Università degli studi di Salerno

Dipartimento di Informatica

***Corso di Laurea in Informatica***

******

***FONDAMENTI DI INTELLIGENZA ARTIFICIALE***

***“Invendum”***

**Studenti:**

##### Nome Matricola

Luigi De Chiara 0512109483

Alex Ferrara 0512106300

## 

## 

**Docente:**

Fabio Palomba

**Docente:**

Fabio Palomba

*Anno Accademico: 2021/22*

INDICE

[1. Obiettivo del progetto 2](#_Toc68874152)

Invendum

# Introduzione

Invendum è una innovativa strategia di marketing realizzata sotto forma di plugin. Nasce dall’esigenza delle aziende di aumentare le vendite di prodotti meno gettonati, che sono sempre stati la causa delle principali problematiche economiche.

Invendum verrà utilizzato da piattaforme “e-commerce”.

# Introduzione-Agente:

L’obiettivo del progetto è quello di realizzare un’agente che sia in grado di suggerire all’utente prodotti, mediante la selezione di prodotti potenzialmente acquistabili sulla base delle transazioni commerciali precedenti. Tale agente cercherà di aumentare le vendite dei prodotti meno acquistati, cosi da ridimensionare le quantità dei prodotti invenduti.

1. **Specifica PEAS**

|  |  |
| --- | --- |
| PEAS | |
| Performance | La capacità dell’agente di riuscire a indicare al cliente prodotti invenduti di determinate categorie correlate. |
| Environment | L’ambiente dove opera l’agente è rappresentato dalla pagina dell’articolo del sito.  L’ambiente è:   * **Completamente osservabile**: abbiamo una panoramica completa delle informazioni relative agli utenti e ai prodotti. * **Stocastico:** poiché l’ambiente non verrà condizionato dalle azioni dell’agente. * **Episodico**: poiché la funzione dell’agente è richiamata nel momento in cui viene visualizzato una scheda articolo. * **Dinamico**: poiché un utente potrebbe variare le categorie di prodotti acquistati. * **Discreto**: in quanto il numero di schede articolo è numericamente limitato. * **Singolo**: in quanto nella pagina dell’articolo le azioni avverranno da parte di un unico agente. |
| Actuators | Gli attuatori del nostro agente sono identificati dagli articoli acquistati dagli utenti. Avremo così un box “Articoli Suggeriti” all’interno della pagina dell’articolo. |
| Sensors | I sensori dell’agente si identificano nella selezione della scheda articolo. |

1. **Analisi del problema**

Il problema proposto avremmo potuto risolverlo utilizzando un algoritmo che controllasse gli acquisti passati degli utenti per ogni singolo ordine.

Questa soluzione avrebbe prodotto risultati inefficienti:

* In quanto lo storico degli ordini, avrebbe consigliato solo prodotti non acquistati dagli utenti.
* Un classico algoritmo non sarebbe risultato dinamico alla scelta di un articolo da consigliare, in quanto banale e privo di informazioni specifiche.

Abbiamo deciso di utilizzare tecniche di **machine** learning per avere un’efficienza ed accuratezza maggiore del problema.

Analizzando tale problema abbiamo verificato che si trattasse di un problema di apprendimento non supervisionato, per questo motivo abbiamo deciso di applicare un algoritmo di **clustering** non avendo a disposizione etichette da assegnare ai dati.

1. **Raccolta, analisi ed elaborazione dei dati**

**Scelta del dataset**

In base ad un’analisi approfondita del nostro problema abbiamo deciso di prelevare un dataset disponibile online e di modellarlo in base alle nostre esigenze, poiché crearne uno da zero avrebbe causato alcune problematiche, come scarsità e inconsistenza dei dati.

**Analisi del dataset**

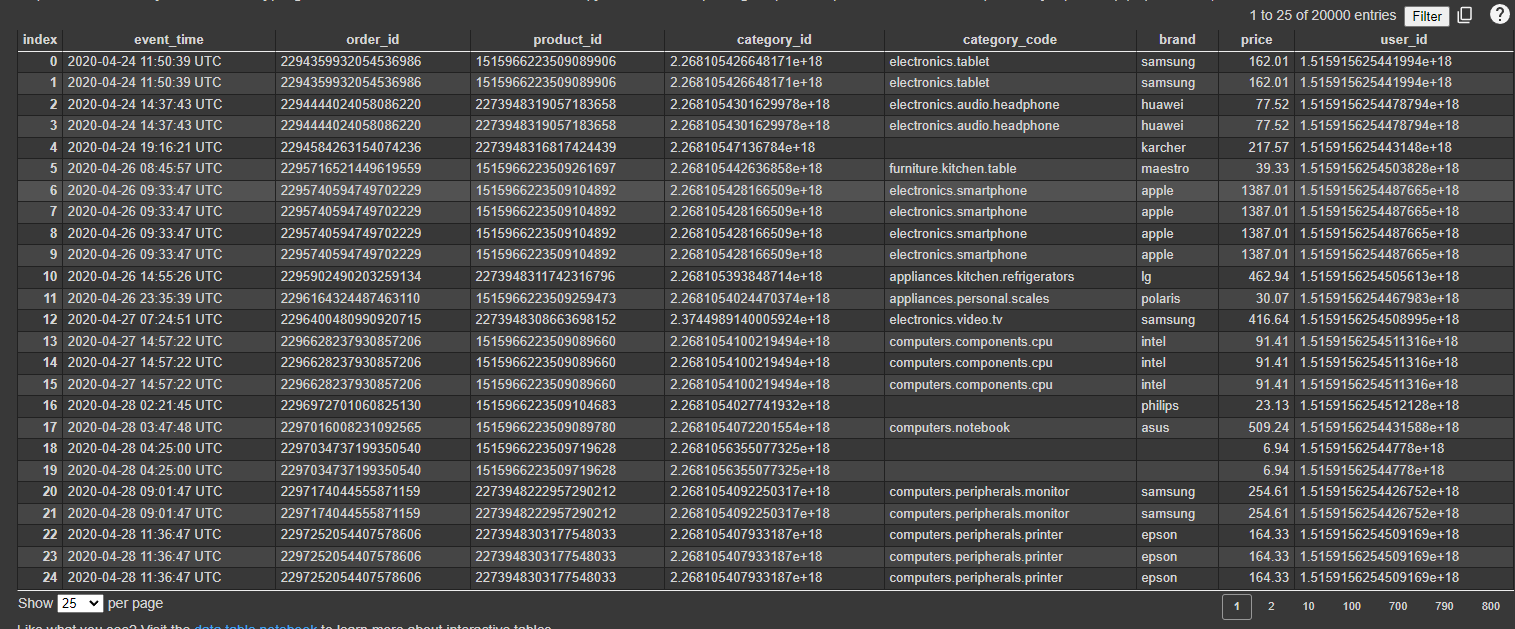
Il dataset in questione è stato accuratamente scelto e scaricato dalla piattaforma <https://www.kaggle.com/datasets>, tale dataset riporta dettagli sulle transazioni commerciali effettuate dai clienti.

Abbiamo deciso di sceglierlo pur non avendo moltissimi features, poiché per lo scopo del nostro progetto erano irrilevanti.

1. **Formattazione dei dati**

Il primo passo di formattazione dei dati è stato quello di effettuare:

* un taglio verticale per eliminare la colonna “event\_time” non utile alla fine della predizione.
* Un taglio orizzontale con “df.dropna(inplace=True)” in modo da eliminare tutte le righe con campi NULL.



Il secondo passo è stato quello di ridurre la dimensione del nostro dataset applicando l’agente solo su un sottogruppo di categorie, poiché gestire tutte le categorie con i relativi record sarebbe stato impensabile, pertanto si è optato di considerare circa trenta categorie:

"SELECT  category\_code,count(category\_code) from categoria where category\_code like 'computers.%'or category\_code like 'electronics.%' group by category\_code"

df=pd.read\_csv(r'/datasetcompleto.csv',sep=',',index\_col=['order\_id'],usecols=['order\_id','product\_id','category\_id','category\_code','brand','price','user\_id'])

q="select \* from df where category\_code like 'computers.%' or  category\_code like 'electronics.%'"

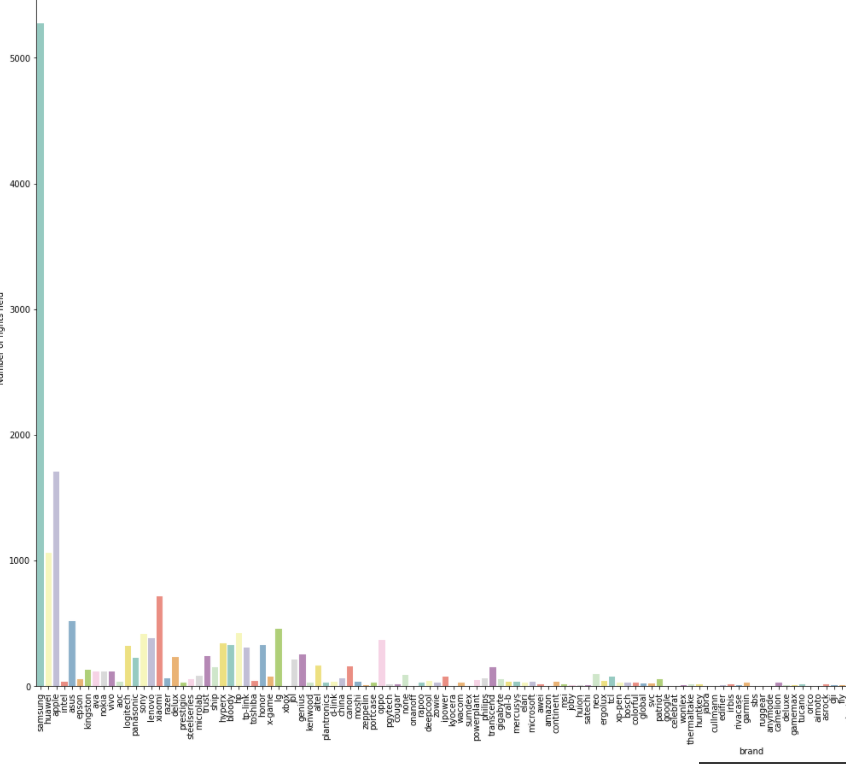
df=sqldf(q,globals())

df.to\_csv('/datasetModificato.csv')

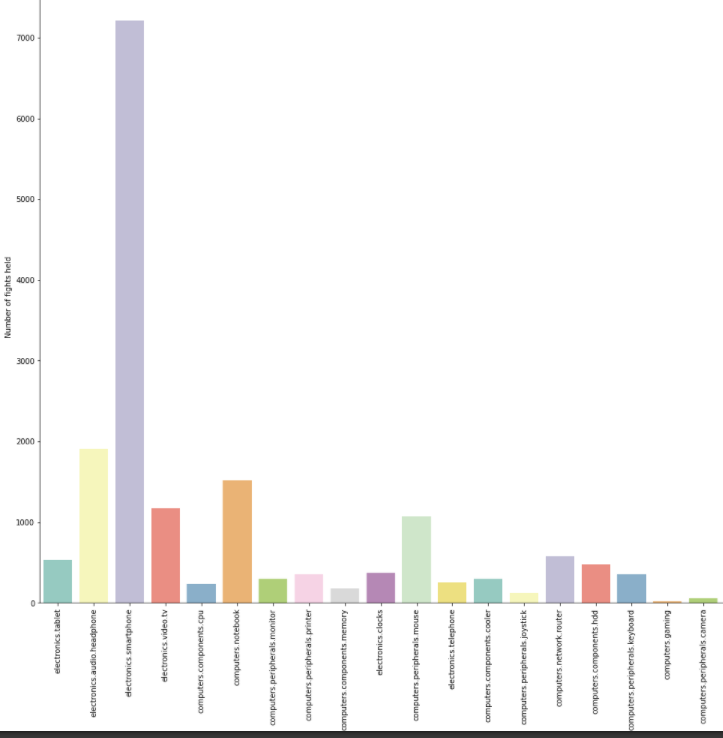
Effettuando un’analisi accurata dei dati, ci siamo resi conto che la categoria “smartphone” avente come brand “Samsung” destabilizzava l’insieme dei dati poiché aveva valori troppo alti, pertanto si è optato di effettuare un taglio orizzontale.

Di seguito vengono riportate le analisi delle categorie e dei brand.

**Analisi dei brand prima del taglio:**



**Analisi delle categorie prima del taglio:**



1. **Algoritmo di clustering**

Dopo aver formattato i dati, ci siamo posti il problema della scelta dell’algoritmo di Clustering da utilizzare. L’indecisione dovuta alle nostre conoscenze era quella di provare tra due diversi algoritmi:

* DBscan: adatto a dataset densi
* K-Means

**DBscan (**Density-Based Spatial Clustering of Applications with Noise)

Il numero di cluster non è necessario fornirlo in quanto sarà l’algoritmo stesso a scoprire quanti cluster esistono in base a due iperparametri che verranno forniti:

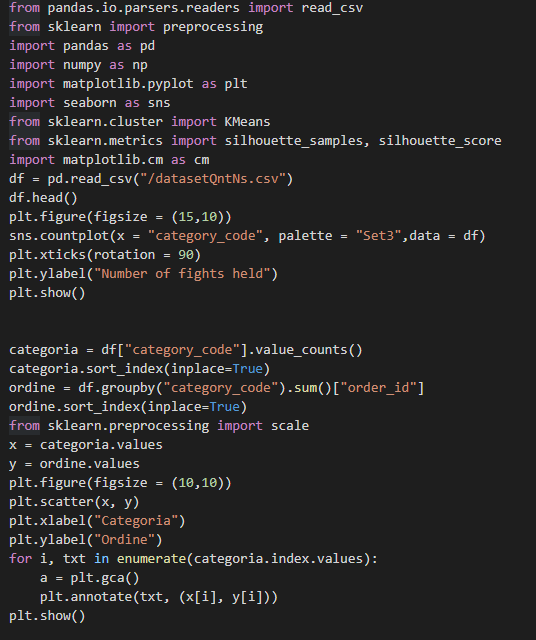
* epsilon o eps (ε): la distanza minima che devono avere due dati in ingresso per essere considerati “vicini”
* minSamples: numero minimo di dati in ingresso, “vicini” tra loro, richiesto per formare un cluster.

**K-Means**

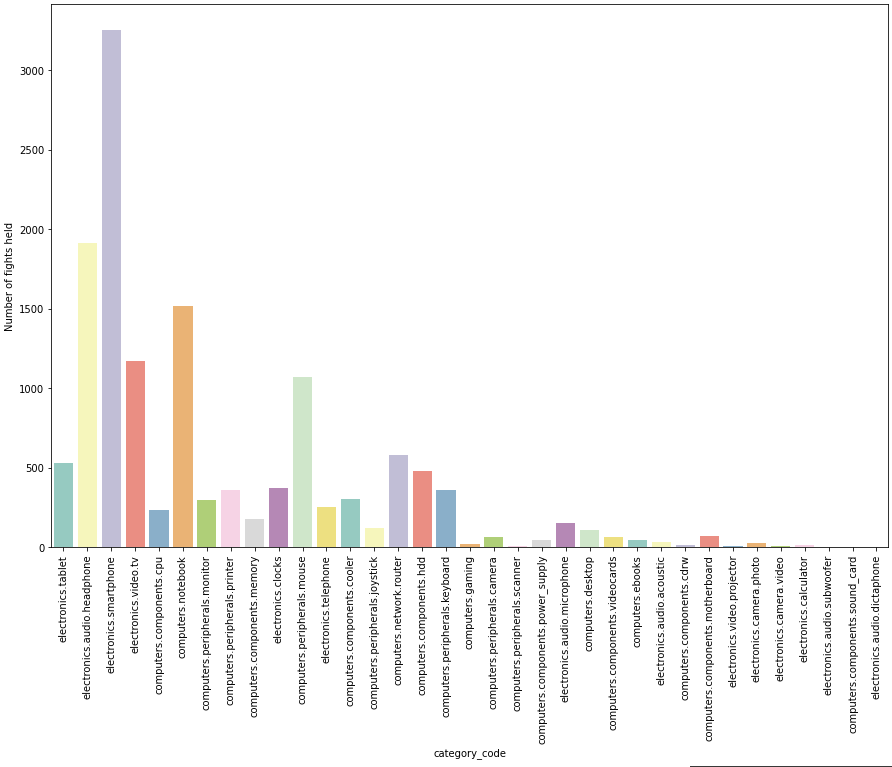
un algoritmo di analisi dei gruppi partizionale che permette di suddividere un insieme di oggetti in k gruppi sulla base della similarità intra-gruppo ed inter-gruppo.

Poiché dovevamo lavorare su dati non estremamente densi, ci siamo affidati all’algoritmo K-Means.

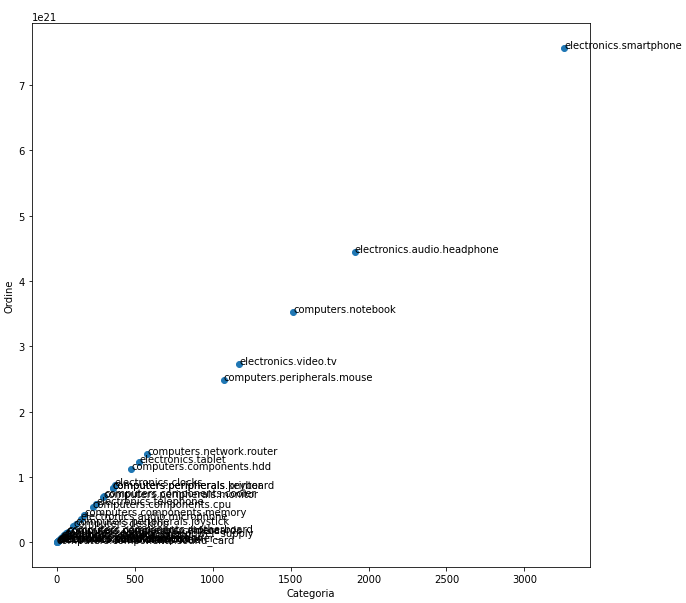
Il primo passo è stato quello di caricare il dataset appena modificato, abbiamo riutilizzato i grafi per verificare che i dati fossero perfettamente bilanciati (dati non troppo distanti fra loro). Abbiamo messo in correlazione le features della category\_code e dell’order\_id per effettuare la predizione delle quantità vendute di ogni singola categoria.



**Analisi delle categorie:**

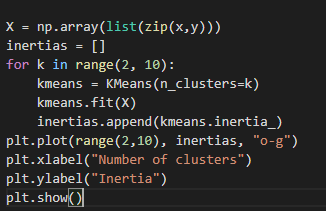


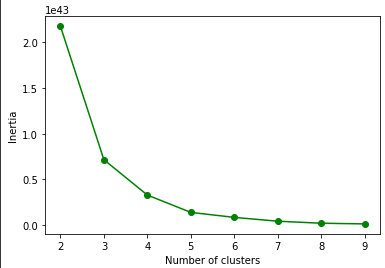
**Grafico a dispersione** basato sul numero di acquisti e sugli ordini effettuati dai clienti.



**Punto di gomito**

Utilizziamo il punto di gomito per valutare il miglior numero di cluster per l’algoritmo K-Means, pertanto andremo a creare un grafo, che ci permetterà di visionare i valori candidati del parametro k rispetto alla somma degli errori quadratici ottenuti dall’algoritmo che genererà i k cluster.





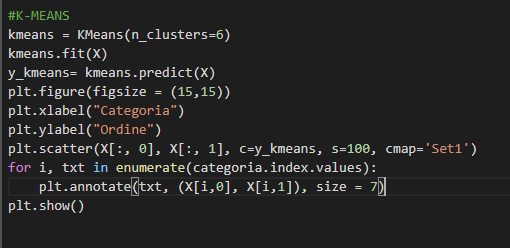
Dal punto di gomito si evince che il numero ideale di cluster da utilizzare è pari a 6.

Dopo aver assegnato tale valore a k(n\_clusters) possiamo implementare l’algoritmo precedentemente scelto.

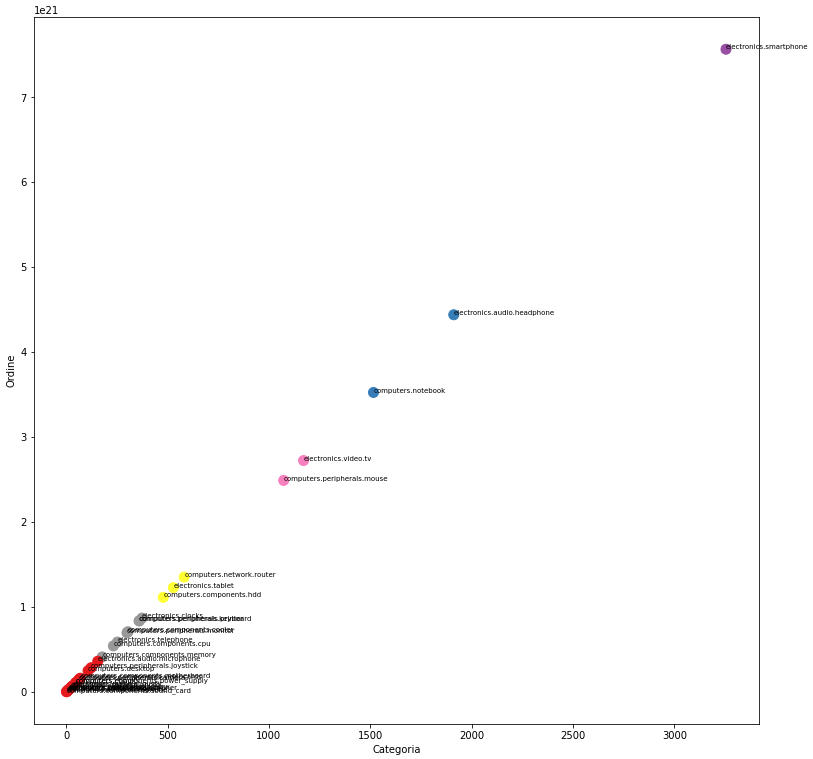
Mediante le funzioni:

* ***.fit()***
* ***.predict()***

Siamo riusciti ad addestrare il modello e ad effettuare previsioni sul set di dati.



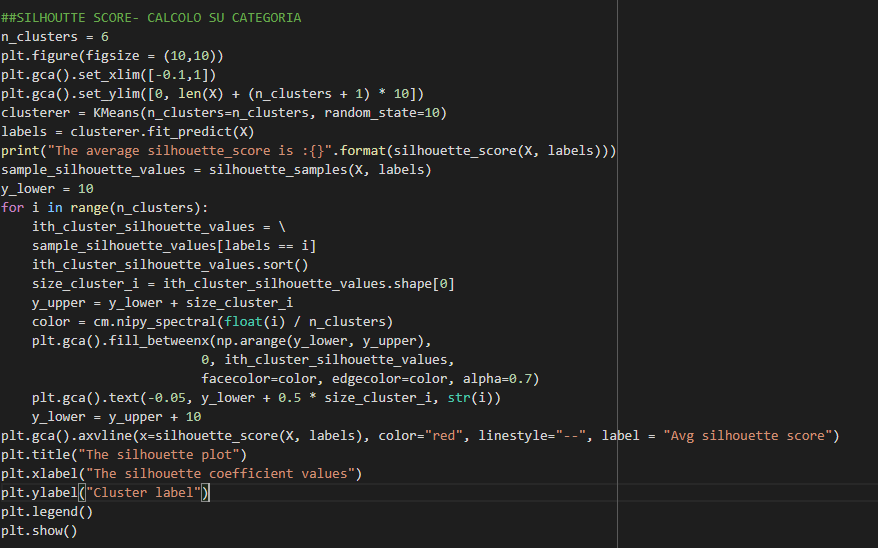
Con .scatter() e .annotate() implementate mediante la libreria “*matplotlib”*, siamo riusciti a creare grafi a dispersione con le annotazioni delle rispettive categorie, così da identificare graficamente ogni singolo cluster.

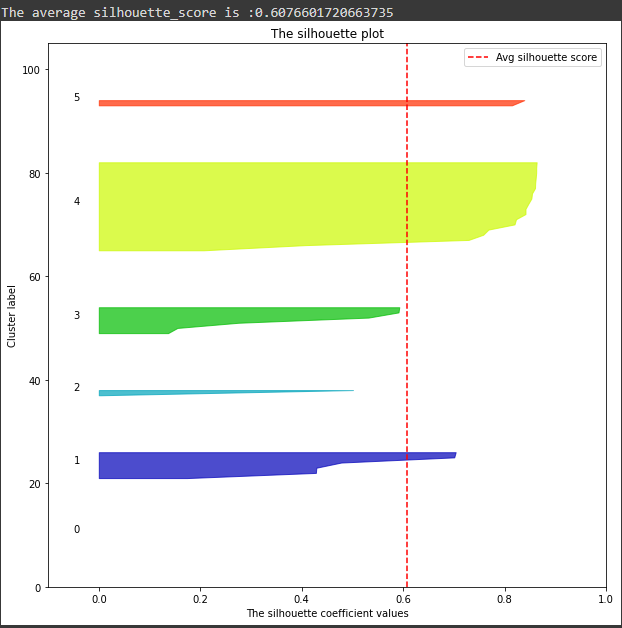


**SILHOUTTE SCORE**

Per poter misurare la consistenza dei cluster, ovvero misurare quanto simili sono gli elementi che compongono ogni singolo cluster, abbiamo utilizzato il “*coefficiente di forma”*.

Implementando il seguente algoritmo avremo come risultato un grafico, il quale ripoterà la bontà del cluster variando da un range di valori [-1,+1].





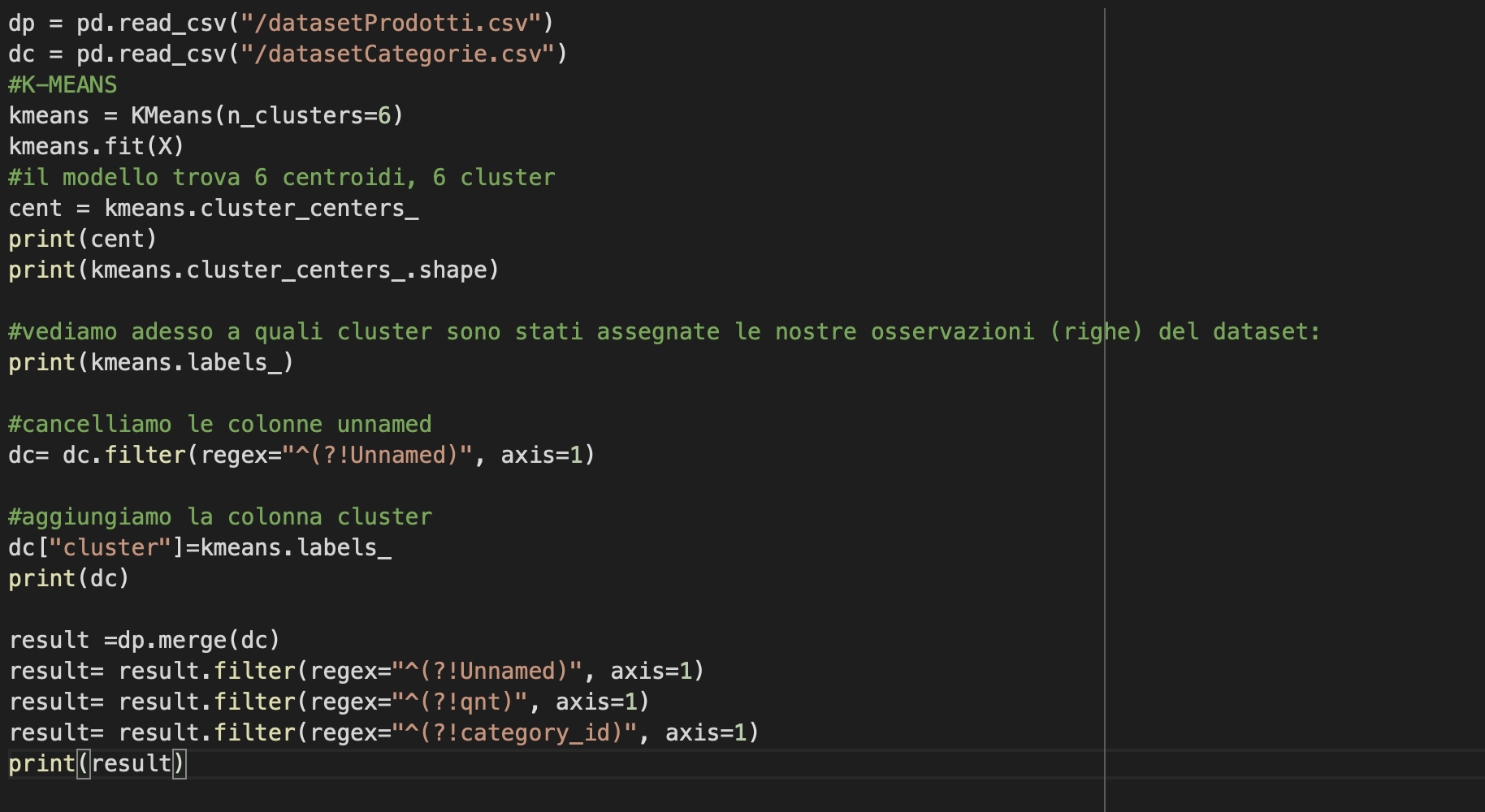
1. **Predizione del cluster di appartenenza dei nuovi campioni**

La predizione verrà effettuata mediante il metodo ***predict()*** al quale passeremo un array X, che conterrà il nome della categoria e il numero d’ordine di ogni transazione commerciale, presente sul nostro Dataset.

Tale metodo verrà eseguito dopo la funzione ***.fit(X)*** che non farà altro che addestrare il nostro modello sulla base dei dati precedentemente indicati.

Grazie a queste funzioni potremo addestrare e fare predizioni sul nostro dataset ogni qualvolta venga aggiornato con nuove transazioni. Infatti la predizione riguarderà la quantità venduta di ogni singola categoria e così facendo verranno definiti cluster contenenti categorie simili.

1. **Implementazione**



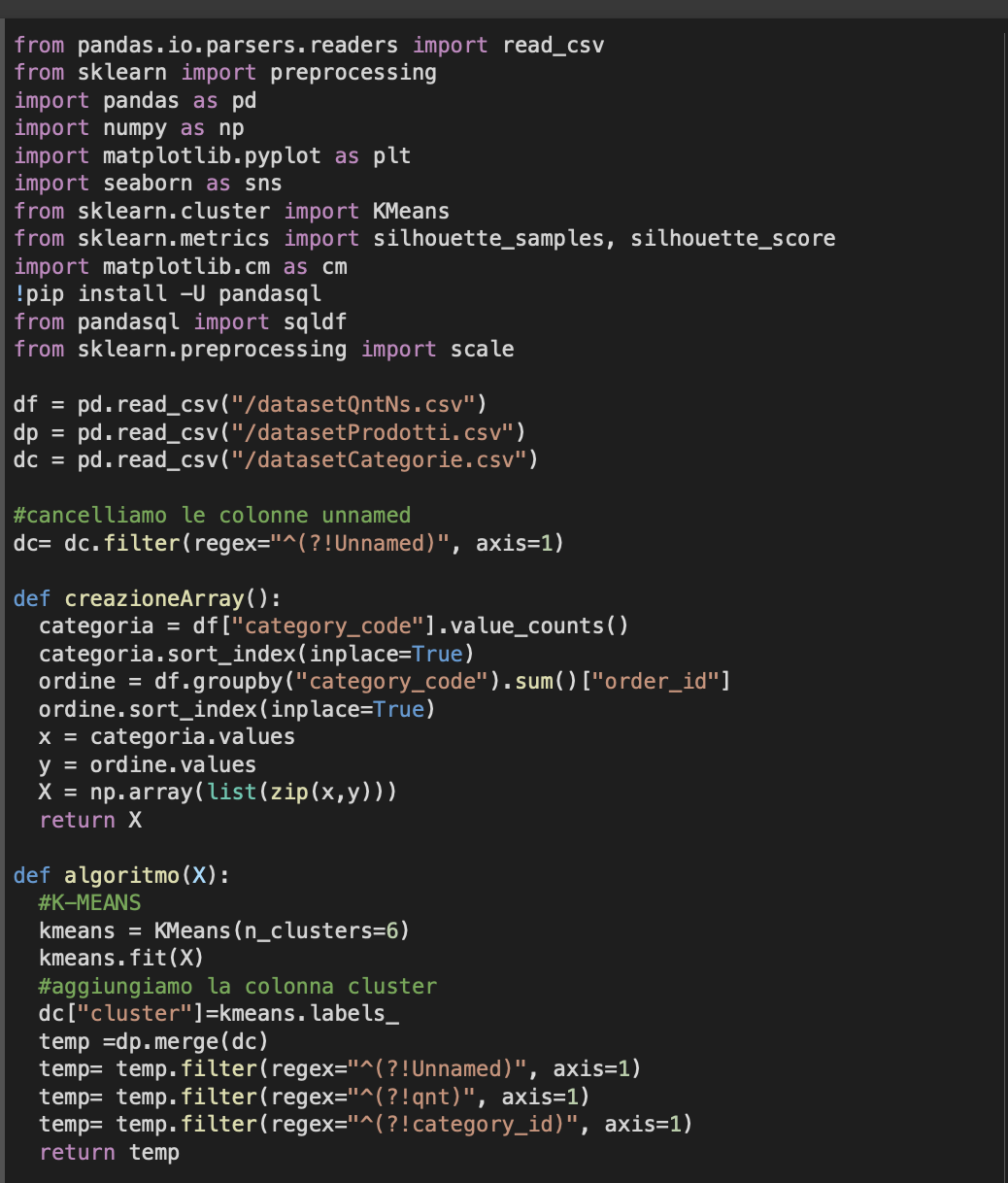
Una volta completata l’analisi dei Cluster, è stato sviluppato un algoritmo che permette di poter visualizzare i risultati prodotti.

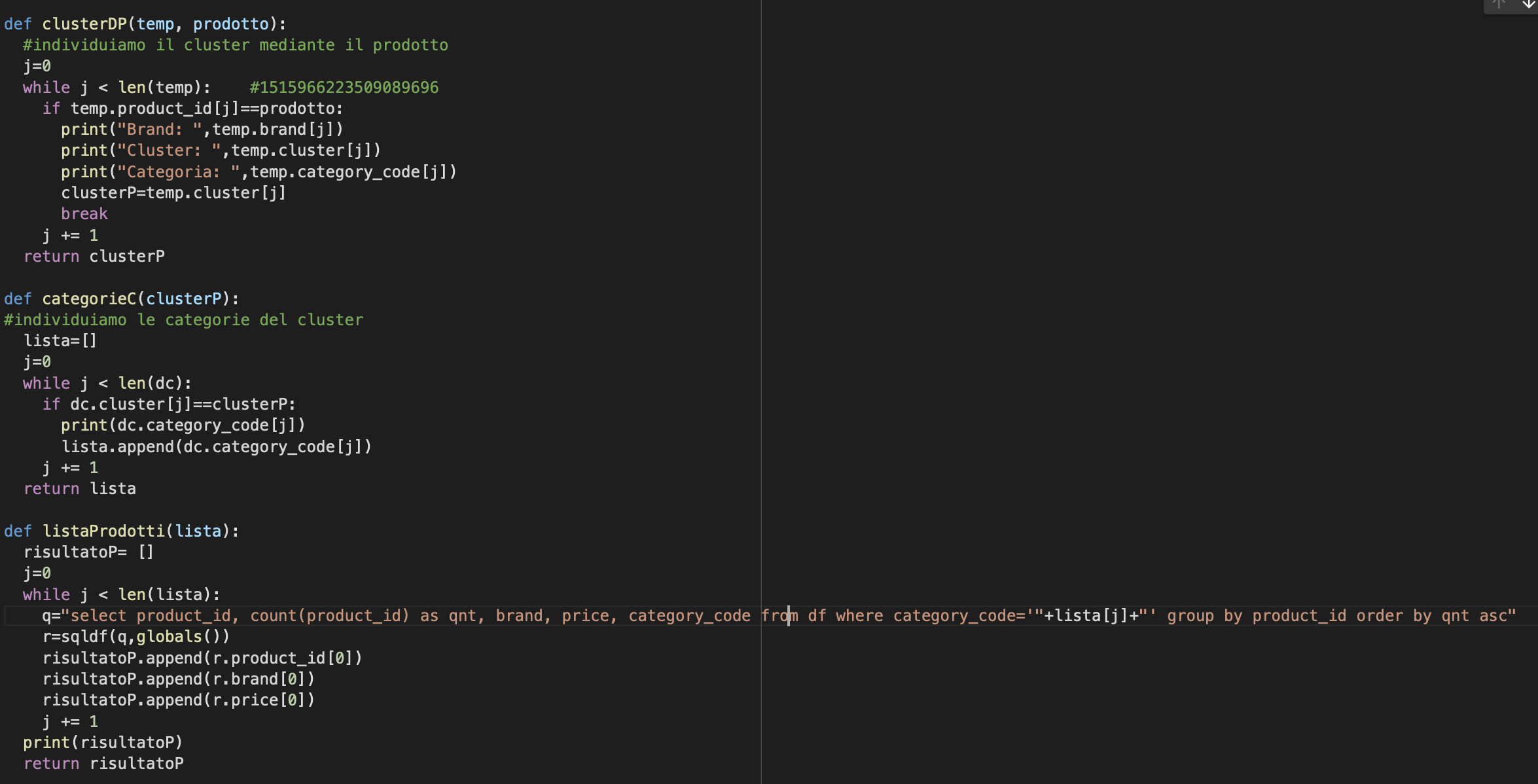
Dopo aver effettuato la funzione ***fit()*** del nostro modello, stampiamo le labels che ci indicheranno il cluster assegnato per ogni campione analizzato. Quest’array verrà poi utilizzato per creare un DataFrame che avrà tutti i prodotti con cluster assegnato mediante la categoria di appartenenza.

Innanzitutto, cancelleremo le colonne unnamed e di seguito andremo a creare una feature “cluster” che conterrà il valore del cluster assegnato per ogni campione in un DataFrame, dopodiché verrà effettuato un merge che metterà in correlazione ogni prodotto con il rispettivo cluster della categoria, dando vita ad un DataFrame temporaneo.

Il nostro algoritmo è strutturato mediante funzioni che ci permettono di:

* Caricare il dataset aggiornato ogni volta che si verifica una nuova transazione commerciale.
* Preparazione delle feature per essere gestite dall’algoritmo K-Means.
* Algoritmo K-Means sui nuovi dati per calcolare i valori dei cluster.
* Funzione che dato in input un prodotto restituisce il cluster della categoria.
* Funzione per individuare il contenuto del cluster precedentemente individuato.
* Funzione che restituisce l’obietto preposto.





Il nostro obiettivo si realizza nell’ultima funzione, dove andremo a restituire una lista di prodotti meno venduti di ogni categoria, correlate sulla base dei cluster individuati.

Il prodotto restituito di ogni singola categoria risulterà essere il prodotto meno venduto. Tale operazione verrà eseguita attraverso la Query:

***"select product\_id, count(product\_id) as qnt, brand, price, category\_code from df where category\_code='"+lista[j]+"' group by product\_id order by qnt asc".***