Marketing"* svelando la "minacciosa" notizia che le aziende stavano iniziando a raccogliere grandi quantità di informazioni personali, con il piano di iniziare nuove campagne di marketing.

Due anni dopo, nel 1996, il termine "Data Science" ha avuto la sua prima apparizione nel titolo della conferenza organizzata dalla *International Federation of Classification Societies (IFCS)* tenuta in Giappone (Figura 1.3).

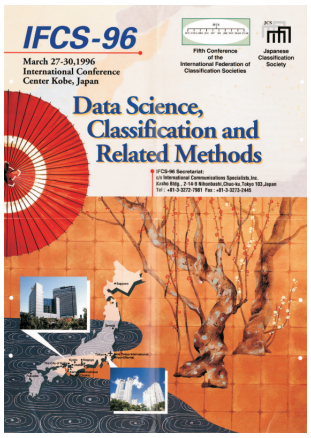


**Figura 1.3.** Poster della quinta IFCS conferenza a Kobe (1996)

Da allora "Data Science" è diventato un termine comunemente usato e un campo in continua crescita.

### 1.1.2 L'importanza

La Data Science mostra i trend e produce "insight" che le aziende possono utilizzare per prendere decisioni più mirate e creare prodotti e servizi più originali. I

dati costituiscono, infatti, la base dell'innovazione, ma il loro valore deriva dalle informazioni che i data scientist possono ottenere e in base alle quali possono agire. Questa nuova Figura professionale, dunque, applica i propri giudizi qualitativi per determinare la rilevanza e il significato delle informazioni prodotte. Tale operazione richiede molta abilità e sforzo, ed è comunemente il passo che i data scientist e gli uomini d'affari trovano più impegnativo. Per questa ragione, un importante aspetto della Data Science è il cosiddetto "last mile problem", il problema dell'ultimo miglio, di cui viene mostrata una descrizione nella Figura 1.4.

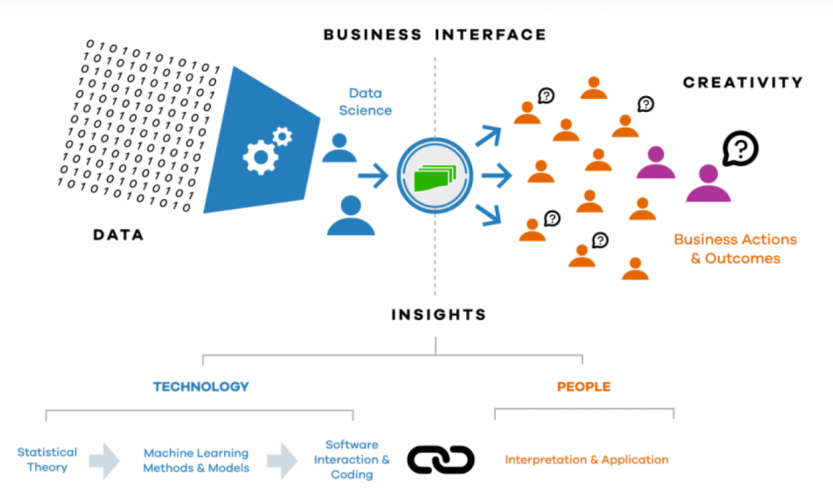


**Figura 1.4.** La Data Science ha un problema di "last mile"

Vediamo quali possono essere i vantaggi per un'azienda che decide di investire in questo campo.

- *Una migliore gestione degli asset*, ovvero delle risorse aziendali, non solo in termini economici. La Data Science e i suoi software di analisi ed elaborazione aiutano a monitorare in tempo reale i processi, a ridurre gli sprechi e a delineare scenari che collaborano a definire e attuare in modo efficace la strategia di impresa. Analizzare i dati significa valutare i rischi, studiare il ciclo di vita di beni e strumenti, calcolare preventivamente gli interventi di manutenzione, tracciare le scadenze.
- *Una manutenzione più efficace*: il monitoraggio in tempo reale di attrezzature e impianti consente di intervenire prima che si verifichi il guasto; tratta della cosiddetta *manutenzione predittiva* o *Condition Based Maintenance*. Tutte le azioni di manutenzione vengono tracciate e archiviate in automatico dai software di elaborazione e rese accessibili su un'unica piattaforma: dagli ordini di lavoro alle richieste di intervento fino alla rendicontazione dei costi e alla tracciatura dei fermi macchina.
- *L'ottimizzazione e il controllo di produzione*: dall'ingresso delle materie prime al prodotto finito, tutti i processi possono essere tracciati, monitorati e verificati.

- *L'aumento della sicurezza negli impianti*: sono un esempio tutti i sistemi che si basano su software che elaborano in tempo reale i dati provenienti da sensori per aiutare a prendere la migliore decisione possibile in relazione al contesto. Fondati sui dati sono la Cybersecurity e i relativi Cybersecurity Management System, ovvero i sistemi integrati di gestione della sicurezza informatica.
- *Il miglioramento dell'esperienza d'acquisto*, attraverso il monitoraggio del comportamento del cliente, che consente di implementare la personalizzazione dell'offerta. Videocamere negli store, reporting dei POS e analisi delle recensioni forniscono dati utili per capire meglio le tendenze di vendita, la percezione dell'offerta, nonché i margini di profitto attesi e realizzati.

## 1.2 La gerarchia dei bisogni

Molte aziende come Google, Amazon, Facebook, grazie alla guida dei dati, sono riuscite a crescere e ad evolvere nel tempo. Sono passate attraverso varie fasi di evoluzione che, fondamentalmente, si basano su una gerarchia di esigenze (Figura 1.5).

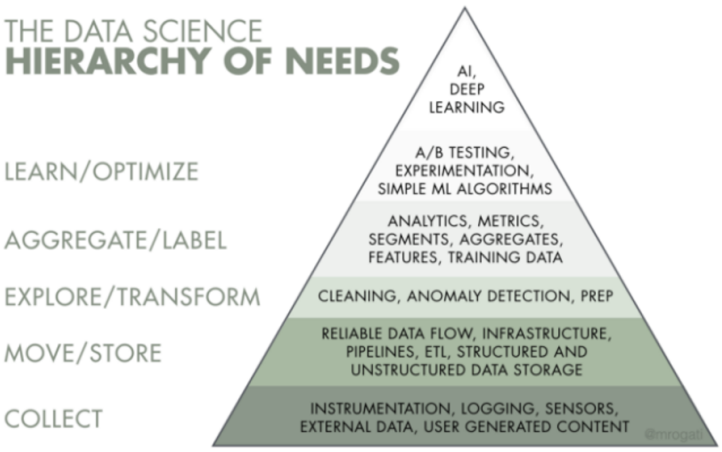


**Figura 1.5.** La gerarchia dei bisogni della Data Science

Questa rappresentazione si ispira alla gerarchia dei bisogni di Maslow dove vengono illustrati i diversi stadi che l'uomo deve attraversare per trovare la felicità; in essa, il concetto fondamentale da tener presente è che bisogna passare attraverso tutte le fasi, senza saltarne nessuna. Infatti, per realizzare correttamente ciascun passo indicato, è necessario utilizzare l'output del passo precedente, e, saltando troppo in fretta ad una fase tardiva, la piramide non è solida e ha un'alta possibilità di fallimento. Analizziamo più da vicino la gerarchia dei bisogni della Data Science.

### 1.2.1 La Raccolta

Alla base della piramide abbiamo la raccolta dei dati. All'inizio questi riguardavano soltanto la registrazione delle transazioni, degli errori e la digitalizzazione dei dati analogici. Successivamente, con l'evoluzione dell'azienda, si è giunti a forme più avanzate di raccolta dati. L'impresa può iniziare a raccogliere dati telemetrici dalle applicazioni, eseguire esperimenti per creare nuovi dati e acquisirli da fonti esterne. I dati sono strutturati e memorizzati in file o tabelle di database con formati particolari: XML, JSON o semplicemente, XLS o CSV, e molte applicazioni permettono di leggerli e gestirli.

### 1.2.2 L'Organizzazione

Il passo successivo riguarda l'organizzazione dei dati, la pulizia e l'archiviazione. È necessaria una fase chiamata ETL (Extract, Transform, Load) che rappresenta circa l'80% del lavoro del Data Scientist. Innanzitutto vengono estratti i dati che risultano più utili all'analisi; questi vengono, successivamente, trasformati da un formato ad un altro. Tale trasformazione è importante perché le sorgenti di partenza possono essere tra loro eterogenee, ma, al momento della memorizzazione nel database finale, tutti i dati devono avere necessariamente un unico formato.

### 1.2.3 L'Analisi

In terzo luogo, abbiamo bisogno di analizzare i nostri dati, ovvero spiegare cosa accade nella nostra organizzazione e perché sta accadendo. Questo, generalmente, inizia con strumenti di analisi dei dati di base, come report, cruscotti e KPI, per poi passare a tecniche più avanzate come quelle di Data Mining. Per Data Mining si intende l'individuazione di informazioni di varia natura (non risapute a priori) tramite estrapolazione mirata da grandi banche dati, singole o multiple. I suoi task si suddividono in due categorie: il *mining descrittivo*, che equivale alla Data Analytics descrittiva e diagnostica, e il *mining predittivo*, che equivale alla Data Analytics predittiva e prescrittiva. Tra le tecniche di Data Mining più utilizzate troviamo:

- *la classificazione*: che ci permette, definite delle classi di interesse, di trovare i profili degli elementi che attribuiamo ad esse;
- *il clustering*: che riunisce oggetti in gruppi omogenei, cioè con caratteristiche simili, senza conoscere le classi di interesse;
- *l'anomaly detection*: che consente di identificare elementi o eventi inaspettati nei set di dati che differiscono dalla norma.

### 1.2.4 La Previsione

Come quarto step, abbiamo bisogno di fare previsioni. Vogliamo sapere cosa accadrà probabilmente in futuro e come dovremmo rispondere a questi potenziali scenari se dovessero verificarsi. Questo generalmente coinvolge l'analitica predittiva, l'analisi prescrittiva e il Machine Learning. In sostanza, il set di dati viene suddiviso in una parte, ad esempio due terzi, per fare *training*, ovvero per addestrare il modello in

modo tale che questo diventi adatto per rappresentare un certo scenario. I restanti dati vengono utilizzati per fare *testing*; cioè, i dati passati non utilizzati per il training permettono di verificare se quel modello che abbiamo costruito è o meno accurato.

### 1.2.5 L'Automatizzazione

Infine abbiamo l'automatizzazione dei processi di Data Science. Quindi, possiamo pensare l'IA (Intelligenza Artificiale) e il Deep Learning alla cima di questa piramide. Quando parliamo di Intelligenza Artificiale ci riferiamo all'abilità di una macchina di imitare in qualche modo il comportamento umano; se applicata correttamente, ovvero guidata dai dati, può minimizzare i costi delle aziende e massimizzare le entrate. Quando parliamo, invece, di Deep Learning ci riferiamo ad un sottoinsieme del Machine Learning che consente ai computer di risolvere problemi più complessi.

## 1.3 L'ecosistema Python

Python (Figura 1.6) è il linguaggio di programmazione in maggior crescita al mondo ed è usato in tantissimi ambiti: dall'Intelligenza Artificiale, all'analisi dei dati, dall'utilizzo di siti web ai videogiochi, dal Machine Learning all'automazione. È utilizzato con successo in tutto il mondo da svariate aziende e organizzazioni, tra le quali Google, la NASA, YouTube, Spotify Ltd, Dropbox.


**Figura 1.6.** Il logo di Python

A partire dal 2010 circa, ha iniziato una traiettoria di crescita che gli ha permesso di competere con altri linguaggi di programmazione di punta, come Java e JavaScript e, per la prima volta in 20 anni, Python ha superato Java in termini di popolarità nel *TIOBE Programming Community Index*, un noto indicatore dedicato alla popolarità dei linguaggi di programmazione che viene riassunto attraverso una classifica le cui posizioni sono definite in base alle query effettuate dagli sviluppatori sui motori di ricerca (Figura 1.7).

Come sottolineato dai ricercatori, C continuerebbe ad essere il linguaggio di programmazione più popolare in assoluto, con un rating del 16.21% e, addirittura, una crescita dello 0.17% nell'edizione di novembre 2020. Python sarebbe, ora, al secondo posto con un indice di popolarità pari a 12.12 punti percentuali, grazie ad un incremento del 2.2%. Terza posizione per Java, in calo di popolarità con l'11.68% e una flessione pari al 4.57%.

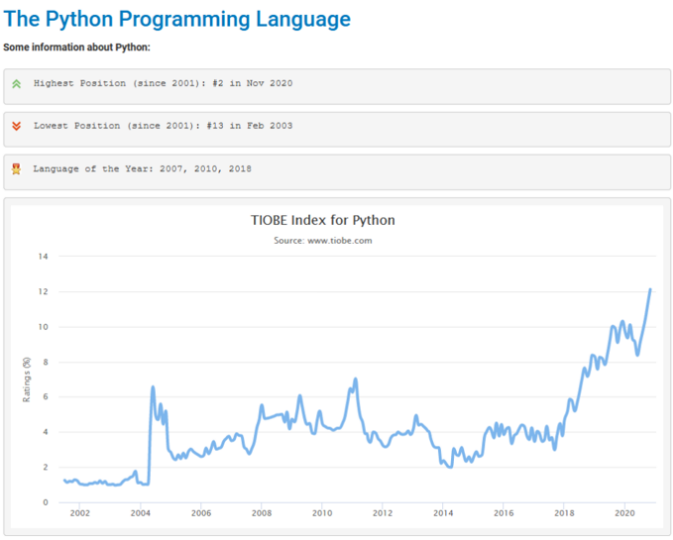


**Figura 1.7.** Il TIOBE Programming Community Index

### 1.3.1 La storia

Python nasce nel dicembre del 1989 per opera dell'informatico olandese Guido van Rossum (Figura 1.8). Dopo aver lavorato per quattro anni (dal 1982 al 1986) allo sviluppo del linguaggio di programmazione ABC, presso il *Centrum voor Wiskunde en Informatica (CWI)* di Amsterdam, dal 1986 Guido inizia a collaborare allo sviluppo di Amoeba, un sistema operativo distribuito, nato anch'esso ad Amsterdam (1981) alla *Vrije Universiteit*.

**Figura 1.8.** Guido van Rossum

Alla fine degli anni Ottanta il gruppo si rende conto che Amoeba necessita di un linguaggio di scripting, cosicché Guido, mentre si trova a casa per le vacanze di Natale del 1989, un po' per hobby e un po' per contribuire allo sviluppo di Amoeba, decide di avviare un suo progetto personale. Cerca di fare mente locale su quanto ha appreso durante il periodo di lavoro su ABC. Quell'esperienza è stata piuttosto frustrante, ma alcune caratteristiche di ABC gli piacciono, tanto da pensare di usarle

come fondamenti del suo nuovo linguaggio: l'indentazione per indicare i blocchi di istruzioni annidate, nessuna dichiarazione delle variabili, stringhe e liste di lunghezza arbitraria.

Su queste basi inizia a scrivere in C, un interprete per il suo futuro linguaggio di programmazione, che battezza con il nome di *Python* in onore della sua serie televisiva preferita: *Monty Python's Flying Circus*. Il 6 marzo 2001 viene fondata la *Python Software Foundation (PSF)*, un'organizzazione not-for-profit che detiene il diritto d'autore su Python e ne promuove la diffusione. La PSF è presieduta da Guido van Rossum e annovera tra i suoi membri il cuore degli sviluppatori di Python, più tante altre personalità nominate in virtù del loro notevole contributo al linguaggio. Nella Figura 1.9 sono riportate diverse versioni di Python che si sono succedute negli anni, più precisamente, dal 1991 al 2020.

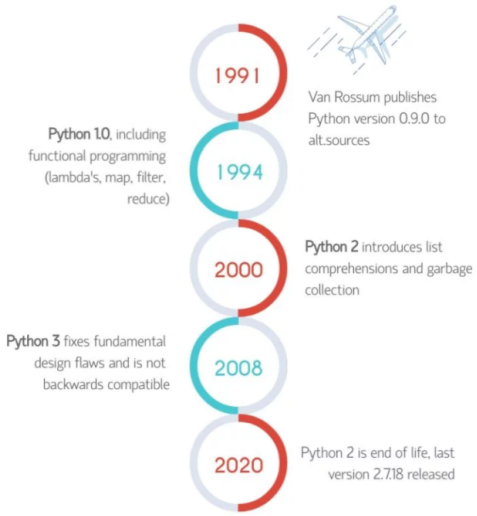


**Figura 1.9.** Diverse versioni di Python nel tempo

### 1.3.2 I punti di forza

Ci sono diverse ragioni per le quali si dovrebbe preferire Python rispetto ad altri linguaggi di programmazione. Vediamo quali sono i suoi punti di forza:

- *È completamente gratuito* ed è possibile usarlo e distribuirlo senza restrizioni di copyright. Si ha la possibilità di scegliere tra una vasta gamma di framework Python open source, librerie e strumenti di sviluppo secondo le proprie precise esigenze.
- *È un linguaggio multi-paradigma*, che supporta sia la programmazione procedurale (che fa uso delle funzioni), sia la programmazione ad oggetti (includendo funzionalità come l'ereditarietà singola e multipla, l'overloading degli operatori,

e il duck typing). Inoltre supporta, anche, diversi elementi della programmazione funzionale (come iteratori e generatori). Presenta, anche, un sistema di tipo dinamico e la gestione automatica della memoria, consentendo al programmatore di usare variabili liberamente, senza doversi preoccupare di dichiararle e di allocare e rilasciare spazi di memoria manualmente. I paradigmi di programmazione e le caratteristiche del linguaggio aiutano ad usare Python per sviluppare applicazioni software grandi e complesse.

- *È un linguaggio robusto, maturo e multi-piattaforma*, ovvero disponibile per tutti i principali sistemi operativi, ed è automaticamente incluso nelle distribuzioni Linux e nei computer Macintosh. Inoltre, fornisce tutti gli strumenti per scrivere in modo semplice programmi portabili, ovvero che si comportano alla stessa maniera se eseguiti su differenti piattaforme.
- *È un linguaggio di alto livello che è al tempo stesso semplice e potente*: la sintassi e i diversi moduli e funzioni che sono già inclusi nel linguaggio sono consistenti, intuitivi, e facili da imparare; il design del linguaggio si basa sul principio del *least astonishment* (cioè della "minor sorpresa": il comportamento del programma coincide con quanto ci si aspetta).
- *Ogni installazione di Python include la standard library*, cioè una collezione di oltre 200 moduli per svolgere i compiti più disparati, come, ad esempio, l'interazione con il sistema operativo e il file system, o la gestione di diversi protocolli. Inoltre, il Python Package Index consente di scaricare ed installare migliaia di moduli aggiuntivi creati e mantenuti dalla comunità.
- *È performante*: anche se Python è considerato un linguaggio interpretato, i programmi vengono automaticamente compilati in un formato chiamato *bytecode* prima ancora di essere eseguiti. Questo formato è più compatto ed efficiente, e garantisce, quindi, prestazione elevate. Inoltre, diverse strutture dati, funzioni, e moduli di Python sono implementati internamente in C per essere ancora più performanti.
- *È integrabile con altri linguaggi*: oltre all'interprete classico scritto in C (e chiamato CPython), esistono anche altri interpreti, che consentono l'integrazione con diversi altri linguaggi.

## 1.4 Python, uno strumento per la Data Science

Python, quindi, è un linguaggio semplice, chiaro e intuitivo e, proprio per questo, i data scientist amano Python. Forse preferiscono entrare rapidamente nel compito principale (ad esempio scoprire l'effetto o la correlazione di una variabile con un risultato), invece di spendere il proprio tempo ad imparare le sfumature di un linguaggio di programmazione "complesso". Python permette loro di entrare rapidamente nel progetto ottenendo, così, preziose intuizioni rapidamente grazie alle sue grandi funzionalità per gestire matematica, statistica e funzioni scientifiche. Python fornisce ottime librerie per gestire le applicazioni di Data Science.

### 1.4.1 Jupyter Notebook

Jupyter Notebook (Figura 1.10) è un ambiente di sviluppo e di supporto utile per lavorare nella Data Science tramite i linguaggi di programmazione R e Python, i più usati nella Data Science e nel Machine Learning. Jupyter offre la possibilità di realizzare, documentare e condividere analisi di dati all'interno di un framework che supporta operazioni di *data cleaning e trasformation*, simulazioni numeriche, modellazione statistica, Machine Learning, e l'esecuzione di applicazioni Scala e Python su piattaforme big data, grazie all'integrazione con Apache Spark.


**Figura 1.10.** Il logo di Jupyter

Jupyter Notebook è, in sostanza, un'applicazione Web open source che permette di catturare l'intero flusso di lavoro dell'analisi in un file, che può essere salvato, ripristinato e riutilizzato in seguito. Al contrario, quando si lavora con un editor di testo o un IDE, il codice, i corrispondenti file di dati e figure, sono sparsi su più file nel file system, e questo comporta uno sforzo significativo per mantenere il lavoro ben organizzato. Jupyter è, dunque, diventato uno standard de-facto per data scientist.

## 1.5 Le librerie di Python per la Data Science

Una delle più grandi risorse di Python è il suo ampio set di librerie, ovvero un gruppo di funzioni volte a svolgere compiti speciali rendendo più facile per gli sviluppatori eseguire compiti complessi senza riscrivere molte righe di codice. Esistono più di 137.000 librerie Python. Analizziamo quelle più importanti che riguardano il campo della Data Science.

### 1.5.1 NumPy

NumPy (Numerical Python) (Figura 1.11) è il fondamento su cui è costruita quasi tutta la potenza del toolkit di Python per la Data Science.


**Figura 1.11.** Il logo di Numpy

È il bundle di base per il calcolo numerico in Python e introduce oggetti per array multidimensionali e matrici, così da permettere agli sviluppatori di eseguire funzioni matematiche e statistiche avanzate su di essi con il minor codice possibile. I vantaggi che NumPy può portare al nostro codice sono i seguenti:

- *più velocità*: usa algoritmi scritti in C che completano in nanosecondi, piuttosto che in secondi;
- *meno loop*: evita che ci siano troppi indici di iterazione;
- *codice più chiaro*: senza loop, il codice sarà più simile alle equazioni che si vuole calcolare;
- *migliore qualità*: ci sono migliaia di collaboratori che lavorano per mantenere NumPy veloce, amichevole e senza bug.

### 1.5.2 Pandas

Pandas (Figura 1.12) mira ad essere l'elemento fondamentale di alto livello per eseguire analisi pratiche e reali dei dati in Python. È in uso in un'ampia varietà di domini accademici e commerciali, tra cui finanza, neuroscienze, economia, statistica, pubblicità, analisi web, e tanto altro ancora. Inoltre, ha l'obiettivo più ampio di diventare lo strumento di analisi / manipolazione dei dati open source più potente e flessibile disponibile in qualsiasi lingua.


**Figura 1.12.** Il logo di Pandas

Le sue principali caratteristiche sono le seguenti:

- Un oggetto DataFrame veloce ed efficiente per la manipolazione dei dati con indicizzazione integrata.
- Strumenti per leggere e scrivere dati tra strutture in memoria e diversi formati: file CSV e di testo, Microsoft Excel, database SQL, e il formato veloce HDF5.
- Allineamento intelligente dei dati e gestione integrata dei dati mancanti: è possibile ottenere l'allineamento automatico basato sull'etichetta nei calcoli e la facile manipolazione dei dati disordinati in una forma ordinata.
- Rimodellamento flessibile e rotazione dei set di dati.
- Le colonne possono essere inserite e cancellate dalle strutture dati per la mutabilità delle dimensioni.
- Aggregazione o trasformazione dei dati con un potente motore di raggruppamento che consente operazioni di divisione-applicazione-combinazione su set di dati;
- Fusione e unione ad alte prestazioni di set di dati.
- L'indicizzazione gerarchica degli assi fornisce un modo intuitivo di lavorare con dati ad alta dimensione in una struttura di dati di dimensione inferiore.

- Funzionalità delle serie temporali: generazione di un range temporale e conversione della frequenza, statistiche della finestra mobile, e spostamento o ritardo delle date.

### 1.5.3 Matplotlib

Matplotlib (Figura 1.13) è una libreria grafica che permette di visualizzare i nostri dati in modo gradevole. Fornisce un'interfaccia simile a MATLAB; l'unica differenza è che usa Python ed è open source. Matplotlib è una libreria standard utilizzata per la creazione di diagrammi e grafici 3D. È piuttosto di basso livello, il che significa che richiede più comandi per generare grafici e figure piacevoli rispetto ad alcune librerie avanzate. Tuttavia, il rovescio della medaglia è la flessibilità. Con un numero sufficiente di comandi, è possibile creare qualsiasi tipo di grafico che si desidera: da istogrammi e grafici a dispersione a grafici con coordinate non cartesiane.


**Figura 1.13.** Il logo di Matplotlib

I concetti base di Matplotlib sono i seguenti:

- *Figure*: è una Figura che contiene uno o più *axes* o *plots*.
- *Axes*: sono gli assi che contengono le figure e che possono contenere due o tre strutture o oggetti tridimensionali. Ogni asse ha un titolo, e le etichette *x-label* e *y-label*.
- *Axis*: si occupa di definire i limiti del grafico.
- *Artists*: sono per lo più legati agli assi, e a qualsiasi cosa si veda sulla Figura , come oggetti testo, oggetti line2d, oggetti collezione.

### 1.5.4 Seaborn

Seaborn (Figura 1.14) è una libreria per creare grafici statistici in Python. È costruita su Matplotlib ed è strettamente integrata con le strutture dati della libreria Pandas. Mira a rendere la visualizzazione una parte centrale dell'esplorazione e della comprensione dei dati. Le sue funzioni di plottaggio orientate ai dati operano su dataframe e array contenenti interi set di dati ed eseguono internamente la mappatura semantica e l'aggregazione statistica necessarie per produrre grafici informativi. Una delle cose che rende eccezionale il pacchetto è l'ampio spettro di grafici disponibili per analizzare le relazioni tra più variabili. Altre caratteristiche interessanti sono i grafici bivariati e le griglie multiplot, per non parlare della varietà di stili di sfondo, palette di colori e altre funzionalità di personalizzazione, che migliorano la comprensione dei dati e rendono le proprie intuizioni di impatto.


**Figura 1.14.** Il logo di Seaborn

### 1.5.5 Scikit-learn

Scikit-learn (Figura 1.15) fornisce una gamma di algoritmi di apprendimento supervisionato e non supervisionato tramite un'interfaccia coerente in Python. È, così, possibile risolvere facilmente algoritmi di regressione, di classificazione (*Naive Bayes, LinearSVC, K-Neighbors Classifier*) e clustering (*K-means, dbscan*). Anche attività quali la trasformazione di dati, la selezione delle funzionalità e i metodi di ensemble possono essere implementate in poche righe. Scikit-learn è basato su SciPy, che gestisce la parte numerica, insieme a NumPy. I moduli o le estensioni per SciPy sono convenzionalmente chiamati *SciKit*.


**Figura 1.15.** Il logo di Scikit-learn

### 1.5.6 Statsmodels

Statsmodel (Figura 1.16) è un modulo Python che fornisce classi e funzioni per la stima di molti modelli statistici diversi, test statistici ed esplorazione dei dati.


**Figura 1.16.** Il logo di Statsmodels

In particolare, fornisce un potente strumento per lavorare con le serie temporali, ovvero una sequenza di osservazioni, relative ad una sola variabile, registrate ad intervalli di tempo regolari. Lo studio di una serie temporale può riguardare svariati ambiti, tra cui quello medico, economico e dell'automazione. Esso si suddivide in due fasi:

- *L'analisi*, che ha lo scopo di cercare e comprendere le caratteristiche della serie, ad esempio se presenta trend, stagionalità, se è possibile visualizzarla attraverso una decomposizione stagionale additiva o moltiplicativa.

- *La predizione*, dove, sulla base delle analisi, viene creato un modello con lo scopo di prevedere i valori futuri della serie temporale. Per fare predizione, la serie deve essere stazionaria e, grazie a Statsmodels, vi sono diversi metodi per verificarlo, come l'*ADF test*, il *KPSS test* e il *PP test*. Se la serie non è stazionaria è possibile renderla tale sempre grazie a funzioni di questa libreria. Tra i principali modelli che è possibile applicare ricordiamo: ARMA, ARIMA, SARIMA, SARIMAX, che lavorano su serie temporali univariate.

# 2

# Descrizione del dataset utilizzato per la campagna

*In questo capitolo introdurremo il concetto generale di dataset, soffermandoci in particolare sulle sue tipologie e caratteristiche. Successivamente presenteremo nel dettaglio il dataset preso in esame per la nostra analisi, che ha come protagonista un innovativo dispositivo di una nota azienda marchigiana.*

## 2.1 Cos'è un dataset

Un dataset è una collezione di dati raccolti per uno scopo specifico. Tali dati possono essere suddivisi in tre grandi categorie: *strutturati*, *semi-strutturati* e *non strutturati*.

- *I dati strutturati* sono organizzati secondo uno schema predefinito e, tipicamente, organizzati in un formato tabellare, nel quale le righe corrispondono a ciascuna osservazione del fenomeno analizzato e le colonne corrispondono alle caratteristiche osservate, dette *features*. I dati strutturati hanno una struttura molto precisa, sono facili da elaborare, ma sono rigidi, perché obbligano a seguire dei modelli che il mondo attuale (che è molto più variegato) non accetta.
- *I dati semi-strutturati*, invece, sono organizzati secondo logiche strutturate e interoperabili, nonostante non non vi siano limiti strutturali al loro inserimento. Questa tipologia di dati rispetta, infatti, alcune delle caratteristiche dei dati strutturati e alcune delle caratteristiche dei dati non strutturati. I file XML, HTML o JSON appartengono a questa categoria, ma anche gli stessi dati prodotti dai sensori sono semi-strutturati, perché, generalmente, hanno una certa struttura, ma anche una certa flessibilità.
- *I dati non strutturati*, infine, sono conservati senza alcuno schema e, per loro natura, non possono essere organizzati come quelli strutturati. I file di testo narrativo prodotti con software di editing testuale oppure file multimediali (ad esempio audio e video) appartengono a questa categoria. Fino a 15 anni fa il 90% dei dati era strutturato; con l'avvento di Internet si stima che l'80/90% dei dati attuali è non strutturato. Questi dati non strutturati sono molto più difficili da elaborare, però sono preziosi per attività quali la *sentiment analysis*, nella quale si vuole capire qual è il *sentiment* di una persona o un gruppo di persone su determinati argomenti.

### 2.1.1 Le tipologie di dataset

In base alla tipologia di dati a disposizione possiamo distinguere tre tipi di dataset: *Record Data, Graph-based Data* e *Ordered Data*.

- I *Record Data* sono un insieme di *record*, ovvero un raggruppamento di campi memorizzati in una tabella, che definisce quali tipi di dati può contenere ciascun oggetto. Questi dati sono spesso memorizzati in "flat file" o in "database relazionali". La forma più elementare di Record Data non ha relazioni esplicite tra i record o i campi di dati, e ogni oggetto ha lo stesso insieme di attributi. Possiamo distinguere tre tipologie di Record Data:
  - *Market Basket Data* (Figura 2.1), che rappresentano una tipologia di dati utilizzati in un contesto commerciale. Questi dati sono organizzati in strutture, che contengono una collezione di articoli acquistati dal cliente in un determinato momento. L'insieme di questi articoli rappresenta i "dati di transizione", che possono essere visti come un insieme di record i cui campi sono attributi asimmetrici.

| TID | ITEMS |
|---|---|
| 1 | Bread, Soda, Milk |
| 2 | Beer, Bread |
| 3 | Beer, Soda, Diaper, Milk |
| 4 | Beer, Bread, Diaper, Milk |
| 5 | Soda, Diaper, Milk |

**Figura 2.1.** Un esempio di dati di transizione

Queste particolari informazioni possono essere molto utili ai grandi rivenditori per comprendere meglio il comportamento di acquisto dei prodotti. In particolare, attraverso la cosiddetta *Market Basket Analysis*, è possibile scoprire associazioni tra coppie di prodotti acquistati insieme, identificando modelli di co-occorrenza.

  - *Data Matrix* (Figura 2.2), che rappresentano collezioni di record i cui campi possiedono tutti lo stesso set di attributi. Ciascun record può essere pensato come un vettore in uno spazio multidimensionale; infatti, questo insieme di dati può essere rappresentato attraverso una matrice $m \times n$, dove le $m$ righe individuano ciascun record e le $n$ colonne rappresentano ciascun attributo che descrive il rispettivo oggetto. Per manipolare e trasformare le informazioni contenute in questo particolare dataset basta applicare le operazioni standard delle matrici.

– *Sparse Data Matrix* (Figura 2.3), a volte anche chiamati *document-data matrix*, sono dei particolari Data Matrix, ma che contengono anche i cosiddetti "dati sparsi", che indicano la presenza di lacune nei dati registrati e il cui valore è pari a zero.

| Projection of x Load | Projection of y Load | Distance | Load | Thickness |
|---|---|---|---|---|
| 10.23 | 5.27 | 15.22 | 27 | 1.2 |
| 12.65 | 6.25 | 16.22 | 22 | 1.1 |
| 13.54 | 7.23 | 17.34 | 23 | 1.2 |
| 14.27 | 8.43 | 18.45 | 25 | 0.9 |

**Figura 2.2.** Un esempio di Data Matrix

| | team | coach | play | ball | score | game | win | lost | timeout | season |
|---|---|---|---|---|---|---|---|---|---|---|
| Document 1 | 3 | 0 | 5 | 0 | 2 | 6 | 0 | 2 | 0 | 2 |
| Document 2 | 0 | 7 | 0 | 2 | 1 | 0 | 0 | 3 | 0 | 0 |
| Document 3 | 0 | 1 | 0 | 0 | 1 | 2 | 2 | 0 | 3 | 0 |

**Figura 2.3.** Un esempio di Sparse Data Matrix

- I *Graph-based Data* sono dataset i cui dati sono organizzati in grafi, una struttura relazionale formata da un numero finito di vertici (o nodi) e un numero finito di segmenti (archi o spigoli) che collegano ogni nodo agli altri. Sia i nodi che gli archi possono essere di tipo diverso e possono avere associati delle etichette o dei pesi. Un esempio tipico di grafo complesso/avanzato è la rete (Figura 2.4).
- Gli *Ordered Data*, infine, sono una raccolta di dati i cui attributi hanno relazioni che implicano un ordine nel tempo e nello spazio. In questa tipologia di dataset possiamo individuare quattro categorie di dati differenti:
  – *Dati sequenziali* (Figura 2.5), chiamati anche *dati temporali*, che possono essere visti come una estensione dei dati memorizzati in record, dove ciascuno di essi ha un tempo associato.
  – *Dati di sequenza* (Figura 2.6), un insieme di dati che può essere visto come una sequenza ordinata di entità individuali, come una sequenza di parole o lettere; essi sono simili ai tipi di dati ma, a differenza di essi, non hanno associati timestamp.
  – *Dati di serie temporali* (Figura 2.7), una tipologia di dati sequenziali in cui ogni record è una serie temporale, cioè una serie di misure raccolte nel tempo.
  – *Dati spaziali* (Figura 2.8), un insieme di oggetti che possiede attributi spaziali, come posizioni o aree, così come altri tipi di attributi.

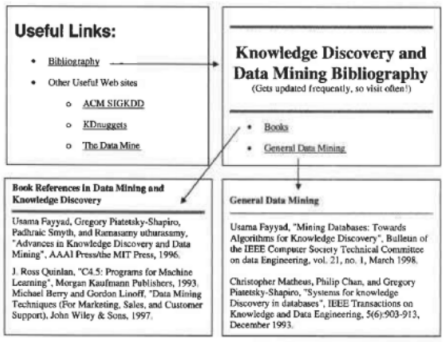


**Figura 2.4.** Un esempio di pagine Web collegate tramite "Links"

| Time | Customer | Items Purchased |
|---|---|---|
| t1 | C1 | A, B |
| t2 | C3 | A, C |
| t2 | C1 | C, D |
| t3 | C2 | A, D |
| t4 | C2 | E |
| t5 | C1 | A, E |

| Customer | Time and Items Purchased |
|---|---|
| C1 | (t1: A,B) (t2:C,D) (t5:A,E) |
| C2 | (t3: A, D) (t4: E) |
| C3 | (t2: A, C) |

**Figura 2.5.** Un esempio di dati sequenziali

```
GGTTCCGCCTTCAGCCCCGCGCC
CGCAGGGCCCGCCCCGCGCCGTC
GAGAAGGGCCCGCCTGGCGGGCG
GGGGGAGGCGGGGCCGCCCGAGC
CCAACCGAGTCCGACCAGGTGCC
CCCTCTGCTCGGCCTAGACCTGA
GCTCATTAGGCGGCAGCGGACAG
GCCAAGTAGAACACGCGAAGCGC
TGGGCTGCCTGCTGCGACCAGGG
```

**Figura 2.6.** Un esempio di dati di sequenza

### 2.1.2 Le caratteristiche di un dataset

Possiamo individuare tre caratteristiche generali per descrivere un dateset, ovvero *Dimensionality, Sparsity* e *Resolution*.

Per *Dimensionality* si intende la "dimensionalità" di un insieme di dati, ovvero il numero di attributi/caratteristiche presenti in un dataset. Se il numero di attributi

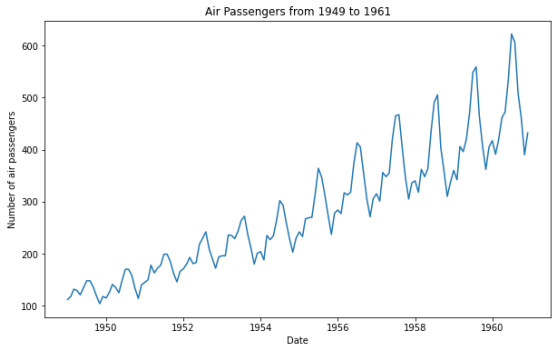


**Figura 2.7.** Un esempio di dati di serie temporali

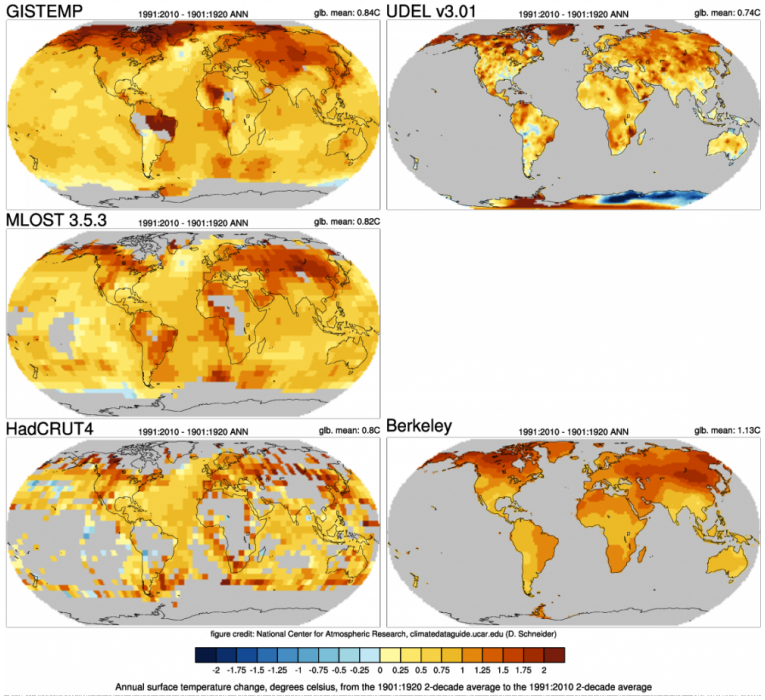


**Figura 2.8.** Un esempio di dati spaziali

è molto alto, generalmente dell'ordine del centinaio o più, si incorre in quello che viene chiamato *Curse of Dimensionality*. Con questo termine si intende un insieme di problemi che provocano difficoltà durante l'analisi, la visualizzazione dei dati per identificare i modelli di Machine Learning o il training di questi modelli. Per ridurre la dimensionalità dei dati è possibile adottare delle tecniche. Una tecnica molto comune prende il nome di *Principal Component Analysis (PCA)*, che prevede l'estrazione delle caratteristiche combinando le variabili di input in un modo specifico ed eliminando le variabili "meno importanti", pur conservando le loro informazioni

più preziose.

Il *Curse of Dimensionality* è strettamente legato alla "Sparsity". Quest'ultiuma è una caratteristica che si riferisce ai "dati sparsi". Se tali dati vengono ben gestiti e pianificati, possono rendere un'intera azienda o organizzazione molto più efficiente. Allo stesso tempo, però, all'aumentare della dimensionalità, i dati diventano sempre più sparsi nello spazio che occupano, causando problemi di classificazione e clustering. Infatti, l'aumento di questi dati porta a conseguenze:

- *per la classificazione*, perchè diventa impossibile creare un modello che assegna in modo affidabile una classe a tutti gli oggetti possibili;
- *per il clustering*, perchè le definizioni di densità e di distanza tra i punti diventano meno significative, quando dovrebbero essere fondamentali.

La *Resolution* indica il "livello di risoluzione" dei dati; e da questo dipendono i rispettivi modelli. Se la risoluzione è troppo "fine", un modello può non essere visibile o può essere coperto dal rumore; se la risoluzione è troppo "grossolana", il modello può scomparire. Questo aspetto non è da sottovalutare; basti pensare ad una risoluzione di tipo temporale: alcuni fenomeni sono visibili su una scala di ore, ma diventano irrilevabili su una scala di mesi.

## 2.2 Origine del dataset utilizzato per la nostra campagna sperimentale

La nostra campagna sperimentale è stata realizzata su un dataset dell'azienda Elica (Figura 2.9), un'impresa marchigiana attiva principalmente nel mercato delle cappe da cucina. Essa viene fondata da Ermanno Casoli nel 1970 a Fabriano e oggi è presieduta dal figlio, Francesco Casoli.

**Figura 2.9.** Logo dell'azienda Elica

La sua piattaforma produttiva è articolata su cinque Paesi tra Italia, Polonia, Messico, India e Cina, e la rende leader assoluta al mondo nel settore delle cappe e tra i leader in Europa nella progettazione, produzione e commercializzazione di motori elettrici per cappe e caldaie da riscaldamento. Infatti, Elica Corporation possiede due "business unit":

- *Cooking BU*: progetta, produce e commercializza cappe da cucina a uso domestico, sia a marchio proprio sia attraverso i brand dei principali produttori internazionali di elettrodomestici e cucine (Whirlpool, Electrolux, Ikea, Indesit Company, Bosch-Siemens, Haier, etc.), piani cottura e, per il mercato asiatico, forni e sterilizzatori.

- *Motors BU*: progetta, produce e commercializza motori elettrici per elettrodomestici, cappe e caldaie da riscaldamento a uso domestico, con il marchio FIME.

## 2.3 Cos'è lo Snap

*Snap "Air Quality Balancer"* (Figura 2.10) è un innovativo sistema di aspirazione che monitora e migliora la qualità dell'aria dell'ambiente in cui è installato, in maniera automatica, attraverso un intervento calibrato sulle esigenze e sulle caratteristiche dell'ambiente stesso. Snap è stato progettato per la casa e per i luoghi pubblici (scuola, pub, hotel, palestra) ed, in particolare, è raccomandato per:

- ogni piano, inclusi gli attici e i seminterrati ristrutturati;
- la cucina;
- il bagno;
- gli spazi living;
- la camera da letto;
- l'ufficio.


**Figura 2.10.** Il dispositivo Snap di Elica

Snap è dotato di 3 sensori che monitorano costantemente l'aria prendendo in esame qualità, temperatura e umidità. In modalità automatica, oppure in modalità manuale, Snap avvia l'aspirazione assicurando il ricambio dell'aria. Esso migliora la qualità dell'aria espellendo odori spiacevoli, agenti allergeni o inquinanti, vapore, batteri e virus derivanti dalle muffe. In più, questo dispositivo assicura lo stesso risultato che si otterrebbe aprendo le finestre, ma senza alcuno spreco di calore ed energia e senza che insetti o altri sgradevoli ospiti possano entrare, con la costante garanzia di un ridotto consumo energetico. Snap riesce a rigenerare l'aria di un ambiente di 25 mq in soli 30 minuti. Inoltre, questo dispositivo, risulta un ottimo supporto alle cappe *Elica Sense* compatibili, poiché, quando queste ultime non sono più in grado di filtrare fumo e odori, possono attivare lo Snap a cui sono associate. Le modalità di funzionamento che è possibile impostare sono sei: *standby, automatic, detox, dry, manual* e *link*.

Snap è dotato di una ventola dal design elegante e di un motore brushless di ultima generazione in grado di garantire consumi ridotti (di soli 7W a velocità di aspirazione massima); il tutto è nascosto da una cover estetica sulla quale è disposto un disco in vetro (Figura 2.11). Questa cover possiede un led centrale luminoso, che segnala la funzione attiva, e una corona luminosa, che segnala la modalità della funzione attiva e restituisce un feedback di stato.

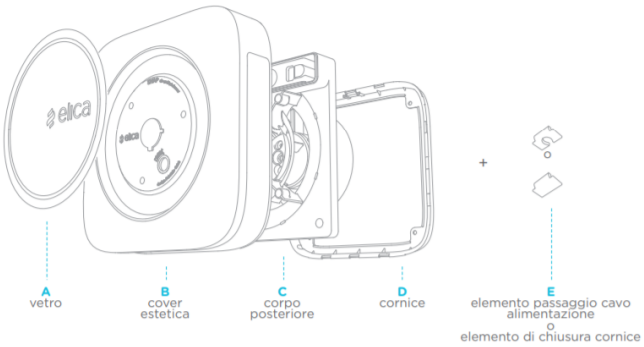


**Figura 2.11.** Componenti di Snap

Per poter funzionare, Snap deve comunicare con l'esterno degli edifici attraverso un foro del diametro di 120 mm e deve essere collegato alla rete elettrica (Figura 2.12).

Snap, dopo il primo collegamento alla rete elettrica, esegue delle operazioni di setting visibili tramite feedback luminosi del led centrale e della corona luminosa. Al termine, esso si prepara per le successive fasi di associazione. Snap può essere controllato da remoto attraverso l'utilizzo di un telecomando, oppure tramite la Snap App, che permette di personalizzare la configurazione e ottimizzare le performance del proprio dispositivo.

- Il telecomando, moderno ed ergonomico, permette di controllare Snap da qualsiasi angolo della stanza. Vetro e tasti touch gli conferiscono stile ed eleganza garantendo, al tempo stesso, una digitazione rapida e precisa.
- Snap App è semplice da utilizzare e ha un design elegante ed intuitivo che permette di controllare e gestire Snap in qualsiasi momento e ovunque ci si trovi. Snap App dà la possibilità di gestire più Snap contemporaneamente, per avere sempre sotto controllo qualità, umidità e temperatura dell'aria di tutti gli ambienti. Essa permette, inoltre, di attivare e gestire, secondo le proprie esigenze, le cinque funzioni principali del dispositivo. In più Snap App consente di gestire da remoto tutte le funzioni speciali dello Snap, ovvero creazione e gestione di gruppi di Snap, controllo delle funzioni della corona luminosa, accesso alla pagina delle statistiche per capire come è cambiata nel tempo la qualità dell'aria degli ambienti.

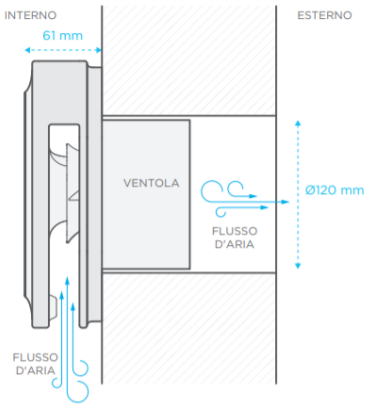


**Figura 2.12.** Preinstallazione di Snap

## 2.4 Struttura del dataset

Il dataset utilizzato per la nostra campagna sperimentale ha come protagonista Snap. Questo dataset appartiene alla categoria dei Record Data ed è una collezione di dati semi-strutturati organizzati in un formato tabellare; quindi, come illustrato nel paragrafo 2.1, i dati sono organizzati in righe e colonne. In particolare, il nostro dataset possiede 3.189.333 dati totali distribuiti su 19 campi che descrivono le caratteristiche del nostro dispositivo. Nella Figura 2.13 sono riportate le prime cinque righe del nostro dataset.

| | id | idSnap | payloadVersion | createdDate | currentMode | fanLevel | temperature | humidity | airQuality |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 71450444 | 1573.0 | 0 | 2020-01-25 23:24:42.0 | 1 | 0 | 23 | 43 | 1 |
| 1 | 71450445 | 2220.0 | 0 | 2020-01-25 23:22:19.0 | 1 | 0 | 26 | 35 | 1 |
| 2 | 71450447 | 1648.0 | 0 | 2020-01-25 23:26:22.0 | 1 | 0 | 21 | 43 | 1 |
| 3 | 71450451 | 1240.0 | 0 | 2020-01-25 23:24:10.0 | 0 | 0 | 19 | 47 | 1 |
| 4 | 71450454 | 2225.0 | 0 | 2020-01-25 23:24:46.0 | 0 | 0 | 21 | 36 | 1 |

| noise | lux | wifiLevel | errors | flags | ping | bytes03 | bytes47 | bytes811 | mode |
|---|---|---|---|---|---|---|---|---|---|
| 35 | 0 | 0 | 0 | 0 | 9138 | 0 | 0 | 0 | AUTOMATIC |
| 8 | 105 | 0 | 0 | 0 | 2131 | 0 | 0 | 1412956160 | AUTOMATIC |
| 40 | 0 | 0 | 0 | 0 | 10436 | 0 | 0 | 1145831424 | AUTOMATIC |
| 36 | 0 | 0 | 0 | 0 | 9363 | 0 | 0 | 0 | STANDBY |
| 34 | 113 | 0 | 0 | 0 | 8707 | 0 | 1999044608 | 1159987200 | STANDBY |

**Figura 2.13.** Il dataset della nostra campagna

### 2.4.1 id & idSnap

Come spiegato nella sezione 2.1.1, i Record Data sono un insieme di record ciascuno dei quali possiede un identificatore univoco, noto come *ID record*. Nel nostro dataset l'ID record viene chiamato *id*; sono presenti tanti id quante sono le righe della tabella, ovvero il numero di dati totali. Il secondo campo riportato nel nostro dataset è chiamato *idSnap*, e rappresenta l'identificatore univoco di ciascun dispositivo analizzato. Nel complesso sono presenti 1.311 Snap differenti.

### 2.4.2 PayloadVersion

In generale il Payload è inteso come "carico utile". Nel nostro dataset tutti i valori di questo campo sono posti a 0, purtroppo non ci sono abbastanza informazioni per poter definir al meglio questa caratteristica.

### 2.4.3 CreatedDate

Questo campo possiede 1.673.167 elementi differenti che indicano la data e l'ora in cui sono state effettuate delle rilevazioni da parte dei sensori di temperatura, umidità e qualità dell'aria di Snap. Osservando il dataset abbiamo potuto notare che le rilevazioni sono state effettuate dal 25 gennaio 2020 al 16 giugno 2020 e che, nella maggior parte dei casi, le rilevazioni sono state eseguite quotidianamente, nelle ore dispari, ogni dieci minuti.

### 2.4.4 CurrrentMode

Snap, dal momento in cui si aziona, si dispone continuamente in fase di ricezione. Esso impara, cioè, a conoscere l'ambiente attraverso rilevamenti continui effettuati dai suoi sensori (qualità, umidità e temperatura dell'aria). In questo modo è in grado di definire una "situazione ideale", che rappresenta il suo standard di riferimento. Il campo currentMode indica la "modalità corrente", ovvero quella di funzionamento con cui Snap sta operando. Vediamole più nel dettaglio:

- *Standby*: è una modalità con la quale si disattiva una qualsiasi funzione attiva; in questo caso, però, i sensori del dispositivo continuano a svolgere il proprio lavoro. Nel nostro dataset questa modalità è indicata con il valore 0, mentre il simbolo di Standby a cui si fa riferimento per il controllo del dispositivo è riportato nella Figura 2.14 ed è lo stesso simbolo utilizzato per l'accensione e lo spegnimento di Snap.

**Figura 2.14.** Il simbolo di Standby

- *Automatic*: è una modalità con la quale Snap si attiva in modo completamente automatico, e solo quando necessario, per ristabilire il comfort dell'ambiente in funzione della qualità dell'aria, della temperatura e dell'umidità. Se uno dei sensori rileva valori differenti rispetto a quanto precedentemente registrato ed identificato come "situazione ideale" il motore si attiva in modalità di aspirazione. Tale modalità riduce odori e sostanze inquinanti, limita l'innalzamento dell'umidità legato, per esempio, ai vapori prodotti in fase di cottura, monitora la temperatura e regola il flusso di aria in uscita, per evitare inutili sbalzi termini e conseguenti sprechi. Nel nostro dataset questa modalità è indicata con il valore 1, mentre il simbolo di Automatic a cui si fa riferimento per il controllo del dispositivo è riportato in Figura 2.15.


**Figura 2.15.** Il simbolo di Automatic

- *Detox*: è una modalità con la quale Snap si concentra sulla qualità dell'aria e, per questo, monitora e attiva l'aspirazione in caso di innalzamento del livello di inquinamento. Come per la modalità Automatic, se il sensore di qualità dell'aria rileva una variazione rispetto a quanto precedentemente registrato ed identificato come "situazione ideale", il motore si attiva in modalità di aspirazione. Questa funzione limita, quindi, la presenza nell'aria di odori, polvere, polline, acari, inquinanti chimici prodotti dal fumo di sigaretta, dall'utilizzo di detergenti per la casa o per l'igiene della persona. Nel nostro dataset essa è indicata con il valore 5, mentre il simbolo di Detox a cui si fa riferimento per il controllo del dispositivo è riportato in Figura 2.16.


**Figura 2.16.** Il simbolo di Detox

- *Dry*: è una modalità con la quale Snap si concentra sul livello di umidità presente nell'ambiente e, per questo, interviene quando i suoi sensori segnalano un incremento di umidità rispetto a quanto solitamente monitorato e registrato. Tale funzione permette di ridurre la formazione di condense derivanti, per esempio, da vapori di cottura o dal vapore acqueo prodotto da una doccia. Snap, in questo modo, limita il possibile sviluppo di muffe, salvaguardando la nostra salute e quella dell'ambiente. Nel nostro dataset questa modalità è indicata con

il valore 6, mentre il simbolo di Dry a cui si fa riferimento per il controllo del dispositivo è riportato in Figura 2.17.


**Figura 2.17.** Il simbolo di Dry

- *Manual*: è una modalità con la quale Snap si attiva al proprio comando, se si avverte la necessità di ventilare l'ambiente. Nel nostro dataset questa modalità è indicata con il valore 2, se la velocità impostata della ventola è la minima, e con il valore 3 se la velocità impostata della ventola è la massima. I simboli di Manual a cui si fa riferimento per il controllo del dispositivo sono riportati in Figura 2.18.

**Figura 2.18.** I simboli di Manual per la velocità min e max

- *Link*: è una modalità con la quale Snap si attiva in connessione alla cappa Elica. Quando necessario, sarà la cappa ad attivare il motore di Snap al fine di potenziare la capacità di rimozione degli odori e dei vapori prodotti in fase di cottura. Nel nostro dataset questa modalità è indicata con il valore 4, mentre il simbolo di Link a cui si fa riferimento per il controllo del dispositivo è riportato in Figura 2.19.


**Figura 2.19.** Il simbolo di Link

### 2.4.5 FanLevel

Questo campo indica la velocità corrente della ventola. Nel nostro dataset vengono individuati quattro valori differenti:

- 0: indica che la ventola non è in funzione;
- 1: indica che la ventola lavora alla velocità; minima
- 2: indica che la ventola lavora alla velocità; media
- 3: indica che la ventola lavora alla velocità massima.

### 2.4.6 Temperature

Il *Temperature Sensor* monitora la temperatura per garantire il miglior comfort termico evitando inutili sprechi. Il sensore può rilevare una temperatura dai -40°C ai 125°C; nel nostro dataset, il valore minimo rilevato è pari a -1°C, mentre il valore massimo è pari a 58°C.

### 2.4.7 Humidity

L'*Humidity Sensor* tiene sotto controllo l'umidità prodotta da vapori di cottura o da altre attività, riducendo l'insorgenza di batteri e muffe sensibili ad ambienti molto umidi. Snap non deumidifica l'aria ma provvede ad assicurare un ricambio completo della stessa quando i livelli di umidità si alzano. L'umidità relativa varia dallo 0%, a cui corrisponde un'aria perfettamente secca, che in natura non è mai raggiunta, al 100%, a cui corrisponde un'aria satura di vapore acqueo. Nel nostro dataset il valore minimo rilevato dal sensore è -1%, mentre il valore massimo è 99%.

### 2.4.8 AirQuality

L'*Air Quality Sensor* rileva la presenza di agenti inquinanti tra cui i VOCs, composti organici volatili provenienti da fumo, detergenti, prodotti spray, materiali di costruzione e fumi di cottura. Snap non ionizza l'aria né la purifica, ma interviene con un completo ricambio dell'aria degli ambienti in cui è installato. Nel nostro dataset, quando il sensore non è ancora pronto per lavorare, il valore di questo campo è 0. Una volta che il sensore è attivo, in base alla contaminazione dell'aria, avremo un valore intero che varia da 1 a 5, dove 1 indica che l'ambiente è ottimale, 5 indica che l'ambiente è molto contaminato.

### 2.4.9 Noise

Questo campo indica il rumore del dispositivo; dal datasheet è possibile conoscere il valore di potenza e pressione sonora in corrispondenza del valore minimo e massimo della velocità della ventola (Figura 2.20). Nel nostro dataset i valori variano da 0 dB a 120 dB.

| | |
|---|---|
| Noise Lw (dBA): | 47 (min speed) - 60 (max speed) |
| Noise Lp at 3 mt (dBA): | 27 (min speed) - 40 (max speed) |

**Figura 2.20.** Dati tecnici riguardo il rumore in Snap

### 2.4.10 Lux

Un'altra caratteristica di Snap è che, al momento dell'accensione e delle iniziali associazioni al telecomando, alla Snap App o alla cappa Elica, la corona luminosa si illumina in modo diverso. In particolare, in base alla modalità corrente di Snap, la corona luminosa del dispositivo si colora differentemente. Vediamo più nel dettaglio quali sono le segnalazioni luminose di Snap per ogni modalità corrente.

- *Standby*: la corona luminosa si spegne ed il led centrale si colora di bianco (Figura 2.21)


**Figura 2.21.** Logout della corona luminosa in modalità Standby

- *Automatic*: la corona luminosa si colora di un colore più o meno intenso comunicando quando Snap sta aspirando e a quale velocità (Figura 2.22)


**Figura 2.22.** Logout della corona luminosa in modalità Automatic

- *Detox*: la corona luminosa ci informa sul livello di inquinamento dell'aria (Figura 2.23)


**Figura 2.23.** Logout della corona luminosa in modalità Detox

- *Dry*: come per la modalità Automatic, la corona luminosa si colora di un colore più o meno intenso comunicando quando Snap sta aspirando e a quale velocità (Figura 2.24)
- *Manual*: la corona luminosa si colora in modo differente a seconda del comando impostato (Figura 2.25)


**Figura 2.24.** Logout della corona luminosa in modalità Dry


**Figura 2.25.** Logout della corona luminosa in modalità Manual

- *Link*: la corona luminosa si colora in modo differente comunicando quando Snap sta aspirando e a quale velocità (Figura 2.26)


**Figura 2.26.** Logout della corona luminosa in modalità Link

Nel nostro dataset i valori per questo campo variano da 0 lx a 127 lx.

### 2.4.11 WifiLevel

Snap si collega a Internet usando una rete Wi-Fi, come quella del router wireless di casa. Dal datasheet possiamo ricavare maggiori informazioni riguardo questo campo (Figura 2.27)

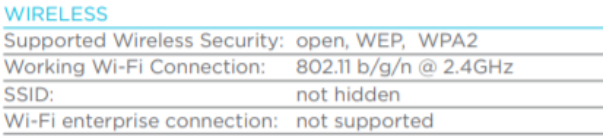


**Figura 2.27.** Dati tecnici riguardo alla connessione Wi-Fi di Snap

### 2.4.12 Errors

Questo dataset fornisce informazioni utili riguardo gli errori registrati sul nostro dispositivo; possiamo individuarne sei tipi di errore, espressi in valore decimale: 0, 2, 4, 9, 13 e 16. Questi valori possono essere codificati attraverso una bitmap illustrata nella seguente tabella (i valori 0 e 1 scritti in grassetto costituiscono la bitmap):

| bit 4 | bit 3 | bit 2 | bit 1 | bit 0 | errore decimale |
|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | 0 |
| **0** | **0** | **0** | **1** | **0** | 2 |
| **0** | **0** | **1** | **0** | **0** | 4 |
| **0** | **1** | **0** | **0** | **1** | 9 |
| **0** | **1** | **1** | **0** | **1** | 13 |
| **1** | **0** | **0** | **0** | **0** | 16 |

A ciascun bit corrisponde un errore di rilevazione:

- bit 0: il Temperature Sensor effettua rilevazioni fuori range (ERR_TEMP);
- bit 1: l'Air Quality Sensor effettua rilevazioni fuori range (ERR_AQ);
- bit 2: l'Humidity Sensor effettua rilevazioni fuori range (ERR_RH);
- bit 3: la comunicazione I2C con i sensori è fallita (ERR_I2C);
- bit 4: il tempo di connessione tra la cappa e Snap in modalità Link è scaduto (ERR_LINK).

Osservando la bitmap e le corrispondenze tra bit ed errore, possiamo definire i seguenti codici decimali:

| **errore decimale** | **tipo di errore** |
|---|---|
| **0** | Nessun Errore |
| **2** | ERR_AQ |
| **4** | ERR_RH |
| **9** | ERR_TEMP, ERR_I2C |
| **13** | ERR_TEMP, ERR_RH, ERR_I2C |
| **16** | ERR_LINK |

### 2.4.13 Flags

Questo campo ci fornisce due informazioni riguardo a Snap, ovvero se la modalità Link è attiva e se Snap lavora o meno in modalità *Relax Mode*, grazie alla quale è possibile personalizzare le sue funzioni Automatic, Dry e Detox assicurando

un'ottima performance in termini di aspirazione e silenziosità. Possiamo individuare quattro tipi di flag espressi in valore decimale: 0, 1, 8 e 9. Questi valori possono essere codificati attraverso una bitmap illustrata nella seguente tabella (i valori 0 e 1 scritti in grassetto costituiscono la bitmap):

| bit 3 | bit 2 | bit 1 | bit 0 | flag decimale |
|---|---|---|---|---|
| **0** | **0** | **0** | **0** | 0 |
| **0** | **0** | **0** | **1** | 1 |
| **1** | **0** | **0** | **0** | 8 |
| **1** | **0** | **0** | **1** | 9 |

A ciascun bit corrisponde l'attivazione o meno della modalità Link o Relax Mode. In particolare:

- bit 0
  - se Snap lavora in modalità standalone bit 0 = 0;
  - se Snap è associato ad una cappa e, quindi, è in modalità Link bit 0 = 1;
- bit 3
  - se Snap non lavora in modalità relax bit 3 = 0;
  - se Snap lavora in modalità relax bit 3 = 1;

Osservando la bitmap e le corrispondenze tra bit e le due modalità di funzionamento prese in considerazione, possiamo definire i seguenti codici decimali:

| **flag decimale** | **modalità attiva** |
|---|---|
| **0** | Nessuna |
| **1** | Link |
| **8** | Relax Mode |
| **9** | Link, Relax Mode |

### 2.4.14 Ping

I VOCs (Volatile Organic Compounds) sono una vasta gamma di sostanze chimiche (composti) a base di carbonio (organico) che si trovano in vari solidi e liquidi naturali e prodotti dall'uomo. Questi evaporano facilmente a temperatura ambiente ordinaria ed è per questo che sono definiti volatili. Il campo Ping restituisce il valore in decimale di queste sostanze rilevate dal sensore di Air Quality.

### 2.4.15 Bytes03, Bytes47 & Bytes 811

Non ci sono abbastanza informazioni per poter definire questi campi del dataset.

### 2.4.16 Mode

Questo è l'ultimo campo del nostro dataset e indica quale è la modalità corrente di Snap. Il campo Mode non è altro che la traduzione in stringa del campo Current Mode precedentemente analizzato.

# 3

# Progettazione della campagna

*In questo capitolo presenteremo le fasi di analisi svolte sul nostro dataset. Utilizzeremo Jupyter Notebook come ambiente di sviluppo e supporto al linguaggio di programmazione Python, mediante il quale verrà svolto il nostro studio. Il primo passo sarà l'attività di ETL, seguiranno l'analisi esplorativa e l'anomaly detection. Per comprendere meglio tutte le fasi del nostro progetto, verranno presentate, prima in generale, e poi in particolare, sui nostri dati.*

## 3.1 La fase di ETL

L'attività di ETL (Extract, Transform and Load), come già anticipato nella Sezione 1.2.2, è una fase fondamentale che precede l'analisi dei dati. I cosiddetti strumenti ETL permettono di integrare schemi eterogenei, nonché di estrarre, trasformare, ripulire, validare, filtrare e caricare i dati.

Le imprese, da diversi anni, si affidano al processo di ETL per ottenere una visione consolidata dei dati, con la quale, veicolano le migliori decisioni aziendali.

Questa fase è, dunque, un tipo di processo di integrazione dei dati composto da quattro processi distinti, ma interconnessi, ovvero *estrazione, pulitura, trasformazione* e *caricamento*, come mostrato più dettagliatamente nella Figura 3.1. L'attività di pulitura è spesso inscindibilmente allacciata e sovrapposta a quella di trasformazione; per questa ragione la integreremo in quest'ultima e le considereremo come un unico processo.

### 3.1.1 Estrazione

Durante questa prima fase i dati rilevanti vengono estratti dalle sorgenti. Le fonti di dati possono includere on-premise database, sistemi CRM, piattaforme di automazione del marketing, data warehouse, file strutturati e non strutturati, applicazioni cloud e qualsiasi altra fonte di dati da cui si desidera trarre spunti attraverso l'elaborazione analitica. Di solito i dati vengono consolidati da sistemi sorgente numerosi e disparati che possono memorizzare le informazioni in un formato diverso. Durante questa fase, quindi, i dati compilati devono essere organizzati in termini di data,

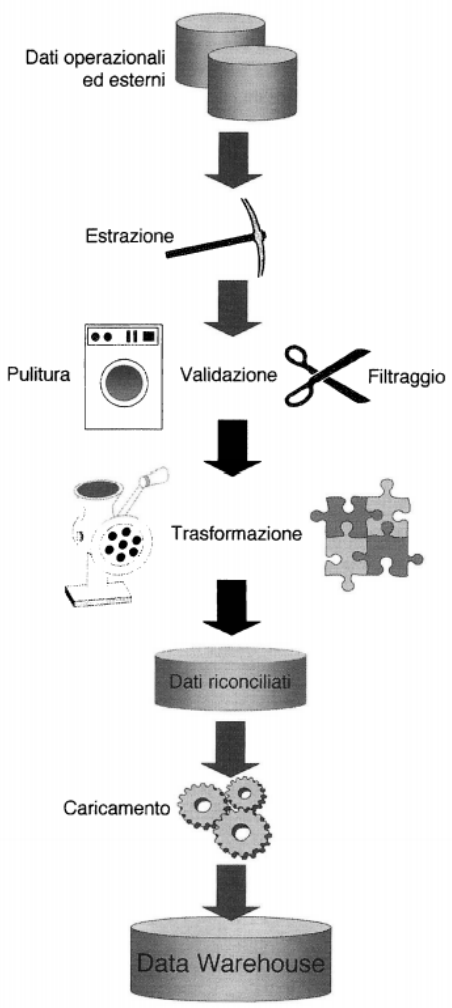


**Figura 3.1.** Estrazione, trasformazione e caricamento (il livello dei dati riconciliati può essere assente)

dimensione e fonte per adattarsi al processo di trasformazione. In tutti i dati è richiesto un certo livello di coerenza in modo che possano essere inseriti nel sistema e convertiti durante la fase successiva. La complessità di questo passo può variare significativamente, a seconda dei tipi, del volume e delle fonti dei dati.

### 3.1.2 Trasformazione

La fase di trasformazione di un processo ETL comporta l'esecuzione di una serie di regole o funzioni sui dati estratti per convertirli in un formato standard. Il livello di manipolazione richiesto dipende esclusivamente dai dati estratti e dalle esigenze del business. Buone fonti di dati richiederanno poca trasformazione, men-

tre altri set di dati potrebbero richiedere trasformazioni significative. Tra gli errori e le inconsistenze tipiche che rendono "sporchi" i dati segnaliamo:

- dati duplicati;
- inconsistenza tra valori logicamente associati;
- dati mancanti;
- uso non previsto di un campo;
- valori impossibili o errati;
- valori inconsistenti per la stessa entità dovuti a differenti convenzioni e abbreviazioni;
- valori inconsistenti per la stessa entità dovuti a errori di battitura.

Tra i processi di trasformazione più comunemente utilizzati ricordiamo:

- *La pulitura* che prevede la correzione, la omogeneizzazione dei dati e l'applicazione di regole proprie del dominio applicativo per stabilire le corrette corrispondenze tra valori.
- *La conversione e la normalizzazione*, che operano sia a livello di formato di memorizzazione sia a livello di unità di misura al fine di uniformare i dati.
- *Il matching*, che stabilisce corrispondenze tra campi equivalenti in sorgenti diverse.
- *La selezione* che eventualmente riduce il numero di campi e di record rispetto alle sorgenti.

### 3.1.3 Caricamento

Questa è l'ultima fase del processo di ETL e consiste nell'importare i dati estratti e trasformati in un database o in un data warehouse, nei quali saranno memorizzati in modo permanente. I dati possono essere caricati attraverso due modalità:

- *Caricamento completo*: i dati trasformati vengono inseriti in nuovi record; ciò produce insiemi di dati che crescono esponenzialmente e possono diventare difficili da mantenere.
- *Caricamento incrementale*: i dati trasformati vengono confrontati con quelli già disponibili, producendo record aggiuntivi solo se le informazioni riscontrate sono nuove e uniche.

## 3.2 La fase di ETL svolta sul nostro dataset

Il primo passo è sottoporre il nostro dataset all'attività di ETL per poterlo preparare alle nostre successive analisi. La libreria di Python molto utile in questa fase è `pandas`, descritta nella Sezione 1.5.2; `pandas` possiede due componenti fondamentali sui quali si basa tutta la sua modalità di funzionamento:

- *serie*: un array, nonchè una semplice colonna;
- *dataframe*: un insieme di serie, ovvero una tabella.

Noi lavoreremo con dataframe.

### 3.2.1 Importazione dei dati

Il nostro dataset è salvato in un file CSV e il primo step sarà quello di creare un dataframe che lo carichi in RAM.

Innanzitutto importiamo le librerie `numpy` e `pandas` nel nostro progetto. Per creare un dataframe a partire da un file CSV, utilizziamo il comando `read_csv`; per visualizzare le sue prime e ultime cinque righe, basta utilizzare, rispettivamente, il comando `head()` e `tail()` (Figura 3.2).

```
In [1]:
import numpy as np
import pandas as pd

In [2]:
data = pd.read_csv(r'C:\Users\Beatrice\Elica\snap_published_status_data2.csv')

In [3]:
data.head()
```

Out[3]:

| | id | idSnap | payloadVersion | createdDate | currentMode | fanLevel | temperature | humidity | airQuality | noise | lux | wifiLevel | errors | flags | ping | bytes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 71450444 | 1573.0 | 0 | 2020-01-25 23:24:42.0 | 1 | 0 | 23 | 43 | 1 | 35 | 0 | 0 | 0 | 0 | 9138 | |
| 1 | 71450445 | 2220.0 | 0 | 2020-01-25 23:22:19.0 | 1 | 0 | 26 | 35 | 1 | 8 | 105 | 0 | 0 | 0 | 2131 | |
| 2 | 71450447 | 1648.0 | 0 | 2020-01-25 23:26:22.0 | 1 | 0 | 21 | 43 | 1 | 40 | 0 | 0 | 0 | 0 | 10436 | |
| 3 | 71450451 | 1240.0 | 0 | 2020-01-25 23:24:10.0 | 0 | 0 | 19 | 47 | 1 | 36 | 0 | 0 | 0 | 0 | 9363 | |
| 4 | 71450454 | 2225.0 | 0 | 2020-01-25 23:24:46.0 | 0 | 0 | 21 | 36 | 1 | 34 | 113 | 0 | 0 | 0 | 8707 | |

```
In [4]:
data.tail()
```

Out[4]:

| | id | idSnap | payloadVersion | createdDate | currentMode | fanLevel | temperature | humidity | airQuality | noise | lux | wifiLevel | errors | flags | ping | l |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3189328 | 83951860 | 1364.0 | 0 | 2020-06-16 13:54:44.0 | 0 | 0 | 26 | 51 | 2 | 2 | 0 | 0 | 0 | 0 | 910 | |
| 3189329 | 83951866 | 1521.0 | 0 | 2020-06-16 13:58:53.0 | 1 | 0 | 25 | 58 | 1 | 7 | 16 | 0 | 0 | 0 | 1796 | |
| 3189330 | 83951956 | 2123.0 | 0 | 2020-06-16 13:56:32.0 | 1 | 0 | 21 | 49 | 2 | 32 | 25 | 0 | 0 | 0 | 8215 | |
| 3189331 | 83951996 | 1548.0 | 0 | 2020-06-16 13:56:58.0 | 1 | 0 | 20 | 54 | 1 | 32 | 29 | 0 | 0 | 0 | 8340 | |
| 3189332 | 83952212 | 1521.0 | 0 | 2020-06-16 13:58:53.0 | 1 | 0 | 25 | 58 | 1 | 7 | 16 | 0 | 0 | 0 | 1796 | |

**Figura 3.2.** Creazione e visualizzazione del dataframe

### 3.2.2 Modifica dei dati

La prima caratteristica che possiamo notare nel nostro dataframe è la numerosità dei campi proposti, tra i quali si evidenziano alcuni poco utili per l'analisi. Abbiamo deciso, quindi, di alleggerire il nostro dataset eliminando le colonne (Figura 3.3) `id, payloadVersion, noise, lux, wifiLevel, bytes03, bytes47` e `bytes811`, perchè non necessari per i nostri scopi. Il comando utilizzato per effettuare questa modifica è `drop`.

```
Data = data.drop(['id','payloadVersion','noise', 'lux', 'wifiLevel','bytes03','bytes47','bytes811'], axis=1)
```

**Figura 3.3.** Eliminazione delle colonne

Successivamente verifichiamo la presenza o meno dei cosiddetti valori `NULL`. Un valore `NULL` è un valore speciale e viene utilizzato quando si vuole indicare un

determinato campo che non ha valore. Grazie al comando `isnull().sum()` (Figura 3.4) è possibile visualizzare quanti valori `NULL` sono presenti per ogni campo.

```
1 Data.isnull().sum()
```

```
idSnap          1
createdDate     0
currentMode     0
fanLevel        0
temperature     0
humidity        0
airQuality      0
errors          0
flags           0
ping            0
mode            0
dtype: int64
```

**Figura 3.4.** I valori `NULL`

Nel nostro dataset è presente un unico valore `NULL`, nel campo `id`. Generalmente, se le righe del dataframe che contengono i valori `NULL` sono poche, possiamo eliminarle (Figura 3.5); a tal proposito utilizziamo il comando `dropna`.

```
1 Data.dropna(inplace=True)
```

**Figura 3.5.** Eliminazione dei valori `NULL`

Infine, eliminiamo i duplicati presenti nel nostro dataframe (Figura 3.6) utilizzando `drop_duplicates`

```
1 df = Data.drop_duplicates()
```

**Figura 3.6.** Eliminazione dei duplicati

Al termine di queste fasi di modifiche otteniamo il dataframe mostrato in Figura 3.7. Esso consiste di 2.909.120 righe e 11 colonne, e rappresenta il punto di partenza delle nostre analisi.

## 3.3 L'analisi esplorativa

L'analisi esplorativa dei dati (Exploration Data ANalysis - EDA) è usata dai data scientist per analizzare e investigare i set di dati e riassumere le loro caratteristiche principali, spesso utilizzando metodi di visualizzazione dei dati. Le tecniche EDA, originariamente sviluppate dal matematico americano John Tukey negli anni '70, continuano ad essere un metodo ampiamente utilizzato nel processo di scoperta dei

```
1 df.shape
```

```
(2909120, 11)
```

```
1 df.head()
```

| | idSnap | createdDate | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping | mode |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1573.0 | 2020-01-25 23:24:42.0 | 1 | 0 | 23 | 43 | 1 | 0 | 0 | 9138 | AUTOMATIC |
| 1 | 2220.0 | 2020-01-25 23:22:19.0 | 1 | 0 | 26 | 35 | 1 | 0 | 0 | 2131 | AUTOMATIC |
| 2 | 1648.0 | 2020-01-25 23:26:22.0 | 1 | 0 | 21 | 43 | 1 | 0 | 0 | 10436 | AUTOMATIC |
| 3 | 1240.0 | 2020-01-25 23:24:10.0 | 0 | 0 | 19 | 47 | 1 | 0 | 0 | 9363 | STANDBY |
| 4 | 2225.0 | 2020-01-25 23:24:46.0 | 0 | 0 | 21 | 36 | 1 | 0 | 0 | 8707 | STANDBY |

**Figura 3.7.** Il dataframe finale

dati. Infatti, queste rappresentano spesso il primo passo nell'analisi dei dati, attuato prima di applicare una qualsiasi tecnica statistica formale.

### 3.3.1 Gli obiettivi dell'analisi esplorativa

L'analisi esplorativa ci permette di ottenere la massima comprensione del nostro set di dati e della sua struttura sottostante. Tra gli obiettivi principali di tale analisi possiamo ricordare:

- il controllo delle ipotesi;
- la selezione preliminare di modelli appropriati;
- l'estrazione di parametri importanti;
- la determinazione delle relazioni tra le variabili;
- l'individuazione di errori e anomalie.

### 3.3.2 I tipi di analisi esplorativa

L'analisi esplorativa dei dati è, generalmente, classificata in due modi. In primo luogo, ogni metodo è grafico o non grafico. In secondo luogo, ogni metodo è univariato o multivariato. I metodi non grafici, generalmente, implicano il calcolo di statistiche riassuntive, mentre i metodi grafici riassumono i dati in modo diagrammatico o pittorico. I metodi univariati guardano una variabile (colonna di dati) alla volta, mentre i metodi multivariati esaminano due o più variabili alla volta.

Possiamo, quindi, individuare quattro tipologie di EDA: univariata non grafica, univariata grafica, multivariata non grafica e multivariata grafica.

- *L'analisi univariata non grafica* è la forma più semplice di analisi dei dati, le cui informazioni sono concentrate in una sola variabile. Lo scopo principale di questa tipologia di analisi è quello di descrivere i dati e trovare i modelli che esistono al loro interno, senza soffermarsi sulle cause e relazioni delle variabili dal momento che viene presa in considerazione soltanto una variabile.
- *L'analisi univariata grafica* è, sicuramente, migliore della precedente, dal momento che i metodi non grafici non forniscono un quadro completo dei dati. Tra i grafici univariati più comuni ricordiamo *lo stem and leaf plot, l'istogramma* e *il boxplot*.

- *Lo stem and leaf plot* è una rappresentazione grafica dei dati che si ottiene disponendo intorno ad un ramo (*stem*) le cifre significative di una distribuzione di frequenza e diramando dal ramo alle foglie (*leaf*) le cifre meno significative. Per rappresentare la distribuzione si costruisce una tabella a due colonne. Nella prima colonna, per ogni valore del cambiamento nei livelli principali della variabile osservata, si riportano le cifre finali corrispondenti. Nella seconda colonna si riportano le relative unità (livelli secondari), indicate tante volte quanti sono i cambiamenti che assumono tali valori. I valori inferiori alla decina sono indicati anteponendo a essi la cifra 0.
  Facciamo un esempio: supponiamo di avere a disposizione i dati 32, 51, 41, 18, 16, 14, 76, 54, 32, 57, 33. Lo stem and leaf plot viene riportato nella tabella:

| **Steam** | **Leaf** |
|---|---|
| 1 | 4, 6, 8 |
| 2 | |
| 3 | 2, 2, 3 |
| 4 | 1 |
| 5 | 1, 4, 7 |
| 6 | |
| 7 | 6 |

**Tabella 3.1.** Un esempio di stem and leaf plot

- *L'istogramma* (Figura 3.8) è un grafico che permette di scoprire e mostrare la distribuzione di frequenza sottostante (forma) ad un insieme di dati continui. L'istogramma è composto da un'insieme di rettangoli tra loro adiacenti. Le basi di questi rettangoli sono tutte allineate sull'asse delle ascisse. Il numero dei rettangoli corrisponde al numero di classi in cui è stata suddivisa la variabile. La larghezza della base dei rettangoli dipende dall'ampiezza di tali classi e, complessivamente, tutte le basi dei rettangoli affiancate devono coprire l'intera gamma dei valori della variabile. L'altezza di ogni rettangolo, invece, può indicare la frequenza oppure la densità dei casi presenti in ogni classe. La frequenza indica quante unità statistiche sono presenti all'interno della classe. La densità, invece, è data dal rapporto tra la frequenza (assoluta o relativa) e l'ampiezza della classe. Tale rapporto è interpretabile come il grado di addensamento delle frequenze nel corrispondente intervallo di valori. Negli istogrammi di densità, infatti, l'altezza dei rettangoli indica la densità della classe, mentre la frequenza è rappresentata dall'area del rettangolo.
- *Il boxplot* (Figura 3.18) permette di studiare graficamente la forma e la variabilità di una distribuzione e confrontare più distribuzioni in termini di forma e variabilità. Questo grafico verrà approfondito nella Sezione 3.5.2.

- *L'analisi multivariata non grafica* si basa sui dati multivariati che derivano da più di una variabile. Questa tecnica non grafica mostra, generalmente, la relazione tra due o più variabili dei dati attraverso tabulazioni incrociate o statistiche.

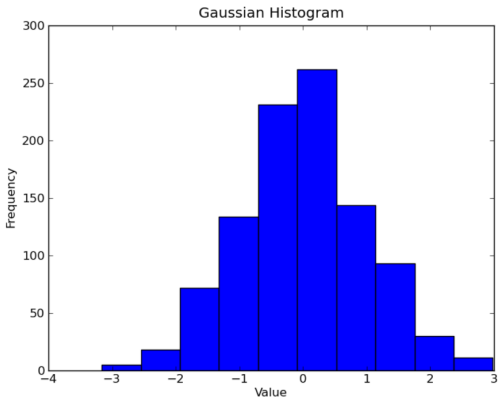


**Figura 3.8.** Un esempio di istogramma

- *L'analisi multivariata grafica* sfrutta i grafici per mostrare le relazioni tra due o più serie di dati. Tra le tipologie più comuni di grafici multivariati ricordiamo *lo scatter plot, il run chart, il bubble chart* e *la heatmap.*
  - *Lo scatter plot* (Figura 3.9) è un grafico di dispersione che utilizza i punti per rappresentare i valori di due diverse variabili numeriche. La posizione di ogni punto sull'asse orizzontale e verticale indica i valori per un singolo punto di dati. I diagrammi di dispersione sono usati per osservare le relazioni tra le variabili.

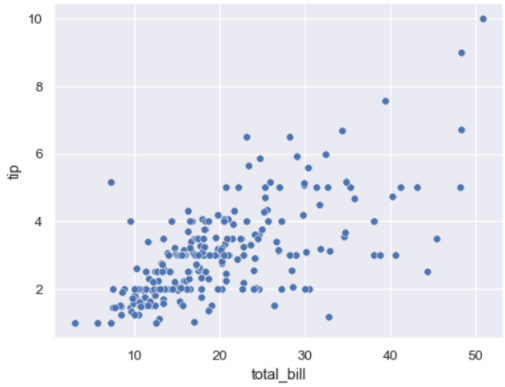


**Figura 3.9.** Un esempio di scatterplot

  - *Il run chart* (Figura 3.10) è un grafico a linee che rappresenta l'andamento dei dati nel tempo. Raccogliendo e tracciando i dati nel tempo si possono trovare tendenze o modelli nel processo. I grafici di esecuzione non possono rivelare se un processo è stabile, poiché non usano limiti di controllo. Tuttavia, possono mostrare come sta funzionando il processo. Il diagramma di esecuzione può essere uno strumento prezioso all'inizio di un progetto,

poiché rivela informazioni importanti su un processo prima di raccogliere abbastanza dati per creare limiti di controllo affidabili.

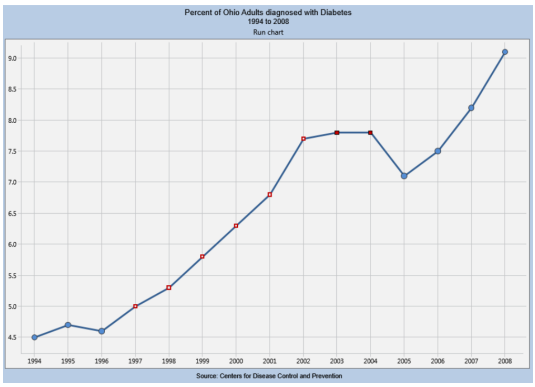


**Figura 3.10.** Un esempio di run chart

– *Il bubble chart* (Figura 3.11) visualizza i dati attraverso cerchi multipli (bolle) in un grafico bidimensionale. Ad ogni punto viene assegnata un'etichetta o una categoria distinguibile attraverso l'utilizzo di colori o aree del cerchio differenti. I grafici a bolle sono tipicamente usati per confrontare e mostrare le relazioni tra cerchi categorizzati, attraverso l'uso di posizionamento e proporzioni. L'immagine complessiva dei grafici a bolle può essere usata per analizzare i modelli/correlazioni. Troppe bolle possono rendere il grafico difficile da leggere, quindi i Bubble Chart hanno una capacità limitata di dimensione dei dati. Questo difetto può essere in qualche modo rimediato dall'interattività, cliccando o passando il mouse sulle bolle per visualizzare informazioni nascoste, avendo un'opzione per riorganizzare o filtrare le categorie raggruppate.

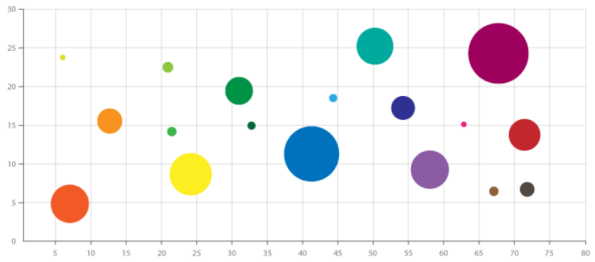


**Figura 3.11.** Un esempio di bubble chart

– *La heatmap* (Figura 3.12) è una rappresentazione grafica dei dati in cui i valori sono visualizzati attraverso uno specifico colore. Lo spettro di colori utilizzato varia da caldo a freddo. Quando le heatmap sono applicate a un formato tabulare, sono utili per l'esame incrociato dei dati multivariati, mo-

strano la varianza e correlazione tra più variabili, rivelando qualsiasi modello e mostrando se alcune variabili sono simili tra loro. In questa tipologia di grafico, tipicamente, tutte le righe rappresentano una categoria, la cui etichetta è visualizzata sul lato sinistro o destro, e tutte le colonne mostrano un'altra categoria, la cui etichetta è visualizzata in alto o in basso. Le singole righe e colonne sono divise in sottocategorie, che incrociandosi tra loro danno origine ad una matrice. I dati contenuti in una cella si basano sulla relazione tra le due variabili nella riga e nella colonna di collegamento. Accanto ad una heatmap è necessaria una legenda per poterla leggere con successo.

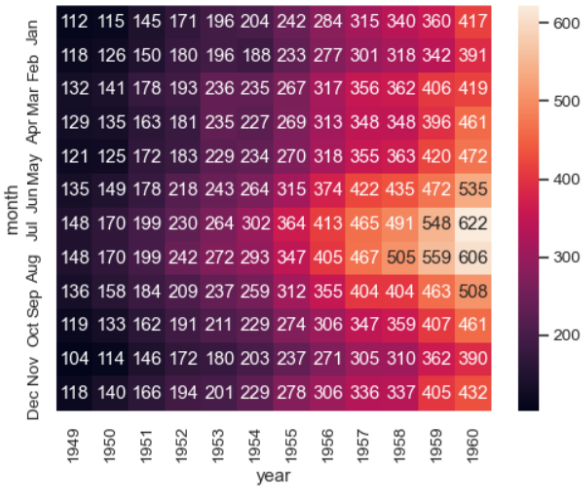


**Figura 3.12.** Un esempio di heatmap

## 3.4 L'anomaly detection

Il processo di anomaly detection ("identificazione delle anomalie") consiste nel riconoscimento di condizioni o circostanze inattese all'interno di un dataset. Per contenuto o evento inatteso si intende qualcosa che differisce dalla norma, di insolito e inaspettato. Con centinaia o migliaia di informazioni da analizzare, il rilevamento delle anomalie può aiutare a indicare dove si sta verificando un errore, migliorando l'analisi della causa principale e ottenendo rapidamente il supporto tecnico sul problema. Questo processo ha molte applicazioni in reti di sensori, nel mondo degli affari, dal rilevamento delle frodi nelle transazioni con carta di credito al rilevamento dei difetti negli ambienti operativi, dal rilevamento delle intrusioni (identificando strani modelli nel traffico di rete che potrebbero segnalare una violazione) al monitoraggio della salute (individuando, per esempio, un tumore maligno in una risonanza magnetica).

### 3.4.1 Cos'è un'anomalia

Il termine anomalia è anche indicato come *outlier*. Un outlier può essere visto come un punto o un comportamento o un pattern che risulta essere statisticamente "inusuale" o "anomalo" rispetto all'intero dataset. In un certo senso, questa definizione lascia all'analista il compito di decidere cosa sarà considerato anormale. Prima di poter individuare eventi anormali è necessario caratterizzare gli eventi normali. Le anomalie possono essere ampiamente classificate in: *anomalie puntuali, anomalie contestuali* e *anomalie collettive*.

- *Le anomalie puntuali* (Figura 3.13) riguardano le singole istanze di dati estremamente diverse e distanti dal resto del dataset. Un tipico esempio di questo genere di anomalie è l'identificazione delle frodi con carte di credito basate sull'importo speso nelle singole transazioni. In questo caso, l'importo di una transazione estremamente elevato rispetto agli importi storici è considerata un'anomalia puntuale.

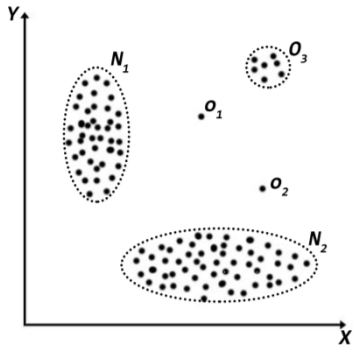


**Figura 3.13.** I punti o1 e o2 sono considerati anomalie puntuali

- *Le anomalie contestuali* sono quelle anomalie specifiche del contesto e sono comuni nei dati di serie temporali; in assenza di un contesto, tutti i dati sembrano normali. Un classico esempio, illustrato in Figura 3.14, è la rilevazione di una temperatura di 5° a Torino che può essere considerata un'anomalia se rilevata durante la stagione estiva (per esempio, nel mese di Giugno) ma nella norma se si considera la stagione invernale (per esempio, nel mese di Dicembre).
- *Le anomalie collettive* (Figura 3.15) possono formarsi a causa di una combinazione di molte istanze, per esempio possono riguardare i dati in sequenza nel log di rete.

### 3.4.2 Le tecniche di anomaly detection

Con il rapido sviluppo del settore IT i dati sono diventati di grandi dimensioni. Trovare manualmente i punti o i pattern anomali non è stato più fattibile in

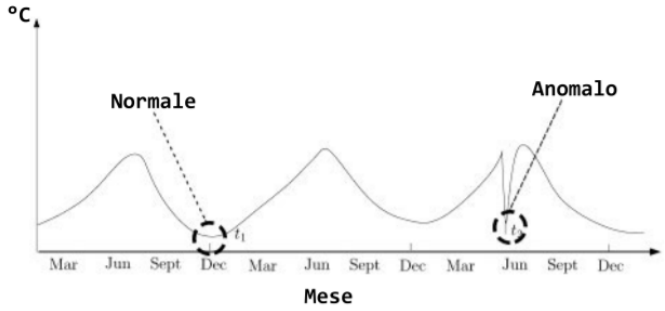


**Figura 3.14.** Un esempio di anomalia contestuale

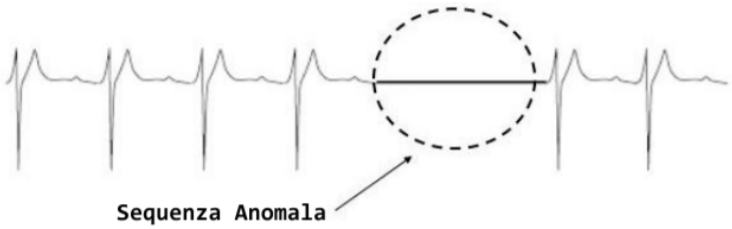


**Figura 3.15.** Un esempio di anomalia collettiva

molte applicazioni. Come risultato di questo fenomeno sono state proposte nuove metodologie basate sulle tecniche tradizionali e applicate ad alcuni contesti per l'anomaly detection dei data stream. In base alle specifiche applicazioni, le tecniche impiegate variano dalle analisi di semplici serie temporali a quelle complesse e multidimensionali degli streaming di dati.

Esistono diversi modelli di anomaly detection ed è possibile categorizzare queste tecniche secondo due tassonomie. La prima, a seconda se i campioni di dati di un dataset vengono forniti con delle etichette assegnate da esperti in materia, classifica le tecniche di anomaly detection in: *modelli con apprendimento supervisionato, semi supervisionato* e *non supervisionato.*

- *Nei modelli con apprendimento supervisionato* gli esperti di dominio esaminano ed etichettano i campioni; dunque l'attività di anomaly detection si riconduce ad un modello di classificazione; gli esperti possono etichettare i dati normali in modo da considerare tutti gli altri come valori anomali.
- Per ciò che concerne *i modelli con apprendimento semi supervisionato*, in alcuni casi è etichettata solo una piccola parte dei dati. Ad esempio, se sono disponibili alcuni oggetti normali etichettati, il modello si addestra su questi e può essere, poi, utilizzato per rilevare i valori anomali perché diversi rispetti a quelli che già si conoscono.
- Per ciò che concerne *i modelli con apprendimento non supervisionato*, in alcuni scenari le etichette dei dati non sono disponibili; pertanto si fanno delle ipotesi. I valori normali sono in qualche modo raggruppati e seguono un modello; invece, quelli anomali si trovano in uno spazio lontano rispetto a tutti gli altri. In

questi casi potrebbe esserci un alto tasso di falsi positivi; quindi i metodi non supervisionati sono meno efficaci a quelli supervisionati. Spesso questo tipo di anomaly detection può essere ricondotta agli algoritmi di clustering.

Una seconda tassonomia per questi modelli si basa sui metodi di separazione delle anomalie rispetto al resto del dataset. In questo caso possiamo classificare i modelli in tre tipologie: *metodi statistici, metodi basati sulla prossimità* e *metodi basati sul clustering.*

- *I metodi statistici* fanno assunzioni sulla normalità dei dati; i dati che non seguono il modello sono considerati outlier.
- *Nei metodi basati sulla prossimità* un oggetto è un valore anomalo se i punti ad esso più vicini sono molto lontani nello spazio delle caratteristiche, ovvero se si discostano significativamente rispetto ai loro vicini.
- *Nei metodi basati sul clustering* si assume che i valori normali appartengono a cluster grandi e densi, mentre gli outlier a cluster piccoli e sparsi o non appartengono a nessun cluster.

## 3.5 Le analisi svolte sul nostro dataset

In questa Sezione entriamo un pò più nel vivo del nostro studio e spieghiamo quali saranno le analisi che descriveranno in dettaglio nel prossimo capitolo.

Prendiamo in considerazione il dataframe di Figura 3.7; osservando il campo `errors`, la prima domanda che ci sorge spontanea è se siano presenti o meno Snap senza errori. A tal proposito, abbiamo implementato un codice (Figura 3.16) che ricercasse nel nostro dataframe questa tipologia di Snap e che ci restituisse il numero di dispositivi senza e con errori.

```
un_snap = df['idSnap'].unique()
cnt_1 = 0
cnt_2 = 1311

for snap in un_snap[::]:
    tmp = df.loc[df['idSnap'] == snap]
    cnt = 0

    for x in tmp.errors:
        cnt += x

    if cnt == 0:
        cnt_1 += 1
        cnt_2 -= 1

print("Sono presenti " + str(cnt_1) + " snap senza errori e " + str(cnt_2) + " snap con errori ")
```

**Figura 3.16.** Il codice per ricercare il numero di Snap senza e con errori

Nel codice appena riportato definiamo:

- il vettore `un_snap`, in cui salviamo tutti gli Snap che compaiono nel file CSV;
- la variabile `cnt_1`, posta inizialmente uguale a 0, e che fungerà da contatore degli Snap senza errori;

- la variabile `cnt_2`, posta uguale al numero di Snap, e che fungerà da contatore degli Snap con errori.

Successivamente implementiamo due cicli for. Il ciclo esterno, che scorre tutti gli elementi del vettore `un_snap`, crea un dataframe temporaneo le cui righe contengono l'idSnap i-esimo; definiamo, poi, la variabile `cnt` e la poniamo uguale a 0. Il ciclo interno scorre gli errori degli Snap salvati nel dataframe temporaneo e li somma, sfruttando la variabile `cnt` precedentemente definita. Se `cnt` è uguale a 0, vuol dire che tutte le righe dello Snap i-esimo sono 0, e quindi non ci sono errori.

L'output di questo codice è riportato in Figura 3.17.

```
Sono presenti 1126 snap senza errori e 185 snap con errori
```

**Figura 3.17.** L'output del codice

Grazie a questo risultato abbiamo scoperto che, nel nostro dataset, sono maggiormente presenti Snap senza errori e che circa il 16% della totalità possiede errori. Le nostre analisi si baseranno proprio su queste due categorie di dispositivi.

### 3.5.1 L'analisi esplorativa degli Snap

L'analisi esplorativa che implementeremo nel capitolo successivo si divide in due fasi: nella prima scegliamo ed analizziamo uno Snap senza errori, nella seconda scegliamo ed analizziamo uno Snap per ogni errore presente nel dataset.

La scelta dello Snap senza errore viene effettuata tra quelli che possiedono la maggior parte delle modalità di utilizzo; tra questi, poi, prendiamo in considerazione quello con più elementi. Invece, lo Snap caratterizzato da una determinata tipologia di errore, viene scelto in base al dispositivo che possiede il maggior numero di errori di quel tipo.

Successivamente, sia nella prima che nella seconda fase, prenderemo in considerazione la temperatura, l'umidità e la qualità dell'aria di ciascuno Snap considerato ed effettueremo analisi principalmente riguardo:

- *L'andamento nel tempo*, visualizzando il comportamento temporale di ciascuno Snap mediante l'utilizzo di lineplot e scatterplot.
- *La distribuzione univariata*, ovvero una distribuzione di frequenze che descrive l'andamento di un "fenomeno" (in questo caso, temperatura, umidità e qualità dell'aria), che presenta una variabilità, o dispersione, intorno ad un valore centrale.
- *La correlazione* tra le coppie di variabili, che esprime la "forza", o "intensità", del loro legame, mediante l'utilizzo di indici, detti *indici di correlazione*.

### 3.5.2 L'anomaly detection degli Snap

Una volta svolte tutte le analisi descritte nella Sezione precedente, il passo successivo sarà l'anomaly detection. Per identificare in modo rapido e preciso valori anomali

e outlier abbiamo utilizzato uno strumento efficace per la visualizzazione dei dati rispetto ai valori centrali, chiamato *boxplot*. Il boxplot è un grafico statistico che si utilizza per variabili quantitative ed è molto utile per capire se la distribuzione è simmetrica oppure asimmetrica, e per confrontare la forma di più distribuzioni. Esso, però, non mostra alcune caratteristiche, come eventuali picchi o valli, che invece si possono osservare con un istogramma.

Il boxplot permette di rappresentare sullo stesso grafico cinque tra le misure di posizione più utilizzate in statistica: *il valore minimo (Q0), il primo quartile (Q1), la mediana (Q2), il terzo quartile (Q3)* ed *il valore massimo (Q4) di una variabile*. Spieghiamo meglio questo concetto attraverso un esempio; consideriamo un piccolo set di dati costituito da sette osservazioni: 1, 6, 5, 4, 4, 7, 8. Innanzitutto organizziamo i dati dal più piccolo al più grande (1, 4, 4, 5, 6, 7, 8); il punto medio è la mediana (5). La mediana divide i dati in due metà (1, 4, 4 e 4, 7, 8). Il punto medio di ogni metà è chiamato "quartile". Così otteniamo due quartili: il 1° quartile è il punto medio della prima metà (4) e il 3° quartile è il punto medio della seconda metà (7). La distanza tra questi due quartili è chiamata *Interquartile Range (IQR)*.

Il boxplot (Figura 3.18) è rappresentato attraverso una "scatola", la cui lunghezza è pari all'IQR, e i valori minimi e massimi sono rappresentati dai suoi "baffi". Il baffo superiore si estende verso l'alto fino al valore massimo, che è minore o uguale a 1,5 volte il range interquartile (IQR). Il baffo inferiore, invece, si estende verso il basso fino al più piccolo valore maggiore o uguale a 1,5 volte il range interquartile. Possiamo, dunque, definire due soglie:

- *la soglia inferiore*, fissata in $Q1 - 1,5 * IQR$
- *la soglia superiore*, fissata in $Q3 + 1,5 * IQR$

Sostanzialmente si moltiplica per 1,5 l'altezza della scatola e si riporta questa distanza al di sopra del terzo quartile ed al di sotto del primo quartile. I valori che non rientrano in tale distanza sono considerati outlier.

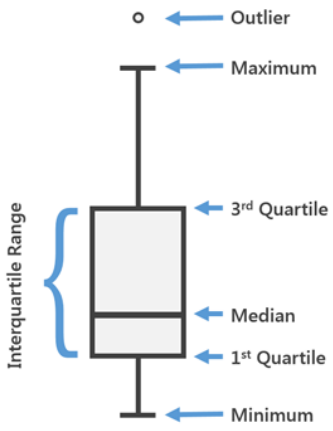


**Figura 3.18.** Un boxplot e le sue caratteristiche

# 4

# Implementazione della campagna

*In questo capitolo implementeremo la campagna sperimentale di data analysis, anticipata nel capitolo precedente. Le analisi verranno svolte su due tipologie di Snap differenti: senza errori e con errori. Per ogni Snap analizzato valuteremo la temperatura, l'umidità e la qualità dell'aria rilevata dai loro rispettivi sensori, prestando particolare attenzione agli outlier.*

## 4.1 Prima analisi: Snap senza errori

Nel nostro dataset, come già anticipato nella Sezione 3.5, sono presenti 1.126 Snap senza errori. La nostra analisi inizia studiando il comportamento di questa tipologia di dispositivi. Abbiamo, quindi, realizzato un codice per ottenere un dataframe contenente solo questa tipologia di Snap, chiamato `snap_ne` (Figura 4.1).

```
1  un_snap = df['idSnap'].unique()
2  snap_ne = df[0:0]
3
4  for snap in un_snap[::]:
5      tmp = df.loc[df['idSnap'] == snap]
6      cnt = 0
7
8      for x in tmp.errors:
9          cnt += x
10
11     if cnt == 0:
12         snap_ne = snap_ne.append(tmp)
```

**Figura 4.1.** Il codice per realizzare un dataframe con Snap senza errori

Questo codice è identico a quello riportato in Figura 3.16, le uniche due differenze sono che, al posto di definire due variabili contatore, definiamo un dataframe vuoto, `snap_ne`, con gli stessi campi del dataset iniziale; inoltre, se la variabile `cnt` è zero, aggiungiamo il dataframe temporaneo `tmp` al dataframe `snap_ne`.

Visualizziamo le statistiche descrittive del nostro nuovo dataframe (Figura 4.2) grazie all'istruzione `describe()` che, per ciascuna delle colonne di `snap_ne`, ci restituisce in output i seguenti parametri:

- *count*, ovvero il numero di variabili non nulle;
- *la media*;
- *la deviazione standard*;
- *il minimo*;
- *il primo quartile*, ovvero il valore corrispondente al 25% degli elementi della distribuzione;
- *il secondo quartile*, ovvero il valore corrispondente al 50% degli elementi della distribuzione;
- *il terzo quartile*, ovvero il valore corrispondente al 75% degli elementi della distribuzione;
- *il massimo*.

```
1 snap_ne.describe()
```

| | idSnap | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping |
|---|---|---|---|---|---|---|---|---|---|
| count | 2.381824e+06 | 2.381824e+06 | 2.381824e+06 | 2.381824e+06 | 2.381824e+06 | 2.381824e+06 | 2381824.0 | 2.381824e+06 | 2.381824e+06 |
| mean | 1.757912e+03 | 1.142553e+00 | 1.280817e-01 | 2.272494e+01 | 4.721371e+01 | 1.223430e+00 | 0.0 | 6.154233e-02 | 5.713018e+03 |
| std | 5.815704e+02 | 1.528549e+00 | 4.061211e-01 | 3.219886e+00 | 1.129228e+01 | 5.870079e-01 | 0.0 | 6.362917e-01 | 3.266685e+03 |
| min | 1.600000e+02 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.0 | 0.000000e+00 | -3.275300e+04 |
| 25% | 1.378000e+03 | 0.000000e+00 | 0.000000e+00 | 2.100000e+01 | 4.000000e+01 | 1.000000e+00 | 0.0 | 0.000000e+00 | 3.343000e+03 |
| 50% | 1.862000e+03 | 1.000000e+00 | 0.000000e+00 | 2.300000e+01 | 4.800000e+01 | 1.000000e+00 | 0.0 | 0.000000e+00 | 5.184000e+03 |
| 75% | 2.241000e+03 | 1.000000e+00 | 0.000000e+00 | 2.500000e+01 | 5.500000e+01 | 1.000000e+00 | 0.0 | 0.000000e+00 | 7.468000e+03 |
| max | 2.628000e+03 | 6.000000e+00 | 3.000000e+00 | 5.800000e+01 | 9.900000e+01 | 5.000000e+00 | 0.0 | 9.000000e+00 | 3.276700e+04 |

**Figura 4.2.** Le statistiche descrittive di `snap_ne`

Prestando particolare attenzione alla temperatura notiamo che, nonostante tutti gli Snap siano senza errori, il massimo rilevato raggiunge il valore di 58 gradi, una temperatura non usuale in un ambiente chiuso. Questo ci fa supporre che ci siano alcuni Snap i cui errori non sono stati rilevati.

Un'altra osservazione che possiamo effettuare su questo dataframe riguarda le modalità di utilizzo. In tutto, le modalità sono 7, ma abbiamo scoperto che non esiste uno Snap che, nel corso delle rilevazioni, le possieda tutte. La modalità *Link* è quella meno frequente (Figura 4.3).

```
1 snap_ml = snap_ne.loc[(snap_ne['currentMode'] == 4)]
2 snap_ml.head()
```

| | idSnap | createdDate | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping | mode |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **2444858** | 1923.0 | 2020-05-21 19:48:36.0 | 4 | 1 | 30 | 42 | 2 | 0 | 1 | 1981 | LINK |
| **2500345** | 2523.0 | 2020-05-23 19:32:58.0 | 4 | 0 | 24 | 43 | 3 | 0 | 1 | 15550 | LINK |

**Figura 4.3.** Gli Snap senza errore con la modalità Link abilitata

Attraverso la funzione `loc` abbiamo salvato in un dataframe temporaneo, chiamato *snap_ml*, tutti gli Snap senza errore, la cui modalità di funzionamento sia *Link*. Soltanto i dispositivi 1.923 e 2.523 hanno tali caratteristiche. Notiamo che il valore di *flags* è 1, come dovrebbe essere, perchè, se fosse stato diverso, avremmo avuto un'incoerenza, in quanto la modalità Link non sarebbe stata abilitata e, di conseguenza, la cappa non sarebbe stata connessa al nostro dispositivo.

### 4.1.1 La scelta dello Snap

Il prossimo passo è quello di scegliere, tra gli Snap senza errore, uno che abbia abilitate tutte le modalità di utilizzo e il maggior numero di elementi. Abbiamo, dunque, implementato un codice (Figura 4.4) che ci restituisse l'idSnap e il suo numero di osservazioni.

```
var = []
best = [0,0]
un_snap = snap_ne['idSnap'].unique()

for snap in un_snap[::]:
    tmp = df.loc[df['idSnap'] == snap]
    if (tmp.currentMode.nunique() == 6):
        flag=True
    else:
        flag=False
    if flag==True:
        var.append(snap)
        if len(tmp) > best[0]:
            best[0] = len(tmp)
            best[1] = snap
print (best)
```

**Figura 4.4.** Il codice per la scelta dello Snap senza errori

In questo codice definiamo, innanzitutto, tre vettori: `var`, `best` e `un_snap`. `Var` contiene tutti gli Snap con errore 0 per ogni riga, `best` contiene la lunghezza dello Snap i-esimo e l'idSnap che possiede il numero maggiore di elementi, e `un_Snap` contiene tutti i dispositivi senza errore.

Successivamente implementiamo un ciclo che scorre tutti gli elementi del vettore `un_Snap` e crea un dataframe temporaneo le cui righe contengono l'idSnap i-esimo. Se lo Snap i-esimo possiede 6 modalità di utilizzo, allora la variabile booleana è posta a `True`, altrimenti è posta a `False`. Se `flag` è `True`, aggiungiamo l'idSnap dello Snap i-esimo nel vettore `var`, grazie alla funzione `append`. Infine, se la lunghezza dello Snap i-esimo è maggiore del massimo finora trovato, salviamo in `best[0]`, la lunghezza dello Snap, e in `best[1]` l'idSnap dello snap i-esimo.

L'output di questo codice è riportato in Figura 4.5.

```
[5554, 2257.0]
```

**Figura 4.5.** Output del codice per la scelta dello Snap senza errori

Dunque, lo Snap che analizzeremo è il 2.257, che possiede 5.554 osservazioni.

## 4.2 Lo Snap 2.257

Innanzitutto, salviamo tutte le righe riferite allo Snap 2.257 in un nuovo dataframe, chiamato `snap2257`. Successivamente, eliminiamo la colonna `idSnap` e stampiamo le sue prime cinque righe. Il tutto è mostrato nella Figura 4.6.

```
snap2257 = df.loc[df['idSnap'] == 2257.0]
snap2257 = snap2257.drop(['idSnap'], axis=1)

snap2257.head()
```

| | createdDate | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping | mode |
|---|---|---|---|---|---|---|---|---|---|---|
| **23** | 2020-01-25 23:26:50.0 | 0 | 0 | 15 | 69 | 1 | 0 | 0 | 8312 | STANDBY |
| **496** | 2020-01-25 23:36:50.0 | 0 | 0 | 15 | 69 | 1 | 0 | 0 | 8007 | STANDBY |
| **976** | 2020-01-25 23:46:50.0 | 0 | 0 | 14 | 69 | 1 | 0 | 0 | 7707 | STANDBY |
| **1441** | 2020-01-25 23:56:50.0 | 0 | 0 | 14 | 70 | 1 | 0 | 0 | 7537 | STANDBY |
| **1888** | 2020-01-26 01:06:50.0 | 0 | 0 | 14 | 69 | 1 | 0 | 0 | 7091 | STANDBY |

**Figura 4.6.** Lo Snap 2.257

Visualizziamo le statistiche descrittive del nostro dataframe (Figura 4.7) grazie all'istruzione `describe()`.

```
snap2257.describe()
```

| | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping |
|---|---|---|---|---|---|---|---|---|
| **count** | 5554.000000 | 5554.000000 | 5554.000000 | 5554.000000 | 5554.000000 | 5554.0 | 5554.000000 | 5554.000000 |
| **mean** | 0.407994 | 0.060317 | 21.215700 | 52.987757 | 1.391430 | 0.0 | 0.048974 | 7640.718041 |
| **std** | 0.510556 | 0.269326 | 4.825565 | 12.594660 | 0.822105 | 0.0 | 0.624068 | 2694.772464 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.000000 | 0.000000 |
| **25%** | 0.000000 | 0.000000 | 17.000000 | 44.000000 | 1.000000 | 0.0 | 0.000000 | 5748.250000 |
| **50%** | 0.000000 | 0.000000 | 23.000000 | 53.000000 | 1.000000 | 0.0 | 0.000000 | 6813.500000 |
| **75%** | 1.000000 | 0.000000 | 25.000000 | 63.000000 | 1.000000 | 0.0 | 0.000000 | 8793.500000 |
| **max** | 6.000000 | 3.000000 | 32.000000 | 81.000000 | 5.000000 | 0.0 | 8.000000 | 20130.000000 |

**Figura 4.7.** Le statistiche descrittive dello Snap 2.257

Prestiamo particolare attenzione al valore minimo di temperatura, umidità e VOCs rilevati dai rispettivi sensori del dispositivo. Notiamo che il valore minimo

è 0 per tutti e tre i campi; questo dato risulta un pò "strano" e richiederà delle osservazioni più approfondite. La media dei valori relativa a questi tre campi, invece, risulta coerente con le aspettative.

Vediamo, ora, quali sono le modalità di utilizzo maggiormente riscontrate dallo Snap 2.257 (Figura 4.8). La funzione utilizzata è `catplot`, che permette di realizzare grafici che mostrano la relazione tra una variabile numerica e una o più variabili categoriche. Con il parametro `kind` possiamo selezionare il tipo di grafico da rappresentare, in questo caso un istogramma. Possiamo notare che le modalità di utilizzo sono prevalentemente Standby (0) e Automatico (1).

```
1 sns.set(rc={'figure.figsize':(8.7, 5.27)})
2 sns.set(style = 'whitegrid')
3 sns.catplot(data=snap2257, x="currentMode", kind='count')
```

```
<seaborn.axisgrid.FacetGrid at 0x193c034c4c0>
```

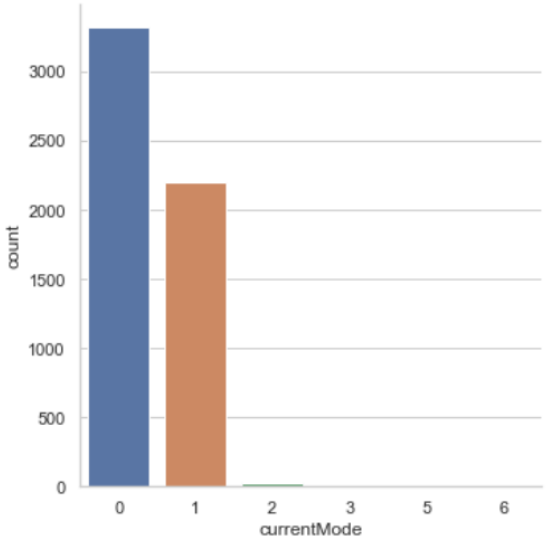


**Figura 4.8.** Le modalità di utilizzo dello Snap 2.257

Valutiamo l'andamento temporale della temperatura, umidità e VOCs registrati dai sensori. Innanzitutto, indicizziamo il nostro dataframe con le date in cui sono state effettuate le rilevazioni (Figura 4.9).

```
1 from datetime import datetime as dt
2 snap2257['createdDate'] = pd.to_datetime(snap2257['createdDate'], infer_datetime_format=True)
3 indexed_snap2257 = snap2257.set_index(['createdDate'])
```

**Figura 4.9.** Indicizzazione del dataframe `snap2257`

Come mostrato in Figura 4.10, importiamo la libreria `matplotlib` e settiamo la dimensione dei font e delle figure. Grazie all'istruzione `subplot`, possiamo realizzare

tre “sotto trame”, distinguibili dall’oggetto `axs`. La funzione `plot` traccia i grafici per ogni “sotto trama”; ad essa vengono passati come parametri, rispettivamente, i valori corrispondenti alla $x$ e alla $y$ di un diagramma cartesiano.

```
import datetime
from matplotlib.pylab import rcParams
import matplotlib.pyplot as plt

plt.rcParams.update({'figure.figsize':(20,10),'figure.dpi':120})
fig, axs = plt.subplots(3, 1,sharex=True)

axs[0].plot(snap1347['createdDate'], snap1347['temperature'])
axs[1].plot(snap1347['createdDate'], snap1347['humidity'])
axs[2].plot(snap1347['createdDate'], snap1347['ping'])

axs[0].set_title('Temperatuta rilevata')
axs[1].set_title('Umidità rilevata')
axs[2].set_title('VOCs rilevati')

inizio = '2020-01-25 00:00:00'
d_inizio = datetime.datetime.strptime(inizio, '%Y-%m-%d %H:%M:%S')
fine = '2020-06-01 00:00:00'
d_fine = datetime.datetime.strptime(fine, '%Y-%m-%d %H:%M:%S')
plt.xlim([d_inizio, d_fine])
```

**Figura 4.10.** Istruzione per la creazione degli andamenti di temperatura, umidità e VOCs rilevati nel tempo

Otteniamo, dunque, i grafici mostrati in Figura 4.11.

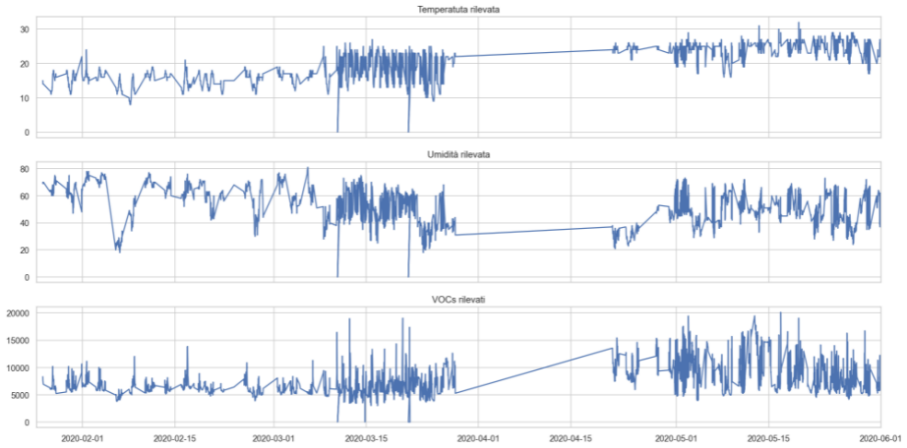


**Figura 4.11.** Gli andamenti nel tempo di temperatura, umidità e VOCs rilevati dallo Snap 2.257

Possiamo subito notare dai grafici che dal 29 Marzo 2020 fino al 22 Aprile 2020 non ci sono state rilevazioni da parte dei sensori. La temperatura, l’umidità e i VOCs verranno analizzati più approfonditamente nelle prossime sezioni.

Per una migliore visualizzazione, dividiamo il dataframe in due parti: `snap_1` e `snap_2` (Figura 4.12), così da non considerare il range temporale in cui non sono state effettuate rilevazioni.

```
snap_1 = snap2257[ (snap2257.createdDate >= '2020-01-25 00:00:00') & (snap2257.createdDate <= '2020-03-29 00:00:00')  ]
snap_2 = snap2257[ (snap2257.createdDate >= '2020-04-22 00:00:00') & (snap2257.createdDate <= '2020-06-17 00:00:00')  ]
```

**Figura 4.12.** Divisione del dataframe `snap2257`

### 4.2.1 Le correlazioni tra le variabili

Quando si parla di correlazione ci si riferisce, generalmente, al grado di relazione tra due variabili. La più importante tipologia di correlazione è quella lineare e viene misurata dall'*indice di correlazione di Pearson* o *indice di correlazione lineare*:

$$r = \frac{\sum_{i=1}^{n} x'_i \cdot y'_i}{\sqrt{\sum_{i=1}^{n} (x'_i)^2 \cdot \sum_{i=1}^{n} (y'_i)^2}}$$

Sapendo che le deviazioni standard delle variabili $x$ e $y$ sono:

$$\sigma_x = \sqrt{\frac{\sum (x'_i)^2}{n}} \quad e \quad \sigma_y = \sqrt{\frac{\sum (y'_i)^2}{n}}$$

e la covarianza di $x$ e $y$ è:

$$\sigma_{xy} = \frac{\sum_{i=1}^{n} x'_i \cdot y'_i}{n}$$

Possiamo effettuare delle sostituzioni e scrivere l'indice di correlazione $r$ come:

$$r = \frac{\sigma_{xy}}{\sigma_x \cdot \sigma_y}$$

L'indice di correlazione gode delle seguenti proprietà:

- $r$ è un valore compreso tra -1 e 1;
- Se $r > 0$, allora esiste una correlazione lineare *diretta* tra le due variabili osservate;
- Se $r = 1$, allora la correlazione diretta è massima;
- Se $r < 0$, allora esiste una correlazione lineare *inversa* tra le due variabili osservate;
- Se $r = -1$, allora la correlazione inversa è massima;
- Se $r = 0$, allora le due variabili osservate non presentano alcuna correlazione lineare (sono, cioè, linearmente incorrelate).

La heatmap (Figura 4.13), come già anticipato nella Sezione 3.2.2, ci permette di determinare graficamente le correlazioni tra le variabili.

La libreria `seaborn` predispone dell'istruzione `heatmap` che ci permette di realizzarla; i parametri passati a tale istruzione sono:

- `correlation`, in cui salviamo le correlazioni tra le coppie delle colonne del dataframe;

- `xticklabels`, che riporta le etichette delle colonne di `correlation`, sull'asse $x$ del grafico;
- `yticklabels`, che riporta le etichette delle colonne di `correlation`, sull'asse $y$ del grafico;
- `annot`, se posto a `True`, scrive il valore degli indici di correlazione in ogni cella.

```
1 import seaborn as sns
2
3 plt.figure(figsize=(14,7))
4 correlation = snap2257.corr()
5 sns.heatmap(correlation, xticklabels=correlation.columns, yticklabels=correlation.columns, annot=True)
```

```
<AxesSubplot:>
```


**Figura 4.13.** Il codice per realizzare la heatmap dello Snap 2.257

Osservando il nostro grafico, notiamo una correlazione moderata; in particolare, tale correlazione viene individuato tra le seguenti coppie di variabili:

- *currentMode* e *ping*;
- *currentMode* e *humidity*;
- *currentMode* e *temperature*;
- *airQuality* e *ping*.

Notiamo, invece, una correlazione debole, in particolare, tra:

- *fanLevel* e *ping*;
- *fanLevel* e *airQuality*;
- *fanLevel* e *humidity*;
- *fanLevel* e *temperature*;
- *fanLevel* e *currentMode*.

Tali correlazioni saranno oggetto della nostra analisi.

### 4.2.2 Il sensore di temperatura

L'analisi sulla temperatura rilevata dal *Temperature Sensor* dello Snap 2.257 inizia esaminando la sua distribuzione. A tal proposito utilizziamo un'istruzione della libreria `seaborn`, chiamata `histplot` (Figura 4.14), che ha il compito di tracciare istogrammi univariati. A tale funzione passiamo, come parametri, la temperatura registrata e i *bins*, ovvero il numero di rettangoli del nostro istogramma, e settiamo `kde` a `True`, in modo che la curva della densità del Kernel venga graficata.

```
sns.set(rc={'figure.figsize':(8.7, 5.27)})
sns.set(style = 'whitegrid')
sns.histplot(snap2257['temperature'], bins=32, kde=True).set_title('Distribuzione della temperatura')
```

**Figura 4.14.** Codice della distribuzione della temperatura rilevata dallo Snap 2.257

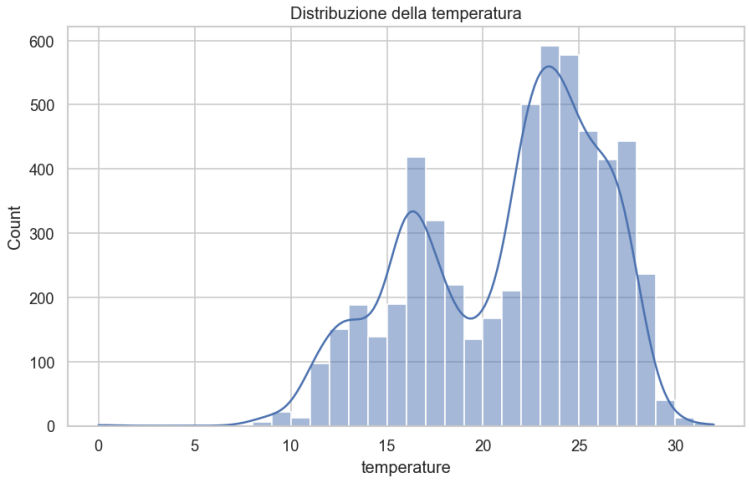


**Figura 4.15.** La distribuzione della temperatura rilevata dallo Snap 2.257

Come mostrato in Figura 4.15, otteniamo, dunque, un istogramma e una curva continua (il kernel) generata dalla somma dei kernel di ogni singolo punto dei dati. Un grafico di densità, quindi, altro non è che una versione smussata e continua di un istogramma. Da questa distribuzione deduciamo che la misura della temperatura più ricorrente è 23 gradi.

Il passo successivo è quello di graficare la relazione tra le modalità di utilizzo e la temperatura, grazie alla funzione `scatterplot` (Figura 4.16) messa a disposizione dalla libreria seaborn.

Lo scatterplot della temperatura rilevata è mostrato in Figura 4.17.

Notiamo che lo Snap, nel primo periodo di rilevazioni (dal 25 Gennaio al 29 Marzo), lavora in modalità Standby, prevalentemente in corrispondenza delle temperature al di sotto dei 20 gradi circa e in modalità Automatica al di sopra dei 20 gradi. Solo alcuni punti si discostano da tale andamento, in particolare nei giorni 10

```
sns.set(rc={'figure.figsize':(18, 10)})

fig, ax = plt.subplots(2, 1)

fig = sns.scatterplot(data=snap_1, x="createdDate", y="temperature", hue="mode", palette="deep", ax=ax[0], legend = 'brief')
fig.set_title('Rilevazioni Gennaio-Aprile')
fig = sns.scatterplot(data=snap_2, x="createdDate", y="temperature", hue="mode", palette="deep", ax=ax[1], legend = 'brief')
fig.set_title('Rilevazioni Aprile-Giugno')
```

**Figura 4.16.** Codice dello scatterplot della temperatura rilevata dallo Snap 2.257

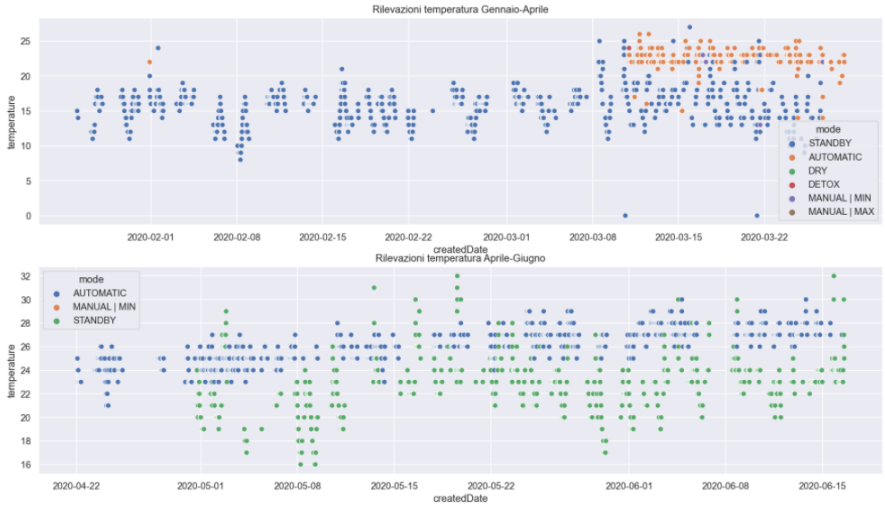


**Figura 4.17.** Scatterplot della temperatura rilevata dallo Snap 2.257

Marzo e 21 Marzo; in questi due casi la temperatura assume valore 0. Nel secondo periodo di rilevazioni (dal 22 Aprile al 17 Giugno), le temperature tendono a salire.

Per concludere le analisi relative a questo campo, visualizziamo un boxplot che metta in relazione le modalità di utilizzo, la temperatura e la velocità della ventola. Questo è possibile grazie all'istruzione `boxplot` (Figura 4.18), messa a disposizione da `seaborn`.

```
sns.set(rc={'figure.figsize':(11.7, 8.27)})
sns.set_theme(style="whitegrid")
sns.boxplot(data=snap2257, x="currentMode", y="temperature", hue='fanLevel')
```

**Figura 4.18.** Codice del boxplot dello Snap 2.257

Dal grafico di Figura 4.19 notiamo che, data la scarsità di valori, l'estremo superiore e inferiore del boxplot, in corrispondenza delle modalità Manuale a velocità massima, Detox e Dry coincidono. In corrispondenza delle modalità Standby, Automatico e Manuale a velocità minima, sono presenti molti più valori, tra i quali vengono individuati degli outlier (Figura 4.20).

Per comprendere meglio l'origine di tali outlier, grafichiamo un subplot (Figura 4.21) costituito da due trame. Nella prima, visualizziamo lo scatterplot dei valori della temperatura rilevata, con le corrispondenti modalità di utilizzo. Nella

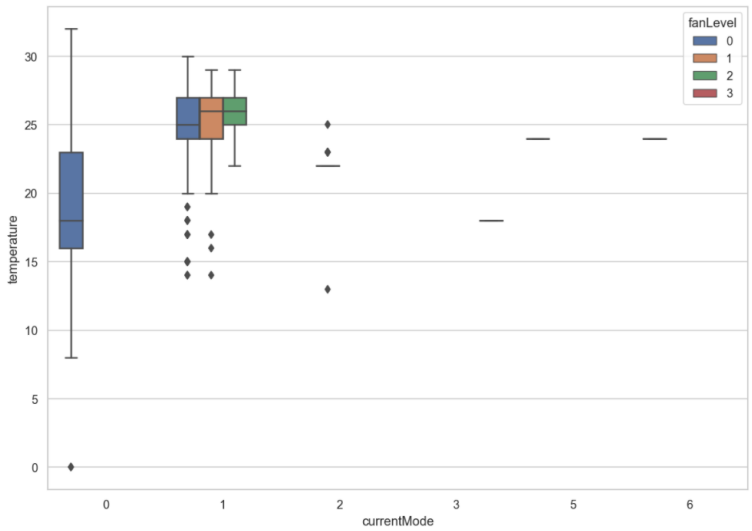


**Figura 4.19.** Il boxplot della temperatura rilevata dallo Snap 2.257

```
outliers_t1 = snap2257.loc[(snap2257['temperature']==0) & (snap2257['currentMode']==0) & (snap2257['fanLevel']==0)]
outliers_t2 = snap2257.loc[(snap2257['temperature']<20) & (snap2257['currentMode']==1) & (snap2257['fanLevel']==0)]
outliers_t3 = snap2257.loc[(snap2257['temperature']<20) & (snap2257['currentMode']==1) & (snap2257['fanLevel']==1)]
outliers_t4 = snap2257.loc[(snap2257['temperature']<20) & (snap2257['currentMode']==2)]
outliers_t5 = snap2257.loc[(snap2257['temperature']>22) & (snap2257['currentMode']==2)]
```

**Figura 4.20.** Gli outlier della temperatura rilevata dallo Snap 2.257

seconda, visualizziamo lo scatterplot di tutte le temperature registrate dal sensore, evidenziando soltanto gli outlier individuati.

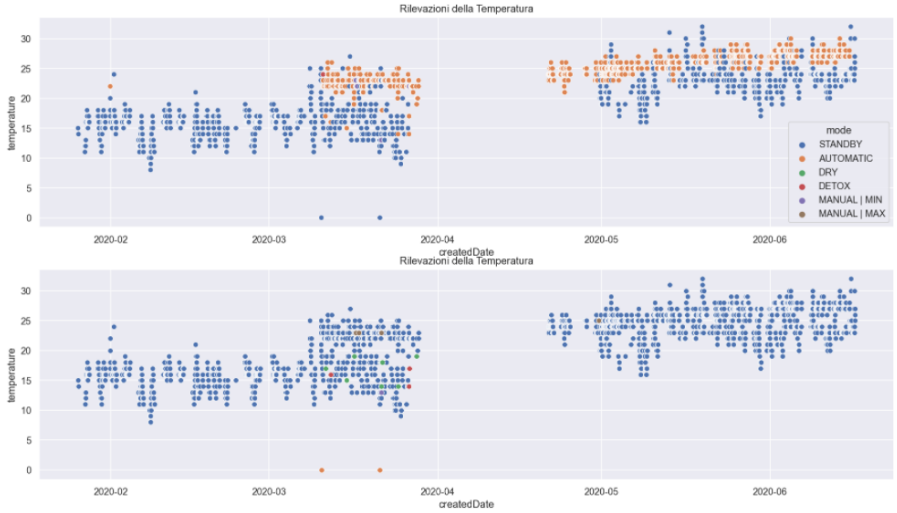


**Figura 4.21.** Il confronto tra gli scatterplot della temperatura rilevata dallo Snap 2.257

Confrontiamo questi due grafici. Innanzitutto notiamo i due outlier in 0, già individuati precedentemente, che potrebbero essere dovuti ad un malfunzionamento. Tutti gli altri outlier, sono definiti come tali in quanto, in corrispondenza di essi, la modalità di utilizzo con cui lavora lo Snap è insolita. Per esempio, l'outlier in corrispondenza della temperatura di 14 gradi è una rilevazione effettuata dallo Snap in modalità Automatica, ma, come affermato precedentemente, il dispositivo lavora in modalità Standby ad una temperatura al di sotto dei 20 gradi fino alla fine del mese di Marzo; dunque, la temperatura di 17 gradi dovrebbe essere rilevata in modalità Standby.

### 4.2.3 Il sensore di umidità

L'analisi sull'umidità rilevata dall'*Umidity Sensor* dello Snap 2.257, analogamente a quanto svolto per la temperatura, inizia dalla sua distribuzione (Figura 4.22).

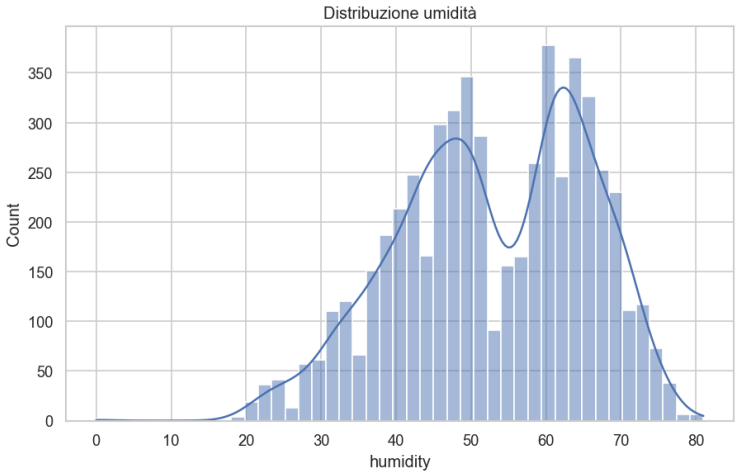


**Figura 4.22.** La distribuzione dell'umidità rilevata dallo Sanp 2.257

Da questo grafico notiamo che l'umidità relativa rilevata con maggior frequenza è intorno al 62%.

Il passo successivo è quello di graficare, attraverso uno scatterplot, la relazione tra le modalità di utilizzo e l'umidità relativa registrata (Figura 4.23).

L'umidità relativa ideale in un ambiente chiuso è compresa tra il 40 e il 65%; il nostro dispositivo rileva un'umidità fino all'80%; però, in corrispondenza di questi valori alti, la modalità di utilizzo risulta sempre Standby e mai Detox. Solo una rilevazione è stata fatta in modalità Detox, e l'umidità relativa è pari al 45%. Notiamo che il livello di umidità relativa in corrispondenza della modalità Automatica, si aggira, prevalentemente, intorno ai valori minori o uguali al 50%. In più, anche in questo scatterplot, osserviamo che nei giorni 10 Marzo e 21 Marzo, i valori rilevati sono pari a 0.


**Figura 4.23.** Scatterplot dell'umidità rilevata dallo Snap 2.257

Per concludere visualizziamo le relazioni tra le modalità di utilizzo, l'umidità e la velocità della ventola, attraverso un boxplot (Figura 4.24), mettendo in evidenza i relativi outlier (Figura 4.25).

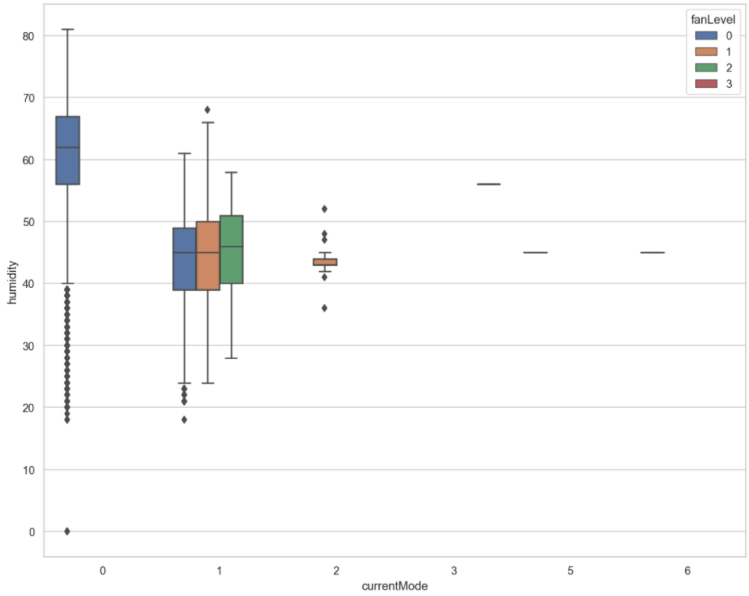


**Figura 4.24.** Il boxplot dell'umidità rilevata dallo Snap 2.257

Per comprendere meglio l'origine di tali outlier, grafichiamo un subplot (Figu-

```
outliers_h1 = snap2257.loc[(snap2257['humidity']<40) & (snap2257['currentMode']==0) & (snap2257['fanLevel']==0)]
outliers_h2 = snap2257.loc[(snap2257['humidity']<24) & (snap2257['currentMode']==1) & (snap2257['fanLevel']==0)]
outliers_h3 = snap2257.loc[(snap2257['humidity']==68) & (snap2257['currentMode']==1) & (snap2257['fanLevel']==1)]
outliers_h4 = snap2257.loc[(snap2257['humidity']>45) & (snap2257['currentMode']==2)]
outliers_h5 = snap2257.loc[(snap2257['humidity']<42) & (snap2257['currentMode']==2)]
```

**Figura 4.25.** Gli outlier dell'umidità rilevata dallo Snap 2.257

ra 4.26) in cui confrontiamo l'andamento dell'umidità nel corso delle rilevazioni, evidenziando le modalità di utilizzo, e l'andamento dell'umidità nel corso delle rilevazioni, i cui outlier sono ben evidenti. Oltre gli outlier in 0, notiamo che, come per la temperatura, anche l'umidità presenta delle anomalie in quanto, in corrispondenza di tali valori, le modalità di utilizzo con cui lavora lo Snap sono inusuali. Infatti, nel primo periodo di rilevazioni, l'umidità relativa al di sotto del 40% viene considerata come outlier perché lo Snap effettua tali rilevazioni in modalità Standby; tuttavia, dato l'andamento generale, dovrebbe rilevarle in modalità Automatica.


**Figura 4.26.** Il confronto tra gli scatterplot dell'umidità rilevata dallo Snap 2.257

### 4.2.4 Il sensore VOCs e della qualità dell'aria

In questa sezione valutiamo i VOCs rilevati dall'*Air Quality Sensor* dello Snap 2.257 e, come prima analisi, esaminiamo la distribuzione di tali fattori inquinanti (Figura 4.27).

Da questo grafico notiamo un picco di frequenza intorno al valore 5.700, misura decimale dei gas organici.

Come per l'umidità e la temperatura, grafichiamo, per mezzo di uno scatterplot, la relazione esistente tra le modalità di utilizzo e i VOCs rilevati (Figura 4.28).

Innanzitutto, dal grafico notiamo che la correlazione tra il ping e la modalità di utilizzo non è così evidente come lo era tra la modalità di utilizzo e la temperatura, e

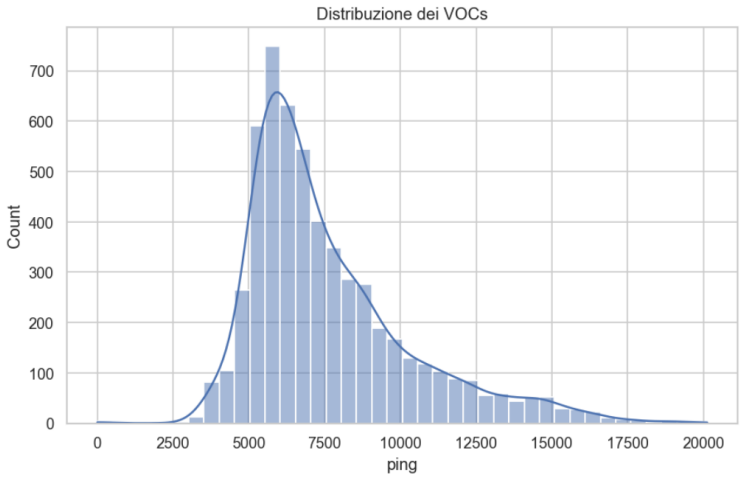


**Figura 4.27.** La distribuzione dei VOCs

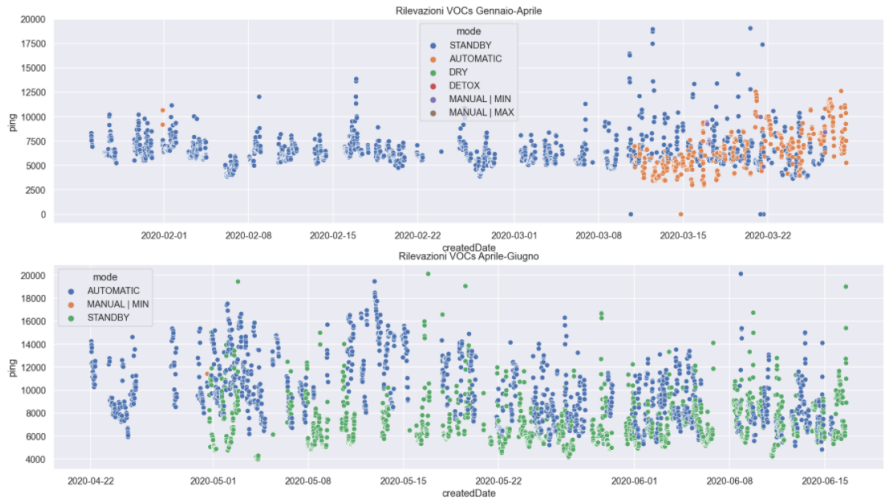


**Figura 4.28.** Scatterplot dei VOCs rilevati dallo Snap 2.257

tra la modalità di utilizzo e l'umidità. A confermarlo è la heatmap precedentemente illustrata, in quanto il coefficiente di Person ottenuto dalla correlazione tra currentMode e ping risulta più piccolo rispetto a quello tra currentMode e temperature, e tra currentMode e humidity. Nonostante questo, osserviamo che i rilevamenti di VOCs compresi all'incirca tra 5.000 e 10.000 sono stati effettuati prevalentemente in modalità Standby. Inoltre, sono stati rilevati quattro valori nulli, due dei quali corrispondono allo stesso giorno e alla stessa ora in cui la temperatura e l'umidità risultano anch'esse nulle.

Visualizziamo, ora, le relazioni tra le modalità di utilizzo, i VOCs e la velocità della ventola attraverso un boxplot (Figura 4.29), mettendo in evidenza i relativi

outlier (Figura 4.30).

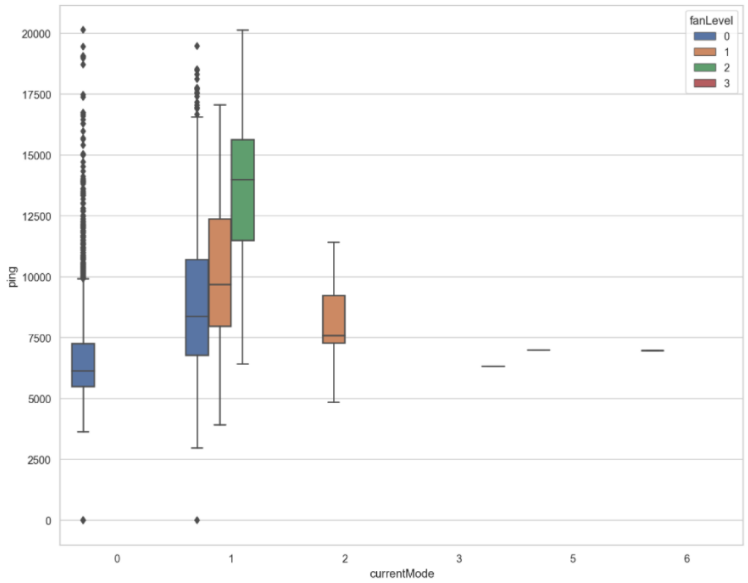


**Figura 4.29.** Il boxplot dei VOCs rilevati in corrispondenza del currentMode dello Snap 2.257

```
outliers_q1 = snap2257.loc[(snap2257['ping']==0) & (snap2257['currentMode']==0) & (snap2257['fanLevel']==0)]
outliers_q2 = snap2257.loc[(snap2257['ping']>=10000) & (snap2257['currentMode']==0) & (snap2257['fanLevel']==0)]
outliers_q3 = snap2257.loc[(snap2257['ping']==0) & (snap2257['currentMode']==1) & (snap2257['fanLevel']==0)]
outliers_q4 = snap2257.loc[(snap2257['ping']>16500) & (snap2257['currentMode']==1) & (snap2257['fanLevel']==0)]
```

**Figura 4.30.** Gli outliers dei VOCs rilevati in corrispondenza del currentMode dello Snap 2.257

Per comprendere meglio l'origine di tali outlier, visualizziamo un subplot (Figura 4.31) in cui confrontiamo l'andamento nel tempo dei VOCs con correlate le modalità di utilizzo, e l'andamento nel tempo dei VOCs in cui vengono messi in evidenza i valori anomali.

Oltre i valori in zero, che potrebbero essere stati provocati da un malfunzionamento, vengono considerati outlier quei valori superiori a 10.000, le cui rilevazioni sono state effettuate in modalità Standby. Tra questi valori notiamo alcuni vicino, se non uguali, a 20.000, e, nonostante l'ambiente risulti più inquinato, la modalità utilizzato non è mai Detox; solo una rilevazione è stata fatta con questa modalità e il ping è uguale a 6.997.

Concludiamo l'analisi visualizzando la correlazione tra il campo ping e il campo airQuality del nostro dataframe per mezzo di un boxplot (Figura 4.32). Poiché il valore di airQuality ci dice (in un range 1-5) quanto è contaminato l'ambiente in


**Figura 4.31.** Il confronto tra gli scatterplot dei VOCs rilevati in corrispondenza del currentMode

modo qualitativo, tramite i valori in decimale dei gas organici rilevati, mostriamo le relative corrispondenze in termini quantitativi.

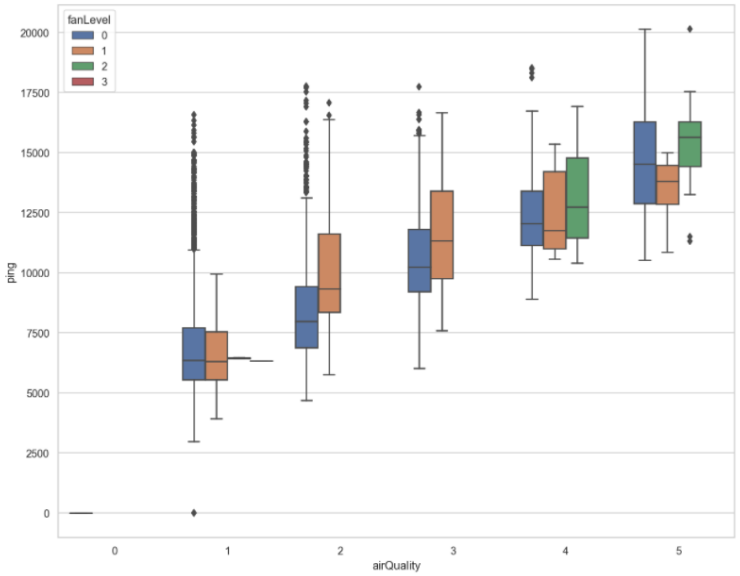


**Figura 4.32.** Il boxplot dei VOCs rilevati in corrispondenza dell'airQuality

Dal boxplot si evidenziano outlier (Figura 4.33) soprattutto in corrispondenza dell'airQuality pari a 1 e 2; infatti, vengono registrati dati il cui valore di gas organico

nell'ambiente è troppo alto per poter poi definire quell'ambiente ottimale o poco contaminato.

```
outliers_qa_1 = snap2257.loc[(snap2257['ping']==0) & (snap2257['airQuality']==1) & (snap2257['fanLevel']==0)]
outliers_qa_2 = snap2257.loc[(snap2257['ping']>=11000) & (snap2257['airQuality']==1) & (snap2257['fanLevel']==0)]
outliers_qa_3 = snap2257.loc[(snap2257['ping']>=13700) & (snap2257['airQuality']==2) & (snap2257['fanLevel']==0)]
outliers_qa_4 = snap2257.loc[(snap2257['ping']>=16250) & (snap2257['airQuality']==2) & (snap2257['fanLevel']==1)]
outliers_qa_5 = snap2257.loc[(snap2257['ping']>16000) & (snap2257['airQuality']==3) & (snap2257['fanLevel']==0)]
outliers_qa_6 = snap2257.loc[(snap2257['ping']>17500) & (snap2257['airQuality']==4) & (snap2257['fanLevel']==0)]
outliers_qa_7 = snap2257.loc[(snap2257['ping']>17500) & (snap2257['airQuality']==5) & (snap2257['fanLevel']==2)]
outliers_qa_8 = snap2257.loc[(snap2257['ping']<12500) & (snap2257['airQuality']==5) & (snap2257['fanLevel']==2)]
```

**Figura 4.33.** Gli outliers dei VOCs rilevati in corrispondenza dell'airQuality

Dunque, non notiamo una netta differenza tra i valori di ping assunti in corrispondenza del sensore di airQuality che indica un ambiente favorevole e i valori di ping assunti in corrispondenza del sensore di airQuality che indica un ambiente più contaminato. Nonostante questo, è presente una certa "scala di valori" ed è così giustificata la moderata correlazione evidenziata nella heatmap.

Spostando la nostra attenzione sul campo fanLevel, notiamo che la velocità della ventola dovrebbe essere più alta man mano che i valori di ping si alzano; tuttavia, nella maggior parte dei casi, la ventola ha una velocità pari a 0. Per visualizzare meglio questa debole correlazione, abbiamo realizzato un `relplot` (Figura 4.34) che mostra nel tempo la relazione esistente tra i campi ping, AirQuality e fanLevel.

## 4.3 Seconda analisi: Snap con errori

Nel nostro dataset, come già anticipato nella Sezione 3.5, sono presenti 185 Snap con errori. Abbiamo realizzato un codice per ottenere un dataframe che contenesse solo questa tipologia di Snap, chiamato `snap_ce` (Figura 4.35).

In questo codice definiamo:

- `snap_err`, un dataframe contenente tutti gli Snap con errore diverso da 0;
- `un_snap_err`, un vettore contenente gli `idSnap` del dataframe `snap_err`;
- `snap_ce`, un dataframe vuoto, con gli stessi campi del dataset iniziale.

Implementiamo, quindi, un ciclo for che scorra tutti gli elementi del vettore `un_snap_err` e, ad ogni iterazione, crei un dataframe temporaneo le cui righe contengano l'idSnap i-esimo e lo salvi in `snap_ce`.

Visualizziamo le statistiche descrittive del dataframe contenente soltanto Snap con errori (Figura 4.36).

In particolare notiamo che il valore minimo di temperatura e umidità è -1, un dato molto insolito e mai incontrato fino ad ora. Considerando il dataframe `snap_err` possiamo visualizzare, attraverso un istogramma, quali sono gli errori maggiormente riscontrati in esso (Figura: 4.37).

Il passo successivo sarà quello di scegliere uno Snap per ogni errore riscontrato nel dataset e valutare quali siano stati gli effetti di ciascun errore sul dispositivo considerato.


**Figura 4.34.** Relazione tra ping, AirQuality e fanLevel dello Snap 2.257

```
snap_err = df.loc[df['errors'] != 0]
un_snap_err = snap_err['idSnap'].unique()
snap_ce=df[0:0]

for snap in un_snap_err[::]:
    tmp = df.loc[df['idSnap'] == snap]
    snap_ce = snap_ce.append(tmp)
```

**Figura 4.35.** Il codice per realizzare un dataframe con Snap con errori

### 4.3.1 La scelta degli Snap

Per scegliere gli Snap abbiamo realizzato un codice per ogni tipologia di errore. Ad esempio, prendiamo in considerazione l'errore 2; il codice realizzato è riportato in Figura 4.38.

In questo codice definiamo:

- `un_snap`, un vettore contenente tutti gli *idSnap* che compaiono nel nostro dataset;
- `var`, un vettore in cui salveremo tutti gli Snap con errore 0;

```
snap_ce.describe()
```

| | idSnap | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping |
|---|---|---|---|---|---|---|---|---|---|
| count | 527296.000000 | 527296.000000 | 527296.000000 | 527296.000000 | 527296.000000 | 527296.000000 | 527296.000000 | 527296.000000 | 527296.000000 |
| mean | 1522.627524 | 1.107497 | 0.112663 | 22.416737 | 47.139315 | 1.218031 | 0.362345 | 0.130196 | 3613.182194 |
| std | 614.695493 | 1.602992 | 0.396884 | 3.473514 | 10.892533 | 0.605018 | 2.232577 | 0.601552 | 3338.004810 |
| min | 170.000000 | 0.000000 | 0.000000 | -1.000000 | -1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 1058.000000 | 0.000000 | 0.000000 | 20.000000 | 40.000000 | 1.000000 | 0.000000 | 0.000000 | 1528.000000 |
| 50% | 1543.000000 | 1.000000 | 0.000000 | 23.000000 | 48.000000 | 1.000000 | 0.000000 | 0.000000 | 2791.000000 |
| 75% | 2081.000000 | 1.000000 | 0.000000 | 25.000000 | 55.000000 | 1.000000 | 0.000000 | 0.000000 | 4815.000000 |
| max | 2616.000000 | 6.000000 | 3.000000 | 40.000000 | 99.000000 | 5.000000 | 16.000000 | 9.000000 | 32767.000000 |

**Figura 4.36.** Le statistiche descrittive di snap_ce

```
sns.set(rc={'figure.figsize':(8.7, 5.27)})
sns.set(style = 'whitegrid')

sns.catplot(data=snap_err, x="errors", kind='count').set(title='Errors')
```

```
<seaborn.axisgrid.FacetGrid at 0x1940fd49910>
```

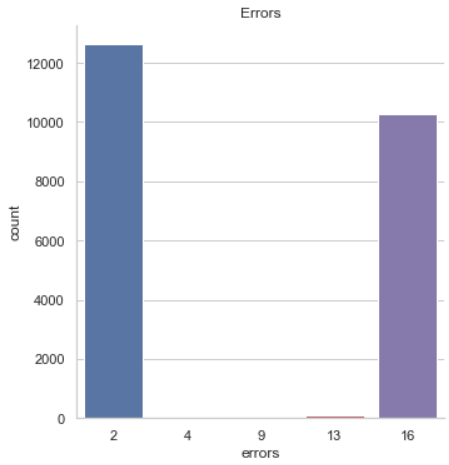


**Figura 4.37.** Gli errori riscontrati nel dataset

```
var = []
un_snap = df['idSnap'].unique()
des_err =  [0, 2]

for snap in un_snap[::]:
    tmp = df.loc[df['idSnap'] == snap]
    err_snap = tmp.errors.unique()
    if (len(err_snap) == len(des_err)):
        if(sorted(err_snap) == sorted(des_err)):
            var.append(snap)
            for snap in var:
                tmp1 = df.loc[df['idSnap'] == snap]
                tmp2 = tmp1.loc[tmp1['errors'] == 2]
            print("id_snap " + str(snap) + " ha "+ str(len(tmp1)) + " valori e ha " + str(len(tmp2)) + " errori '2' \n ")
```

**Figura 4.38.** Il codice per la scelta dello Snap con errore 2

- `des_err`, un vettore che contenga gli errori desiderati.

Implementiamo due cicli for. Il ciclo esterno, che scorre tutti gli elementi del vettore `un_snap`, ad ogni iterazione crea un dataframe temporaneo le cui righe contengono l'idSnap i-esimo, e un vettore `err_snap` che contenga gli elementi del cam-

po `errors` del dataframe temporaneo. Se la lunghezza del vettore appena creato è uguale alla lunghezza del vettore `des_err`, allora valutiamo se anche gli elementi di questi due vettori sono gli stessi; in caso affermativo salviamo il nostro *snap* nel vettore `var`. Il ciclo interno, che scorre tutti gli elementi del vettore `var`, salva nel dataframe temporaneo `tmp_1` l'idSnap del dataset originario e nel dataframe temporaneo `tmp_2` gli errori pari a 2 contenuti nel dataframe `tmp_1`.

L'output di questo codice è dato da una lista in cui, per ogni Snap, viene stampato il numero di elementi e il numero di errori (Figura 4.39).

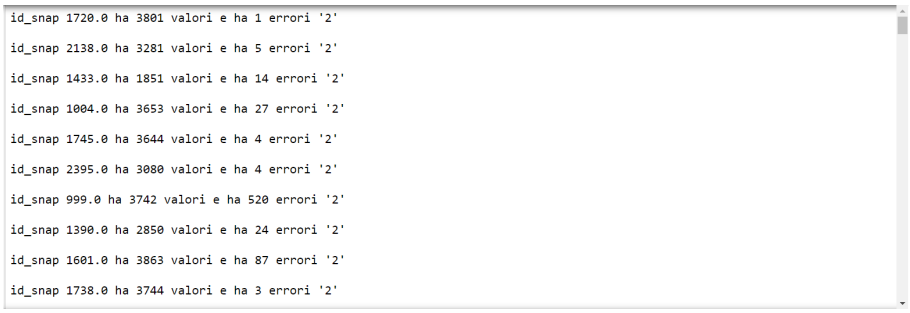


**Figura 4.39.** Output del codice per la scelta dello Snap con errore 2

Tra questi Snap verrà scelto quello che possiede il maggior numero di errori.

Il codice appena descritto verrà applicato anche per gli altri errori; basterà cambiare il vettore `des_err` e il dataframe temporaneo `tmp_2`.

Gli Snap scelti sono:

- *1.347*, per l'errore 2;
- *1.838*, per l'errore 16;
- *1.320*, per l'errore 4;
- *1.872*, per l'errore 9 e 13.

## 4.4 Lo Snap 1.347

Innanzitutto, salviamo tutte le righe riferite allo Snap 2.257 in un nuovo dataframe, chiamato `snap1347`. Successivamente, eliminiamo la colonna `idSnap` e stampiamo le sue prime cinque righe. Il tutto è mostrato nella Figura 4.40.

Visualizziamo le statistiche descrittive del nostro dataframe (Figura 4.41).

La temperatura e l'umidità si aggirano intorno a valori più che normali, mentre il ping varia da 0 a 32.767, il valore più alto di gas organico riscontrato fino ad ora. Visualizziamo l'andamento nel tempo di questi tre campi (Figura 4.42).

Il ping ha un andamento molto particolare.

Visualizziamo la heatmap (Figura 4.43) per valutare le correlazioni tra le variabili.

Le uniche coppie di variabili moderatamente correlate sono *airQuality e ping* e *errors e ping*. Tutte le altre coppie di variabili hanno una correlazione molto bassa, compresa la coppia *errors* ed *airQuality*.

```
snap1347 = df.loc[df['idSnap'] == 1347.0]
snap1347 = snap1347.drop(['idSnap'], axis=1)

snap1347.head()
```

| | createdDate | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping | mode |
|---|---|---|---|---|---|---|---|---|---|---|
| **106** | 2020-01-25 23:26:30.0 | 0 | 0 | 18 | 53 | 4 | 2 | 0 | 32469 | STANDBY |
| **576** | 2020-01-25 23:36:30.0 | 0 | 0 | 18 | 53 | 1 | 2 | 0 | 2750 | STANDBY |
| **1058** | 2020-01-25 23:46:30.0 | 0 | 0 | 18 | 53 | 1 | 2 | 0 | 21710 | STANDBY |
| **1528** | 2020-01-25 23:56:30.0 | 0 | 0 | 18 | 54 | 5 | 2 | 0 | 32674 | STANDBY |
| **7165** | 2020-01-26 05:04:37.0 | 0 | 0 | 18 | 56 | 1 | 0 | 0 | 322 | STANDBY |

**Figura 4.40.** Lo Snap 1.347

```
snap1347.describe()
```

| | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping |
|---|---|---|---|---|---|---|---|---|
| **count** | 3724.000000 | 3724.000000 | 3724.000000 | 3724.000000 | 3724.000000 | 3724.000000 | 3724.0 | 3724.000000 |
| **mean** | 0.000806 | 0.000806 | 20.074919 | 48.057734 | 1.472342 | 1.440387 | 0.0 | 11329.932331 |
| **std** | 0.049161 | 0.049161 | 2.889128 | 6.542503 | 1.204583 | 0.897929 | 0.0 | 12579.107028 |
| **min** | 0.000000 | 0.000000 | 11.000000 | 24.000000 | 1.000000 | 0.000000 | 0.0 | 0.000000 |
| **25%** | 0.000000 | 0.000000 | 18.000000 | 44.000000 | 1.000000 | 0.000000 | 0.0 | 393.750000 |
| **50%** | 0.000000 | 0.000000 | 20.000000 | 49.000000 | 1.000000 | 2.000000 | 0.0 | 4546.500000 |
| **75%** | 0.000000 | 0.000000 | 22.000000 | 53.000000 | 1.000000 | 2.000000 | 0.0 | 23286.250000 |
| **max** | 3.000000 | 3.000000 | 27.000000 | 67.000000 | 5.000000 | 2.000000 | 0.0 | 32767.000000 |

**Figura 4.41.** Le statistiche descrittive dello Snap 1.347

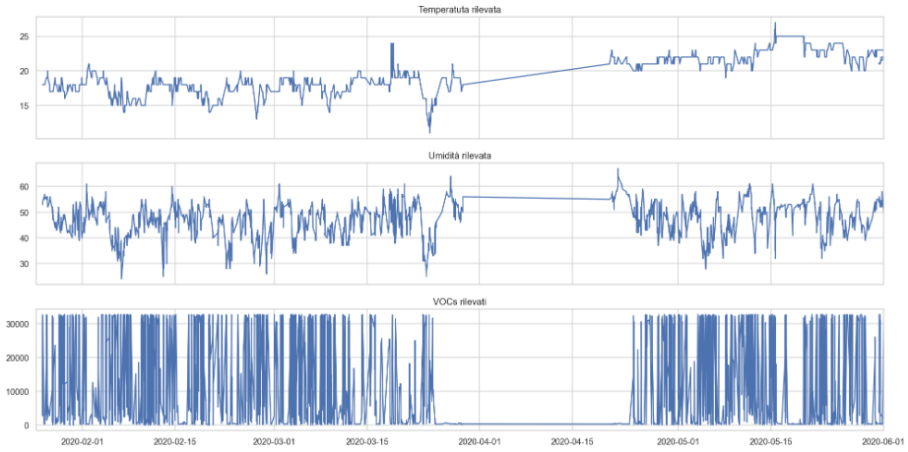


**Figura 4.42.** Gli andamenti nel tempo di temperatura, umidità e VOCs rilevati dallo Snap 1.347

```
1 plt.figure(figsize=(14,7))
2 correlation = snap1347.corr()
3 sns.heatmap(correlation, xticklabels=correlation.columns, yticklabels=correlation.columns,annot=True)
```

```
<AxesSubplot:>
```


**Figura 4.43.** La heatmap dello Snap 1.347

### 4.4.1 Gli effetti dell'errore "2" sullo Snap

L'errore "2" indica che il valore registrato dal sensore di *airQuality* risulta fuori range. Visualizziamo, attraverso uno scatterplot (Figura 4.44), la relazione esistente tra *errors* e *ping*.

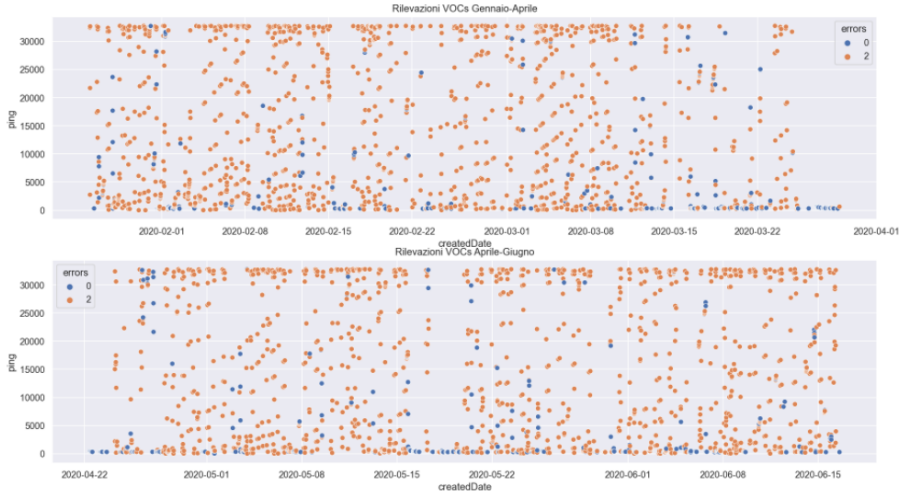


**Figura 4.44.** Scatterplot dei VOCs rilevati dallo Snap 1.347

Questo Snap è molto particolare perché gli errori, oltre a prevalere, non pre-

sentano una vera e propria relazione con i VOCs, nonostante il loro coefficiente di correlazione sia moderato. Quasi tutti i valori assunti dal ping sono soggetti all'errore "2".

Visualizziamo un boxplot (Figura 4.45) che metta in relazione il ping, gli errori e il sensore di airQuality.

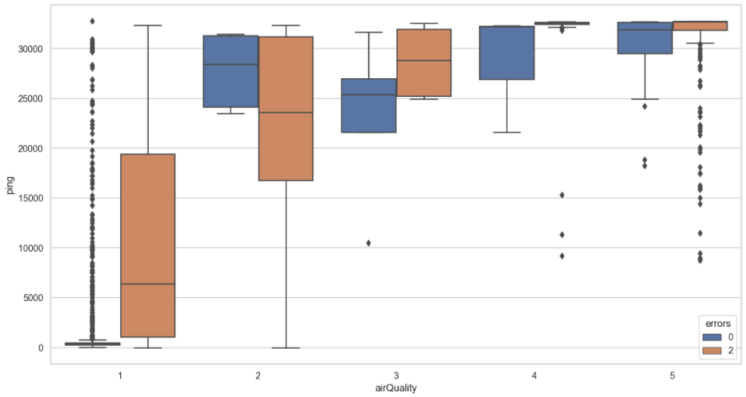


**Figura 4.45.** Il boxplot dello Snap 1.347

Notiamo che viene considerato come ambiente ottimale, un ambiente il cui valore di VOCs nell'aria è minore all'incirca di 1.200, mentre tutti gli altri valori non registrati come outlier (Figura 4.46) vengono considerati errori. Ma, osservando l'airQuality pari a 2, notiamo che, in corrispondenza di tale sensore, vengono attribuiti valori di ping molto alti, non registrati come errori e che dovrebbero essere valutati in corrispondenza di un ambiente molto contaminato. Il sensore di qualità dell'aria dello Snap 1.347 non distingue in modo adeguato un ambiente ottimale da uno molto contaminato.

```
outliers_q1 = snap1347.loc[(snap1347['ping']>1250) & (snap1347['airQuality']==1) & (snap1347['errors']==0)]
outliers_q2 = snap1347.loc[(snap1347['ping']<2000) & (snap1347['airQuality']==3) & (snap1347['errors']==0)]
outliers_q3 = snap1347.loc[(snap1347['ping']<32500) & (snap1347['airQuality']==4) & (snap1347['errors']==2)]
outliers_q4 = snap1347.loc[(snap1347['ping']<25000) & (snap1347['airQuality']==5) & (snap1347['errors']==0)]
outliers_q5 = snap1347.loc[(snap1347['ping']<31000) & (snap1347['airQuality']==5) & (snap1347['errors']==2)]
```

**Figura 4.46.** Gli outlier dello Snap 1.347

## 4.5 Lo Snap 1.838

Innanzitutto, salviamo tutte le righe riferite allo Snap 1.838 in un nuovo dataframe, che chiamiamo `snap1838`. Successivamente, eliminiamo la colonna `idSnap` e stampiamo le sue prime cinque righe. Il tutto è mostrato nella Figura 4.47.

Visualizziamo le statistiche descrittive del nostro dataframe (Figura 4.48).

```
snap1838 = df.loc[df['idSnap'] == 1838.0]
snap1838 = snap1838.drop(['idSnap'], axis=1)

snap1838.head()
```

| | createdDate | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping | mode |
|---|---|---|---|---|---|---|---|---|---|---|
| **61643** | 2020-01-28 01:04:23.0 | 0 | 0 | 20 | 46 | 1 | 0 | 1 | 15485 | STANDBY |
| **62095** | 2020-01-28 01:14:59.0 | 0 | 0 | 20 | 49 | 1 | 0 | 1 | 14357 | STANDBY |
| **62530** | 2020-01-28 01:24:59.0 | 0 | 0 | 20 | 45 | 5 | 0 | 1 | 12556 | STANDBY |
| **62967** | 2020-01-28 01:34:59.0 | 0 | 0 | 20 | 43 | 5 | 0 | 1 | 11083 | STANDBY |
| **63399** | 2020-01-28 01:44:59.0 | 0 | 0 | 20 | 42 | 5 | 0 | 1 | 9751 | STANDBY |

**Figura 4.47.** Lo Snap 1.838

```
snap1838.describe()
```

| | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping |
|---|---|---|---|---|---|---|---|---|
| **count** | 3435.000000 | 3435.000000 | 3435.000000 | 3435.000000 | 3435.000000 | 3435.000000 | 3435.000000 | 3435.000000 |
| **mean** | 1.631732 | 0.070160 | 22.595051 | 41.399709 | 1.214556 | 6.264920 | 1.002329 | 5960.512664 |
| **std** | 1.779906 | 0.338765 | 1.942064 | 7.795960 | 0.534993 | 7.810714 | 0.136498 | 1305.093299 |
| **min** | 0.000000 | 0.000000 | 18.000000 | 25.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 |
| **25%** | 1.000000 | 0.000000 | 21.000000 | 36.000000 | 1.000000 | 0.000000 | 1.000000 | 5037.000000 |
| **50%** | 1.000000 | 0.000000 | 22.000000 | 41.000000 | 1.000000 | 0.000000 | 1.000000 | 5746.000000 |
| **75%** | 1.000000 | 0.000000 | 24.000000 | 46.000000 | 1.000000 | 16.000000 | 1.000000 | 6615.000000 |
| **max** | 5.000000 | 3.000000 | 30.000000 | 82.000000 | 5.000000 | 16.000000 | 9.000000 | 18279.000000 |

**Figura 4.48.** Le statistiche descrittive dello Snap 1.838

Notiamo che i valori rilevati dai sensori sono più che normali e possiamo visualizzare l'andamento nel tempo di temperatura, umidità e VOCs (Figura 4.49).


**Figura 4.49.** Gli andamenti nel tempo di temperatura, umidità e VOCs rilevati dallo Snap 1.838

Visualizziamo la heatmap (Figura 4.50) per valutare le correlazioni tra le variabili.

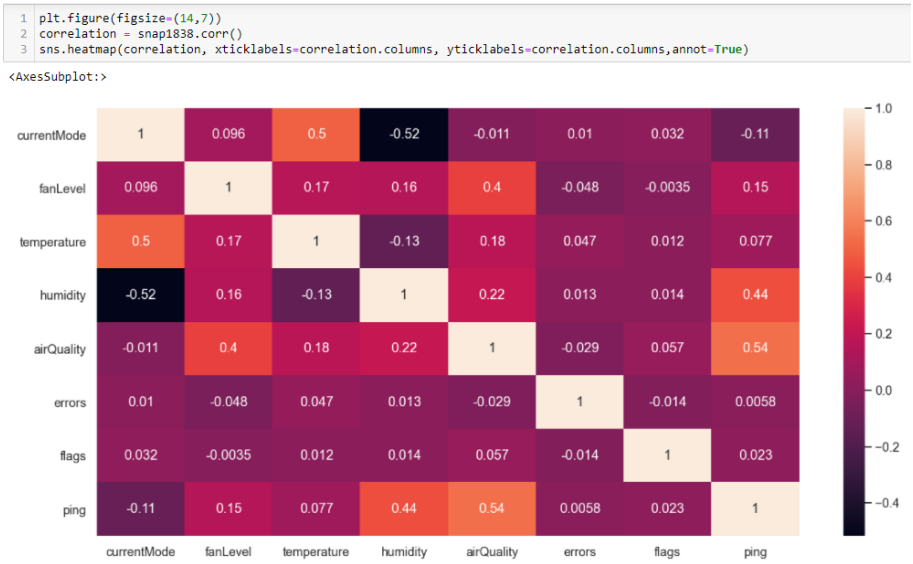


**Figura 4.50.** La heatmap dello Snap 1.838

Notiamo una moderata correlazione tra

- *humidity* e *currentMode*;
- *temperature* e *currentMode*;
- *ping* e *airQuality*.

Non c'è, però, correlazione tra *ping* ed *errors* e *airQuality* ed *errors*.

### 4.5.1 Gli effetti dell'errore "16" sullo Snap

Una particolarità dello Snap 1.838, fino ad ora non ancora incontrata in nessuno Snap, è che lavora anche in modalità Link. La modalità Link si attiva nel momento in cui lo Snap non è in grado di filtrare bene l'ambiente e, per questo, automaticamente, interviene la cappa. Visualizziamo le rilevazioni effettuate con questa modalità (Figura 4.51) e che abbiamo salvato in un dataframe chiamato `link`.

Notiamo che due di queste rilevazioni possiedono un *createdDate* errato perché appartengono ad un range temporale diverso da quello del nostro progetto e, per questo, non le consideriamo. Osservando gli altri valori del dataframe `link`, notiamo che tutti i flag sono pari ad 1, e questo significa che la connessione con la cappa è abilitata. Ma, attraverso un'analisi più approfondita, abbiamo scoperto che tutti i valori di flag sono pari a 1, tranne uno che è pari a 9 (è abilitata la modalità Link e relaxMode). Nonostante questo, l'errore "16", che indica che il tempo concesso per la connessione con la cappa è trascorso e la richiesta in quel momento risulta scaduta, è presente in modo molto frequente e anche in corrispondenza di quei

```
link = snap1838.loc[snap1838['currentMode']==4]
link
```

| | createdDate | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping | mode |
|---|---|---|---|---|---|---|---|---|---|---|
| **1663846** | 1899-12-31 01:30:13 | 4 | 1 | 25 | 41 | 2 | 16 | 1 | 6547 | LINK |
| **1664028** | 1899-12-31 01:40:13 | 4 | 1 | 25 | 40 | 2 | 16 | 1 | 6292 | LINK |
| **236682** | 2020-02-02 21:54:24 | 4 | 1 | 24 | 57 | 3 | 16 | 1 | 8912 | LINK |
| **2672850** | 2020-05-29 21:50:11 | 4 | 0 | 30 | 41 | 3 | 0 | 1 | 8613 | LINK |
| **3129194** | 2020-06-14 15:04:56 | 4 | 1 | 24 | 61 | 1 | 16 | 1 | 9018 | LINK |
| **3188351** | 2020-06-16 13:38:00 | 4 | 0 | 24 | 73 | 5 | 0 | 1 | 13106 | LINK |

**Figura 4.51.** Le rilevazioni dello Snap 1.838 in modalità Link

valori le cui rilevazioni sono state effettuate in modalità Link. Visualizziamo uno scatterplot (Figura 4.52) che metta in evidenza gli errori e i valori di VOCs rilevati.

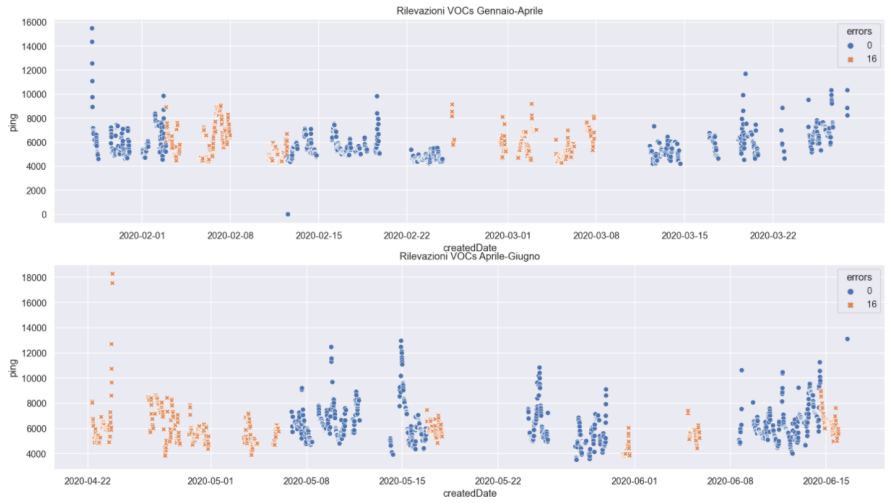


**Figura 4.52.** Scatterplot dei VOCs rilevati dallo Snap 1.838

Come ci aspettavamo dalla heatmap, tra l'errore e i valori di gas organico, non c'è correlazione (il coefficiente di correlazione è quasi vicino allo 0).

La presenza di questo errore prescinde dal funzionamento del sensore di airQuality che è moderatamente correlato con il ping e la cui distribuzione dei valori è mostrata nel boxplot in Figura 4.53.

## 4.6 Lo Snap 1.320

Innanzitutto, salviamo tutte le righe riferite allo Snap 1.320 in un nuovo dataframe, che chiameremo `snap1320`. Successivamente, eliminiamo la colonna `idSnap` e stampiamo le sue prime cinque righe. Il tutto è mostrato nella Figura 4.54.

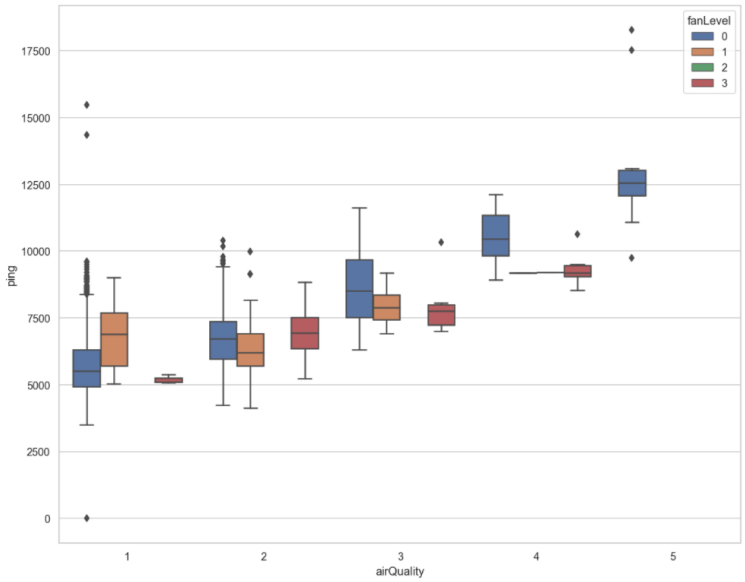


**Figura 4.53.** Il boxplot dello Snap 1.838

```
snap1320 = df.loc[df['idSnap'] == 1320.0]
snap1320 = snap1320.drop(['idSnap'], axis=1)

snap1320.head()
```

| | createdDate | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping | mode |
|---|---|---|---|---|---|---|---|---|---|---|
| **402** | 2020-01-25 23:26:45.0 | 1 | 0 | 28 | 18 | 1 | 0 | 0 | 4759 | AUTOMATIC |
| **880** | 2020-01-25 23:36:45.0 | 1 | 0 | 28 | 18 | 1 | 0 | 0 | 4760 | AUTOMATIC |
| **1348** | 2020-01-25 23:46:45.0 | 1 | 0 | 28 | 18 | 1 | 0 | 0 | 4757 | AUTOMATIC |
| **1674** | 2020-01-25 23:56:45.0 | 1 | 0 | 28 | 18 | 1 | 0 | 0 | 4733 | AUTOMATIC |
| **2260** | 2020-01-26 01:06:45.0 | 1 | 0 | 28 | 18 | 1 | 0 | 0 | 4697 | AUTOMATIC |

**Figura 4.54.** Lo Snap 1.320

Visualizziamo le statistiche descrittive del nostro dataframe (Figura 4.55).

Notiamo che sia la temperatura che l'umidità minime rivelate sono pari a -1 e il valore massimo raggiunto da quest'ultima è pari a 99. Visualizziamo l'andamento nel tempo dei campi di temperatura, umidità e VOCs (Figura 4.56).

Visualizziamo la heatmap (Figura 4.57) per valutare le correlazioni tra le variabili.

In particolare notiamo che il campo *errors* non è correlato con nessun campo.

### 4.6.1 Gli effetti dell'errore "4" sullo Snap

L'errore "4" indica che l'umidità rilevata dal sensore è fuori range. Visualizziamo attraverso uno scatterplot (Figura 4.58) l'umidità rilevata dal sensore ed evidenziamo in corrispondenza di quale valore abbiamo gli errori.

```
1 snap1320.describe()
```

| | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping |
|---|---|---|---|---|---|---|---|---|
| count | 3808.000000 | 3808.000000 | 3808.000000 | 3808.000000 | 3808.000000 | 3808.000000 | 3808.0 | 3808.000000 |
| mean | 1.038866 | 0.026523 | 27.614758 | 23.257353 | 1.075630 | 0.002101 | 0.0 | 4703.729779 |
| std | 0.392418 | 0.160706 | 1.880726 | 9.650142 | 0.277999 | 0.091658 | 0.0 | 1253.293807 |
| min | 1.000000 | 0.000000 | 23.000000 | -1.000000 | 1.000000 | 0.000000 | 0.0 | 1362.000000 |
| 25% | 1.000000 | 0.000000 | 26.000000 | 17.000000 | 1.000000 | 0.000000 | 0.0 | 4010.000000 |
| 50% | 1.000000 | 0.000000 | 28.000000 | 21.000000 | 1.000000 | 0.000000 | 0.0 | 4575.000000 |
| 75% | 1.000000 | 0.000000 | 29.000000 | 27.000000 | 1.000000 | 0.000000 | 0.0 | 5313.000000 |
| max | 5.000000 | 1.000000 | 33.000000 | 54.000000 | 3.000000 | 4.000000 | 0.0 | 10421.000000 |

**Figura 4.55.** Le statistiche descrittive dello Snap 1.320

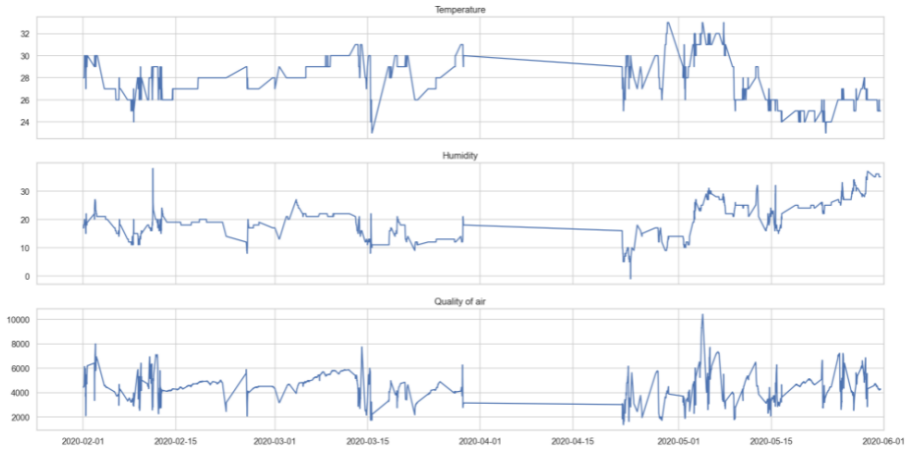


**Figura 4.56.** Gli andamenti nel tempo di temperatura, umidità e VOCs rilevati dallo Snap 1.320

Sono presenti soltanto due errori in corrispondenza del valore di minimo -1 (come già notato nelle statistiche descrittive); tale valore anomalo dal momento che il valore minimo di umidità relativa è pari a 0%. A confermarlo è il boxplot in Figura 4.59 in cui vengono mostrati i relativi outlier.

## 4.7 Lo Snap 1.872

Innanzitutto, salviamo tutte le righe riferite allo Snap 1.872 in un nuovo dataframe, che chiameremo `snap1872`. Successivamente, eliminiamo la colonna `idSnap` e stampiamo le sue prime cinque righe. Il tutto è mostrato nella Figura 4.60.

Visualizziamo le statistiche descrittive del nostro dataframe (Figura 4.61).

Un dato che attira molto la nostra attenzione è il valore di minimo dell'umidità e della temperatura rilevata.

```
plt.figure(figsize=(14,7))
correlation = snap1320.corr()
sns.heatmap(correlation, xticklabels=correlation.columns, yticklabels=correlation.columns,annot=True)
```

```
<AxesSubplot:>
```


**Figura 4.57.** La heatmap dello Snap 1.320

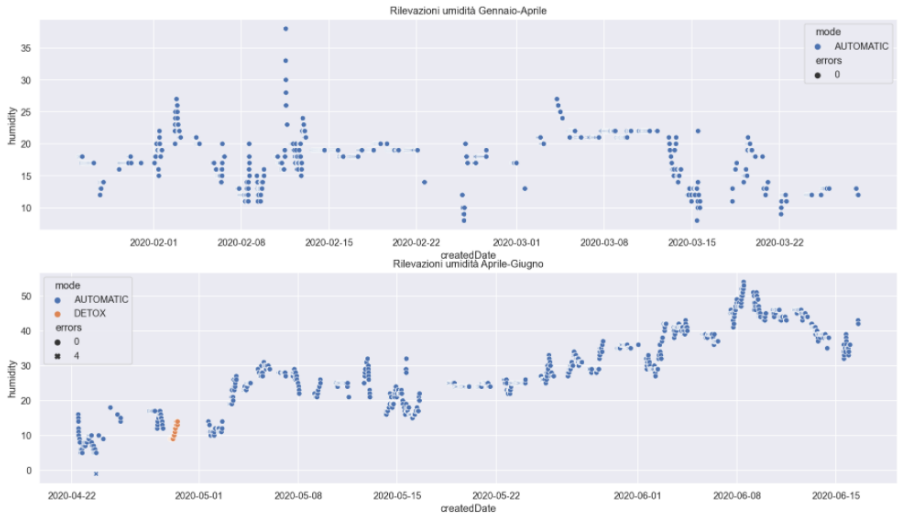


**Figura 4.58.** Scatterplot dell'umidità rilevata dallo Snap 1.320

Visualizziamo l'andamento nel tempo dei campi di temperatura, umidità e VOCs (Figura 4.62).

Visualizziamo la heatmap (Figura 4.63) per valutare le correlazioni tra le variabili.

In particolare notiamo che il campo *errors* è fortemente correlato al campo *temperature* e moderatamente correlato al campo *humidity*.

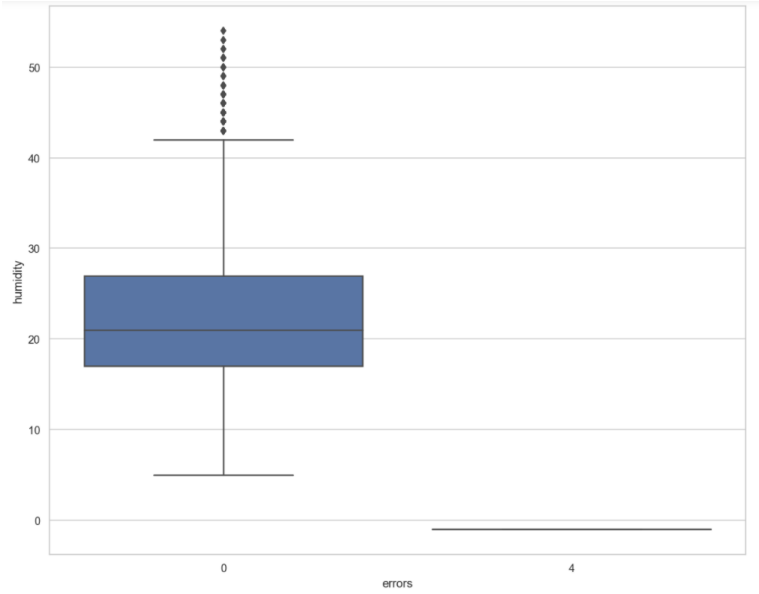


**Figura 4.59.** Il boxplot dello Snap 1.320

```
snap1872 = df.loc[df['idSnap'] == 1872.0]
snap1872 = snap1872.drop(['idSnap'], axis=1)

snap1872.head()
```

| | createdDate | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping | mode |
|---|---|---|---|---|---|---|---|---|---|---|
| **287678** | 2020-02-04 15:06:09.0 | 0 | 0 | 24 | 36 | 1 | 0 | 0 | 6785 | STANDBY |
| **288121** | 2020-02-04 15:16:09.0 | 0 | 0 | 24 | 36 | 1 | 0 | 0 | 6756 | STANDBY |
| **288575** | 2020-02-04 15:26:09.0 | 0 | 0 | 24 | 36 | 1 | 0 | 0 | 6755 | STANDBY |
| **289040** | 2020-02-04 15:36:09.0 | 0 | 0 | 24 | 36 | 1 | 0 | 0 | 6786 | STANDBY |
| **289498** | 2020-02-04 15:46:09.0 | 0 | 0 | 24 | 36 | 1 | 0 | 0 | 6755 | STANDBY |

**Figura 4.60.** Lo Snap 1.872

### 4.7.1 Gli effetti dell'errore "9" e "13" sullo Snap

L'errore "9" viene utilizzato per indicare che il sensore di temperatura rileva un valore fuori range ed è avvenuta una mancata comunicazione I2C con i sensori. L'errore "13", invece, viene utilizzato per indicare che i sensori di temperatura e umidità rilevano un valore di umidità fuori range e che la comunicazione I2C con i sensori non è avvenuta.

Visualizziamo due scatterplot: nel primo (Figura 4.64) evidenziamo le relazioni tra la temperatura rilevata dal sensore e gli errori, nel secondo (Figura 4.65) evidenziamo le relazioni tra l'umidità e gli errori.

Innanzitutto notiamo in entrambi i grafici dei "buchi" nelle rilevazioni. Nel primo grafico notiamo che l'errore "9" e l'errore "13" sono rilevati in corrispondenza del valore di temperatura -1, e questo spiega il dato anomalo. Nel secondo grafico

```
1 snap1872.describe()
```

| | currentMode | fanLevel | temperature | humidity | airQuality | errors | flags | ping |
|---|---|---|---|---|---|---|---|---|
| count | 2359.000000 | 2359.000000 | 2359.000000 | 2359.000000 | 2359.000000 | 2359.000000 | 2359.0 | 2359.000000 |
| mean | 0.007206 | 0.005511 | 24.730818 | 42.824502 | 1.248834 | 0.546418 | 0.0 | 5830.248410 |
| std | 0.160678 | 0.125143 | 5.993142 | 12.686026 | 0.581352 | 2.513801 | 0.0 | 1346.088549 |
| min | 0.000000 | 0.000000 | -1.000000 | -1.000000 | 1.000000 | 0.000000 | 0.0 | 0.000000 |
| 25% | 0.000000 | 0.000000 | 24.000000 | 37.000000 | 1.000000 | 0.000000 | 0.0 | 4917.000000 |
| 50% | 0.000000 | 0.000000 | 26.000000 | 46.000000 | 1.000000 | 0.000000 | 0.0 | 5814.000000 |
| 75% | 0.000000 | 0.000000 | 27.000000 | 50.000000 | 1.000000 | 0.000000 | 0.0 | 6431.000000 |
| max | 5.000000 | 3.000000 | 40.000000 | 99.000000 | 5.000000 | 13.000000 | 0.0 | 18950.000000 |

**Figura 4.61.** Le statistiche descrittive dello Snap 1.872

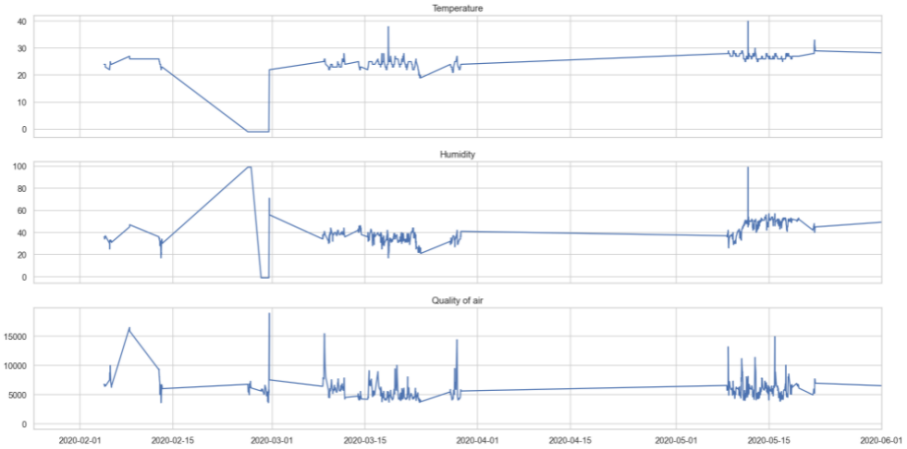


**Figura 4.62.** Gli andamenti nel tempo di temperatura, umidità e VOCs rilevati dallo Snap 1.872

notiamo che l'errore "9" è rilevato in corrispondenza del valore di umidità relativa pari al 99%, e l'errore "13" in corrispondenza del valore di umidità relativa pari a -1. Possiamo, quindi, confermare quanto appreso dalla heatmap.

```
plt.figure(figsize=(14,7))
correlation = snap1872.corr()
sns.heatmap(correlation, xticklabels=correlation.columns, yticklabels=correlation.columns,annot=True)
```

```
<AxesSubplot:>
```


**Figura 4.63.** La heatmap dello Snap 1.872

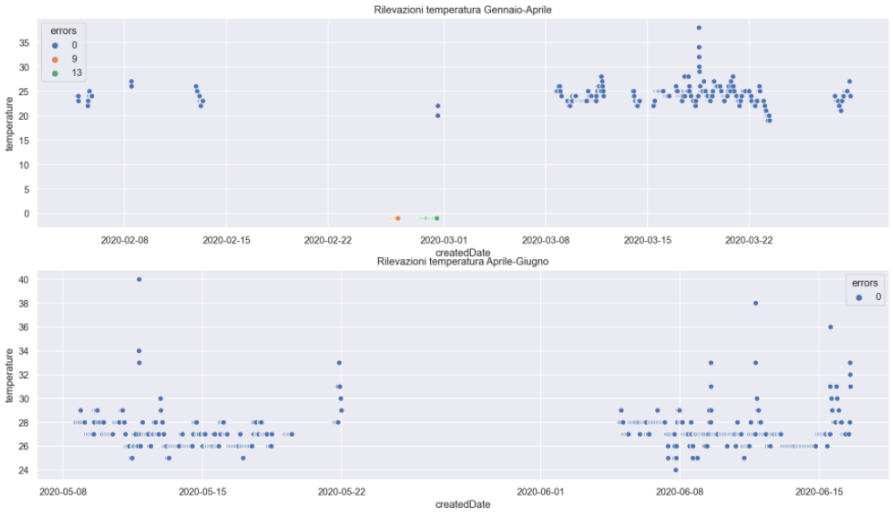


**Figura 4.64.** Scatterplot dell'umidità rilevata dallo Snap 1.872

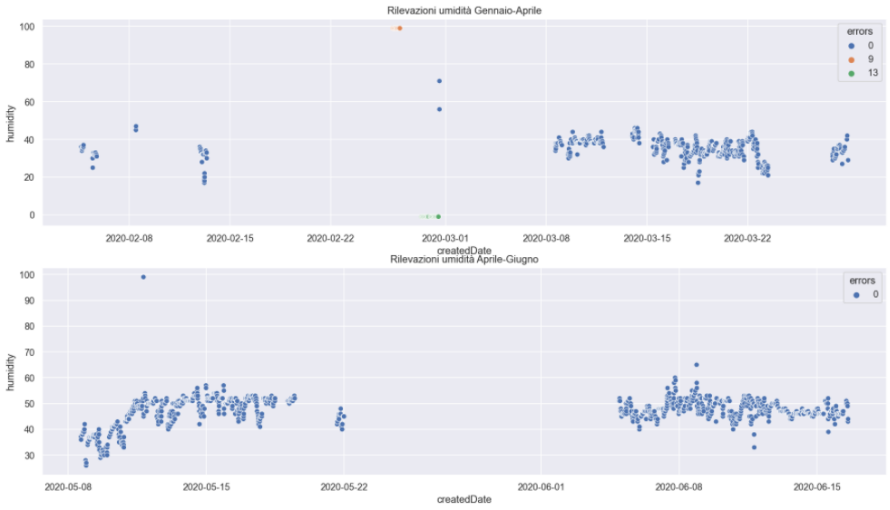


**Figura 4.65.** Scatterplot dell'umidità rilevata dallo Snap 1.872

# 5

# Discussione in merito ai risultati ottenuti

*In questo capitolo presenteremo i principali risultati ottenuti nella nostra campagna sperimentale e analizzeremo le lezioni più importanti apprese in tale progetto.*

## 5.1 Compendio dei principali risultati ottenuti

L'analisi esplorativa svolta su questo dataset ha avuto l'obiettivo di comprendere al meglio il funzionamento di Snap "Air Quality Balancer" e delle eventuali anomalie nel comportamento dei suoi sensori.

Grazie alle nostre analisi abbiamo potuto osservare quali temperature, umidità e livelli di VOC nell'aria circostante registrasse il dispositivo.

Alcuni studi ritengono che il corpo umano preferisca la temperatura dai circa 20°C ai 24°C e un'umidità relativa dell'aria tra il 40 e il 60% a qualsiasi temperatura; per tale ragione, la temperatura di 20°C e l'umidità relativa del 50% possono essere considerate misure ideali in ambiente chiuso. La media delle misure registrate dal *Temperature Sensor* è di circa 22°C, mentre la media delle misure registrate dall'*Humidity Sensor* è del circa 47%; possiamo, dunque, considerarle rilevazioni ottime, in termini di media.

Per quanto riguarda gli agenti inquinanti registrati dall'*Air Quality Sensor*, invece, a fronte delle analisi svolte, Snap considera come ottimale un ambiente il cui livello di ping, espresso in decimale, si concentra su valori al di sotto di 5000, mentre considera un ambiente molto contaminato, un ambiente il cui livello di ping si concentra su valori al di sopra di 10000. Notiamo, però, che, in corrispondenza dei cinque livelli del sensore di airQuality, la concentrazione di ping non sia troppo distinta tra un livello e l'altro e, dalle nostre analisi, ci saremmo aspettati una "scala di valori" differente. Nonostante questo, dagli Snap analizzati abbiamo potuto osservare come il dispositivo ha la capacità di rigenerare l'aria dell'ambiente in breve tempo, una peculiarità che viene presentata dall'azienda stessa; abbiamo notato, infatti, che lo Snap, una volta individuati valori elevati di ping, riesce a ristabilire i valori che si aggirano intorno alla media.

Circa l'84% dei dispositivi del nostro dataset viene registrato senza errori; abbiamo scelto lo *Snap 2.257* come "modello" di questo insieme di dati. Effettuando analisi approfondite su di esso, abbiamo scoperto una buona correlazione tra le

variabili; in particolare, abbiamo osservato le relazioni tra il currentMode e le tre misure fondamentali di queste analisi, quali la temperatura, l'umidità e il livello di VOC; abbiamo, infine, notato come la modalità corrente varia in base ai valori registrati di ciascun campo. La modalità Standby prevale sulle altre, e ci saremmo aspettati per essa, delle modalità di utilizzo più variegate; d'altronde la sua preponderanza potrebbe essere dovuta alla volontà di garantire un maggior risparmio energetico sul dispositivo.

In seguito, abbiamo analizzato su questo Snap gli outlier e, tra questi valori anomali, abbiamo individuato due misure il cui valore di temperatura, umidità e VOC risulta essere 0. Dunque, supponiamo che, in corrispondenza di questi monitoraggi, ci siano stati dei malfunzionamenti e, di conseguenza, interruzioni del dispositivo. Approfondendo le nostre ricerche, abbiamo scoperto che circa l'8% dei dispositivi del nostro dataset presentano questo genere di anomalia.

Le analisi sono proseguite sui dispositivi registrati con errori e, per ciascuna tipologia di errore abbiamo scelto uno Snap.

Abbiamo scelto lo *Snap 1.347* come "modello" dei dispositivi con errore 2 e, attraverso i nostri grafici, abbiamo dimostrato come non ci sia alcuna correlazione tra il sensore di airQuality e il campo ping, tanto da non riuscire a distinguere un ambiente ottimale da un ambiente poco contaminato. Osservando altri Snap con questa tipologia di errore, abbiamo notato che, rispetto al precedente, il sensore di airQuality possiede una maggiore correlazione con i valori di ping e vengono registrati come errori valori di VOC molto alti, che si discostano dall'andamento generale. Tali valori, però, non sempre vengono registrati come errori in altre tipologie di Snap.

Abbiamo scelto lo *Snap 1.838* come "modello" dei dispositivi con errore 16 e abbiamo notato che, come esso, tutti i dispositivi con questa tipologia di errore lavorano anche in modalità *Link*. Nel nostro dataset, i dispositivi che lavorano con questa modalità sono 15, di cui soltanto 2 non presentano questa tipologia di errore, mentre i dispositivi la cui modalità Link risulta essere abilitata sono 32. Dunque, soltanto il 2% dei dispositivi totali è stata connessa ad una cappa "Elica Sense", e solo l'1% ha sfruttato la modalità Link, ma con poco successo.

Abbiamo scelto lo *Snap 1.320* come "modello" dei dispositivi con errore 4; in corrispondenza di tali errori vengono individuati valori la cui misura di umidità risulta pari all'1%.

Infine, abbiamo scelto lo *Snap 1.872* come "modello" dei dispositivi con errore 9 e 13; in corrispondenza dell'errore 9 sono stati individuati valori anomali la cui temperatura rilevata risulta -1 e l'umidità pari al 99%, mentre, in corrispondenza dell'errore 13 sono stati individuati valori anomali la cui temperatura rilevata risulta sempre -1, ma l'umidità pari al -1%. Notiamo, però, che una rilevazione di umidità pari al 99%, che rappresenta un valore piuttosto alto per un ambiente chiuso, viene rilevata come outlier, ma non come errore.

L'osservazione degli Snap con errori ha avuto l'obiettivo di comprendere quali fossero i valori che il dispositivo registra come errori, ma, anche, grazie a queste analisi, abbiamo scoperto che tra gli Snap senza errori sono presenti valori anomali che non sono stati registrati come tali. Infatti, nel nostro dataset, sono presenti Snap senza errori che rilevano, per esempio, temperature comprese tra i 45°C e i 58°C, valori insoliti in un ambiente chiuso, umidità relative al di sopra del 90%, registrate

da circa l'8% dei dispositivi senza errore, e valori negativi di VOC, monitorati, però, da dispositivi le cui rilevazioni sono state interrotte nei mesi di Febbraio/Marzo, probabilmente per problemi evidenti nel loro utilizzo.

## 5.2 Le lezioni apprese

In questa tesi abbiamo analizzato il dataset dell'azienda Elica, un'azienda marchigiana che si occupa di cappe da cucina, nonché produttore di un innovativo dispositivo chiamato *Snap "Air Quality Balancer"*, oggetto del nostro studio. Tale dispositivo è dotato di tre sensori che hanno il compito di monitorare e migliorare la qualità dell'aria nell'ambiente in cui essi sono installati. L'obiettivo della nostra campagna sperimentale è stato quello di effettuare un'analisi esplorativa che permettesse di verificare e comprendere il funzionamento di esso, determinando, nel contempo, eventuali anomalie registrate nei diversi monitoraggi. Il nostro studio si è basato sui concetti fondamentali della Data Science e, con l'aiuto di Python, siamo riusciti a raggiungere gli obiettivi prefissati.

In particolare, le lezioni apprese durante tale campagna sperimentale riguardano:

- l'analisi dei dati;
- il linguaggio di programmazione Python;
- lo studio di un dataset reale;
- l'anomaly detection come forma di manutenzione.

Esse saranno esaminate più nel dettaglio nelle prossime sottosezioni.

### 5.2.1 L'analisi dei dati

In un mondo sempre più centralizzato intorno alla tecnologia dell'informazione, grandi quantità di dati vengono prodotti e memorizzati ogni giorno. La presente tesi ci ha permesso di approfondire il campo della Data Science, un ambito che, negli ultimi tempi, si è diffuso sempre maggiormente in ambiti aziendali, e non solo. Nel nostro caso specifico, i dati sono stati registrati da sensori ed è stato nostro compito estrarre informazioni da essi, cercando di dare loro un senso.

Le fasi del nostro studio hanno principalmente seguito gli step dettati dalla gerarchia dei bisogni della Data Science. Innanzitutto, è stata effettuata da terzi una raccolta dati attraverso simulazioni da parte dei sensori di 1.311 Snap, effettuate dal 25 gennaio 2020 al 16 giugno 2020; i dati sono stati salvati in un file CSV, che l'azienda ci ha messo a disposizione. Abbiamo, poi, organizzato il nostro dataset eseguendo l'attività di ETL che ci ha permesso di trasformare e ripulire i dati in modo più consono per i nostri obiettivi. Sono seguite analisi che ci hanno permesso di determinare il funzionamento e il malfunzionamento dei dispositivi. Una tecnica applicata nella presente tesi è *anomaly detection*, che ci ha consentito di identificare eventi inaspettati che differivano dalla norma. Le fasi di previsione e automatizzazione non sono state effettuate nella nostra campagna sperimentale, ma, attraverso tecniche più complesse, potrebbero essere oggetto di studi futuri.

Grazie, quindi, al nostro studio, abbiamo potuto comprendere meglio tali fasi non solo da un punto di vista teorico, ma anche da un punto di vista pratico, che ci aiuterà, sicuramente, nell'analisi di eventuali dataset futuri.

### 5.2.2 Il linguaggio di programmazione Python

Le nostre analisi sono state effettuate sfruttando il linguaggio di programmazione Python supportato dall'ambiente di sviluppo Jupyter Notebook. Tale linguaggio è tra i più utilizzati nel mondo grazie, anche, alla sua indubbia versatilità. Con la presente tesi abbiamo potuto sfruttare le principali funzionalità di cui dispone per effettuare un'analisi esplorativa.

Python ci ha permesso di eseguire le fasi di ETL in modo semplice e rapido, mediante l'utilizzo di singole funzioni appartenenti alla libreria `Pandas`. Nel nostro caso, questa fase non ha richiesto troppi step: abbiamo eliminato i campi per noi superflui, i duplicati e un singolo valore `NULL`.

Python ci ha permesso di implementare codici per ottenere nuovi dataframe che sono stati sottoposti ad analisi più approfondite. Tale linguaggio ci è risultato molto più intuitivo rispetto a quelli da noi conosciuti.

Infine, l'aspetto più interessante di Python è stata la realizzazione di grafici, grazie a semplici funzioni messe a disposizione dalla libreria `Seaborn`. Ci ha sorpreso la molteplicità di rappresentazioni che è possibile effettuare con tale libreria e di quante varianti è possibile apportare con il semplice cambiamento dei paramenti di ciascuna funzione. Abbiamo realizzato istogrammi, lineplot, heatmap, scatterplot e boxplot che ci hanno permesso di dedurre le nostre informazioni.

### 5.2.3 Lo studio di un dataset reale

La presente tesi ci ha permesso di approfondire il concetto di dataset e, attraverso la nostra campagna sperimentale, abbiamo avuto l'opportunità di studiare dati reali. Le possibili strade da intraprendere nell'analisi dei dati sono molteplici, ma ognuna di esse può portare a scoperte interessanti.

Il primo passo nel nostro studio è stato la documentazione riguardo Snap. Essa è stata fondamentale e ci ha aiutato nella progettazione della nostra campagna sperimentale. Abbiamo ricercato sul sito di Elica quante più informazioni possibili riguardo il dispositivo, in modo da comprenderne l'utilizzo e il funzionamento. Questo ha creato delle aspettative nel dataset e nell'andamento dei dati stessi, alcune delle quali si sono trasformate in ipotesi da verificare nel nostro progetto.

Essendo un dataset reale la probabilità di trovare, ad esempio, duplicati, è alta; lo stesso vale per anomalie o date non corrette. Inoltre, data la numerosità di valori registrati, è impossibile analizzare ciascun dispositivo; per questo abbiamo optato per uno studio che ci permettesse di analizzare gli Snap per noi più interessanti e da cui avremmo potuto trarre conclusioni generali. Abbiamo, infatti, realizzato differenti dataframe con caratteristiche specifiche, partendo con la divisione del dataset in dispositivi senza e con errori.

### 5.2.4 L'anomaly detection come forma di manutenzione

Il fulcro della nostra campagna sperimentale riguarda l'anomaly detection. Ci sono diversi metodi e funzioni messe a disposizione da librerie di Python che permettono di affrontare tale analisi; nella presente tesi abbiamo sfruttato il boxplot, uno strumento grafico molto semplice e immediato che ci ha permesso di determinare

tutti i valori anomali che riguardano le rilevazioni di temperatura, umidità e VOC effettuate dagli Snap presi in considerazione.

Come spiegato nel compendio dei risultati ottenuti, sono state rilevate alcune anomalie e questi dati potrebbero essere un buono spunto per l'apporto di future modifiche ai dispositivi. Si parla, dunque, di manutenzione, ovvero la combinazione di tutte le azioni tecniche, amministrative e gestionali, durante il ciclo di vita di un'entità, volte a mantenerla o riportarla in uno stato in cui possa eseguire la funzione richiesta. L'obiettivo della manutenzione riguarda la minimizzazione dei guasti, delle interruzioni di dispositivi e dei costi operativi.

La manutenzione può essere suddivisa in due tipi di strategie: correttiva e preventiva.

- *La manutenzione correttiva* è seguita dal riconoscimento di un guasto e ha il compito di riportare le prestazioni del sistema al momento precedente al problema. Questo tipo di approccio comporta costi molto elevati legati sia alle perdite di produzione sia ai guasti improvvisi.
- *La manutenzione preventiva*, invece, avviene prima dell'arrivo del guasto ed è eseguita ad intervalli periodici a seconda dello stato di salute del sistema; essa è volta a ridurre la probabilità del guasto o il degrado del funzionamento di un'entità. Esistono diverse categorie di manutenzione preventiva. Tra queste individuiamo la *La manutenzione predittiva* (Figura 5.1). Essa rappresenta uno dei temi più importanti dell'industria 4.0. Il suo scopo principale è quello di predire quando un guasto o un malfunzionamento potrebbe verificarsi, così da prevenirli attraverso l'attività di manutenzione. Questa attività consente di intervenire solo quando necessario, ottimizzando i tempi e i costi di intervento. È importante capire cosa ci si aspetta dal processo di manutenzione predittiva in modo da selezionare i dati che servono allo scopo. Il numero di record di dati per addestrare i modelli predittivi deve essere adeguato; più sono gli errori o le failure a disposizione e più accurato sarà il risultato. Dunque, l'anomaly detection, può essere considerata un punto di partenza per il miglioramento dei dispositivi presi in esame, in modo da renderli più efficienti ed efficaci, prevedendo gli andamenti futuri ed evitando guasti o errori nel loro utilizzo.

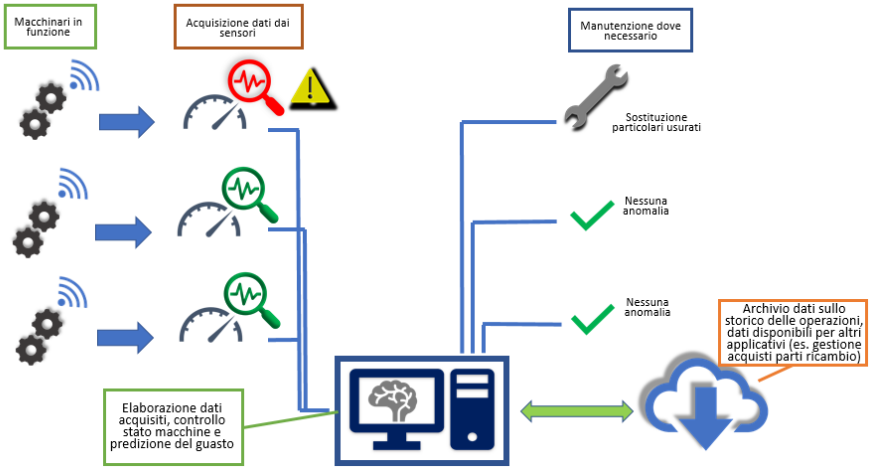


**Figura 5.1.** Il processo di manutenzione predittiva

# 6

# Conclusioni

Nella presente tesi è stata effettuata un'analisi esplorativa su un dataset dell'azienda Elica, un'impresa marchigiana attiva principalmente nel mercato delle cappe da cucina. Abbiamo studiato il comportamento di un innovativo dispositivo, chiamato *Snap "Air Quality Balancer"*, un sistema di aspirazione dotato di tre sensori che monitorano costantemente l'aria prendendo in esame qualità, temperatura e umidità.

Il primo passo è stato quello di contestualizzare il nostro progetto introducendo la Data Science, un campo interdisciplinare fondamentale nell'attuale contesto di trasformazione digitale; abbiamo, altresì, presentato Python, uno strumento molto amato dai data scientist e utilizzato per le nostre analisi.

Successivamente abbiamo descritto dettagliatamente la struttura del nostro dataset, approfondendo il significato di ciascun campo.

La fase di ETL (Extract, Trasform and Load) è stato il primo step per la progettazione della nostra campagna sperimentale e, grazie all'utilizzo di funzioni in Python, abbiamo potuto filtrare i dati più utili per il nostro progetto.

Le nostre analisi si dividono prevalentemente in due fasi:

- Durante la prima fase abbiamo osservato gli Snap senza errori, soffermandoci, in particolare, sullo studio dei sensori di temperatura, umidità e qualità dell'aria di un "modello" scelto per questa categoria di dispositivi.
- Durante la seconda fase abbiamo osservato gli Snap con errori, soffermandoci sullo studio degli effetti che ciascun errore ha apportato su quattro differenti "modelli" scelti per questa categoria di dispositivi.

In entrambe le fasi è stata svolta l'anomaly detection, che rappresenta il fulcro del nostro progetto, con lo scopo di individuare eventuali anomalie nel comportamento dei sensori dei dispositivi scelti per le nostre analisi.

I risultati ottenuti nel corso di questo progetto ci hanno permesso di comprendere i punti di forza di Snap, che lo rendono un dispositivo utile ed innovativo, nonché le sue debolezze, da cui è possibile partire per apportare modifiche e miglioramenti.

L'impostazione di questo progetto potrebbe essere un punto di partenza per analisi esplorative più approfondite su Snap o su dispositivi simili ad esso. Grazie alle molteplici funzionalità di Python, potrebbero essere realizzati grafici aggiuntivi per trarre nuove informazioni da questi dati.

Inoltre, potrebbero essere applicate ulteriori tecniche di Data Mining come la *classificazione*, grazie alla quale, una volta definite delle classi di interesse, è possibile trovare profili di elementi che attribuiamo ad esse, oppure il *clustering*, grazie al quale è possibile raggruppare oggetti in gruppi omogenei, cioè con caratteristiche simili, senza conoscere le classi di interesse. Attraverso molteplici algoritmi, la libreria `Scikit-Learn` di Python ci consente di applicare tali analisi.

Infine, potrebbero essere effettuate analisi predittive per creare un modello con il quale prevedere i valori futuri di una serie temporale multivariata. Tale analisi si basa sul Machine Learning, ovvero algoritmi di autoapprendimento che imparano a conoscere lo stato ottimale di funzionamento di ogni dispositivo, rilevando in anticipo potenziali anomalie. Questi studi sono vantaggiosi soprattutto perché minimizzano il rischio di guasti imprevisti e improvvisi, obbligando manutenzioni correttive che comportano costi molto elevati. La libreria `Statsmodel` di Python fornisce un potente strumento per lavorare con le serie temporali.

# Riferimenti bibliografici


1. Histograms and Density Plots in Python. https://towardsdatascience.com/histograms-and-density-plots-in-python-f6bda88f5ac0, 2018.
2. What is Exploratory Data Analysis? https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15, 2018.
3. ETL - Understanding It and Effectively Using It. https://medium.com/hashmapinc/etl-understanding-it-and-effectively-using-it-f827a5b3e54d, 2019.
4. Journey into Data Science. https://towardsdatascience.com/journey-into-data-mining-3b5ccfa5343, 2019.
5. Types of Data Sets in Data Science, Data Mining & Machine Learning. https://towardsdatascience.com/types-of-data-sets-in-data-science-data-mining-machine-learning-eb47c80af7a#_=_, 2019.
6. Why Data Science has a 'last-mile' problem. https://deepconnect.cloud/data-science-is-hard/, 2019.
7. Boxplot for anomaly detection. https://towardsdatascience.com/boxplot-for-anomaly-detection-9eac783382fd, 2020.
8. Exploratory Data Analysis. https://www.ibm.com/cloud/learn/exploratory-data-analysis, 2020.
9. Seaborn: statistical data visualization. https://seaborn.pydata.org/, 2020.
10. About pandas. https://pandas.pydata.org/about/, 2021.
11. About statsmodels. https://www.statsmodels.org/stable/about.html, 2021.
12. Jupyter Notebook: strumento per data science. https://daf-dataportal-it-docs.readthedocs.io/it/latest/datascience/jupyter/, 2021.
13. Matplotlib: Visualization with Python. https://matplotlib.org/, 2021.
14. NumPy v1.20 Manual. https://numpy.org/doc/stable/, 2021.
15. Snap by Elica. https://elica.com/IT-it/snap-air-quality-balancer, 2021.
16. The Data Science Hierarchy of Needs. https://matthewrenze.com/articles/the-data-science-hierarchy-of-needs/, 2021.
17. M. Buttu. *Programmare con Python. Guida completa.* Edizioni Lswr, 2014.
18. R. Johansson. *Numerical Python.* Apress, 2018.
19. P. Morgan. *Data Analysis from Scratch with Python.* Ai Sciences, 2017.
20. F. Nelli. *Python Data Analytics.* Apress, 2018.
21. G. Press. A Very Short History Of Data Science. *Forbes*, 2013.