



UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Homework 3 – Big Data Engineering

Anno Accademico 2022-2023

Riccio Emanuele – M63001339

Tammaro Ferdinando – M63001380

Prof. Moscato Vincenzo

Sommario

- Introduzione 3
 - Configurazione 3
- Kafka 4
 - Producer 4
 - Consumer 4
- Hive 7
- Query e Dashboard 9
 - Query 10
 - Query 1 10
 - Query 2 10
 - Query 3 10
 - Query 4 11
 - Query 5 11
 - Query List 11
 - Esecuzione query e creazione grafici su Dashboard 11
 - Dashboard 12

Introduzione

Il task da eseguire per il terzo homework del corso di Big Data Engineering consiste nell'acquisizione di uno stream di dati tramite Apache Kafka e memorizzarlo in un log su HDFS, per poi effettuarci delle query utilizzando PySpark, Pig, oppure Hive (già utilizzati nel primo homework) e visualizzare i risultati su un'apposita dashboard.

In questo homework, sono stati utilizzati, oltre a Kafka e HDFS, Hive per effettuare la query e il framework Streamlit per creare la dashboard tramite il linguaggio Python. Kafka, Hadoop e Hive sono stati installati in dei container Docker per semplificare le operazioni di setup, mentre tutte le operazioni eseguite tramite Python sono state eseguite su host Windows. Per verificare l'esito delle richieste è stato inoltre utilizzato il software *Postman*. I dati presi in esame provengono da un dataset CSV fornito dalla protezione civile italiana riguardante l'epidemia di Covid-19 in Italia, e sono aggiornati a novembre 2022.

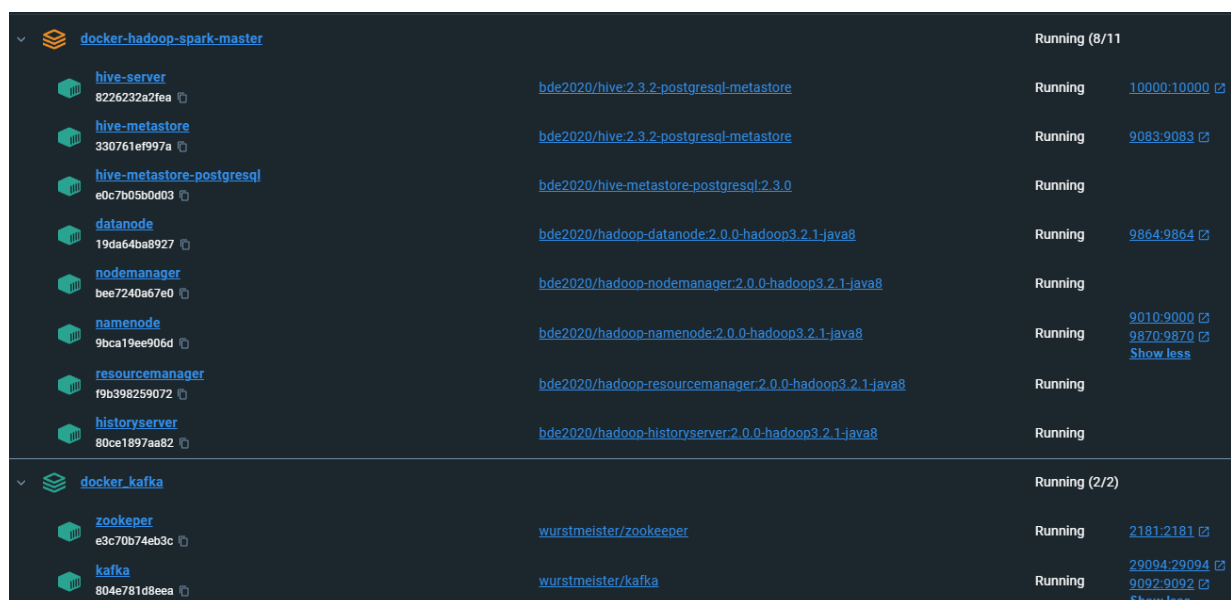
Configurazione

Su Docker abbiamo utilizzato i seguenti container visualizzati in figura 1.

Kafka è stato esposto sulla porta 9092, invece il *namenode* di *Hadoop* è stato messo sulla porta 9870. Nel caso in esame, si è deciso di utilizzare un singolo *datanode*; il server *Hive* è stato esposto sulla porta 10000.

Per problemi di compatibilità, nel file host di windows si è dovuto aggiungere il seguente codice per poter correttamente tradurre gli host name.

```
127.0.0.1    localhost datanode namenode
::1         localhost datanode namenode
```



docker-hadoop-spark-master		Running (8/11)
hive-server	bde2020/hive:2.3.2-postgresql-metastore	Running 10000:10000
hive-metastore	bde2020/hive:2.3.2-postgresql-metastore	Running 9083:9083
hive-metastore-postgresql	bde2020/hive-metastore-postgresql:2.3.0	Running
datanode	bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8	Running 9864:9864
nodemanager	bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8	Running
namenode	bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8	Running 9010:9000 9870:9870
resourcemanager	bde2020/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8	Running
historyserver	bde2020/hadoop-historyserver:2.0.0-hadoop3.2.1-java8	Running
docker_kafka		Running (2/2)
zookeeper	wurstmeister/zookeeper	Running 2181:2181
kafka	wurstmeister/kafka	Running 29094:29094 9092:9092

Figura 1 - Container di Docker

Kafka

Trattandosi di un problema con un singolo consumatore e un singolo produttore, si è scelto di utilizzare un topic con una singola partizione e con un replication factor di 1.

Questo codice è stato eseguito direttamente nella shell:

```
kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic dati_covid
```

Producer

In python si è poi provveduto a definire la funzione del produttore, partendo dall'oggetto `KafkaProducer`, che legge riga per riga il file csv presente nel filesystem di Windows e la scrive sul topic di Kafka come messaggio. Per questioni di complessità computazionale, ci si è limitati alle prime 10.000 righe del dataset.

```
1. from kafka import KafkaProducer
2. import json
3.
4. topic = "dati_covid"
5.
6. def producer():
7.     producer = KafkaProducer(
8.         bootstrap_servers=['localhost:9092'],
9.     )
10.    count = 0
11.    with open("C:\\Users\\eleun\\Downloads\\covid19.csv", encoding="utf8") as
        file:
12.        for line in file:
13.            producer.send(topic, json.dumps(line).encode('utf-8'))
14.            count = count + 1
15.            if (count==10000):
16.                break
17.    print(f"count: {count}")
```

Consumer

La funzione consumer è un po' più complessa, in quanto deve interagire sia con l'HDFS sia con Kafka. Per questo motivo viene istanziato un oggetto `InsecureClient` che si collega con l'HDFS sulla porta 9870 tramite protocollo webHDFS. Viene innanzitutto cancellato e ricreato un file testuale vuoto sull'HDFS, chiamato "dati_covid.txt". Nella funzione viene poi definito un oggetto `KafkaConsumer`, che si occupa di leggere i




messaggi dal topic di Kafka. Infine, viene riaperto il file testuale su HDFS in modalità di scrittura da parte del client, e per ogni messaggio che viene letto dal topic, viene aggiornato il file mettendo in append il messaggio appena letto. Il codice python è il seguente:

```


1. import time
2. from kafka import KafkaConsumer
3. from hdfs import InsecureClient
4. import json
5.
6. topic = "dati_covid"
7. client = InsecureClient('http://localhost:9870', user='bigdata')
8.
9. #http://localhost:9870/webhdfs/v1/user/bigdata/data/dati_covid.txt?op=OPEN
10.
11. file = "data/dati_covid.txt"
12. client.delete(file)
13. with client.write(file, encoding='utf-8') as writer:
14.     pass
15.
16. def consumer():
17.     consumer = KafkaConsumer(
18.         topic,
19.         bootstrap_servers='localhost:9092',
20.         auto_offset_reset='earliest',
21.         consumer_timeout_ms=10000
22.     )
23.     with client.write(file, append=True, encoding='utf-8') as writer:
24.         for message in consumer:
25.             d = json.loads(message.value)
26.             print(f"Consumer " + str(d))
27.             writer.write(d)
28.

```

Browse Directory

/user/bigdata/data   

Show entries Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
<input type="checkbox"/>	-rwxr-xr-x	bigdata	supergroup	1.63 MB	May 31 10:29	3	128 MB	dati_covid.txt 

Showing 1 to 1 of 1 entries Previous **1** Next

Hadoop, 2019.

Figura 2 - HDFS

Tramite queste funzioni, sull'HDFS viene creato il file testuale che contiene riga per riga le informazioni ottenute tramite i messaggi di Kafka, come è possibile vedere in figura 2.

Hive

Con Hive sono state eseguite delle query partendo dai dati presenti sull'HDFS. Per fare ciò c'è stato bisogno di far comunicare Hive con HDFS e importare i dati in una tabella Hive. Per far comunicare questi due bastardi si è usato il seguente codice, che inizializza la tabella Hive. In particolare, con la query `create_table_query` si è creata la tabella definendo la struttura come riportato qui sotto, specificando come fossero formattate le colonne con il comando `ROW FORMAT DELIMITED FIELDS TERMINATED BY ','`, siccome le righe presenti sul file `.csv` nell'HDFS usavano come delimitatore proprio la virgola. Infine, il comando `LOAD DATA INPATH {hdfs_path} INTO TABLE {table_name}` carica i dati letti dal file `dati_covid.txt` su HDFS in una tabella.

```
1. import puretransport
2. from pyhive import hive
3.
4. transport = puretransport.transport_factory(host='localhost',
5.                                             port=10000,
6.                                             username='username',
7.                                             password='secret')
8. hive_con = hive.connect(username='username', thrift_transport=transport,
9.                          database='default')
10.
11. table_name = 'dati_covid'
12. column_names = [
13.     "data",
14.     "stato",
15.     "codice_regione",
16.     "denominazione_regione",
17.     "ricoverati_con_sintomi",
18.     "terapia_intensiva",
19.     "totale_ospedalizzati",
20.     "isolamento_domiciliare",
21.     "totale_positivi",
22.     ...
23.]
24. column_types = [
25.     "STRING",
26.     "STRING",
27.     "INT",
28.     "STRING",
29.     "INT",
30.     "INT",
31.     "INT",
```

```

32.     "INT",
33.     "INT",
34.     ...
35.]
36.
37.hdfs_path = '/user/bigdata/data/dati_covid.txt'
38.hdfs = '/user/bigdata/data/'
39.
40.create_table_query = f"CREATE EXTERNAL TABLE IF NOT EXISTS {table_name} ({',
    '.join([f'{col_name} {col_type}' for col_name, col_type in zip(column_names,
        column_types)]))" \
41.                 f"ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS
    TEXTFILE " \
42.                 f"LOCATION '{hdfs}'"
43.
44.print(create_table_query)
45.# Query per load
46.with hive_con.cursor() as cursor:
47.    cursor.execute(create_table_query)
48.
49.
50.load_query = f"LOAD DATA INPATH '{hdfs_path}' INTO TABLE {table_name}"
51.print(load_query)
52.with hive_con.cursor() as cursor:
53.    cursor.execute(load_query)
54.
55.

```


Query e Dashboard

Una volta creata la tabella, si è proceduto a creare la dashboard con il framework streamlit. Dopo aver importato le necessarie librerie e fatto le configurazioni del caso, è stata effettuata la connessione ad Hive per accedere alla tabella precedentemente definita e poter effettuare le query. Questo ha permesso la creazione di una dashboard interattiva, dove è possibile effettuare un filtraggio dei risultati in base alle regioni desiderate.

```
1. import plotly_express as px
2. import streamlit as st
3. import puretransport
4. from pyhive import hive
5. import pandas as pd
6. from pandas import DataFrame
7.
8. st.set_page_config(
9.     page_title="BDE HW 3 Riccio - Tammaro",
10.    page_icon=":syringe:",
11.    layout="wide"
12.)
13.st.title("BDE HW 3 Riccio - Tammaro")
14.transport = puretransport.transport_factory(host='localhost',
15.                                             port=10000,
16.                                             username='username',
17.                                             password='secret')
18.
19.hive_con = hive.connect(username='username', thrift_transport=transport,
20.                          database='default')
21.table_name = 'dati_covid'
22.
```

Nel seguente codice, con la prima query effettuata si ottengono i nomi distinti delle regioni, così da poter creare il filtro interattivo sulla sidebar e selezionare le regioni di interesse.

```
23.info_query = [
24.    f"SELECT DISTINCT(denominazione_regione) FROM {table_name}"
25.]
26.
27.with hive_con.cursor() as cursor:
28.    cursor.execute(info_query[0])
29.    results = cursor.fetchall()
30.    regioni = DataFrame.from_records(results)
31.
```

```

32. # filtering sidebar
33. st.sidebar.header("Filtri")
34. states = st.sidebar.multiselect(
35.     "Regioni",
36.     options=regioni[regioni.columns[0]],
37.     default=regioni[regioni.columns[0]]
38.)
39.
40. escaped_list = [f"'%s'" % (value.replace("'", "\\')) for value in states]
41. lista_string = ", ".join(escaped_list)
42.
43. where_regioni = f"WHERE denominazione_regione IN ({lista_string})"

```

Query

Per effettuare le query è stata definita una lista di stringhe che potesse essere iterata tramite un ciclo for. Al suo interno sono presenti le seguenti query:

Query 1

Nella prima query, si visualizza semplicemente il numero di casi totali per regione nel periodo specificato dal dataset.

```

1. SELECT denominazione_regione, SUM(totale_casi) AS casi_totali FROM
   {table_name} {where_regioni} GROUP BY denominazione_regione

```

Query 2

Nella seconda query, si calcola la media di casi giornalieri per ogni mese

```

1. SELECT DATE_FORMAT(data, 'yyyy-MM') AS mese, AVG(nuovi_positivi) AS
   media_casi_giornalieri FROM {table_name} {where_regioni} GROUP BY
   DATE_FORMAT(data, 'yyyy-MM') ORDER BY mese

```

Query 3

Nella terza query, viene calcolata la percentuale di persone guarite rispetto al numero totale di casi per ogni regione

```

1. SELECT denominazione_regione, (SUM(dimessi_guariti) / SUM(totale_casi)) * 100
   AS percentuale_guariti FROM {table_name} {where_regioni} GROUP BY
   denominazione_regione

```

Query 4

Nella quarta query, viene ricercato quale sia stato il giorno dove è stato effettuato il maggior numero di tamponi

```
1. SELECT data, tamponi FROM {table_name} {where_regioni} ORDER BY tamponi DESC
   LIMIT 1
```

Query 5

Nella quinta query, viene calcolato il numero medio di tamponi eseguiti in ogni mese.

```
1. SELECT DATE_FORMAT(data, 'yyyy-MM') AS mese, AVG(tamponi) AS
   media_tamponi_giornalieri FROM {table_name} {where_regioni} GROUP BY
   DATE_FORMAT(data, 'yyyy-MM') ORDER BY mese
```

Query List

In streamlit, queste sono state riportate in questa lista:

```
44.query_list = [
45.     query1, query2, query3, query4, query5
46.]
```

Esecuzione query e creazione grafici su Dashboard

Iterando su questa lista di query, abbiamo definito e creato gli opportuni grafici tramite la libreria `plotly_express`, assegnando ad ognuno di essi un titolo. Per la query 4, essendo il risultato puramente numerico, non sono stati creati grafici ma ci si è limitati a riportare l'output in una tabella.

Il codice scritto è il seguente:

```
47.i = 0
48.query_title = ["Totali casi per regione", "Media casi giornalieri per mese",
   "Percentuale di guariti per regione", "Giorno con maggior numero di tamponi",
   "Media tamponi giornalieri per mese"]
49.if len(states)>0:
50.     for query in query_list:
51.         # Esegui la query di selezione
52.         with hive_con.cursor() as cursor:
53.             cursor.execute(query)
54.             # Recupera i risultati
55.             results = cursor.fetchall()
56.             df = DataFrame.from_records(results)
57.
58.             st.markdown(f"## Query {i+1}: {query_title[i]}")
59.
```

```

60.         st.write(df)
61.         if i!=3:
62.             # Crea grafico a torta
63.             if len(df.columns) == 2:
64.                 fig = px.pie(df, names=df.columns[0], values=df.columns[1],
65.                             title='Distribuzione')
66.                 st.plotly_chart(fig)
67.             # Crea istogramma
68.             if len(df.columns) == 2:
69.                 fig = px.bar(df, x=df.columns[0], y=df.columns[1],
70.                             title='Andamento')
71.                 st.plotly_chart(fig)
72.                 i = i + 1

```

Dashboard

Possiamo vedere un estratto della dashboard risultante nelle seguenti immagini:

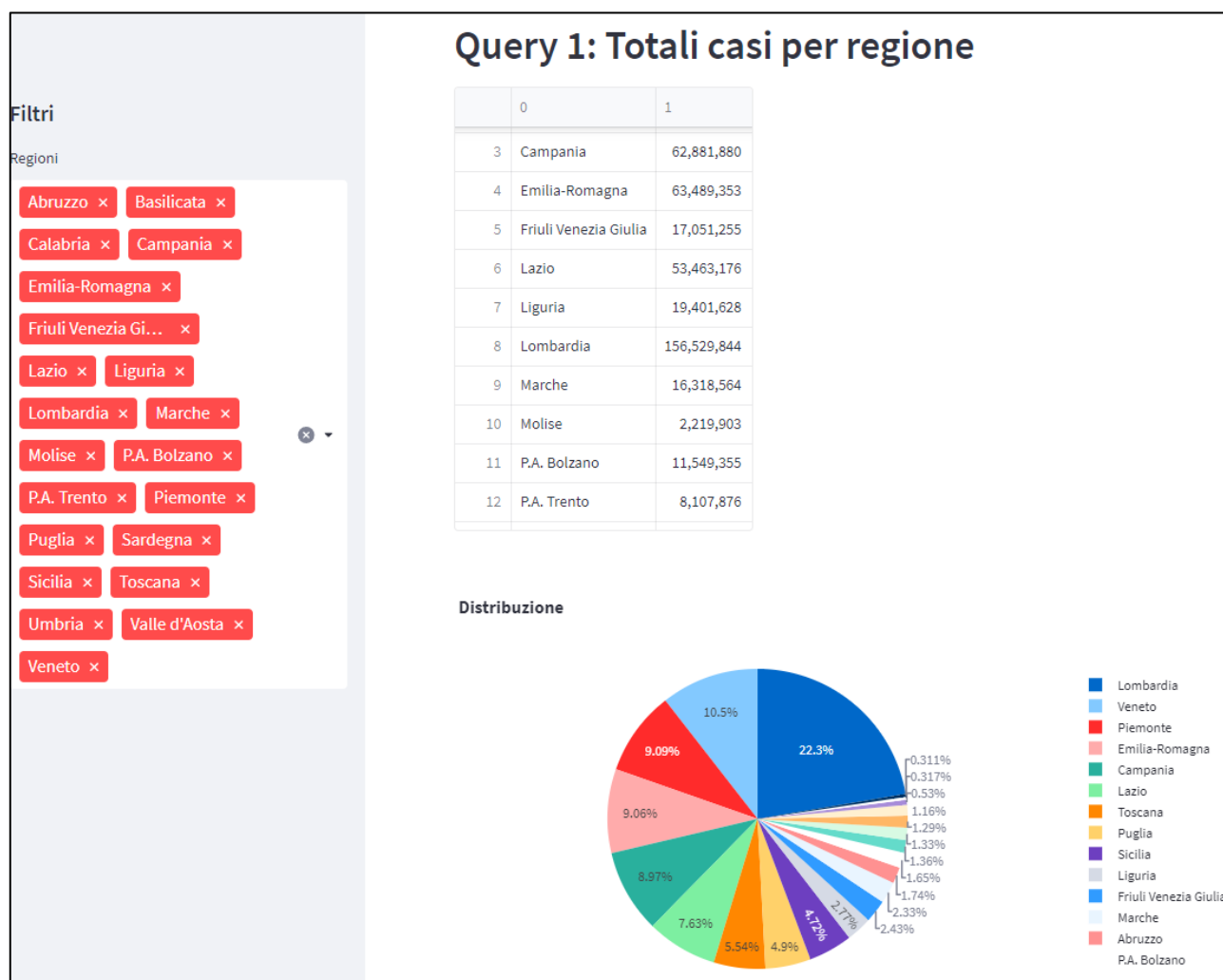


Figura 3 - Estratto della dashboard su Streamlit

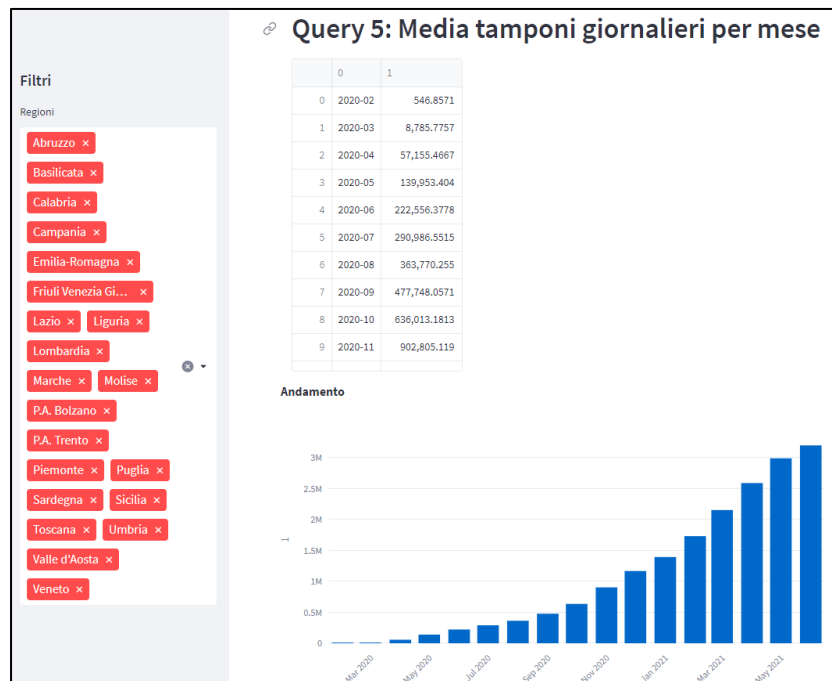


Figura 4 - Estratto della dashboard su Streamlit

La dashboard è completamente interattiva, infatti andando a rimuovere\aggiungere le regioni dal filtro presente nella sidebar, è possibile rimuoverle anche dalle tabelle e dai grafici delle query che verranno quindi rieseguite:

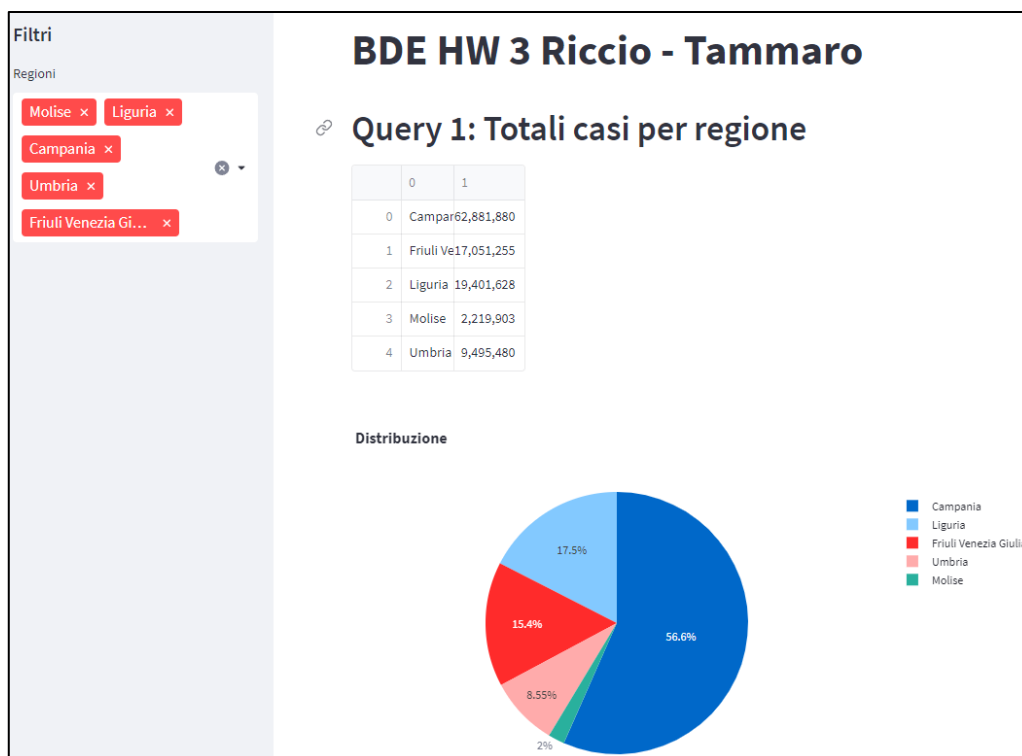


Figura 5 - Estratto della dashboard filtrata

