Università Politecnica delle Marche

FACOLTA' DI INGEGNERIA

Corso di Laurea triennale Ingegneria Gestionale

Analisi statistica di magazzini complessi in Python

Statistical Analysis of complex warehouses in Python

Relatore:

Zingaretti Primo

Tesi di Laurea:

Colombo Gianluca

Correlatore:

Romeo Luca

A.A. 2020/2021

Capitolo 1: Introduzione

Il presente elaborato è stato realizzato a seguito a uno studio su una struttura dati della Luxottica Group . L'impresa in questione commercializza occhiali di Ray-Ban, Oakley, Vogue Eyewear, Persol, Oliver Peoples, Arnette, Costa del fascia alta, di lusso e sportivi, con un portafoglio marchi forte e ben bilanciato sia di proprietà, tra cui Mar e Alain Mikli, sia in licenza, come Giorgio Armani, Burberry, Bulgari, Chanel, Dolce&Gabbana, Ferrari, Michael Kors, Prada, Ralph Lauren, Tiffany & Co., Valentino e Versace. L'azienda vanta circa 80 milioni di dipendenti, 91 milioni di prodotti, , 9000 negozi e 4 centri distribuiti nel mondo. L'obiettivo della tesi è, principalmente, quello di analizzare statisticamente la struttura dati andando a identificare i fattori chiave per una descrizione sintetica dell'oggetto di studio che ne permetta l'interpretazione e la previsione futura.

La tesi è suddivisa in quattro capitoli. Il primo è dedicato alle definizioni teoriche di statistica, la conoscenza di tali nozioni è fondamentale per una corretta analisi statistica. Il secondo capitolo tratterà l'implementazione della base di dati su un ambiente integrato in SQL (Structured Query Language), e sulla scelta del linguaggio di programmazione nel quale poter eseguire il lavoro, susseguito dalle motivazioni che hanno spinto alla scelta dei seguenti Software. Nel terzo capitolo si procederà con l'individuazione dei fattori chiave e nella ricerca delle correlazioni. Tramite esse, sono stati analizzati i costi di Stoccaggio e individuato qual è il componente più utilizzato e qual è il modello costituito dal numero maggiore di componenti, e identificato il magazzino più economico. Si è ottenuta una statistica di base per ogni modello, calcolando media, varianza, e deviazione standard dei costi dei componenti che lo compongono. Il quarto capitolo tratterà l'analisi di regressione lineare con lo studio attraverso la retta di regressione e proponendo un'interpretazione grafica utilizzata per prevedere il valore di una determinata variabile y in base al valore di un'altra variabile x. Nella parte finale del quarto capitolo verranno effettuati alcuni test per comparare

le medie calcolate in precedenza, e concludendo con una possibile espansione del progetto tramite una multi-regressione.

Capitolo 2: Definizioni e concetti generali

2.1 Statistica

La statistica studia un fenomeno con lo scopo di metterne in evidenza gli aspetti essenziali, risalendo eventualmente alle leggi che lo regolano. Nella maggior parte dei casi rappresenta il mezzo più efficace per ridurre il margine di incertezza delle nostre scelte.

Il primo problema che ci si trova ad affrontare è quello di sintetizzare la massa di dati grezzi in pochi numeri o indicatori particolarmente informativi, utilizzando metodiche grafiche o numeriche, che siano in grado di descrivere la massa di dati, senza alterarne il senso complessivo. Questa parte della statistica è nota con il nome di statistica descrittiva. Il procedimento inferenziale consente poi di prendere una decisione quanto più possibile obiettiva circa l'ipotesi formulata.

Le variabili statistiche possono essere qualitative, se esprimono una qualità dell'individuo, o quantitative, che possono essere misurate su una scala discreta (numero di insetti suscettibili ad un certo insetticida, numero di semi germinati in certe condizioni ambientali...) o su una scala continua (produzione delle piante o altezza degli alberi...).

Avendo a che fare con un numero elevato di dati, è conveniente considerare le frequenze delle unità sperimentali: la frequenza assoluta non è altro che il numero degli individui che presentano una certa misura (per un carattere quantitativo) o una certa modalità (per un carattere qualitativo).

2.2.1 Indici di tendenza centrale: Media, Moda e Mediana

Dato un insieme di dati, è possibile calcolare degli indici aggiuntivi che rispecchino il più possibile le informazioni contenute nell'insieme dei dati. Un'informazione fondamentale è quella relativa alla tendenza centrale, espressa, tra gli altri, da tre indicatori, cioè la media, la moda e la mediana. La media aritmetica è un concetto molto intuitivo ed esprime, in genere, quanta

parte dell'intensità totale del fenomeno compete, in media, a ciascuna unità sperimentale. Si indica con μ e si calcola facendo la somma dei valori relativi alla variabile rilevata in tutti gli individui, e dividendola per il numero degli individui del collettivo.

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n}$$

Quando si ha a che fare con distribuzioni di frequenze, la media può essere calcolata moltiplicando il valore centrale di una classe per il numero degli individui che appartengono a quella classe, secondo la seguente espressione.

$$\mu = \frac{\sum_{i=1}^{n} f_i \cdot x_i}{n}$$

Il valore centrale di una classe è dato dalla semisomma degli estremi della classe stessa.

La moda è invece la classe che presenta la maggior frequenza.

La mediana è data dal termine che bipartisce la distribuzione di frequenza in modo da lasciare lo stesso numero di termini a sinistra e a destra.

Se abbiamo una serie di individui ordinati in graduatoria, la mediana è data dall'individuo che occupa il posto (n + 1)/2 o, se gli individui sono in numero pari, dalla media delle due osservazioni centrali.

2.2.2 Indici di variabilità dei fenomeni: Devianza, varianza, deviazione standard e coefficiente di variabilità

Gli indici di tendenza centrale non ci informano su come i dati tendono ad assumere misure che sono diverse gli uni dagli altri. Quindi, quando si vuole descrivere un insieme di dati, è necessario utilizzare non solo un indice della tendenza centrale, ma anche un indice di variabilità, che ci consenta di stabilire come si colloca ogni singolo dato rispetto alla tendenza centrale dell'insieme.

Esistono diversi indici di variabilità, tra cui i più diffusi sono la devianza, la varianza, la deviazione standard ed il coefficiente di variabilità. L'indice SS:

$$SS = \sum_i (x - \mu)^2$$

Costituisce la somma dei quadrati degli scarti (SS) ed è noto con il termine di devianza. Dividendo la devianza per il numero dei gradi di libertà (numero degli individui del collettivo meno uno) si ottiene la varianza (generalmente indicata con σ 2):

$$\sigma^2 = \frac{SS}{n-1} = \frac{\sum_i (x - \mu)^2}{n-1}$$

La radice quadrata della varianza costituisce la deviazione standard, che si indica con σ. Il coefficiente di variabilità è un indice percentuale, dato dal rapporto fra la deviazione standard e la media, moltiplicato per 100. E' interessante per confrontare tra di loro le variabilità di due o più collettivi e/o variabili.

$$CV = \frac{\sigma}{\mu} \times 100$$

2.3 Coefficiente di correlazione

Un indicatore statistico per descrivere il grado di variazione congiunta di due variabili è il coefficiente di correlazione. Il calcolo è abbastanza semplice: dato un collettivo statistico composto da n unità sperimentali, sulle quali sono state rilevate due variabili statistiche (X1i e X2i con i che va da 1 ad n e medie rispettivamente pari a μX1 e μX2), definiamo coefficiente di correlazione (r), la misura:

$$r = \frac{\sum_{i=1}^{n} [(X1_i - \mu_{x1})(X2_i - \mu_{x2})]}{\sqrt{\sum_{i=1}^{n} (X1_i - \mu_{x1})^2 \sum_{i=1}^{n} (X2_i - \mu_{x2})^2}}$$

La quantità al numeratore viene detta codevianza (o somma dei prodotti), mentre si può notare che al numeratore, sotto radice, abbiamo il prodotto delle devianze delle due variabili. Il coefficiente di correlazione varia tra –1 e

+1 (la dimostrazione di questa proprietà non è necessaria): un valore pari a +1 indica concordanza perfetta (tanto aumenta una variabile, tanto aumenta l'altra), mentre un valore pari a –1 indica discordanza perfetta (tanto aumenta una variabile tanto diminuisce l'altra). Un valore pari a 0 indica assenza di qualunque grado di variazione congiunta tra le due variabili (assenza di correlazione). Valori intermedi tra quelli anzidetti indicano correlazione positiva (se positivi) e negativa (se negativi).

2.4 Analisi di regressione

In alcuni casi le due variabili rilevate sulle unità sperimentali sono tali che possiamo ipotizzare che una relazione di dipendenza diretta, sulla base di considerazioni biologiche, sociali, chimiche, fisiche ecc… In sostanza, è possibile individuare una variabile dipendente (detta anche variabile regressa) e una variabile indipendente (detta anche regressore). In questo caso, la conoscenza del semplice grado di correlazione tra le due variabili può non essere sufficiente per i nostri scopi, mentre potrebbe essere necessaria la conoscenza diretta della funzione matematica che lega la variabile dipendente alla variabile indipendente. In questa sede, per motivi di semplicità, restringiamo il nostro interesse alle funzioni lineari e, in particolare, all'equazione di una retta. Nel momento in cui ipotizziamo che tra le due variabili esiste una relazione lineare, rappresentabile con una linea retta di equazione generica:

$Y = mX + q$

o meglio (in statistica):

14 $Y = b1\ X + b0$

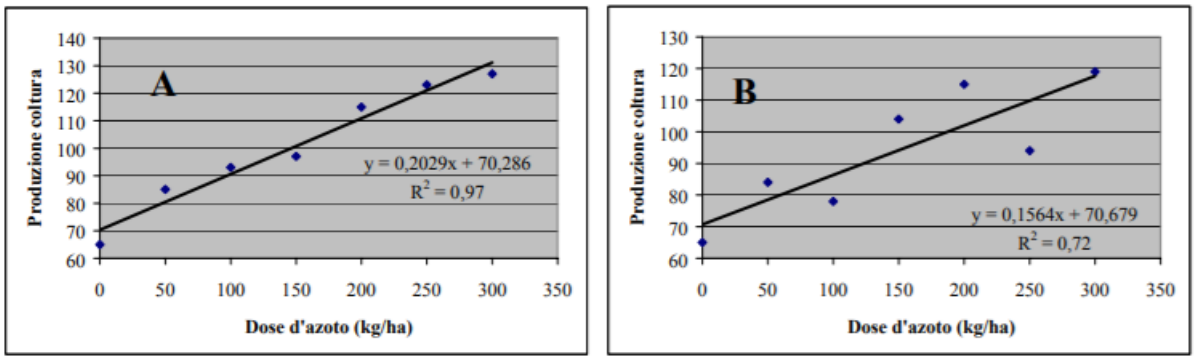
Il problema è ridotto alla determinazione dei valori di b1 (detto in statistica coefficiente di regressione) e b0 che sono rispettivamente la pendenza della retta e l'intercetta (intersezione con l'asse delle Y). L'esigenza di fare una analisi di regressione si presenta, in genere, perché vogliamo essere in grado di prevedere i valori della Y qualunque sia il valore della X (o viceversa). Il problema sarebbe assolutamente banale se i punti fossero perfettamente allineati, il che non si verifica mai in statistica, almeno per due motivi: 1) le relazioni biologiche non sono quasi mai perfettamente lineari, ma lo sono

solo approssimativamente; 2) le variabili osservate sulle unità sperimentali fluttuano a causa del possibile errore sperimentale. E' quindi necessaria una procedura di interpolazione, che viene eseguita analiticamente ricorrendo alle formule seguenti (n è il numero di unità sperimentali mentre μX e μY sono le medie delle due variabili):

$$b_1 = \frac{\sum_{i=1}^{n}[(X_i - \mu_X)(Y_i - \mu_Y)]}{\sum_{i=1}^{n}(X_i - \mu_X)^2}$$

$$b_0 = \mu_Y - b_1\mu_X$$

Mentre la formula per il calcolo di b0 è banale, la formula per il calcolo di b1 porta al numeratore la codevianza di X e Y ed al denominatore la devianza di X. Quando questo calcolo viene eseguito con l'aiuto del computer, l'output dell'analisi comprende in genere un indicatore detto coefficiente di determinazione (R2). Questo indicatore numericamente è il quadrato del coefficiente di correlazione lineare, ma concettualmente indica la quota parte della variabilità della Y che è attribuibile alla dipendenza lineare dalla X; in sostanza si tratta di un indicatore della bontà della regressione: più è vicino ad 1 e più la regressione è buona. La figura sottostante mostra due esempi di regressione con diversi valori di R2 . E' comprensibile visivamente come la regressione in (A) si più attendibile di quella in (B).



Due esempi di regressioni con diversi coefficienti di determinazione

2.5 Confronto tra due medie: il test t di Student

Si ha spesso interesse a considerare due popolazioni per scoprire se queste sono diverse per il carattere o i caratteri considerati. Più in particolare, siccome ognuna delle popolazioni sarà descritta dalla sua media, saremo

interessati a rispondere al quesito se l'eventuale differenza rilevata tra le due medie e da ritenersi una differenza reale, effettiva e con un preciso significato biologico. In sostanza, in termini statistici, dovremo stabilire se la differenza tra le medie è significativa oppure da attribuire a fattori casuali e quindi non significativa. E' intuitivo comprendere che, anche se il problema può sembrare banale, esso non lo è; basti ripensare al fatto che ogni media stimata si porta dietro un alone di incertezza, definito appunto dall'intervallo di confidenza.

La decisione dovrà essere basata su due aspetti:

1) l'ampiezza della differenza tra le medie: più la differenza tra le due medie è alta e più è probabile che essa sia significativa;

2) l'ampiezza dell'errore standard. Più è elevata la variabilità dei dati e quindi l'errore di stima è più è bassa la probabilità che le differenze osservate tra le medie siano significative. Questi due aspetti sono stati utilizzati per definire il cosiddetto test di t :

$$t = \frac{\overline{X}_1 - \overline{X}_2}{s_{\overline{X}_1 - \overline{X}_2}}$$

La quantità al numeratore è la differenza tra le medie dei due campioni, mentre la quantità al denominatore è il cosiddetto errore standard della differenza tra due medie, che si calcola a partire dalla media ponderata delle deviazioni standard dei due campioni, secondo la formula seguente:

$$s_{\overline{X}_1 - \overline{X}_2} = \sqrt{\overline{s}^2 \frac{n_1 + n_2}{n_1 n_2}}$$

Dove s è la varianza mediata dei due campioni e si calcola sommando le devianze dei due campioni e dividendole per la somma dei rispettivi gradi di libertà:

$$\overline{s}^2 = \frac{SS_1 + SS_2}{n_1 + n_2 - 2}$$

Secondo quanto detto in precedenza, è evidente che il test di t assume un valore tanto più alto, quanto più è significativa la differenza tra le due medie. O meglio: tanto più è elevato il valore del test di t, quanto più bassa è la

probabilità di sbagliare affermando che la differenza tra le due medie è significativa. La probabilità d'errore che riteniamo accettabile va fissata all'inizio dell'esperimento ed è in genere pari al 5% o all' 1% (p = 0.05 o p = 0.01). Ora, fissata la probabilità d'errore che riteniamo accettabile, per rispondere alla domanda fatta all'inizio dobbiamo sapere qual è il valore minimo del test che ci consente di concludere che la differenza tra le medie è significativa. Questo valore può essere desunto dalle tavole di t riportate in precedenza, considerando un numero di gradi di libertà pari ad [(n1-1)+(n2-2)], con n1 ed n2 pari alle numerosità dei due campioni a confronto.

Capitolo 3: Programmi utilizzati

3.1 Python

Python nasce come linguaggio di programmazione generico e ad oggetti, non orientato esclusivamente all'ambito scientifico come lo sono Matlab, R o Julia, non possiede nativamente le classi necessarie per i calcoli e le computazioni avanzate.

Python risulta essere un linguaggio semplice ma potente, leggibile e versatile. I benefici che avremmo affidandoci a questo linguaggio sono di gran lunga superiori agli svantaggi per non parlare dei numerosi e vari campi di applicazione. Vediamo i punti di forza che fanno di Python un linguaggio così tanto apprezzato:

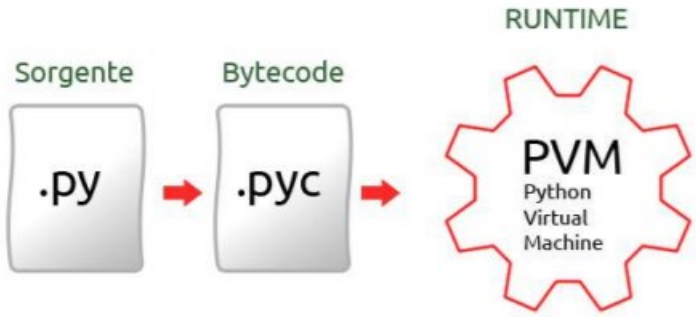
• Open source: Python `e completamente free; il suo uso e la sua distribuzione sono libere da copyright. Inoltre, è supportato da una community enorme e sempre attiva.

• Multi-paradigma: Python supporta sia la programmazione ad oggetti, sia la programmazione procedurale. Esso supporta, inoltre, diversi elementi della programmzione funzionale.

• Ricco di librerie: Ogni installazione di Python include la standard library, cio`e una collezione di oltre 200 moduli per svolgere i compiti più disparati. Inoltre, il Python Package Index consente di scaricare e installare migliaia di moduli aggiuntivi creati e mantenuti dalla comunità.

• Integrabile con altri linguaggi: esistono interpreti che permettono l'integrazione di Python con altri linguaggi, per esempio IronPython, per lavorare con framework, o Jython, per Java.

In Python l'interprete interattivo può essere lanciato da riga di comando attraverso il comando python. L'interprete legge e valuta le istruzioni o gli script lanciati dall'utente; se sono sintatticamente corretti li esegue; non `e prevista una fase di compilazione a parte. Il codice viene scansionato per token, ognuno dei quali viene analizzato dentro una struttura logica ad albero che rappresenta il programma. Tale struttura viene, infine, trasformata in bytecode. Per poter eseguire questi bytecode si utilizza un apposito interprete noto come macchina virtuale Python (PVM) .



Per poterne quindi dimostrare le potenzialità nel settore della visualizzazione, ci si deve appoggiare al Python Scientific Stack: un set di librerie scritte in Python, facilmente integrabili e open-source (come il linguaggio stesso), nato per ovviare a questa mancanza; è sponsorizzato dall'associazione PyData, che ne sostiene economicamente lo sviluppo e la diffusione, anche grazie all'organizzazione di svariati eventi annuali su scala mondiale.

La velocità di esecuzione di Python non è il suo punto di forza se lo paragoniamo ai linguaggi compilati come C. La situazione cambia se paragoniamo Python ad altri linguaggi interpretati, come PHP o Ruby; possiamo, in una certa misura, paragonare le sue prestazioni a Java. Python può superare i propri limiti di prestazioni pure scrivendo estensioni in C da utilizzare all'interno di codice Python, sfruttando, così, le prestazioni del linguaggio compilato nelle parti di codice dove è necessario.

Molti dei software facenti parte dello Stack (la lista completa è disponibile alla pagina https://pydata.org/downloads.html), principalmente quelli appartenenti alla suite principale SciPy, saranno brevemente introdotti.

SciPy è un ecosistema di librerie e tools open-source facente parte della suite PyData: aggiunge importanti funzionalità in campo matematico, scientifico e ingegneristico.

https://it.mathworks.com/products/matlab.html

https://www.r-project.org

https://julialang.org

https://pydata.org

https://www.scipy.org

In particolare per l'analisi di dati sotto forma di matrici, le componenti principali dello stack sono Numpy, Scipy (omonima della suite), Pandas, Matplotlib, altre librerie per il machine learning, l'algebra e una console interattiva.

3.1.1 NumPy

NumPy è una libreria che aggiunge all'installazione base di Python funzionalità per l'elaborazione di dati matriciali; implementa principalmente:

• Definizione di tipi di dati arbitrari .

• Array multidimensionali efficienti (visti come oggetti/classi) .

• Operazioni algebriche tra matrici e array .

• Funzioni per leggere e scrivere array di dati su disco .

• Funzioni per l'algebra lineare, trasformate di Fourier e un generatore di numeri pseudo-casuali .

• Funzionalità per l'integrazione in Python di codice c/c++ (oltre a quelle già presenti di default) e fortran Si presta quindi molto bene per fare da contenitore a dataset tabellari o multidimensionali, facilmente collegabili ai database.

3.1.2 SciPy

SciPy oltre a essere il nome della collezione, è anche il nome della principale libreria in essa contenuta, costruita sopra la base fornita da NumPy.
Aggiunge a quanto già presente funzionalità come:

- Altre routine per le matrici, il calcolo integrale e differenziale
- Algoritmi per le radici
- Algoritmi per processare segnali
- Distribuzioni di probabilità, funzioni statistiche

http://www.numpy.org licenza BSD-3

https://www.scipy.org/scipylib licenza BSD-3

3.1.3 Pandas

Pandas è una libreria che fornisce funzioni di analisi e strutture di memorizzazione orientate a lavorare nello specifico con dati relazionali (es: database). Può accettare come parametri gli array di NumPy, sono tra loro compatibili. Ambisce ad essere il più potente tool open-source del settore per la manipolazione: molte delle routine di basso livello sono state ottimizzate scambiando codice Python con codice c, precompilato e ottimizzato, più veloce nell'esecuzione. In particolare è ottimizzata per lavorare con:

- Dati tabellari, con colonne di vari tipi primitivi (come in sql o in excel).
- Serie temporali, anche non necessariamente ordinate.
- Dati matriciali, con righe e colonne definite da label, etichette testuali.
- Dataset statistici Si compone principalmente di due strutture per la memorizzazione, che coprono quasi tutte le esigenze di rappresentazione dei dati usati in finanza, statistica, scienza e ingegneria: Series unidimensionali e DataFrame9 bidimensionali. Alcune funzioni significative:
- Gestione dei dati mancanti (rappresentati come NaN) per tutti i tipi.
- Modificabilità di colonne e righe da altri DataFrame.

• Allineamento dati mediante una serie di label specificate dall'utente o in modo automatico.

• Funzioni di raggruppamento, divisione, combinazione, merge e join tra dataset .

• Funzioni di indicizzazione tramite label .

• Funzioni di i/o per interagire con dati salvati in file csv , excel, database .

http://pandas.pydata.org, licenza BSD-3

Simile a dataframe di r, ma con molte più funzionalità

https://it.wikipedia.org/wiki/Comma-separated_values

3.1.4 Dataframe

Un DataFrame rappresenta una struttura dati tabellare (e.g. foglio di calcolo) che consta di una collezione ordinata di colonne, ognuna delle quali può essere di un tipo di dati diverso. Il DataFrame possiede sia un index per le righe che un index per le colonne e può essere costruito passando un dizionario di liste di uguale lunghezza, in tale situazione, il vettore index verrà generato automaticamente, e le colonne saranno disposte in modo ordinato. Nel caso si vogliano avere le colonne ordinate in un preciso ordine sarà sufficiente passare i nomi delle colonne nell'ordine desiderato in una lista, al parametro columns del costruttore; mentre per definire un index sarà necessario passare una lista di indici al parametro index.

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

Out []:

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

Esempio in cui si generano numeri da 0 a 15 e passiamo una lista a index 1 e a columns.

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

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

Out [ ]:

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

Un altro modo è assegnando alla colonna una lista di uguale lunghezza della colonna. In questo caso si generano una lista di 4 numeri che a partire da 200, distano 7 uno dall’altro:

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

Out [ ]:

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

L’eliminazione di righe o colonne è altrettanto semplice, sarà necessario il metodo drop in cui si indica l’indice della riga che si voglia eliminare, mentre nel caso delle colonne, oltre all’indice, sarà necessario passare il parametro axis=1.

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

Out []:

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

Per terminare questa breve infarinatura sui DataFrame, si andrà a vedere come, nelle operazioni aritmetiche tra tabelle, l'allineamento sugli indici sia gestito in modo automatico. Si considerino i 2 seguenti DataFrame:

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

Out []:

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

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

Out []:

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

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

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

Out [ ]:

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

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

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

Out [ ]:

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

3.1.4 Strutture per la visualizzazione, il Matplotlib

Matplotlib è stata la prima libreria a disposizione degli utenti Python per poter disegnare grafici, in particolar modo bidimensionali. Ha origine dall'interfaccia di Matlab (ma è indipendente poiché scritta completamente con classi Python) e NumPy. È l'unica libreria grafica delle proposte parte integrante della suite SciPy, nata dall'idea di creare una libreria che permettesse di produrre in modo semplice grafici di alta qualità e facile comprensione. concettualmente diviso in tre parti:

1. L'interfaccia pylab: disponibile in matplotlib.pylab, consente all'utente di creare i grafici con un'interfaccia simile a quella di matlab.

2. Frontend: il set di classi adibite alla creazione e disposizione di figure, testi, grafici, legende.

3. Backend: classi che trasformano e salvano la rappresentazione ottenuta in un file (come ps, svg, gtk, agg per il formato png, pdf, ecc. . .).

http://matplotlib.org, licenza basata su PSF

https://it.wikipedia.org/wiki/Encapsulated_PostScript

https://it.wikipedia.org/wiki/Portable_Network_Graphics

3.2 Database e MySQL

Oggigiorno l'evoluzione della tecnologia informatica consente una ricerca più rapida ed efficace di qualsiasi tipo di informazione ed in qualsiasi postazione ci si trovi. L'innovazione più importante per questo tipo di evoluzione è il database, un insieme di dati suddivisi per categorie secondo uno schema logico, le tabelle, e poi ordinati per caratteristiche, i campi. Ogni informazione contenuta all'interno del database viene quindi inserita in una particolare tabella e caratterizzata da valori specificati dai vari campi: tale informazione viene chiamata record.

I database vengono creati, gestiti ed interrogati da un sistema software chiamato DataBase Management System, o DBMS: è un sistema di gestione dei dati che garantisce un livello di sicurezza ai dati, permettendone una condivisione sicura ed affidabile agli utenti.

Il database quindi deve essere progettato e costruito per consentire l'accesso, tramite i DBMS, alle informazioni in esso contenute ed alla sua struttura da parte degli utenti: per questo sono stati sviluppati diversi tipi di linguaggi, a seconda del loro utilizzo:

• Data Definition Language (DDL): consente di definire la struttura della base di dati e le autorizzazioni per l'accesso;

• Device Media Control Language (DMCL): permette alla struttura fisica del database di far riferimento alle particolari unit`a di memoria di massa utilizzate dal sistema;

• Data Manipulation Language (DML): consente di interrogare e aggiornare le istanze della base di dati;

• Data Control Language (DCL): permette la gestione dell'accesso al database con relative restrizioni di operazioni come aggiornamento, selezione e cancellazione;

• Query language (QL): permette di interrogare il database al fine di ritrovare i dati relativi alla chiave di ricerca impostata dall'utente. Il linguaggio pi`u diffuso e basilare per interagire con un database `e il linguaggio SQL.

3.2.2 Il linguaggio SQL

L'SQL (Structured Query Language) è un linguaggio di tipo dichiarativo usato in sistemi basati sul modello relazionale per leggere, modificare e gestire dati memorizzati nei database, per creare e modificare schemi di database, per creare e gestire strumenti di controllo ed accesso ai dati. Le funzioni principali del linguaggio SQL sono quello di interrogare, aggiornare ed inserire dati all'interno di un database; tali operazioni, basilari del linguaggio SQL, vengono effettuate attraverso l'uso di particolari istruzioni, denominate query e Data Manipulation Language (DML). Queste operazioni permettono all'utente di descrivere quali sono le informazioni a lui necessarie, lasciando al DBMS la responsabilità di amministrarle, disporle e effettuare le operazioni fisiche per produrne visivamente i risultati. Sono righe di comando, in codice SQL, che descrivono al DBMS le operazioni da effettuare: vengono definite attraverso l'uso di parole chiave che ne indicano le istruzioni da eseguire, e da un elenco di attributi che indica su quali tabelle, campi o record sono da effettuare le operazioni. Mentre le istruzioni DML possono modificare la struttura del database, le query servono per richiedere un set particolare di informazioni; inoltre le query possono essere collegate tra loro grazie a particolari operatori insiemistici, atti a confrontare risultati di più richieste, oppure possono essere utilizzate per formulare tabelle necessarie ad un'altra query.

3.2.3 Istruzioni di modifica, eliminazione e creazione

I comandi SQL di modifica, eliminazione e creazione possono interagire con la struttura del database: esse, accedendo ai dati, possono modificare, creare o cancellare tabelle, campi e record. Sono delle operazioni delicate, che possono compromettere l'integrità delle informazioni contenute nella base dati. Per questo, per determinati elementi, vengono creati dei permessi affinchè solo chi ne è in possesso possa operare su di essi; in questo modo si previene la modifica al database da parte di utenti non autorizzati, lasciandone la responsabilità solo a chi di competenza. Le parole chiave sono molteplici per queste operazioni, di seguito verranno elencate le più comuni:

• UPDATE: ha la funzione di modificare i dati delle tabelle; può modificare, a seconda della sintassi usata, un solo campo, più campi contemporaneamente, tutti i campi oppure solo una determinata selezione dei campi utilizzando il comando WHERE;

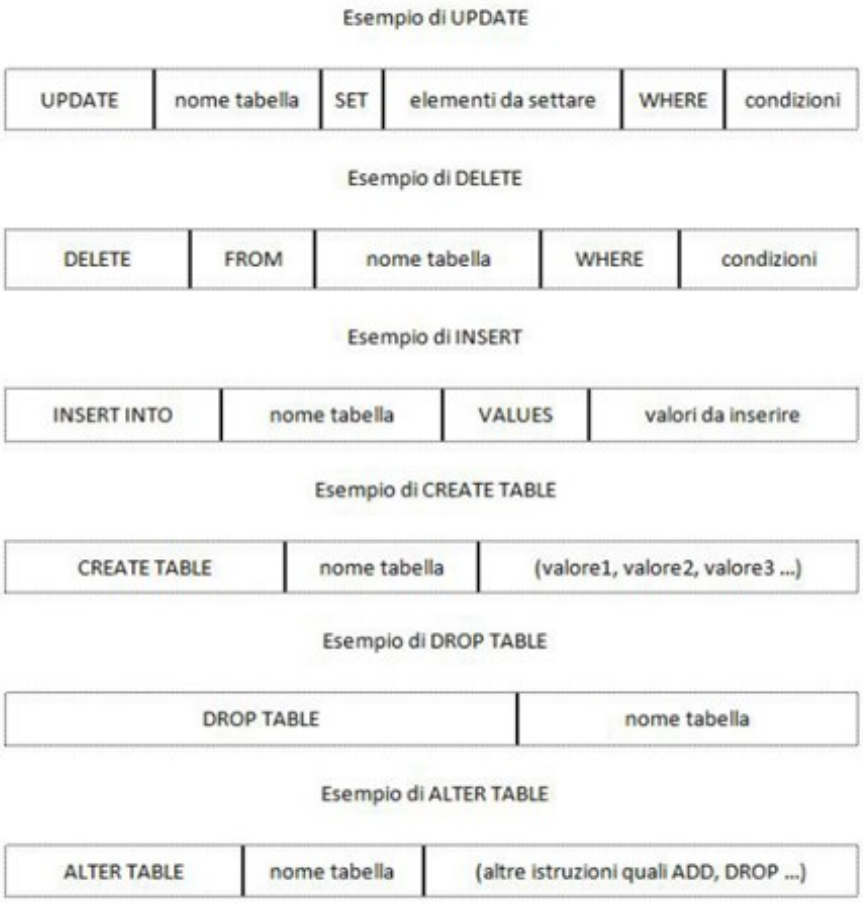
• DELETE: ha la funzione di cancellare i dati dalle tabelle; come il comando update anche delete può operare in modo generico cancellando tutti i record della tabella, oppure può identificare i record da cancellare mediante la parola chiave aggiuntiva WHERE e la condizione (o le condizioni) ad essa associata;

• INSERT: ha la funzione di inserire i dati nelle tabelle;

• CREATE TABLE: ha la funzione di creare tabelle all'interno della base dati; dopo il comando si possono specificare, record per record, i campi e i relativi valori da associarvi;

• DROP TABLE: ha la funzione di eliminare una tabella;

• ALTER TABLE: ha la funzione di modificare i dati contenuti in una tabella.



3.2.4 Le query

La query è il costrutto più importante e più usato del linguaggio SQL: è una funzione che verrà utilizzata ampliamente nel progetto e permette di interrogare il database in modo da ottenere delle informazioni ivi contenute filtrandole, a seconda delle esigenze, con l'introduzione di parole chiave. La sintassi di una query è ben definita a priori, con regole precise che ne identificano ogni singola componente; le due parole chiave obbligatorie per la formulazione di una query sono:

• SELECT: indica i campi che si intendono ricercare nelle tabelle specificate e ne evidenziano il risultato;

• FROM: indica le tabelle in cui l'utente intende ricercare i valori dei campi richiesti. Questo semplice costrutto, sebbene poco articolato, è già in grado di produrre un risultato. Nel linguaggio SQL sono state implementate altre

parole chiave per le query che, seppur facoltative, ne aumentano la potenza espressiva, e per questo sono frequentemente utilizzate:

• WHERE: questa parola chiave indica delle condizioni che devono essere verificate nell'interrogazione al database: infatti verranno visualizzate nel risultato solo ed esclusivamente i record che abbiano verificato le condizioni qui espresse. L'omissione di questa clausola indica che la query selezionerà tutti i record presenti nel risultato del prodotto cartesiano delle tabelle nella clausola FROM;

• GROUP BY: i record risultanti dalla query verranno suddivisi in gruppi a seconda del valore del campo qui inserito, chiamato campo di raggruppamento;

• HAVING: usata solo in presenza della clausola GROUP BY, viene utilizzata per specificare delle condizioni su ciascun gruppo, corrispondente ad ogni distinto valore del campo di raggruppamento: solo i gruppi per i quali la condizione `e soddisfatta entrano a far parte del risultato della query;

• ORDER BY: questa parola chiave identifica quali campi vengono utilizzati per ordinare i risultati ed in che senso vengono ordinati (crescente o decrescente). Senza questa specifica, il risultato della query `e semplicemente non definito. Il risultato di una query `e quindi una tabella costituita da record:

• che appartengono alle tabelle elencate nella clausola FROM;

• che soddisfano le condizioni espresse della clausola WHERE (ove presente);

• che sono proiettate sulla lista di attributi specificati nella clausola SELECT;

• raggruppati secondo il campo espresso in GROUP BY (ove presente);

• rispettanti la condizione di raggruppamento inclusa in HAVING (ove presente);

• ordinati secondo le specifiche di ORDER BY (ove presente).

Lo standard SQL prevede una logica a tre valori per la soluzione di operazioni tra valori nulli: vero, falso e sconosciuto. In generale, le clausole presenti in maggior parte nelle query sono SELECT, FROM e WHERE;

l’omissione di quest’ultima non crea problemi nella correttezza della query, ma porta il sistema a fornire un risultato corposo, privo di ogni logica e dispendioso in termini di tempo di calcolo. Ognuna delle clausole sopra elencate e i loro successivi attributi, formano una parte importante nella formulazione di una query. Non solo, ognuna di esse al suo interno è dotata di altre parole chiave che apportano altre condizioni ed operazioni interne alle specifiche e concorrenti nella formazione del risultato.

Capitolo 4: Strutturazione dati Luxottica

4.1 Analisi e creazione della base di dati

E’ stato analizzato un documento in Excel dell’azienda Luxottica che contiene le seguenti informazioni:

Il file è strutturato in diversi fogli.

- Model Components Hubs: 3 colonne indipendenti con la lista dei componenti, dei modelli e dei centri di produzione (hubs).

- BOMxComponents: tabella che rappresenta la lista dei componenti utilizzati per la produzione di ogni modello, con la quantità utilizzata e la relativa unità di misura.

- MinLotProfit: tabella che rappresenta la quantità minima di produzione e il profitto in ogni prodotto.

- ComponentsStocked: quantità di componenti stoccati (immagazzinati) per ogni centro.

- ProductionCostComponents: costo di produzione per ogni componente di ogni centro.

- StockedCostComponents: valore dei componenti stoccati per ogni centro.

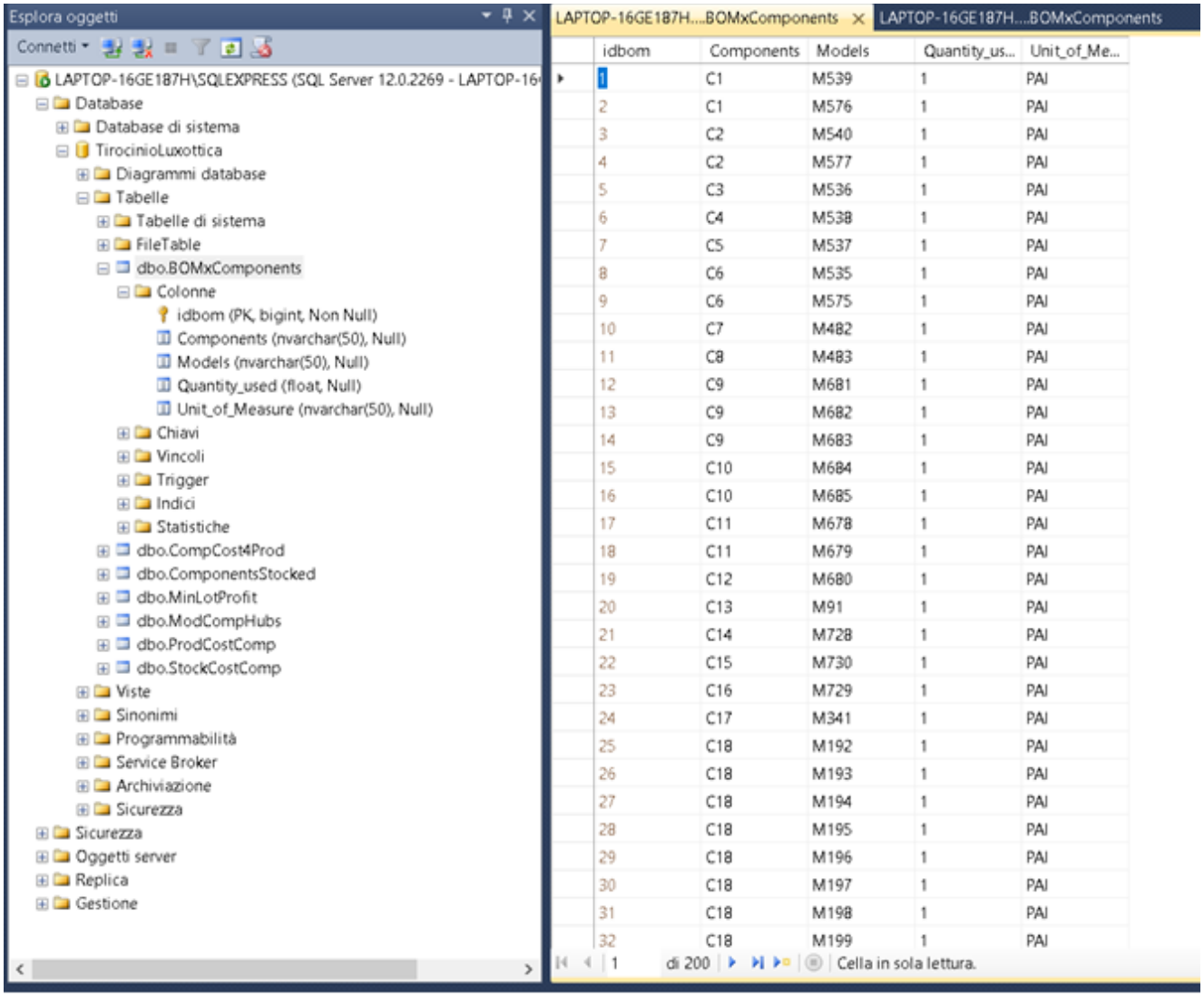
- ComponentsCost4Production: costo dei componenti usati per la produzione.

- Measures: unità di misura per ogni componente.

- TransferCost: costo di trasporto di ogni componente tra i vari centri di produzione.

Per gestire il tutto, l'insieme di dati è stato trasferito su di un database.

La base di dati è stata realizzata in un server locale con il software SQL Server Management Studio.



Esempio di visualizzazione del foglio BOMxComponents nel database.

E con le seguenti linee di comando in Python è stato possibile interagire con la basi di dati attraverso il software Pycharm, tramite l'implementazione della libreria Pandas descritta nel capitolo precedente.

```
cs = 'DRIVER={SQL Server};SERVER=LAPTOP-16GE187H\SQLEXPRESS;DATABASE=TirocinioLuxottica;Trusted_Connection=yes'

connStr = pyodbc.connect(cs)
cursor = connStr.cursor()
```

4.2 Identificazione dei fattori chiave

Il prossimo obiettivo è quello di eseguire un'analisi statistica tramite l'individuazione di opportune correlazioni tra le varie voci e ottenere una descrizione sintetica del fenomeno oggetto di studio che ne permetta l'interpretazione e la previsione.

4.2.1 Correlazione Components - Models:

La prima correlazione è stata individuata dalla struttura dati "Componenti" e "Modelli" nel foglio BOMxComponents ,che permetterà di individuare qual è il modello con più componenti tramite la funzione .count, che nel linguaggio python conta il numero delle occorrenze in una lista. Innanzitutto sono state selezionate le voci interessate tramite una query nel database , eseguendo una SELECT (spiegazione capitolo 2) di tutti i componenti e modelli presenti nel foglio. Successivamente quest'informazione è stata inserita in un dataframe denominato dfBOM. Una volta raggruppati i dati selezionati tramite la funzione di aggregazione GROUP-BY vengono conteggiati tutti i componenti utilizzati per ogni modello tramite la funzione .count. Lo stesso, è stato iterato per calcolare il componente più utilizzato. Successivamente tramite la funzione di ordinamento .sort si è ottenuto una lista di dati ordinata per modello (con più componenti) e per frequenza (numero di componenti usati maggiornamente) .

Components	Freq	Models	CompxModel
C1570	370	M243	32
C1570	370	M247	32
C1573	303	M243	32
C1573	303	M247	32
C1054	124	M243	32
C1054	124	M247	32
C2612	103	M243	32
C2612	103	M247	32
C2521	83	M243	32
C2521	83	M247	32
C1055	62	M243	32
C1056	62	M247	32
C2631	50	M243	32
C2631	50	M247	32
C837	47	M243	32
C837	47	M247	32
C833	45	M243	32
C833	45	M247	32
C2344	43	M243	32
C2344	43	M247	32

Prendendo in esame il componente C1570 la rappresentazione ci mostra che in totale il componente è utilizzato 370 volte, è presente nel modello M243 e che a sua volta quest'ultimo contiene 32 componenti.

Il componente più utilizzato è il C1204 e il modello con più componenti è l'M892.

Components	Freq	Models	CompxModel
C1204	466	M892	37

Per un'ulteriore verifica è stata eseguita direttamente la query nel database che estrapola i medesimi risultati. Le query sono le seguenti:

```
stringaQuerySelect='SELECT Models, count(*) as Quantità FROM BOMxComponents '
                   'GROUP BY Models order by Quantità DESC'
```

Lista con ordinamento dal modello con più componenti.

```
stringaQuerySelect='SELECT Components, count(*) as Quantità FROM BOMxComponents '
                   'GROUP BY Components order by Quantità DESC'
```

Lista con ordinamento dal componente più utilizzato.

4.2.2 Correlazione Hub - StockedCostComponents:

La seconda correlazione è stata eseguita tra i vari centri di produzione (Hubs) e i costi di stoccaggio, per poter individuare il magazzino con più spese. Identicamente come con la correlazione precedente, viene eseguita una select sul foglio di dati interessati dove quest'ultimi vengono inseriti in un DataFrame. I dati presi in esame sono i costi di stoccaggio nei centri H1 , H2, H3 e H4.

Sono state inizializzate quattro variabili che fungono da contatore, attraverso un ciclo for vengono scansionati i costi di stoccaggio tramite la funzione .max che individua il valore massimo della riga i-esima. Con l'istruzione if viene imposta la condizione che se il valore massimo equivale al costo di stoccaggio nel magazzino H1 allora verrà incrementato il contatoreH1 .

H1Stock	H2Stock	H3Stock	H4Stock
7.90	3.18	7.90	7.90

Nella riga seguente il valore massimo che equivale ad H1Stock, H3Stock e H4Stock. Dunque:

contatoreH1=contatoreH1+1

contatoreH3=contatoreH3+1

contatoreH4=contatoreH4+1

Il risultato finale è il seguente:

Contatore_H1	Contatore_H2	Contatore_H3	Contatore_H4
2856.00	2968.00	3083.00	3032.00

Il magazzino H3 è il magazzino con più spese.

4.2.3 Correlazione BOMxComponents - ProductionCostComponents:

Per poter fornire un indicatore di sintesi alquanto di rilievo nell'ambito aziendale, ovvero la media aritmetica dei costi dei modelli, è stata eseguita

una terza correlazione tra la tabella BOMxComponents e la tabella ProductionCostComponents.

Come in precedenza viene eseguita la select nell'insieme di dati che si vuole prendere in esame. In questo caso viene eseguita nel foglio BOMxComponents per poter individuare il numero di componenti presenti in ogni modello, e nel foglio ProductionCostComponents per individuare i costi di ogni componente del modello in questione. Nel ciclo for vengono inizializzate le seguenti variabili:

sommatoriaH1=0; sommatoriaH2=0; sommatoriaH3=0; sommatoriaH4=0 .

In seguito vengono conteggiati il numero dei componenti che compongono il modello in questione tramite la funzione .count e con un secondo ciclo for interno viene scannerizzato il foglio ProductionCostComponents e viene eseguita la sommatoria per ogni costo del componente presente.

```
sommatoriaH1 += costoComponente['H1Prod'].iloc[0]
sommatoriaH2 += costoComponente['H2Prod'].iloc[0]
sommatoriaH3 += costoComponente['H3Prod'].iloc[0]
sommatoriaH4 += costoComponente['H4Prod'].iloc[0]
```

.iloc estrapola il contenuto della prima riga, ovvero il contenuto del costo del primo componente. Alla fine del ciclo for sommatoriaH1 corrisponderà alla sommatoria di tutti i costi dei componenti.

Tramite l'operazione:

```
mediaH1=  sommatoriaH1 / len(components)
mediaH2 = sommatoriaH2 / len(components)
mediaH3 = sommatoriaH3 / len(components)
mediaH4 = sommatoriaH4 / len(components)
```

si ottiene la media aritmetica dei costi dei componenti di un singolo modello. La funzione len restituisce il numero di oggetti contenuti in components (il numero di componenti del modello).

Il risultato finale applicato a tutti i centri e applicato ad ogni modello è il seguente:

	Models	MEDIA_H1	MEDIA_H2	MEDIA_H3	MEDIA_H4	NumeroComponenti
538	M584	14.70067	14.70021	14.70029	14.70021	15
470	M522	14.69766	14.69719	14.69728	14.69719	15
537	M583	14.46770	14.46724	14.48999	14.46724	15
469	M521	14.46708	14.46661	14.49670	14.46661	15
536	M582	14.39486	14.39440	14.39448	14.39440	15
468	M520	14.13854	14.13808	14.16616	14.13808	15
775	M798	13.92298	14.07211	14.08607	14.08602	13
301	M370	12.40887	13.02736	13.00629	13.00629	3
781	M802	12.02226	12.00707	12.02501	12.02495	12

MediaH1 è la media dei costi del modello in questione, con la seguente rappresentazione si può notare il numero dei componenti che non è sempre proporzionale al costo di produzione del modello, come nel caso di M802.

	Models	MEDIA_H1	MEDIA_H2	MEDIA_H3	MEDIA_H4	NumeroComponenti
763	M787	2.01666	2.09318	2.09266	2.09264	31

O come nel modello M787.

4.2.4 Correlazione MinLot - Profit:

Analizzando la tabella MinLotProfit è stata calcolata la spesa minima di produzione per ciascun magazzino. Come descritta in precedenza, la seguente tabella descrive sostanzialmente la quantità minima di produzione di ciascun modello con i relativi prezzi. Innanzitutto è stata preparata la struttura dati con cui andare a eseguire ogni calcolo interessato e di seguito sono state concatenate le due strutture dati:

```
dfconcat = pd.concat([dfLot, dfAnalisi], axis=1 )
```

dfAnalisi è il dataframe nel quale è stata riportata la media aritmetica. Nel nuovo dataframe dfconcat (che contiene tutte le informazioni del foglio dati MinLotProfit e della media aritmetica) è stata creata un'ulteriore colonna denominata CostProdH1 (che conterrà i costi di produzione di ogni singolo centro) nel modo seguente:

```
dfconcat = pd.concat([dfLot, dfAnalisi], axis=1 )
dfconcat['CostProd_H1']=0
```

In questo modo è stata creata la colonna CostProd_H1 di tipo float, inizializzata a zero.

Per calcolare la spesa minima di produzione è stato eseguito in un ciclo for (in modo tale da reiterare il calcolo per ogni modello) il seguente calcolo:

```
dfLotCost.at[id, 'CostProd_H1'] = ((costomodelloH1)*(lottominimo))
```

La funzione .at trascrive l'informazione (nella colonna CostProd_H1) contenuta nell'operazione di destra, ovvero il costo del modello in H1 moltiplicato per la sua quantità di produzione minima. Le variabili costomodelloH1 e lottominimo sono variabili che cambiano per ogni ciclo e che contengono le informazioni descritte in precedenza tramite le seguenti linee di codice:

```
for id, row in df4.iterrows():
    costomodelloH1=dfLotCost[dfLotCost['Models']==row['Models']]['COSTmodel_H1']
    costomodelloH2=dfLotCost[dfLotCost['Models']==row['Models']]['COSTmodel_H2']
    costomodelloH3=dfLotCost[dfLotCost['Models']==row['Models']]['COSTmodel_H3']
    costomodelloH4=dfLotCost[dfLotCost['Models']==row['Models']]['COSTmodel_H4']
```

(Trova nel seguente dataframe e nella riga i-esima, il costo del modello i-esimo).

```
lottominimo=dfLotCost[dfLotCost['Models']==row['Models']]['MinLot']
```

(Trova nel seguente dataframe e nella riga i-esima, la quantità di produzione del modello i-esimo).

Ed eseguendo la somma di tutti i costi produzione di ogni magazzino possiamo trovare la spesa minima di produzione di quest'ultimi con le seguenti linee di codice:

```
for id, row in dfLotCost.iterrows():

    costoH1=dfLotCost['CostProd_H1']
    costoH2=dfLotCost['CostProd_H2']
    costoH3=dfLotCost['CostProd_H3']
    costoH4=dfLotCost['CostProd_H4']
```

In un ciclo for vengono estrapolati i costi di produzione del magazzino e consecutivamente vengono sommati e inseriti nella variabile sommatioriaH1min.

```
sommatoriaH1min += costoH1.iloc[0]
sommatoriaH2min += costoH2.iloc[0]
sommatoriaH3min += costoH3.iloc[0]
sommatoriaH4min += costoH4.iloc[0]
```

Il risultato finale è il seguente:

Spesa_MagazzinoH1	8752053.80
Spesa_MagazzinoH2	8923697.70
Spesa_MagazzinoH3	8932554.26
Spesa_MagazzinoH4	8932523.63

4.3 Regressione lineare

In questo capitolo si andrà ad introdurre il modello statistico di regressione lineare; lo stesso consentirà di valutare il grado di dipendenza di una serie di variabili (solitamente indicate da Y) confrontandole con una serie di altre variabili (note come variabili indipendenti, o casuali, solitamente indicate da X).

L'obiettivo è capire se esiste una relazione di tipo lineare tra i costi di produzione e i prezzi di vendita dei modelli, dove il costo di produzione dei modelli è la variabile dipendente (y) mentre il prezzo di vendita è la variabile indipendente (X. Viene calcolato in primo luogo il prezzo di vendita , analizzando il profitto e i costi di produzione di ciascun modello:

Sapendo il profitto dalla tabella 'MinLotProfit', il prezzo di vendita sarà la somma tra il costo di produzione e il profitto di ciascun modello. Concatenando i dataframe 'Models'(Elenco di tutti i modelli) , 'COSTmodel_H1'(Costo di produzione) e 'Profit' (Profitto) è stato calcolato il prezzo di vendita.

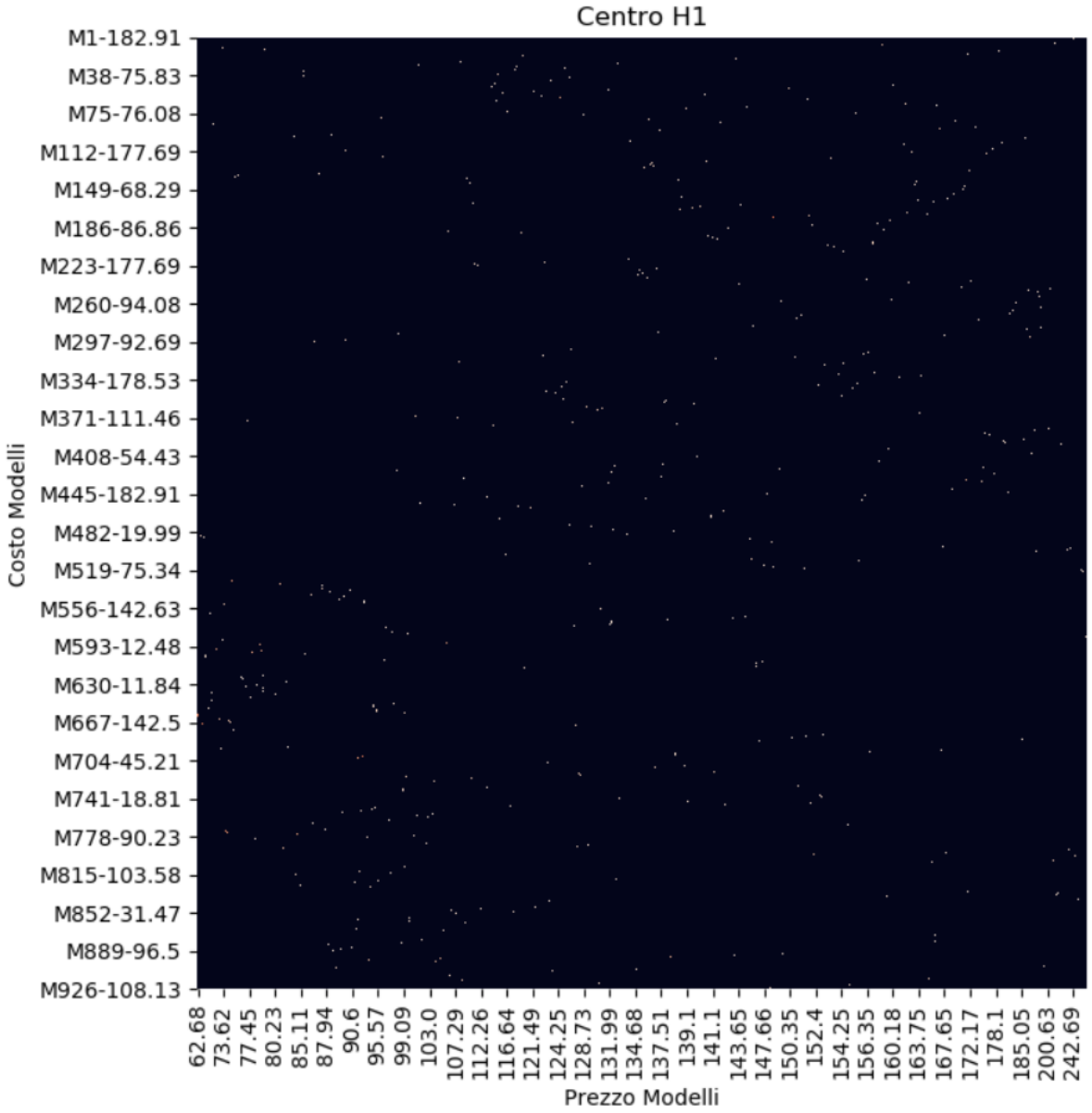
```
dfcostmodelH1 = dfregModel[['COSTmodel_H1']]
dfModels = dfBOM['Models']
dfprofitReg = dfLot[['Profit']]
dfLotCost['sellprice_H1']=0
dfLotCost['sellprice_H1']=dfLotCost['sellprice_H1'].astype(float)
dfLotCost['sellprice_H2']=0
dfLotCost['sellprice_H2']=dfLotCost['sellprice_H2'].astype(float)
dfLotCost['sellprice_H3']=0
dfLotCost['sellprice_H3']=dfLotCost['sellprice_H3'].astype(float)
dfLotCost['sellprice_H4']=0
dfLotCost['sellprice_H4']=dfLotCost['sellprice_H4'].astype(float)
```

```
for id, row in dfLotCost.iterrows():
    costomodelloH1=dfLotCost[dfLotCost['Models']==row['Models']]['COSTmodel_H1']
    costomodelloH2=dfLotCost[dfLotCost['Models']==row['Models']]['COSTmodel_H2']
    costomodelloH3=dfLotCost[dfLotCost['Models']==row['Models']]['COSTmodel_H3']
    costomodelloH4=dfLotCost[dfLotCost['Models']==row['Models']]['COSTmodel_H4']
    profit=dfLotCost[dfLotCost['Models']==row['Models']]['Profit']
    dfLotCost.at[id, 'sellprice_H1'] = ((costomodelloH1)+(profit))
    dfLotCost.at[id, 'sellprice_H2'] = ((costomodelloH2)+(profit))
    dfLotCost.at[id, 'sellprice_H3'] = ((costomodelloH3)+(profit))
    dfLotCost.at[id, 'sellprice_H4'] = ((costomodelloH4)+(profit))
    dfLotCost['sellprice_H1']=dfLotCost['sellprice_H1'].round(decimals=2)
    dfLotCost['sellprice_H2']=dfLotCost['sellprice_H2'].round(decimals=2)
    dfLotCost['sellprice_H3']=dfLotCost['sellprice_H3'].round(decimals=2)
    dfLotCost['sellprice_H4']=dfLotCost['sellprice_H4'].round(decimals=2)
```

A seguire è stato creato il dataframe ‘dfregModels’ con i dati necessari per l’analisi del modello statistico.

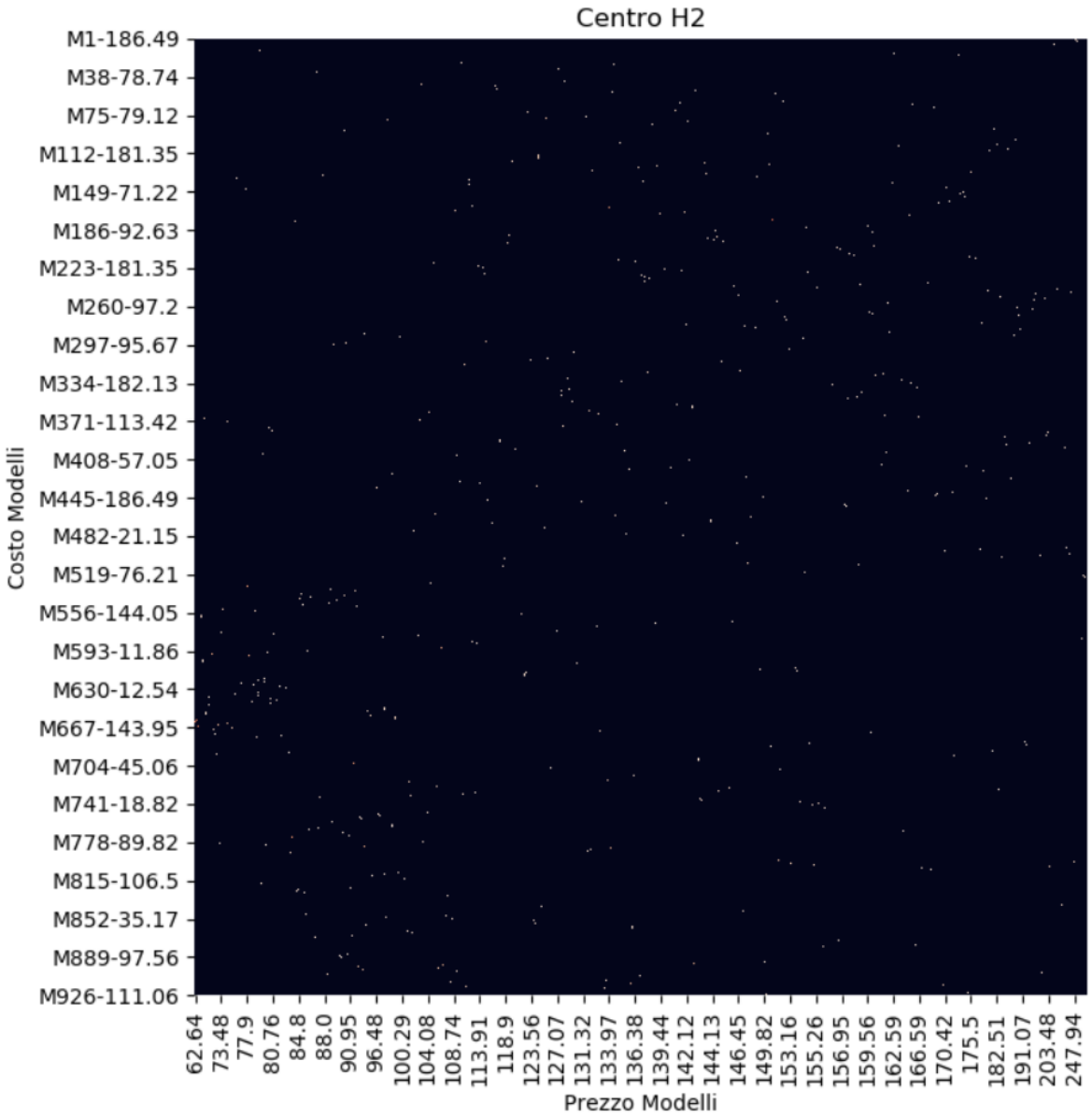
```
dfregModels = pd.concat([dfprofitReg, dfregModel, dfModels, dfLotCost], axis=1)
dfregModels= (dfLotCost[['Models','Profit','COSTmodel_H1','COSTmodel_H2','COSTmodel_H3',
                  'COSTmodel_H4','sellprice_H1','sellprice_H2','sellprice_H3','sellprice_H4']])
```

Prima di creare il modello di regressione, è stata rappresentata graficamente la distribuzione dei profitti attraverso una mappa di calore per capire se esiste una correlazione tra i costi dei diversi centri. Per tale rappresentazione è servito l’utilizzo del pivot sul dataframe ‘dfregModels’ per rappresentare i modelli in questione con i relativi costi sull’asse delle ordinate, e i prezzi di vendita associato sull’asse delle ascisse.



Ad ogni modello è associato un costo di produzione (asse y) alla quale è associato un prezzo di vendita (asse x), e l'intersezione degli assi rappresenta il profitto (nodo bianco).

Rappresentando il grafico anche sul centro H2 si è ottenuto il seguente risultato:



Si può notare a prima vista come la distribuzione dei profitti tra i due centri è similare. Cio' significa che il costo di produzione tra i vari centri cambia esiguamente. Difatti il costo di produzione dei modelli (calcolato nel capitolo precedente) è il seguente:

	Profit	COSTmodel_H1	COSTmodel_H2	COSTmodel_H3	COSTmodel_H4
M1	68.00000	182.90604	186.49316	186.67825	186.67761
M2	65.00000	177.69396	181.34611	182.39107	182.39043
M3	67.00000	177.69396	181.34611	182.39107	182.39043
M4	67.00000	178.52851	182.13183	182.24887	182.24823
M5	64.00000	182.90604	186.49316	186.67825	186.67761
M6	58.00000	142.63120	144.05327	144.03553	144.03489
M7	67.00000	142.49596	143.94862	144.12872	144.12808
M8	65.00000	90.22524	89.81950	90.01451	90.01451
M9	62.00000	96.49663	97.55644	97.14792	97.14792
M10	67.00000	9.38555	8.78598	10.22032	10.21968

Se i valori fossero stati maggior distanti tra di loro avremo avuto una distribuzione dei nodi diversificata. In aggiunta anche i profitti sono molto somiglianti., in caso contrario avremo avuto una tonalità del colore diversa:

Tonalità dei nodi rappresentante i profitti.

Si è verificato che anche i centri H3 e H4 hanno all'incirca la stessa distribuzione.

Per la creazione del modello di regressione si è provveduto a suddividere il dataset in set di dati di test e di allenamento. In questo caso è stato considerato il 33% del set di dati come dato di test, mentre il 66% sarà il set di dati di addestramento.

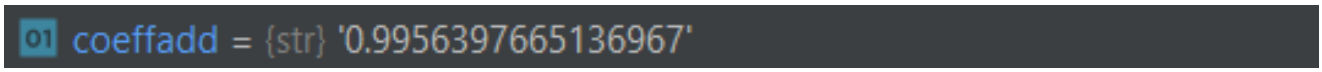
```
X_train, X_test, y_train, y_test=train_test_split(dfregModels['sellprice_H1'],
                                                  dfregModels['COSTmodel_H1'], test_size=0.33, random_state=42)
```

E con il seguente script è stato creato il modello:

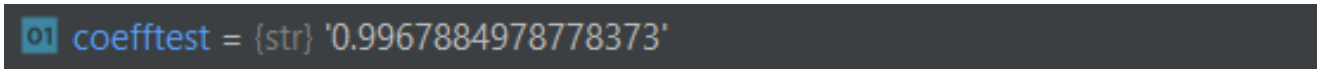
```
reg=linear_model.LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
reg.fit(X_train.values.reshape(-1, 1), y_train.values.reshape(-1, 1))
reg.coef_
coeffadd=str(reg.score(X_train.values.reshape(-1, 1), y_train.values.reshape(-1, 1)))
coefftest=str(reg.score(X_test.values.reshape(-1, 1), y_test.values.reshape(-1, 1) ))
```

Viene assegnata alla variabile reg il valore del modello di regressione ed eseguito poi l'adattamento del modello di regressione ai dati.

Con il termine reg.coef vengono stimati i coefficienti del problema di regressione e stampate dalle successive righe di codice il coefficiente di determinazione, e il coefficiente previsto del dataset di test).



Coefficiente di determinazione relativo al dataset di addestramento.

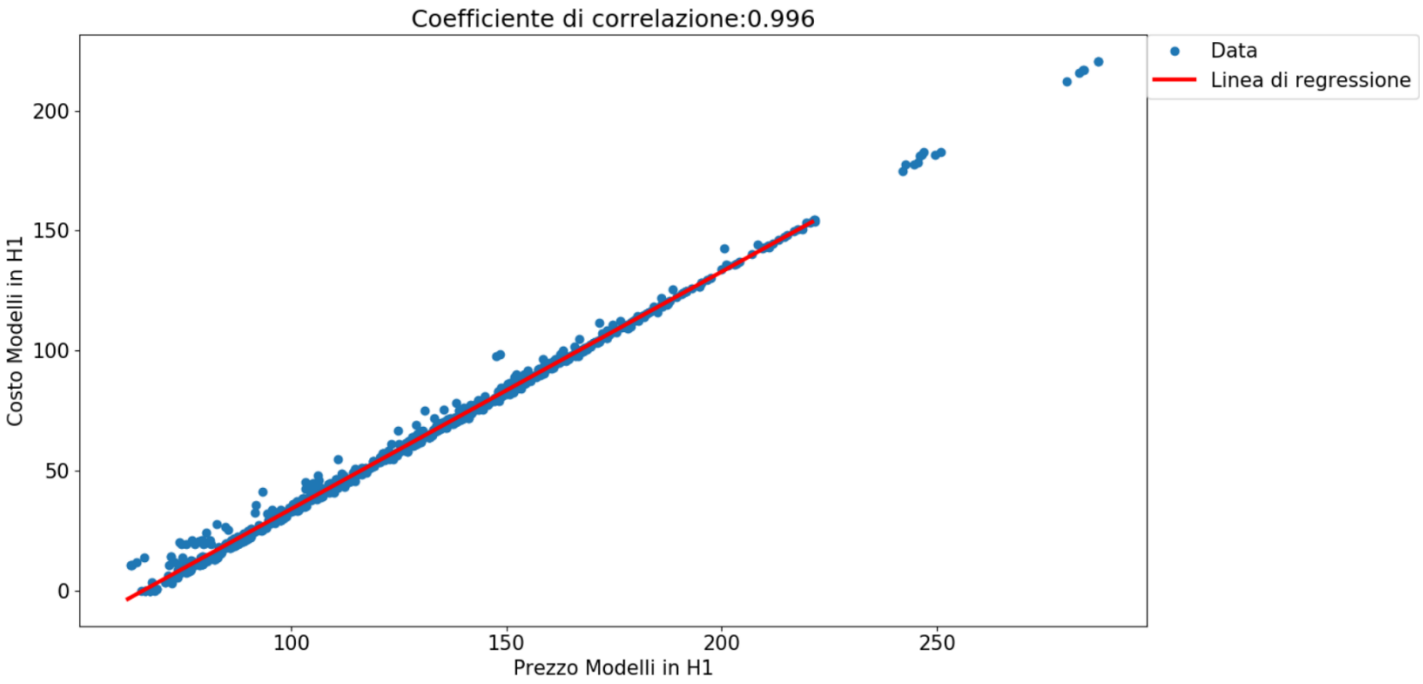


Coefficiente di determinazione previsto del dataset di test.

Ed è stato rappresentato quanto trovato con le seguenti righe di codice:

```
font={'size': 15}
plt.rc('font', **font)
plt.plot(dfregModels['sellprice_H1'], dfregModels['COSTmodel_H1'], 'o', label='Data')
I=np.linspace(np.floor(min(dfregModels['sellprice_H1'])), np.ceil(max(dfregModels['COSTmodel_H1'])),50)
plt.plot(I, reg.predict(I.reshape(-1, 1)), color='r', linewidth=3, label='Linea di regressione')
plt.legend(bbox_to_anchor=(1, 1), borderaxespad=0.)
plt.xlabel('Prezzo Modelli in H1')
plt.ylabel('Costo Modelli in H1')
Preds=reg.predict(dfregModels['sellprice_H1'].values.reshape(-1, 1))
R2=r2_score(dfregModels['COSTmodel_H1'],Preds)
plt.title('Coefficiente di correlazione:'+ str(np.round(R2,decimals=3)))
plt.show()
```

Vengono impostati i caratteri, le etichette, i titoli e le varie dimensioni del grafico e viene ottenuto il seguente risultato:



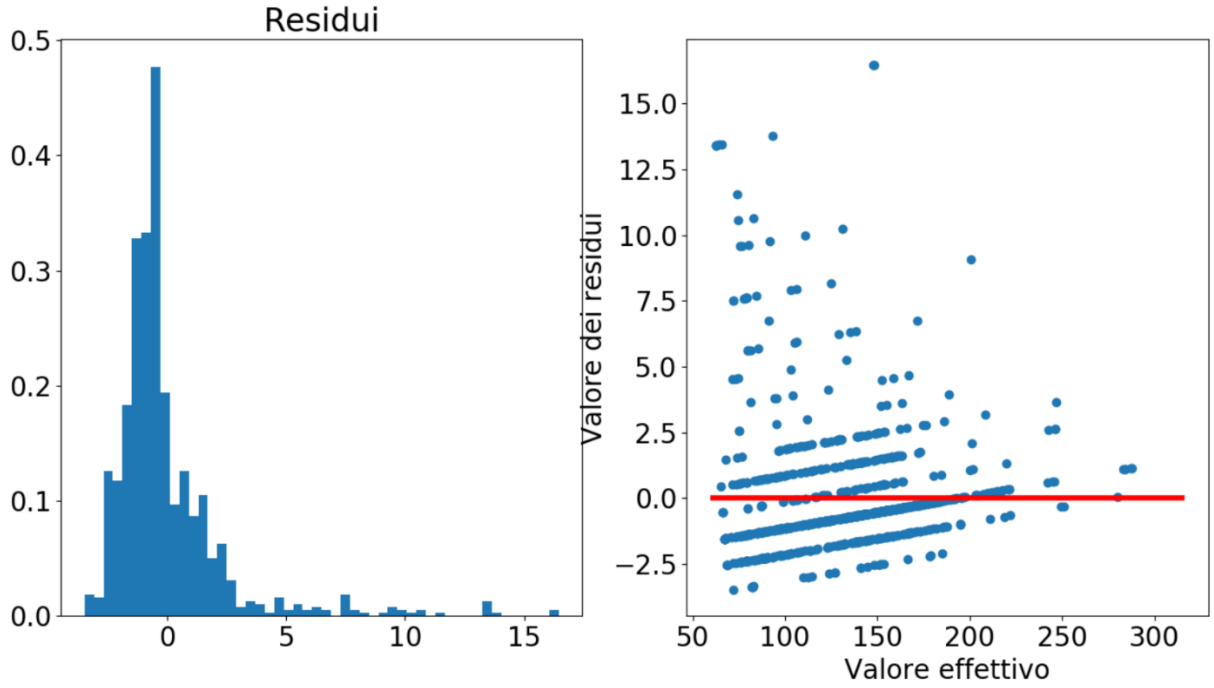
Si può notare che il valore ottenuto è un alto valore del coefficiente di correlazione (R2 = 0.996). Da ciò si può confermare come i costi di produzione dei componenti che compongono i modelli influenzino il prezzo dei modelli.

Per concludere è stato rappresentato graficamente il valore dei residui, ossia una stima osservabile dell'errore statistico. Anche in questo caso vengono definite le proprietà dei grafici:

```
Residuals=dfregModels['COSTmodel_H1'].values.reshape(-1, 1)-Preds
font={'size': 20}
plt.rc('font', **font)
fig, ax=plt.subplots(1,2,figsize=(20,10))
num_bins=50
n, bins, patches=ax[0].hist(Residuals, num_bins, density=1)
ax[0].title.set_text('Residui')
ax[1].plot(dfregModels['sellprice_H1'],Residuals, 'o')
ax[1].set(xlabel='Valore effettivo', ylabel='Valore dei residui')
ax[1].hlines(0, np.min(dfregModels['sellprice_H1'])*0.95,
             np.max(dfregModels['sellprice_H1'])*1.1, colors='r', linestyles='solid', zorder=10, linewidth=4)
plt.show()
```

Tale script da come output il seguente risultato: il primo rappresenta la distribuzione dei residui per i prezzi dei modelli, mentre il secondo la

distribuzione dei residui per quanto riguarda i costi.



4.4 T-test di Student

E' stata effettuata un'ultima analisi sui costi con il test t di Student, controllando le medie di due campioni (in questo caso i costi dei modelli tra i vari centri) per capire se sono significativamente diversi l'uno dall'altro. E' stato calcolato l'errore standard per trovare il valore t e vedere quanto sia probabile che i due campioni hanno la stessa media (differenza uguale zero, ovvero ipotesi nulla), in altre parole se la differenza osservata nei costi dei due centri sia dovuta solo all'errore campionario.

Analizzando i costi tra i centri H1 e H2 con le seguenti linee di codice:

```
tH1H2=0
N=957

#std deviation
sH1H2 = np.sqrt(((dfAnalisi['COSTmodel_H1'].var())+(dfAnalisi['COSTmodel_H2'].var()))/2)
## Calculate the t-statistics
tH1H2 = ((dfAnalisi['COSTmodel_H1'].mean())-(dfAnalisi['COSTmodel_H2'].mean()))/((sH1H2*np.sqrt(2/N)))
```

Il valore trovato è il seguente:

```
01 tH1H2 = {float64} -1.1463602880521513
```

Il valore t calcolato dal test può essere interpretato confrontandolo con i valori critici della distribuzione. Il valore critico può essere calcolato

utilizzando i gradi di libertà e un livello di significatività con la funzione del punto percentuale (PPF).

```
# Compare with the critical t-value :
#Degrees of freedom
dfH2H3 = 2*N - 2
#p-value after comparison with the
pH2H3 = 1 - stats.t.cdf(tH2H3,df=dfH2H3)
```

Se il valore assoluto di t <= p-value: ipotesi nulla.

Se il valore assoluto di t > p-value: ipotesi alternativa.

```
# Compare with the critical t-value :
#Degrees of freedom
dfH1H2 = 2*N - 2
#p-value after comparison with the t
pH1H2 = 1 - stats.t.cdf(tH1H2,df=dfH1H2)
```

Il valore trovato è il seguente:

```
pH1H2 = {float64} 0.8741052248028757
```

|tH1H2|> pH1H2.

I costi tra i due centri differiscono tra di loro, è stata verificata l'ipotesi alternativa.

E' stato effettuato il test anche nei restanti centri e il risultato è il seguente:

```
tH3H1 = {float64} 1.13378341900295
```

```
pH3H1 = {float64} 0.12851377143647014
```

|tH3H1|> pH3H1. Tra H1 e H3 l'ipotesi è alternativa.

```
tH1H4 = {float64} -1.1135948386687373
```

```
pH1H4 = {float64} 0.8672034929623589
```

|tH1H4| > pH1H4. Tra H1 e H4 l'ipotesi è alternativa.

```
tH2H3 = {float64} 0.016643612314440906
```

```
pH2H3 = {float64} 0.49336133413002126
```

|tH2H3|< pH2H3. Tra H2 e H3 l'ipotesi è nulla.

```
tH2H4 = {float64} 0.03626000948731955
```

```
pH2H4 = {float64} 0.4855394109282575
```

|tH2H4|< pH2H4. Tra H2 e H4 l'ipotesi è nulla.

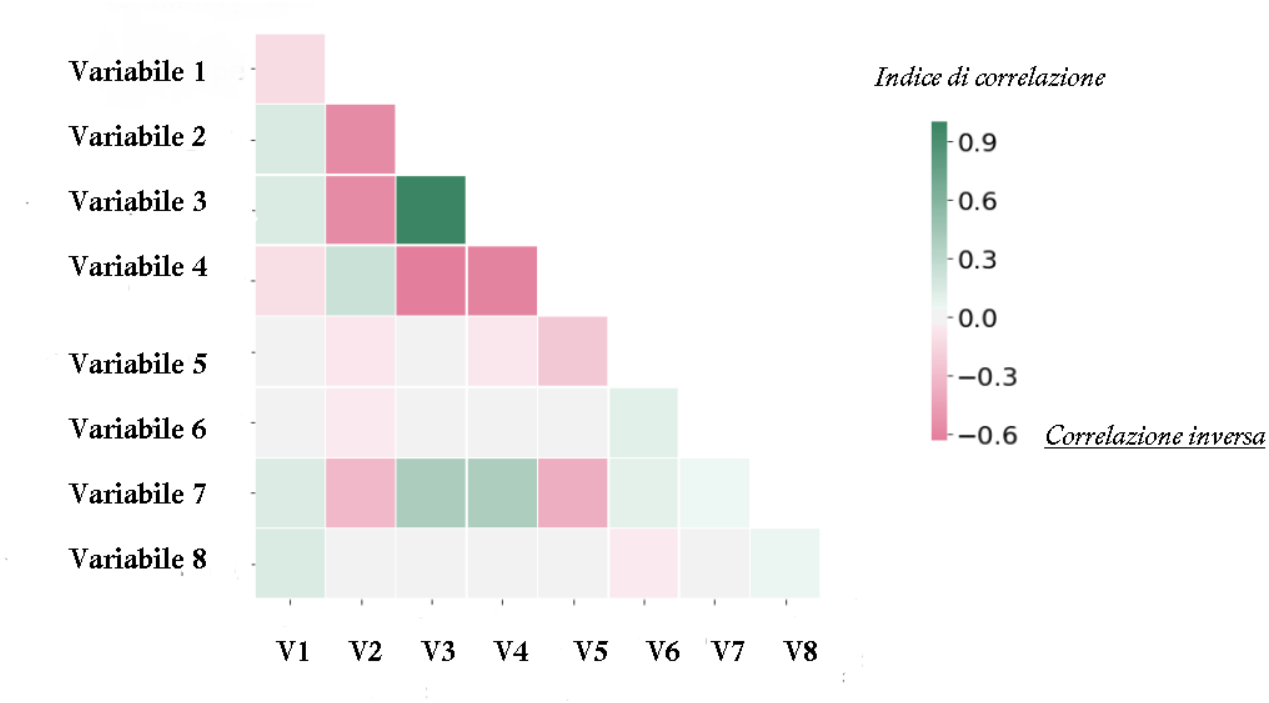
```
tH3H4 = {float64} 0.01969265184792866
```

```
pH3H4 = {float64} 0.4921453036690323
```

|tH3H4|< pH3H4. Tra H3 e H4 l'ipotesi è nulla.

5.1 Conclusioni e sguardo al futuro

Nella tesi sono state illustrate e descritte le varie fasi che hanno portato ad un'analisi statistica discreta a supporto dell'azienda Luxottica. La struttura dati è stata trasferita su di una base di dati e modellata a piacimento attraverso il linguaggio MySQL. Successivamente è stata eseguita l'analisi sfruttando le potenzialità del linguaggio orientato agli oggetti Python. La combinazione dei due linguaggi ha portato nell'insieme a una gestione piuttosto efficiente del lavoro. Per quanto riguarda il futuro, una delle caratteristiche che andrebbero, probabilmente, implementate è una mappa di calore che identifichi nuove correlazioni con le variabili.



E indicando gli spazi vuoti le variabili da inserire, in correlato una bozza del codice:

```
28
29 modeling_data=tempo.copy()
30 modeling_data=modeling_data.drop(['          ','          '], axis=1)
31 le = LabelEncoder()
32 modeling_data['       ']=le.fit(modeling_data['       ']).transform
33 (modeling_data[       '])
34 le2 = LabelEncoder()
35 modeling_data['          ']=le2.fit(modeling_data['          ']).transform
36 (modeling_data['          '])
37
```

Il Label Encoder convertirà opportuni dati di testo categoriali in dati numerici comprensibili al modello.

Un'ulteriore implementazione potrebbe essere un miglioramento del modello di regressione lineare trasformandolo in un modello di multi regressione nel quale si avrà in rappresentazione di più variabili dipendenti (variabili y) da poter studiare e analizzare con opportune variabili indipendenti.

In correlazione una bozza del codice ancora in fase di implementazione.

```
#REGRESSIONE LINEARE MULTIPLA
df4modelM1 = df4[df4['Models'] == 'M1']
df4modelM333 = df4[df4['Models'] == 'M333']
df4modelM539 = df4[df4['Models'] == 'M539']
df4modelM1.drop(df4modelM1.columns[[0, 1, 3, 4, 5, 6, 7, 8, 9]], axis = 1, inplace = True)
df4modelM333.drop(df4modelM333.columns[[0, 1, 3, 4, 5, 6, 7, 8, 9]], axis = 1, inplace = True)
df4modelM539.drop(df4modelM539.columns[[0, 1, 3, 4, 5, 6, 7, 8, 9]], axis = 1, inplace = True)

dfregM1 = dfregM1_transposed.append(df4modelM1)
dfregM1 = dfregM1.max().to_frame().T
dfregM333 = dfregM333_transposed.append(df4modelM333)
dfregM333 = dfregM333.max().to_frame().T
dfregM539 = dfregM539_transposed.append(df4modelM539)
dfregM539 = dfregM539.max().to_frame().T
dfREG = dfregM1.append([dfregM333, dfregM539])
dfREG = dfREG.replace(np.nan, 0, regex=True)

X = dfREG.iloc[:, :-1].values
y = dfREG.iloc[:, 66].values
X[:, 65] = labelencoder.fit_transform(X[:, 65])
onehotencoder = OneHotEncoder(categorical_features = [65])
X = onehotencoder.fit_transform(X).toarray()
X=X[:, 1:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
```

Bibliografia

[1] ELEMENTI DI STATISTICA DESCRITTIVA ED INFERENZIALE
file:///C:/Users/Utente%20PC/Desktop/dispensestatistica.pdf

[2] Linguaggio e concetti della statistica

di Giorgio Celant Di Battista Marco edito da CLEUP, 2018 1999.

[3] Il coefficiente di correlazione e la retta di regressione http://www.quadernodiepidemiologia.it/epi-mobile/cause/coef_cor.htm

[4] Python Community: https://stackoverflow.com/ , https://www.geeksforgeeks.org/python-pandas-

[5] Regressione lineare con Python https://www.lorenzogovoni.com/regressione-lineare-python/

[6] How to Code the Student's t-Test from Scratch in Python https://machinelearningmastery.com/how-to-code-the-students-t-test-from-scratch-in-python/

[7] Using Pandas iloc, loc, & ix to select rows and columns in DataFrames
https://www.shanelynn.ie/pandas-iloc-loc-select-rows-and-columns-dataframe/

[8] MySQL 8.0 Reference Manual :: 13.2.10 SELECT Statement
https://dev.mysql.com/doc/refman/8.0/en/select.html

[9] pandas.DataFrame — pandas 1.3.1 documentation
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

[10] Pandas Tutorial: DataFrames in Python – DataCamp
https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python

[11] NumPy quickstart — NumPy v1.21 Manual https://numpy.org/doc/stable/user/quickstart.html

[12] Python Plotting With Matplotlib (Guide) – Real Python https://realpython.com/python-matplotlib-guide/

[13] Get started | PyCharm – JetBrains https://www.jetbrains.com/help/pycharm/quick-start-guide.html

[14] Running Queries Using SQL Server Management Studio
https://community.snowsoftware.com/s/article/runningqueriesusingsqlservermanagementstudio

[15] seaborn.heatmap — seaborn 0.11.1 documentation
https://seaborn.pydata.org/generated/seaborn.heatmap.html

[16] Codice in Python con cui è stato eseguito il lavoro:

```
import pyodbc
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
from collections import OrderedDict
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
```

```
from scipy import stats
import statsmodels.formula.api as sm
import random
import pandas.util.testing as tm

cs = 'DRIVER={SQL Server};SERVER=LAPTOP-
16GE187H\SQLEXPRESS;DATABASE=TirocinioLuxottica;Trusted_Connection=yes'

connStr = pyodbc.connect(cs)
cursor = connStr.cursor()



#1^ Correlazione: Components -Models [BOMxComponents]: Calcolare qual'è il Modello con più
Componenti e qual'è il Componente più utilizzato .


stringaQuerySelect='SELECT Components, Models FROM BOMxComponents'
pd.options.display.float_format = "{:,.2f}".format
dfBOM=pd.read_sql(stringaQuerySelect,connStr)
dfBOM.groupby('Models').count()
dfBOM['CompxModel'] = dfBOM.groupby('Models')['Models'].transform('count')
dfBOM.groupby('Components').count()
dfBOM['Freq'] = dfBOM.groupby('Components')['Components'].transform('count')
dfBOM.sort_values(by=['CompxModel','Freq'], ascending=False, inplace=True)
dfBOM2 = dfBOM[['Components','Models','CompxModel']]
#dfBOM2['sort'] = dfBOM2['Components'].str.extract('(\d+)', expand=False).astype(int)
#dfBOM2.sort_values('sort',inplace=True, ascending=True)
dfBOM2.set_index('Components', inplace=True)

#Verifica Query:

stringaQuerySelect='SELECT Models, count(*) as Quantità FROM BOMxComponents GROUP BY Models
order by Quantità DESC'



stringaQuerySelect='SELECT Components, count(*) as Quantità FROM BOMxComponents GROUP BY
Components order by Quantità DESC'

dataframeComponenti=pd.read_sql(stringaQuerySelect,connStr)
#2^ Correlazione: Hub - [StockedCostComponents]: Analizzare i Costi Di Stoccaggio e vedere qual'è il
Magazzino più economico.
stringaQuerySelect = 'SELECT H1Stock,H2Stock,H3Stock,H4Stock FROM StockCostComp'
dfStock = pd.read_sql(stringaQuerySelect,connStr)

contatoreH1=0
contatoreH2=0
contatoreH3=0
```

```
contatoreH4=0

for id, row in dfStock.iterrows():
   Min=dfStock.iloc[id].min()
   if (Min==dfStock.H1Stock.iloc[id]):
      contatoreH1=contatoreH1+1
   if (Min==dfStock.H2Stock.iloc[id]):
      contatoreH2=contatoreH2+1
   if (Min==dfStock.H3Stock.iloc[id]):
      contatoreH3=contatoreH3+1
   if (Min==dfStock.H4Stock.iloc[id]):
      contatoreH4=contatoreH4+1

dfMinHub = pd.DataFrame(index=range(1,1),
columns=['Contatore_H1','Contatore_H2','Contatore_H3','Contatore_H4'], dtype='int')
dfMinHub.at[id, 'Contatore_H1'] = contatoreH1
dfMinHub.at[id, 'Contatore_H2'] = contatoreH2
dfMinHub.at[id, 'Contatore_H3'] = contatoreH3
dfMinHub.at[id, 'Contatore_H4'] = contatoreH4

#3^ Correlazione: [BOMxComponents - ProductionCostComponents] : Calcolo per ogni Modello la Media
aritmetica dei costi dei Componenti che lo compongono.

stringaQuerySelect='select distinct Models from BOMxComponents order by Models'
dfAnalisi=pd.read_sql(stringaQuerySelect,connStr)#sono tutti i modelli
dfAnalisi.sort_values(by=['Models'], ascending=False, inplace=True)

dfAnalisi['MEDIA_H1']=0
dfAnalisi['MEDIA_H1']=dfAnalisi['MEDIA_H1'].astype(float)
dfAnalisi['MEDIA_H2']=0
dfAnalisi['MEDIA_H2']=dfAnalisi['MEDIA_H2'].astype(float)
dfAnalisi['MEDIA_H3']=0
dfAnalisi['MEDIA_H3']=dfAnalisi['MEDIA_H3'].astype(float)
dfAnalisi['MEDIA_H4']=0
dfAnalisi['MEDIA_H4']=dfAnalisi['MEDIA_H4'].astype(float)
dfAnalisi.reset_index(drop=True)
stringaQuerySelect='SELECT * FROM BOMxComponents'
dataframeBOMxComponents=pd.read_sql(stringaQuerySelect,connStr) #sono tutti i componenti

stringaQuerySelect='SELECT * FROM ProdCostComp'
dataframeProdCostComp=pd.read_sql(stringaQuerySelect,connStr) #sono tutti i costi
dataframeProdCostComp['H1Prod'] = dataframeProdCostComp['H1Prod'].astype(float)
dataframeProdCostComp['H2Prod'] = dataframeProdCostComp['H2Prod'].astype(float)
dataframeProdCostComp['H3Prod'] = dataframeProdCostComp['H3Prod'].astype(float)
dataframeProdCostComp['H4Prod'] = dataframeProdCostComp['H4Prod'].astype(float)
dfAnalisi['NumeroComponenti']=0
dfAnalisi['MEDIA_H1']=dfAnalisi['MEDIA_H1'].astype(float)
```

```
for id, row in dfAnalisi.iterrows():

components=dataframeBOMxComponents[dataframeBOMxComponents['Models']==row['Models']]['Comp
onents']
   sommatoriaH1=0
   sommatoriaH2=0
   sommatoriaH3=0
   sommatoriaH4=0
   dfAnalisi.at[id, 'NumeroComponenti'] = components.count()
   for component in components:
      costoComponente=dataframeProdCostComp[dataframeProdCostComp['Components']==component]
      sommatoriaH1 += costoComponente['H1Prod'].iloc[0]
      sommatoriaH2 += costoComponente['H2Prod'].iloc[0]
      sommatoriaH3 += costoComponente['H3Prod'].iloc[0]
      sommatoriaH4 += costoComponente['H4Prod'].iloc[0]

      dfAnalisi.at[id, 'COSTmodel_H1'] = sommatoriaH1
      dfAnalisi.at[id, 'COSTmodel_H2'] = sommatoriaH2
      dfAnalisi.at[id, 'COSTmodel_H3'] = sommatoriaH3
      dfAnalisi.at[id, 'COSTmodel_H4'] = sommatoriaH4

   mediaH1=  sommatoriaH1 / len(components)
   mediaH2 = sommatoriaH2 / len(components)
   mediaH3 = sommatoriaH3 / len(components)
   mediaH4 = sommatoriaH4 / len(components)

   dfAnalisi.at[id, 'MEDIA_H1'] = mediaH1
   dfAnalisi.at[id, 'MEDIA_H2'] = mediaH2
   dfAnalisi.at[id, 'MEDIA_H3'] = mediaH3
   dfAnalisi.at[id, 'MEDIA_H4'] = mediaH4

dfAnalisi.sort_values(by=['MEDIA_H1','COSTmodel_H1'], ascending=False, inplace=True)

#Spesa minima di produzione per ogni Magazzino

stringaQuerySelect = 'SELECT * FROM MinLotProfit'
dfLot=pd.read_sql(stringaQuerySelect,connStr)

df3 = pd.concat([dfLot, dfAnalisi], axis=1 )
df3['CostProd_H1']=0
df3['CostProd_H1']=df3['CostProd_H1'].astype(float)
df3['CostProd_H2']=0
df3['CostProd_H2']=df3['CostProd_H2'].astype(float)
df3['CostProd_H3']=0
df3['CostProd_H3']=df3['CostProd_H3'].astype(float)
df3['CostProd_H4']=0
df3['CostProd_H4']=df3['CostProd_H4'].astype(float)
df4=
(df3[['Models','Minlot','Profit','COSTmodel_H1','COSTmodel_H2','COSTmodel_H3','COSTmodel_H4','CostPr
```

```
od_H1','CostProd_H2','CostProd_H3','CostProd_H4']])

for id, row in df4.iterrows():
   costomodelloH1=df4[df4['Models']==row['Models']]['COSTmodel_H1']
   costomodelloH2=df4[df4['Models']==row['Models']]['COSTmodel_H2']
   costomodelloH3=df4[df4['Models']==row['Models']]['COSTmodel_H3']
   costomodelloH4=df4[df4['Models']==row['Models']]['COSTmodel_H4']
   lottominimo=df4[df4['Models']==row['Models']]['Profit']
   df4.at[id, 'CostProd_H1'] = ((costomodelloH1)*(lottominimo))
   df4.at[id, 'CostProd_H2'] = ((costomodelloH2)*(lottominimo))
   df4.at[id, 'CostProd_H3'] = ((costomodelloH3)*(lottominimo))
   df4.at[id, 'CostProd_H4'] = ((costomodelloH4)*(lottominimo))

sommatoriaH1min=0
sommatoriaH2min=0
sommatoriaH3min=0
sommatoriaH4min=0

dfProdMin =
pd.DataFrame(index=range(0,957),columns=['Spesa_MagazzinoH1','Spesa_MagazzinoH2','Spesa_Magazzin
oH3','Spesa_MagazzinoH4'], dtype='float')

for id, row in df4.iterrows():

   costoH1=df4['CostProd_H1']
   costoH2=df4['CostProd_H2']
   costoH3=df4['CostProd_H3']
   costoH4=df4['CostProd_H4']
   sommatoriaH1min += costoH1.iloc[0]
   sommatoriaH2min += costoH2.iloc[0]
   sommatoriaH3min += costoH3.iloc[0]
   sommatoriaH4min += costoH4.iloc[0]

dfProdMin.at[id, 'Spesa_MagazzinoH1'] = sommatoriaH1min
dfProdMin.at[id, 'Spesa_MagazzinoH2'] = sommatoriaH2min
dfProdMin.at[id, 'Spesa_MagazzinoH3'] = sommatoriaH3min
dfProdMin.at[id, 'Spesa_MagazzinoH4'] = sommatoriaH4min
dfProdMin = [dfProdMin.iloc[-1]]
stringaQuerySelect='SELECT H1Prod,H1Stock from ProdCostComp INNER JOIN StockCostComp ON
ProdCostComp.id_prodcostcomp=StockCostComp.id_compstock'

#Mappa di calore

dfregModel = pd.DataFrame(index=range(0,957), dtype='float')
for id, row in dfAnalisi.iterrows():
   models=dfAnalisi[dfAnalisi['Models']==row['Models']]['Models']
   for model in models:
      costoModello=dfAnalisi[dfAnalisi['Models']==model]
      dfregModel.at[id, 'COSTmodel_H1'] = costoModello['COSTmodel_H1'].iloc[0]
```

```
        dfregModel.at[id, 'COSTmodel_H2'] = costoModello['COSTmodel_H2'].iloc[0]
        dfregModel.at[id, 'COSTmodel_H3'] = costoModello['COSTmodel_H3'].iloc[0]
        dfregModel.at[id, 'COSTmodel_H4'] = costoModello['COSTmodel_H4'].iloc[0]

dfcostmodelH1 = dfregModel[['COSTmodel_H1']]
dfModels = dfBOM['Models']
dfprofitReg = dfLot[['Profit']]
df4['sellprice_H1']=0
df4['sellprice_H1']=df4['sellprice_H1'].astype(float)
df4['sellprice_H2']=0
df4['sellprice_H2']=df4['sellprice_H2'].astype(float)
df4['sellprice_H3']=0
df4['sellprice_H3']=df4['sellprice_H3'].astype(float)
df4['sellprice_H4']=0
df4['sellprice_H4']=df4['sellprice_H4'].astype(float)

for id, row in df4.iterrows():
    costomodelloH1=df4[df4['Models']==row['Models']]['COSTmodel_H1']
    costomodelloH2=df4[df4['Models']==row['Models']]['COSTmodel_H2']
    costomodelloH3=df4[df4['Models']==row['Models']]['COSTmodel_H3']
    costomodelloH4=df4[df4['Models']==row['Models']]['COSTmodel_H4']
    profit=df4[df4['Models']==row['Models']]['Profit']
    df4.at[id, 'sellprice_H1'] = ((costomodelloH1)+(profit))
    df4.at[id, 'sellprice_H2'] = ((costomodelloH2)+(profit))
    df4.at[id, 'sellprice_H3'] = ((costomodelloH3)+(profit))
    df4.at[id, 'sellprice_H4'] = ((costomodelloH4)+(profit))
    df4['sellprice_H1']=df4['sellprice_H1'].round(decimals=2)
    df4['sellprice_H2']=df4['sellprice_H2'].round(decimals=2)
    df4['sellprice_H3']=df4['sellprice_H3'].round(decimals=2)
    df4['sellprice_H4']=df4['sellprice_H4'].round(decimals=2)

dfregModels = pd.concat([dfprofitReg, dfregModel, dfModels, df4], axis=1)
dfregModels=
(df4[['Models','Profit','COSTmodel_H1','COSTmodel_H2','COSTmodel_H3','COSTmodel_H4','sellprice_H1','s
ellprice_H2','sellprice_H3','sellprice_H4']])

new_col = dfregModels['COSTmodel_H1']
piv =pd.pivot_table(dfregModels, index=["Models"], columns=["sellprice_H1"], values="Profit",
fill_value=0)
dfregModels.set_index('Models', inplace=True)
dfregModels['sort'] = dfregModels.index.str.extract('(\d+)', expand=False).astype(int)
dfregModels.sort_values('sort',inplace=True, ascending=True)
dfregModels = dfregModels.drop('sort', axis=1)
piv['sort'] = piv.index.str.extract('(\d+)', expand=False).astype(int)
piv.sort_values('sort',inplace=True, ascending=True)
piv = piv.drop('sort', axis=1)
piv.reset_index(level=0, inplace=True)
piv.insert(loc=0, column='COSTmodel_H1', value=new_col)
piv.set_index('Models','COSTmodel_H1', inplace=True)
```

```
piv['COSTmodel_H1']=piv['COSTmodel_H1'].round(decimals=2)
piv.set_index('COSTmodel_H1', append=True, inplace=True)
ax = sns.heatmap(piv, square=True, cbar=False)
plt.show()


new_col2 = dfregModels['COSTmodel_H2']
piv2 =pd.pivot_table(dfregModels, index=["Models"], columns=["sellprice_H2"], values="Profit",
fill_value=0)
dfregModels.set_index('Models', inplace=True)
dfregModels['sort'] = dfregModels.index.str.extract('(\d+)', expand=False).astype(int)
dfregModels.sort_values('sort',inplace=True, ascending=True)
dfregModels = dfregModels.drop('sort', axis=1)
piv2['sort'] = piv2.index.str.extract('(\d+)', expand=False).astype(int)
piv2.sort_values('sort',inplace=True, ascending=True)
piv2 = piv2.drop('sort', axis=1)
piv2.reset_index(level=0, inplace=True)
piv2.insert(loc=0, column='COSTmodel_H2', value=new_col2)
piv2.set_index('Models','COSTmodel_H2', inplace=True)
piv2['COSTmodel_H2']=piv2['COSTmodel_H2'].round(decimals=2)
piv2.set_index('COSTmodel_H2', append=True, inplace=True)
ax = sns.heatmap(piv2, square=True, cbar=False)

new_col3 = dfregModels['COSTmodel_H3']
piv3 =pd.pivot_table(dfregModels, index=["Models"], columns=["sellprice_H3"], values="Profit",
fill_value=0)
dfregModels.set_index('Models', inplace=True)
dfregModels['sort'] = dfregModels.index.str.extract('(\d+)', expand=False).astype(int)
dfregModels.sort_values('sort',inplace=True, ascending=True)
dfregModels = dfregModels.drop('sort', axis=1)
piv3['sort'] = piv3.index.str.extract('(\d+)', expand=False).astype(int)
piv3.sort_values('sort',inplace=True, ascending=True)
piv3 = piv3.drop('sort', axis=1)
piv3.reset_index(level=0, inplace=True)
piv3.insert(loc=0, column='COSTmodel_H3', value=new_col3)
piv3.set_index('Models','COSTmodel_H3', inplace=True)
piv3['COSTmodel_H3']=piv3['COSTmodel_H3'].round(decimals=2)
piv3.set_index('COSTmodel_H3', append=True, inplace=True)
ax = sns.heatmap(piv3, square=True, cbar=False)

new_col4 = dfregModels['COSTmodel_H4']
piv4 =pd.pivot_table(dfregModels, index=["Models"], columns=["sellprice_H4"], values="Profit",
fill_value=0)
dfregModels.set_index('Models', inplace=True)
dfregModels['sort'] = dfregModels.index.str.extract('(\d+)', expand=False).astype(int)
dfregModels.sort_values('sort',inplace=True, ascending=True)
dfregModels = dfregModels.drop('sort', axis=1)
piv4['sort'] = piv4.index.str.extract('(\d+)', expand=False).astype(int)
piv4.sort_values('sort',inplace=True, ascending=True)
```

```
piv4 = piv4.drop('sort', axis=1)
piv4.reset_index(level=0, inplace=True)
piv4.insert(loc=0, column='COSTmodel_H4', value=new_col4)
piv4.set_index('Models','COSTmodel_H4', inplace=True)
piv4['COSTmodel_H4']=piv4['COSTmodel_H4'].round(decimals=2)
piv4.set_index('COSTmodel_H4', append=True, inplace=True)
ax = sns.heatmap(piv4, square=True, cbar=False)


#Regressione lineare
X_train, X_test, y_train, y_test=train_test_split(dfregModels['sellprice_H1'],
                                    dfregModels['COSTmodel_H1'], test_size=0.33, random_state=42)
reg=linear_model.LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
reg.fit(X_train.values.reshape(-1, 1), y_train.values.reshape(-1, 1))
reg.coef_
coeffadd=str(reg.score(X_train.values.reshape(-1, 1), y_train.values.reshape(-1, 1)))
coefftest=str(reg.score(X_test.values.reshape(-1, 1), y_test.values.reshape(-1, 1) ))

font={'size': 15}
plt.rc('font', **font)
plt.plot(dfregModels['sellprice_H1'], dfregModels['COSTmodel_H1'], 'o', label='Data')
I=np.linspace(np.floor(min(dfregModels['sellprice_H1'])), np.ceil(max(dfregModels['COSTmodel_H1'])),50)
plt.plot(I, reg.predict(I.reshape(-1, 1)), color='r', linewidth=3, label='Linea di regressione')
plt.legend(bbox_to_anchor=(1, 1), borderaxespad=0.)
plt.xlabel('Prezzo Modelli in H1')
plt.ylabel('Costo Modelli in H1')
Preds=reg.predict(dfregModels['sellprice_H1'].values.reshape(-1, 1))
R2=r2_score(dfregModels['COSTmodel_H1'],Preds)
plt.title('Coefficiente di correlazione:'+ str(np.round(R2,decimals=3)))

#Valori residui
Residuals=dfregModels['COSTmodel_H1'].values.reshape(-1, 1)-Preds
font={'size': 20}
plt.rc('font', **font)
fig, ax=plt.subplots(1,2,figsize=(20,10))
num_bins=50
n, bins, patches=ax[0].hist(Residuals, num_bins, density=1)
ax[0].title.set_text('Residui')
ax[1].plot(dfregModels['sellprice_H1'],Residuals, 'o')
ax[1].set(xlabel='Valore effettivo', ylabel='Valore dei residui')
ax[1].hlines(0, np.min(dfregModels['sellprice_H1'])*0.95,
        np.max(dfregModels['sellprice_H1'])*1.1, colors='r', linestyles='solid', zorder=10, linewidth=4)
plt.show()


tH1H2=0
N=957

#std deviation
```

```
sH1H2 = np.sqrt(((dfAnalisi['COSTmodel_H1'].var())+(dfAnalisi['COSTmodel_H2'].var()))/2)
## Calculate the t-statistics
tH1H2 = ((dfAnalisi['COSTmodel_H1'].mean())-(dfAnalisi['COSTmodel_H2'].mean()))/((sH1H2*np.sqrt(2/N)))

# Compare with the critical t-value :
#Degrees of freedom
dfH1H2 = 2*N - 2
#p-value after comparison with the t
pH1H2 = 1 - stats.t.cdf(tH1H2,df=dfH1H2)

#T-TEST H2-H3
tH2H3=0

#std deviation
sH2H3 = np.sqrt(((dfAnalisi['COSTmodel_H2'].var())+(dfAnalisi['COSTmodel_H3'].var()))/2)
## Calculate the t-statistics
tH2H3 = ((dfAnalisi['COSTmodel_H2'].mean())-(dfAnalisi['COSTmodel_H3'].mean()))/((sH2H3*np.sqrt(2/N)))

# Compare with the critical t-value :
#Degrees of freedom
dfH2H3 = 2*N - 2
#p-value after comparison with the
pH2H3 = 1 - stats.t.cdf(tH2H3,df=dfH2H3)

#T-TEST H3-H1
tH3H1=0

#std deviation
sH3H1 = np.sqrt(((dfAnalisi['COSTmodel_H3'].var())+(dfAnalisi['COSTmodel_H1'].var()))/2)
## Calculate the t-statistics
tH3H1 = ((dfAnalisi['COSTmodel_H3'].mean())-(dfAnalisi['COSTmodel_H1'].mean()))/((sH3H1*np.sqrt(2/N)))

# Compare with the critical t-value :
#Degrees of freedom
dfH3H1 = 2*N - 2
#p-value after comparison with the t
pH3H1 = 1 - stats.t.cdf(tH3H1,df=dfH3H1)

#T-TEST H1-H4
tH1H4=0

#std deviation
sH1H4 = np.sqrt(((dfAnalisi['COSTmodel_H1'].var())+(dfAnalisi['COSTmodel_H4'].var()))/2)
## Calculate the t-statistics
tH1H4 = ((dfAnalisi['COSTmodel_H1'].mean())-(dfAnalisi['COSTmodel_H4'].mean()))/((sH1H4*np.sqrt(2/N)))

# Compare with the critical t-value :
#Degrees of freedom
dfH1H4 = 2*N - 2
```

```
#p-value after comparison with the t
pH1H4 = 1 - stats.t.cdf(tH1H4,df=dfH1H4)

#T-TEST H2-H4
tH2H4=0

#std deviation
sH2H4 = np.sqrt(((dfAnalisi['COSTmodel_H2'].var())+(dfAnalisi['COSTmodel_H4'].var()))/2)
## Calculate the t-statistics
tH2H4 = ((dfAnalisi['COSTmodel_H2'].mean())-(dfAnalisi['COSTmodel_H4'].mean()))/((sH2H4*np.sqrt(2/N)))

# Compare with the critical t-value :
#Degrees of freedom
dfH2H4 = 2*N - 2
#p-value after comparison with the t
pH2H4 = 1 - stats.t.cdf(tH2H4,df=dfH2H4)

#T-TEST H3-H4
tH3H4=0

#std deviation
sH3H4 = np.sqrt(((dfAnalisi['COSTmodel_H3'].var())+(dfAnalisi['COSTmodel_H4'].var()))/2)
## Calculate the t-statistics
tH3H4 = ((dfAnalisi['COSTmodel_H3'].mean())-(dfAnalisi['COSTmodel_H4'].mean()))/((sH3H4*np.sqrt(2/N)))

# Compare with the critical t-value :
#Degrees of freedom
dfH3H4 = 2*N - 2
#p-value after comparison with the t
pH3H4 = 1 - stats.t.cdf(tH3H4,df=dfH3H4)


#Metriche di Quantificazione dell'errore
'''X = dfregModel.iloc[:, :-1].values.reshape(-1, 1)
y = dfregModel.iloc[:, 2].values.reshape(-1, 1)
regressor = LinearRegression()
regressor.fit(X, y)
y_pred = regressor.predict(X)'''

#REGRESSIONE LINEARE MULTIPLA
'''df4modelM1 = df4[df4['Models'] == 'M1']
df4modelM333 = df4[df4['Models'] == 'M333']
df4modelM539 = df4[df4['Models'] == 'M539']
df4modelM1.drop(df4modelM1.columns[[0, 1, 3, 4, 5, 6, 7, 8, 9]], axis = 1, inplace = True)
df4modelM333.drop(df4modelM333.columns[[0, 1, 3, 4, 5, 6, 7, 8, 9]], axis = 1, inplace = True)
df4modelM539.drop(df4modelM539.columns[[0, 1, 3, 4, 5, 6, 7, 8, 9]], axis = 1, inplace = True)

dfregM1 = dfregM1_transposed.append(df4modelM1)
dfregM1 = dfregM1.max().to_frame().T
```

```
dfregM333 = dfregM333_transposed.append(df4modelM333)
dfregM333 = dfregM333.max().to_frame().T
dfregM539 = dfregM539_transposed.append(df4modelM539)
dfregM539 = dfregM539.max().to_frame().T
dfREG = dfregM1.append([dfregM333, dfregM539])
dfREG = dfREG.replace(np.nan, 0, regex=True)

X = dfREG.iloc[:, :-1].values
y = dfREG.iloc[:, 66].values
X[:, 65] = labelencoder.fit_transform(X[:, 65])
onehotencoder = OneHotEncoder(categorical_features = [65])
X = onehotencoder.fit_transform(X).toarray()
X=X[:, 1:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

date = pd.date_range('2017-02-23', periods=10*12, freq='2h')
freq = np.random.poisson(lam=2, size=(len(date)))
df = pd.DataFrame({"freq":freq}, index=date)
df["hours"] = df.index.hour
df["days"] = df.index.map(lambda x: x.strftime('%b-%d'))
piv = pd.pivot_table(df, values="freq",index=["hours"], columns=["days"], fill_value=0)

pivBOM =pd.pivot_table(dfBOM2, index=["Components"], columns=["Models"], values="CompxModel",
fill_value=0, )
pivBOM['sort'] = pivBOM.index.str.extract('(\d+)', expand=False).astype(int)
pivBOM.sort_values('sort',inplace=True, ascending=True)
pivBOM = pivBOM.drop('sort', axis=1)
f, ax=plt.subplots(figsize=(3, 3))
ax = sns.heatmap(pivBOM, square=True)'''
```