Università degli studi di Salerno
Dipartimento di Informatica
Corso di Laurea in Informatica

FONDAMENTI DI INTELLIGENZA ARTIFICIALE

"Invendum"

Docente:
Fabio Palomba

Studenti:

Nome	*Matricola*
Luigi De Chiara	0512109483
Alex Ferrara	0512106300

https://github.com/alexferrara74/Invendum_FIA.git

Anno Accademico: 2021/22

INDICE

Invendum

1. **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".

2. **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, così da ridimensionare le quantità dei prodotti invenduti.

3. **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.

4. Analisi del problema

Il problema proposto poteva essere risolto 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 si è deciso di applicare un algoritmo di **clustering** non avendo a disposizione etichette da assegnare ai dati.

5. Raccolta, analisi ed elaborazione dei dati

5.1 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.

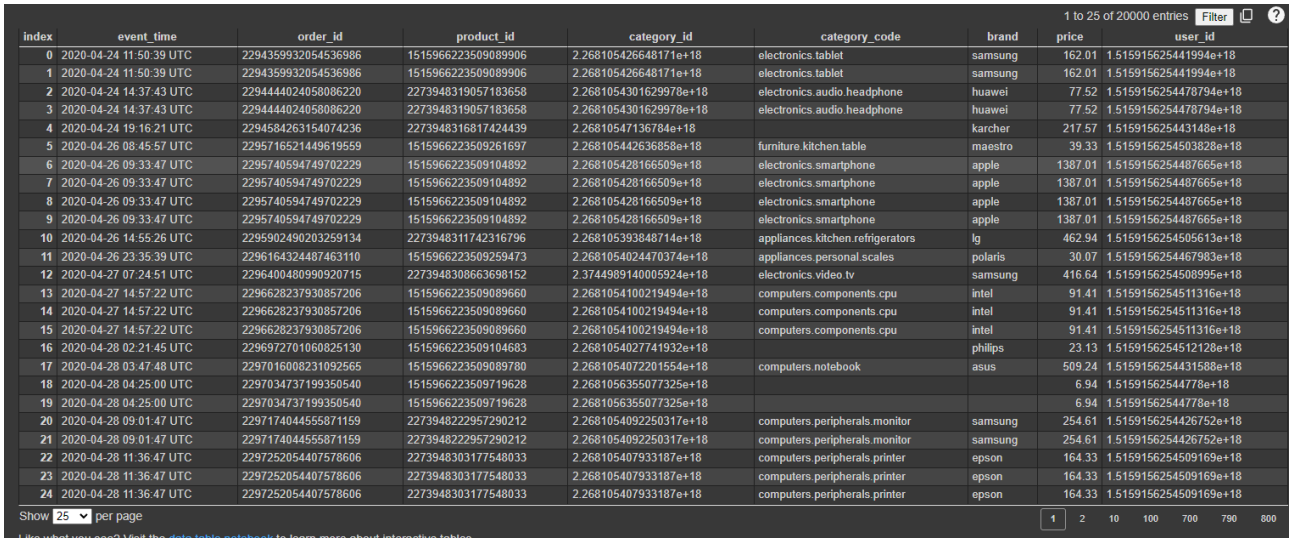
5.2 Analisi del dataset

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.

6. 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.



1 to 25 of 20000 entries Filter

index	event_time	order_id	product_id	category_id	category_code	brand	price	user_id
0	2020-04-24 11:50:39 UTC	2294359932054536986	1515966223509089906	2.268105426648171e+18	electronics.tablet	samsung	162.01	1.515915625441994e+18
1	2020-04-24 11:50:39 UTC	2294359932054536986	1515966223509089906	2.268105426648171e+18	electronics.tablet	samsung	162.01	1.515915625441994e+18
2	2020-04-24 14:37:43 UTC	2294444024058086220	2273948319057183658	2.268105430162997 8e+18	electronics.audio.headphone	huawei	77.52	1.515915625447879 4e+18
3	2020-04-24 14:37:43 UTC	2294444024058086220	2273948319057183658	2.268105430162997 8e+18	electronics.audio.headphone	huawei	77.52	1.515915625447879 4e+18
4	2020-04-24 19:16:21 UTC	2294584263154074236	2273948316817424439	2.26810547136784e+18		karcher	217.57	1.515915625443148e+18
5	2020-04-26 08:45:57 UTC	2295716521449619559	1515966223509261697	2.268105442636858e+18	furniture.kitchen.table	maestro	39.33	1.515915625450382 8e+18
6	2020-04-26 09:33:47 UTC	2295740594749702229	1515966223509104892	2.268105428166509e+18	electronics.smartphone	apple	1387.01	1.515915625448766 5e+18
7	2020-04-26 09:33:47 UTC	2295740594749702229	1515966223509104892	2.268105428166509e+18	electronics.smartphone	apple	1387.01	1.515915625448766 5e+18
8	2020-04-26 09:33:47 UTC	2295740594749702229	1515966223509104892	2.268105428166509e+18	electronics.smartphone	apple	1387.01	1.515915625448766 5e+18
9	2020-04-26 09:33:47 UTC	2295740594749702229	1515966223509104892	2.268105428166509e+18	electronics.smartphone	apple	1387.01	1.515915625448766 5e+18
10	2020-04-26 14:55:26 UTC	2295902490203259134	2273948311742316796	2.268105393848714e+18	appliances.kitchen.refrigerators	lg	462.94	1.515915625450561 3e+18
11	2020-04-26 23:35:39 UTC	2296164324487463110	1515966223509259473	2.268105402447037 4e+18	appliances.personal.scales	polaris	30.07	1.515915625446798 3e+18
12	2020-04-27 07:24:51 UTC	2296400480990920715	2273948308663698152	2.374498914000592 4e+18	electronics.video.tv	samsung	416.64	1.515915625450899 5e+18
13	2020-04-27 14:57:22 UTC	2296628237930857206	1515966223509089660	2.268105410021949 4e+18	computers.components.cpu	intel	91.41	1.515915625451131 6e+18
14	2020-04-27 14:57:22 UTC	2296628237930857206	1515966223509089660	2.268105410021949 4e+18	computers.components.cpu	intel	91.41	1.515915625451131 6e+18
15	2020-04-27 14:57:22 UTC	2296628237930857206	1515966223509089660	2.268105410021949 4e+18	computers.components.cpu	intel	91.41	1.515915625451131 6e+18
16	2020-04-28 02:21:45 UTC	2296972701060825130	1515966223509104683	2.268105402774193 2e+18		philips	23.13	1.515915625451212 8e+18
17	2020-04-28 03:47:48 UTC	2297016008231092565	1515966223509089780	2.268105407220155 4e+18	computers.notebook	asus	509.24	1.515915625443158 8e+18
18	2020-04-28 04:25:00 UTC	2297034737199350540	1515966223509719628	2.268105635507732 5e+18			6.94	1.51591562544778e+18
19	2020-04-28 04:25:00 UTC	2297034737199350540	1515966223509719628	2.268105635507732 5e+18			6.94	1.51591562544778e+18
20	2020-04-28 09:01:47 UTC	2297174044555871159	2273948222957290212	2.268105409225031 7e+18	computers.peripherals.monitor	samsung	254.61	1.515915625442675 2e+18
21	2020-04-28 09:01:47 UTC	2297174044555871159	2273948222957290212	2.268105409225031 7e+18	computers.peripherals.monitor	samsung	254.61	1.515915625442675 2e+18
22	2020-04-28 11:36:47 UTC	2297252054407578606	2273948303177548033	2.268105407933187e+18	computers.peripherals.printer	epson	164.33	1.515915625450916 9e+18
23	2020-04-28 11:36:47 UTC	2297252054407578606	2273948303177548033	2.268105407933187e+18	computers.peripherals.printer	epson	164.33	1.515915625450916 9e+18
24	2020-04-28 11:36:47 UTC	2297252054407578606	2273948303177548033	2.268105407933187e+18	computers.peripherals.printer	epson	164.33	1.515915625450916 9e+18

Show 25 per page

1 2 10 100 700 790 800

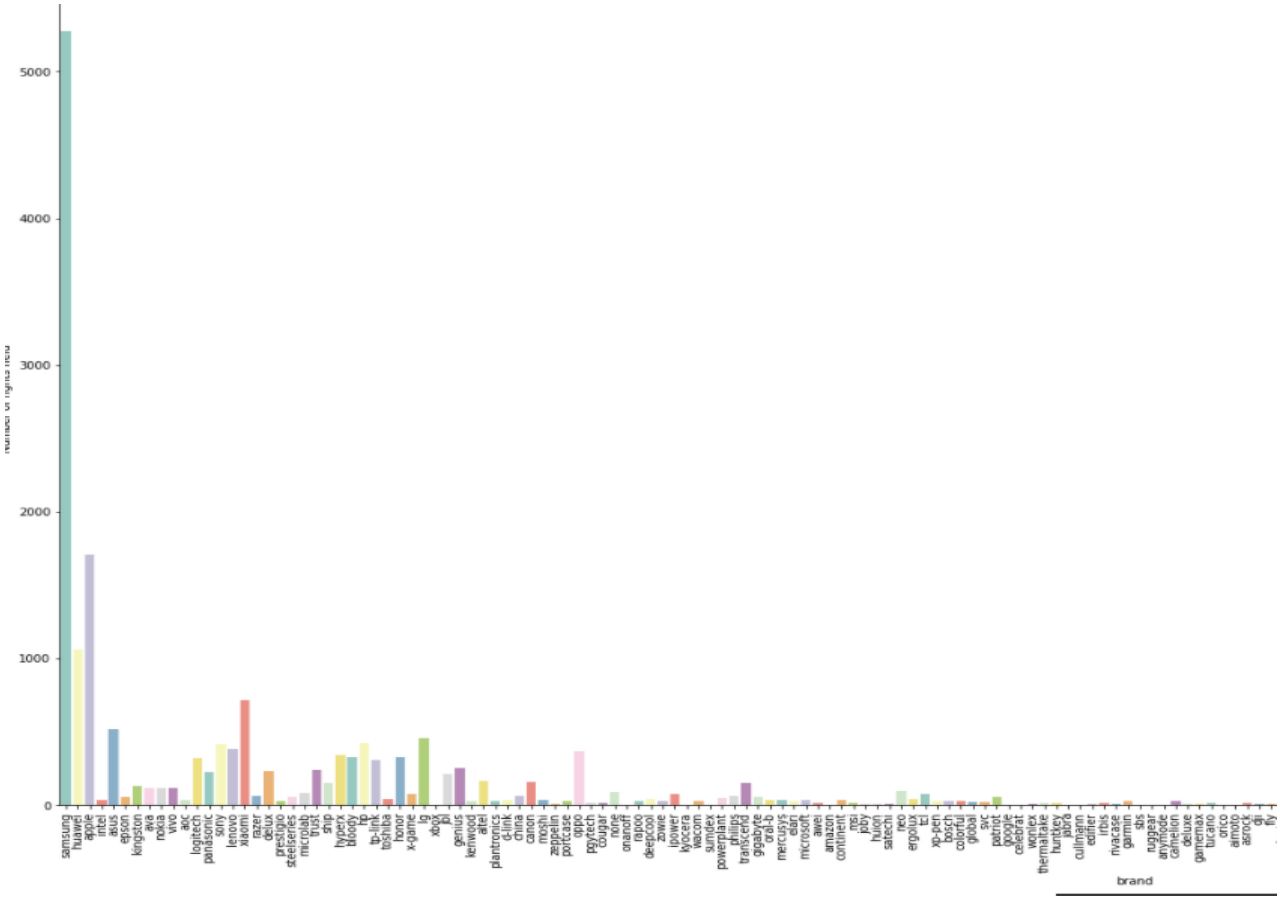
Il secondo passo è stato quello di ridurre la dimensione del nostro dataset applicando l’agente solo su un sottogruppo di categorie, poiché gestire tutte le categorie con i relativi record sarebbe stato impensabile; pertanto, si è optato di considerare circa trenta categorie:

```
"SELECT  category_code,count(category_code) from categori
a where category_code like 'computers.%'or category_code
like 'electronics.%' group by category_code"
```

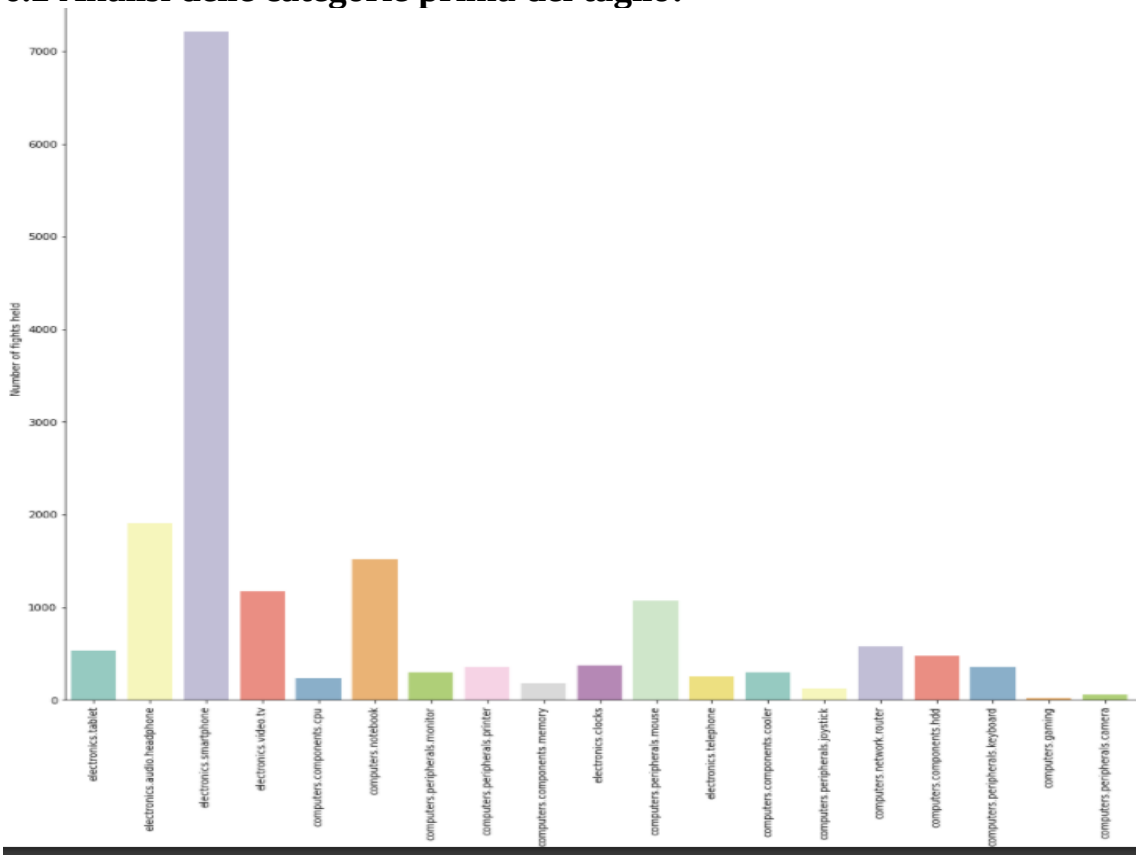
```
df=pd.read_csv(r'/datasetcompleto.csv',sep=',',index_col=['order_id'],us
ecols=['order_id','product_id','category_id','category_code','brand','pr
ice','user_id'])
q="select * from df where category_code like 'computers.%' or  category_
code like 'electronics.%'"
df=sqldf(q,globals())

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.

6.1 Analisi dei brand prima del taglio:



6.2 Analisi delle categorie prima del taglio:



7. 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

7.1 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.

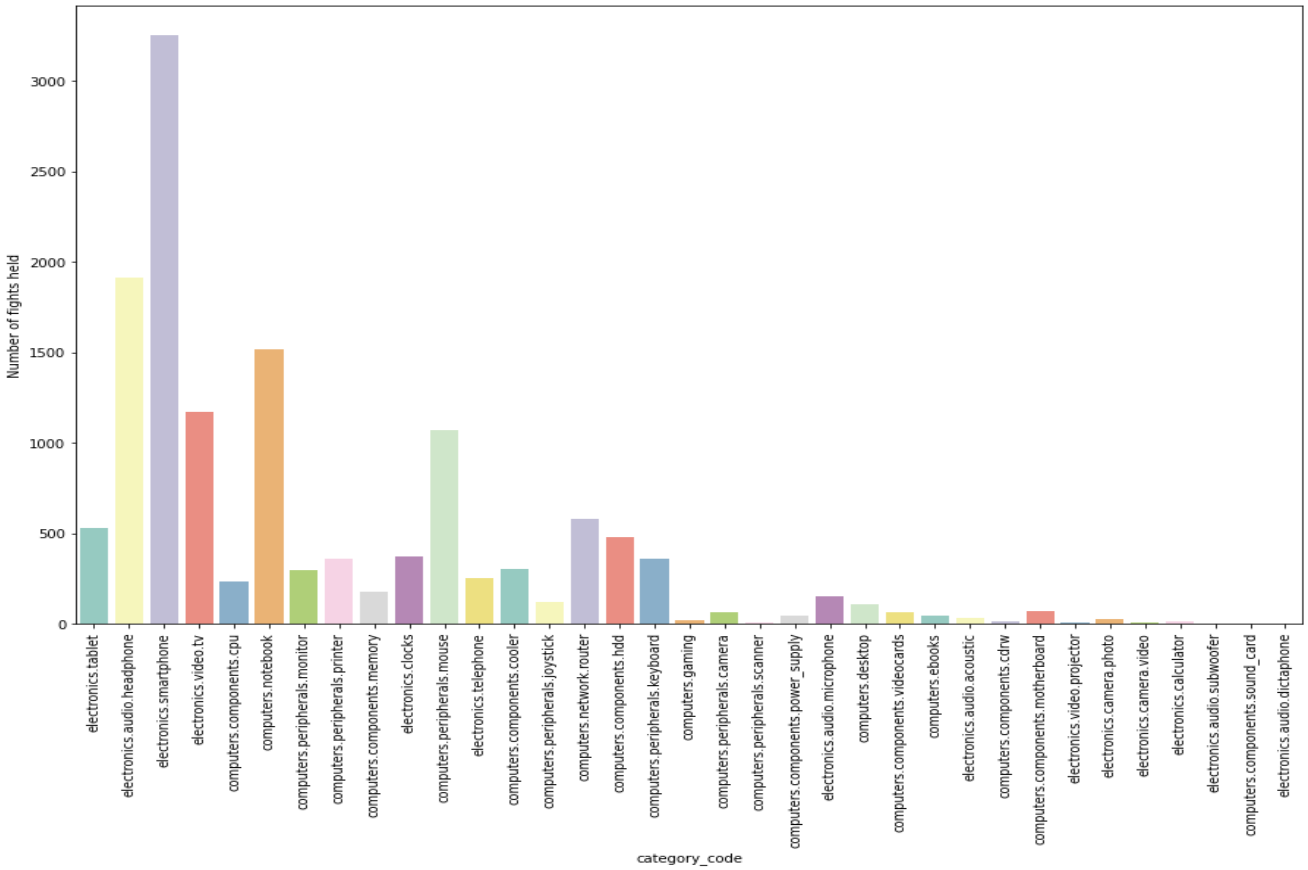
7.2 K-Means

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 category_code e order_id per effettuare la predizione delle quantità vendute per ogni singola categoria.

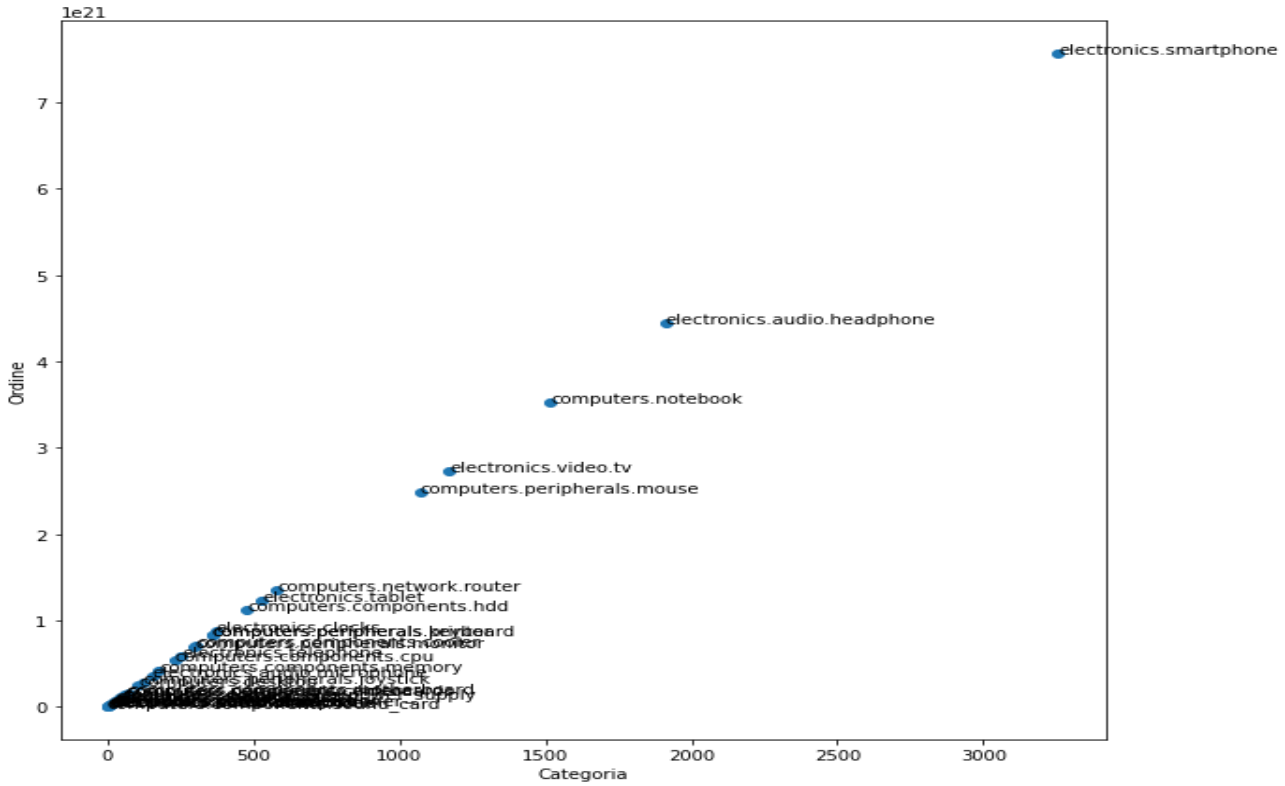
```
from pandas.io.parsers.readers import read_csv
from sklearn import preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm
df = pd.read_csv("/datasetQntNs.csv")
df.head()
plt.figure(figsize = (15,10))
sns.countplot(x = "category_code", palette = "Set3",data = df)
plt.xticks(rotation = 90)
plt.ylabel("Number of fights held")
plt.show()


categoria = df["category_code"].value_counts()
categoria.sort_index(inplace=True)
ordine = df.groupby("category_code").sum()["order_id"]
ordine.sort_index(inplace=True)
from sklearn.preprocessing import scale
x = categoria.values
y = ordine.values
plt.figure(figsize = (10,10))
plt.scatter(x, y)
plt.xlabel("Categoria")
plt.ylabel("Ordine")
for i, txt in enumerate(categoria.index.values):
    a = plt.gca()
    plt.annotate(txt, (x[i], y[i]))
plt.show()
```

7.3 Analisi delle categorie:



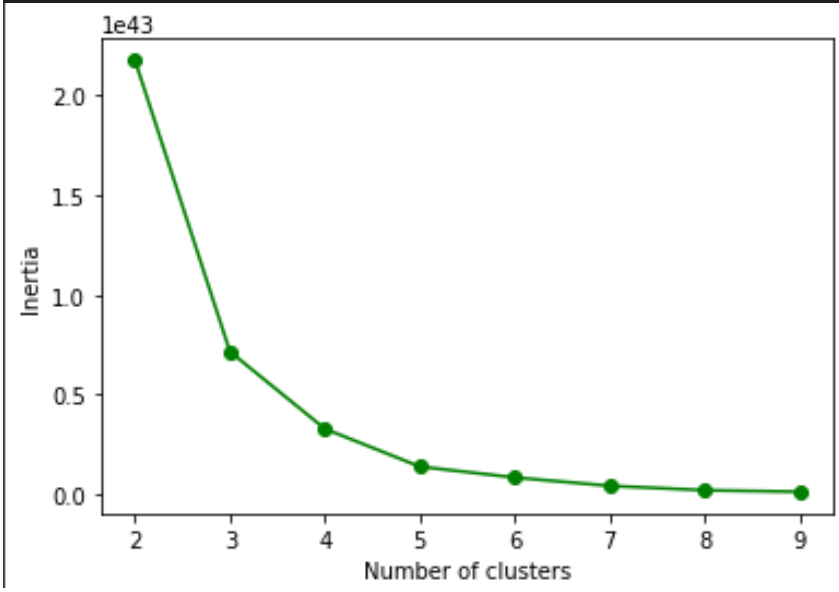
7.4 Grafico a dispersione:

Basato sul numero di acquisti e sugli ordini effettuati dai clienti.



8. **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.

```
X = np.array(list(zip(x,y)))
inertias = []
for k in range(2, 10):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    inertias.append(kmeans.inertia_)
plt.plot(range(2,10), inertias, "o-g")
plt.xlabel("Number of clusters")
plt.ylabel("Inertia")
plt.show()
```



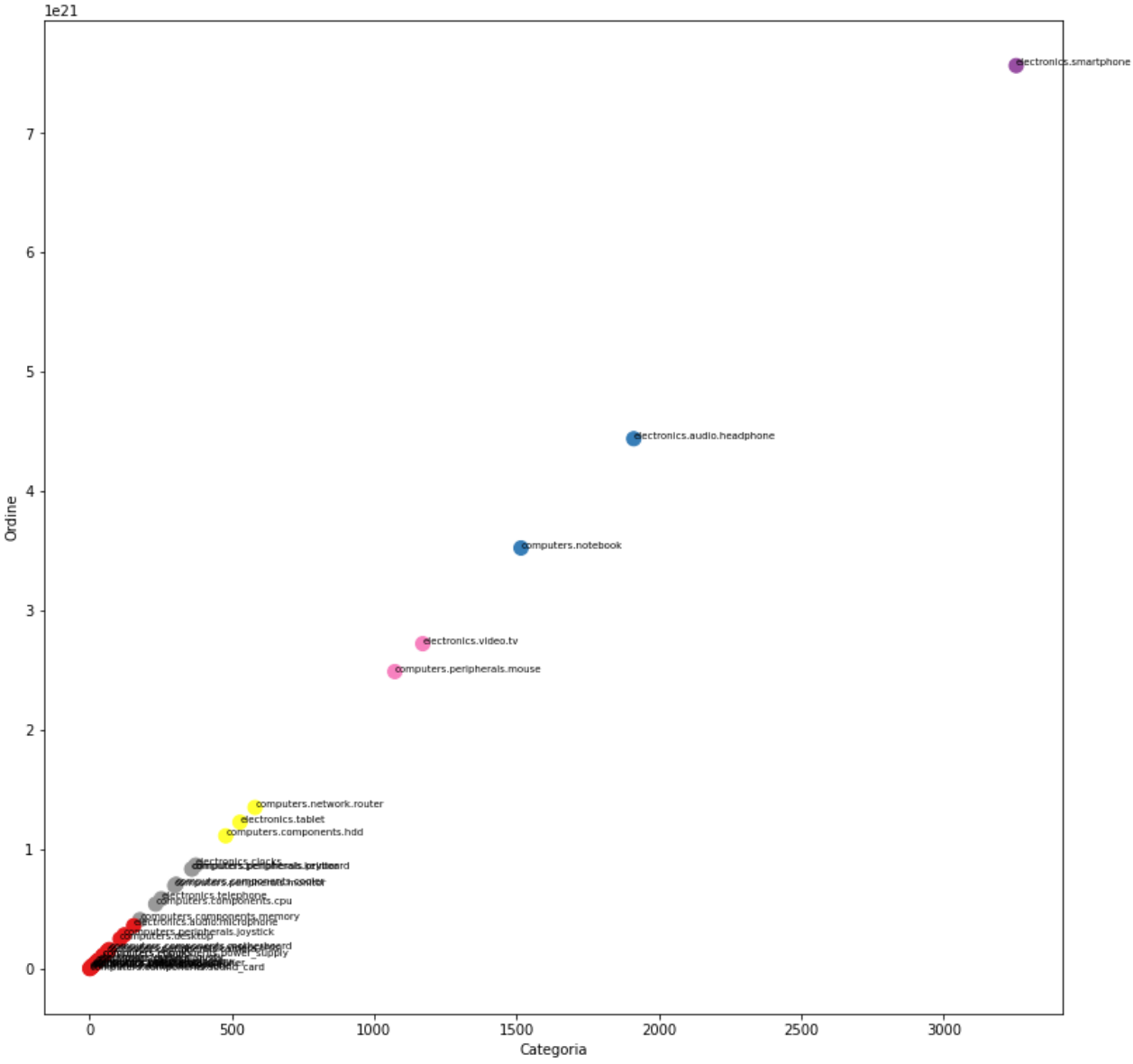
Dal punto di gomito si evince che il numero ideale di cluster da utilizzare è pari a 6. Dopo aver assegnato tale valore a k(n_clusters) possiamo implementare l'algoritmo precedentemente scelto.
Mediante le funzioni:

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

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

```python
#K-MEANS
kmeans = KMeans(n_clusters=6)
kmeans.fit(X)
y_kmeans= kmeans.predict(X)
plt.figure(figsize = (15,15))
plt.xlabel("Categoria")
plt.ylabel("Ordine")
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=100, cmap='Set1')
for i, txt in enumerate(categoria.index.values):
    plt.annotate(txt, (X[i,0], X[i,1]), size = 7)
plt.show()
```

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.



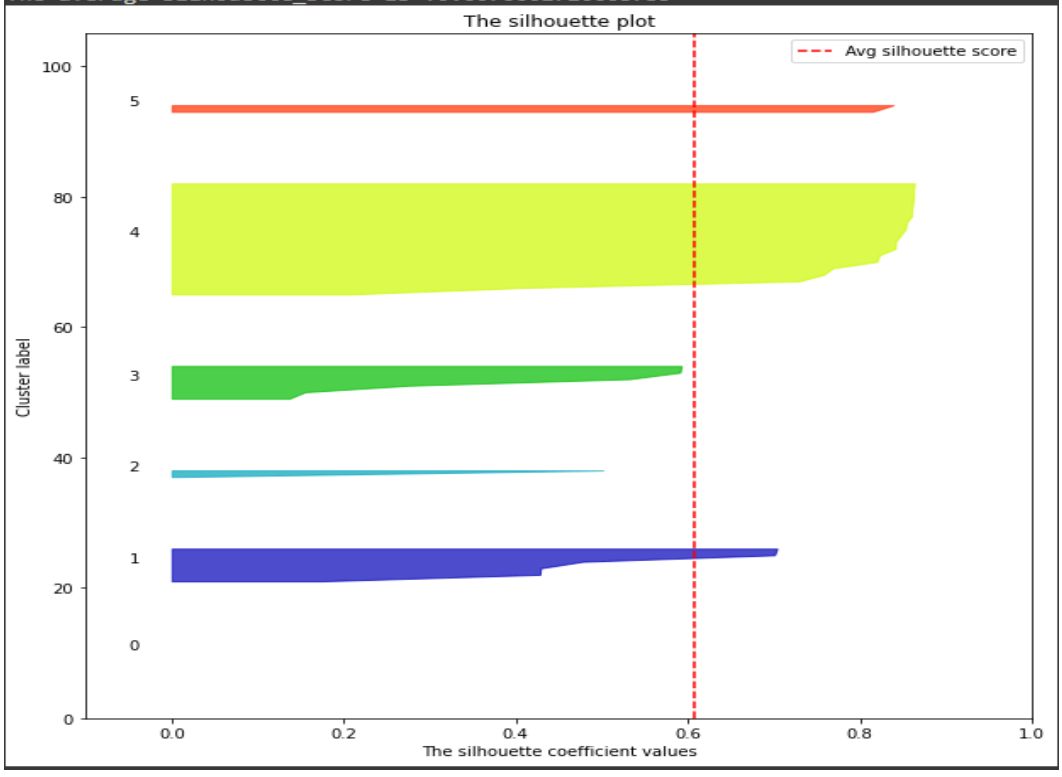
9. SILHOUTTE SCORE

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].

```
##SILHOUTTE SCORE- CALCOLO SU CATEGORIA
n_clusters = 6
plt.figure(figsize = (10,10))
plt.gca().set_xlim([-0.1,1])
plt.gca().set_ylim([0, len(X) + (n_clusters + 1) * 10])
clusterer = KMeans(n_clusters=n_clusters, random_state=10)
labels = clusterer.fit_predict(X)
print("The average silhouette_score is :{}".format(silhouette_score(X, labels)))
sample_silhouette_values = silhouette_samples(X, labels)
y_lower = 10
for i in range(n_clusters):
    ith_cluster_silhouette_values = \
    sample_silhouette_values[labels == i]
    ith_cluster_silhouette_values.sort()
    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i
    color = cm.nipy_spectral(float(i) / n_clusters)
    plt.gca().fill_betweenx(np.arange(y_lower, y_upper),
                            0, ith_cluster_silhouette_values,
                            facecolor=color, edgecolor=color, alpha=0.7)
    plt.gca().text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
    y_lower = y_upper + 10
plt.gca().axvline(x=silhouette_score(X, labels), color="red", linestyle="--", label = "Avg silhouette score")
plt.title("The silhouette plot")
plt.xlabel("The silhouette coefficient values")
plt.ylabel("Cluster label")
plt.legend()
plt.show()
```

```
The average silhouette_score is :0.6076601720663735
```



10. 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.

11. Implementazione

```
dp = pd.read_csv("/datasetProdotti.csv")
dc = pd.read_csv("/datasetCategorie.csv")
#K-MEANS
kmeans = KMeans(n_clusters=6)
kmeans.fit(X)
#il modello trova 6 centroidi, 6 cluster
cent = kmeans.cluster_centers_
print(cent)
print(kmeans.cluster_centers_.shape)

#vediamo adesso a quali cluster sono stati assegnate le nostre osservazioni (righe) del dataset:
print(kmeans.labels_)

#cancelliamo le colonne unnamed
dc= dc.filter(regex="^(?!Unnamed)", axis=1)

#aggiungiamo la colonna cluster
dc["cluster"]=kmeans.labels_
print(dc)

result =dp.merge(dc)
result= result.filter(regex="^(?!Unnamed)", axis=1)
result= result.filter(regex="^(?!qnt)", axis=1)
result= result.filter(regex="^(?!category_id)", axis=1)
print(result)
```

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.

```
from pandas.io.parsers.readers import read_csv
from sklearn import preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm
!pip install -U pandasql
from pandasql import sqldf
from sklearn.preprocessing import scale

df = pd.read_csv("/datasetQntNs.csv")
dp = pd.read_csv("/datasetProdotti.csv")
dc = pd.read_csv("/datasetCategorie.csv")

#cancelliamo le colonne unnamed
dc= dc.filter(regex="^(?!Unnamed)", axis=1)

def creazioneArray():
  categoria = df["category_code"].value_counts()
  categoria.sort_index(inplace=True)
  ordine = df.groupby("category_code").sum()["order_id"]
  ordine.sort_index(inplace=True)
  x = categoria.values
  y = ordine.values
  X = np.array(list(zip(x,y)))
  return X

def algoritmo(X):
  #K-MEANS
  kmeans = KMeans(n_clusters=6)
  kmeans.fit(X)
  #aggiungiamo la colonna cluster
  dc["cluster"]=kmeans.labels_
  temp =dp.merge(dc)
  temp= temp.filter(regex="^(?!Unnamed)", axis=1)
  temp= temp.filter(regex="^(?!qnt)", axis=1)
  temp= temp.filter(regex="^(?!category_id)", axis=1)
  return temp
```

```python
def clusterDP(temp, prodotto):
  #individuiamo il cluster mediante il prodotto
  j=0
  while j < len(temp):    #1515966223509089696
    if temp.product_id[j]==prodotto:
      print("Brand: ",temp.brand[j])
      print("Cluster: ",temp.cluster[j])
      print("Categoria: ",temp.category_code[j])
      clusterP=temp.cluster[j]
      break
    j += 1
  return clusterP

def categorieC(clusterP):
#individuiamo le categorie del cluster
  lista=[]
  j=0
  while j < len(dc):
    if dc.cluster[j]==clusterP:
      print(dc.category_code[j])
      lista.append(dc.category_code[j])
    j += 1
  return lista

def listaProdotti(lista):
  risultatoP= []
  j=0
  while j < len(lista):
    q="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"
    r=sqldf(q,globals())
    risultatoP.append(r.product_id[0])
    risultatoP.append(r.brand[0])
    risultatoP.append(r.price[0])
    j += 1
  print(risultatoP)
  return risultatoP
```

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".

12. Applicazione

Dopo aver eseguito tutte le operazioni elencate nei passi precedenti, avremo risultati fruibili da un applicativo.
Abbiamo creato una rappresentazione fedele di un possibile evento che si verificherà nei principali e-commerce.
Il nostro applicativo simula la scelta di un prodotto con la quale verrà richiamata la funzione agente, che effettuerà la predizione e la stampa dei prodotti consigliati.
Tale stampa all'interno di un sito e-commerce farà riferimento ad uno *span* dedicato ai prodotti consigliati.