Laboratory for Data Science

Alma Stira, 658699

Antonia Giovinazzi, 667576

Sommario

1. Introduzione

La prima parte del lavoro per il nostro progetto è relativa alla costruzione di un datawarehouse a partire dal file “computer_sales.csv”, che contiene una panoramica completa delle vendite di computer registrate tra marzo 2013 e aprile 2018. Il file è composto da 3.412.325 righe e da 25 colonne [tabella1], in cui sono riportati non solo dettagli delle transazioni di vendita, ma anche le specifiche tecniche delle configurazioni di CPU, GPU e RAM per ciascun dispositivo venduto, il che permette di avere una visione chiara delle caratteristiche dei prodotti acquistati.

Inoltre, il file correlato, “geography.xml”, arricchisce il dataset aggiungendo informazioni geografiche sulle località associate a ciascuna vendita.

Colonna	Tipo di valore
Time_code Geo_id	INT 64
Ram_sales_currency Cpu_sales_currency Gpu_sales_currency Ram_size Ram_clock Cpu_n_cores Gpu_memory Sales_currency	Float64
Currency Ram_vendor_name Cpu_vendor_name Gpu_vendor_name Ram_brand Cpu_brand Gpu_brand Ram_name Cpu_name Gpu_name Ram_type Cpu_series Cpu_socket Gpu_processor Gpu_processor_manufacturer Gpu_memory_type	Object

Tabella 1: Tipologia dei dati

2. Datawarehouse

2.1 Costruire un datawarehouse

Al fine di creare un nuovo datawarehouse ci siamo concentrate sul capire il significato e i valori associati alle colonne che caratterizzavano il nostro

dataset. L'approccio principale utilizzato per questo compito è stato quello di controllare il set di dati tramite l'utilizzo di *python* e in particolar modo della libreria csv. Durante la parte di comprensione dei dati non sono stati trovati outliers né valori mancanti, tuttavia si è fatta attenzione sui valori 'geizhals_unknown' e 'pricespy_unknown' che erano rispettivamente presenti nelle colonne 'cpu_vendor_name', 'gpu_vendor_name', 'ram_vendor_name'. Così distribuiti:

geizhals_unknown	2.043.095
pricespy_unknown	321.750

Tabella 2 : Valori univoci di unkonown per Ram

geizhals_unknown	1.016.568
pricespy_unknown	78.993

Tabella 3: Valori univoci di unknown per Cpu

geizhals_unknown	98.457
pricespy_unknown	89.090

Tabella 2: Valori univoci di unknown di Gpu

Analizzando il significato dei due valori:

- geizhals_unknown: probabilmente fa riferimento a dati provenienti al sito di comparazione prezzi Geizhals, dove il nome del venditore potrebbe non essere stato disponibile o non riconosciuto al momento della registrazione
- pricespy_unknown: fa riferimento al sito PriceSpy, e indica che l'informazione relativa al venditore non era disponibile o non è stata identificata

di conseguenza dato che nessuno dei due fa effettivamente riferimento ai nomi dei venditori ma a siti di rivenditori si è deciso, per mantenere una coerenza dei dati, di eliminare qualsiasi riga avesse uno dei due valori. A questo punto il nostro dataset si è dimezzato a 1.032.928 record.

Al fine di creare le basi del nostro datawarehouse si è proceduto alla creazione delle colonne: 'year', 'month', 'quarter',' dayofweek', 'day', estrapolate dalla colonna 'time_code', mentre le colonne 'cpu_sales_usd', 'gpu_sales_usd', 'ram_sales_usd' create moltiplicando le rispettive colonne 'cpu_sales_currency', 'gpu_sales_currency' e 'ram_sales_currency' con un tasso di cambio.

```python
def create_time_columns(time_code):
    #usiamo un nuovo formato data
    date_obj = datetime.datetime.strptime(time_code, '%Y%m%d')

    year = date_obj.year
    month = date_obj.month
    day = date_obj.day  # numerico
    week = date_obj.isocalendar()[1]
    quarter = (month - 1) // 3 + 1
    day_of_week = date_obj.strftime("%A")  # Giorno della settimana in nome

    return {
        'year': year,
        'month': month,
        'day': day,
        'week': week,
        'quarter': f"Q{quarter}",
        'day_of_week': day_of_week
    }

# qua per convertire le vendite in USD
def convert_to_usd(amount, currency):
    exchange_rate = 1.1  # impostare il tasso
    if currency == 'USD':
        return float(amount)
    else:
        return float(amount) * exchange_rate

file_path = r'C:\Users\User\LDS\filtered_computer_sales.csv'
output_file_path = r'C:\Users\User\LDS\c_s_time.csv'

new_sales_data = []

with open(file_path, mode='r', newline='', encoding='utf-8') as file:
    csv_reader = csv.DictReader(file)

    # Salva i fieldnames originali (colonne esistenti)
    fieldnames = csv_reader.fieldnames + ['year', 'month', 'day', 'week', 'quarter', 'day_of_week',
                                          'ram_sales_usd', 'cpu_sales_usd', 'gpu_sales_usd', 'total_sales_usd']

    for row in csv_reader:
            time_columns = create_time_columns(row['time_code'])

        # Conversione delle vendite in USD
        ram_sales_usd = convert_to_usd(row['ram_sales_currency'], row['currency'])
        cpu_sales_usd = convert_to_usd(row['cpu_sales_currency'], row['currency'])
        gpu_sales_usd = convert_to_usd(row['gpu_sales_currency'], row['currency'])
        total_sales_usd = convert_to_usd(row['sales_currency'], row['currency'])

        # si agg tutte le informazioni alla riga
        new_row = row.copy()
        new_row.update(time_columns)
        new_row['ram_sales_usd'] = ram_sales_usd
        new_row['cpu_sales_usd'] = cpu_sales_usd
        new_row['gpu_sales_usd'] = gpu_sales_usd
        new_row['total_sales_usd'] = total_sales_usd

        # si agg la nuova riga modificata ai nuovi dati
        new_sales_data.append(new_row)


with open(output_file_path, mode='w', newline='', encoding='utf-8') as output_file:
    writer = csv.DictWriter(output_file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(new_sales_data)

print("Dati processati e salvati con successo in 'c_s_time.csv'!")
```

Figura 1: Codice creazione nuove colonne

Successivamente ci siamo dedicate all'unione del nostro attuale csv con le informazioni geografiche, il che è stato possibile grazie alla colonna 'geo_id' già presente nel nostro file 'computer_sales.csv', di conseguenza abbiamo ottenuto come risultato finale un file csv composto da 1.032.928 record e 38 colonne, che combina i dati di vendita con le informazioni geografiche per ogni record.

I passaggi successivi riguardano la creazione di diversi dizionari per tenere traccia delle combinazioni uniche che stanno ad indicare gli id delle nostre future tabelle. Infine abbiamo utilizzato questi dati per creare una "fact table"

destinata ad essere utilizzata in un database relazionale, che raccoglie informazioni dettagliate sulla vendita di componenti hardware (Cpu, Ram, Gpu) associandole a dimensioni come geografia, tempo e i diversi componenti.

2.1 Popolare un datawarehouse

In questa fase riguarda la popolazione del nostro data warehouse. Per iniziare, su SQL Management Studio abbiamo creato le tabelle 'Geography', 'Ram', 'Gpu', 'Cpu', 'Time' e 'Computer_sales'. Successivamente, utilizzando Python e la libreria *pyodbc*, abbiamo stabilito una connessione al database e proceduto con il popolamento delle varie tabelle con il seguente codice:

```
# Connessione al database
server = 'tcp:lds.di.unipi.it'
username = 'Group_ID_2'
password = 'VGGIZA0H'
database = 'Group_ID_2_DB'
connectionString = 'DRIVER={ODBC Driver 17 for SQL Server};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+password

try:
    cnxn = pyodbc.connect(connectionString)
    cursor = cnxn.cursor()
    print("Connessione effettuata!")

    # Apri il file della Geography
    with open('Geography.csv', 'r') as file:
        reader = csv.reader(file)
        next(reader)

        for row in reader:
            # Preparare i dati da inserire
            geo_id = row[0]
            country = row[1]
            region = row[2]
            continent = row[3]
            currency = row[4]


            # inseriamo i dati nella tabella Geography
            cursor.execute("""
                INSERT INTO Geography (geo_id, country, region, continent, currency)
                VALUES (?, ?, ?, ?, ?)
            """, geo_id, country, region, continent, currency)

    # Commit delle operazioni di inserimento
    cnxn.commit()
    print("Dati inseriti correttamente nella tabella Geography!")

except pyodbc.Error as ex:
    print(f"Errore di connessione al database: {ex}")

finally:
    # Chiudi la connessione al database!!!!!
    cnxn.close()
```

Figura 2: Codice popolazione data warehouse

Completato il processo di popolazione delle tabelle, il nostro data warehouse ha assunto il seguente schema.

Figura 3: Schema data warehouse

3. Costruire un processo ETL

Per la costruzione di un flusso di lavoro ETL abbiamo utilizzato il pacchetto SSIS di Visual Studio. La task richiedeva: “identificare gli ID computer associati alle vendite più elevate di CPU, inoltre, aumentare il risultato includendo la percentuale di vendite w.r.t. alle vendite totali di tutti i computer all'interno della stessa serie di CPU”.

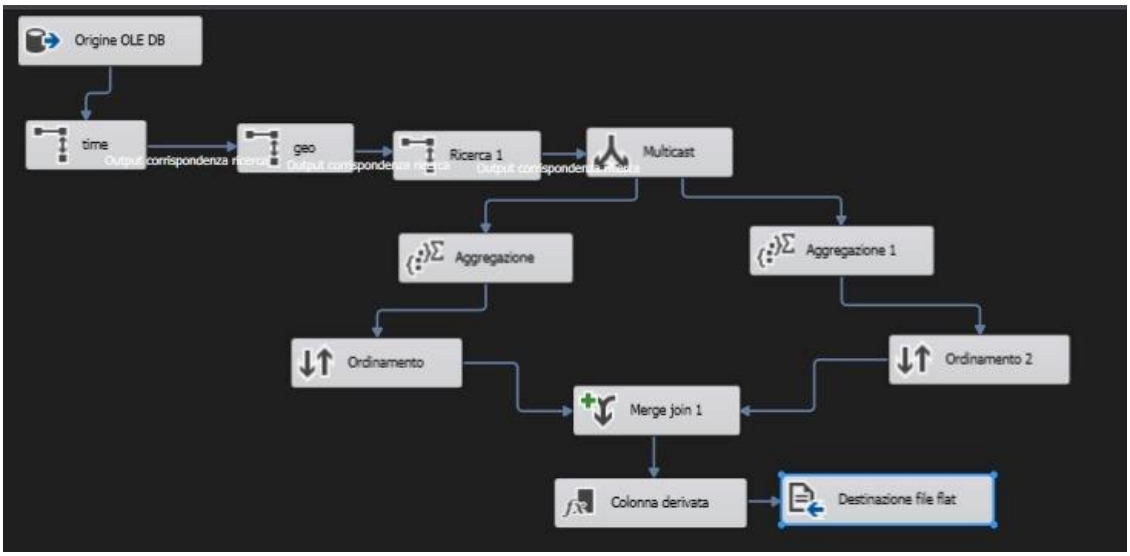

Figura 4: Flusso dati ETL

Come mostrato in figura, il primo passo è stato connettersi al database remoto tramite una fonte OLE DB, che ci ha permesso di accedere alla tabella dei fatti "computer_sales". Utilizzando due ricerche dei nodi, abbiamo mappato i record della fact table con le tuple presenti nelle tabelle "Time", "Geography" e "Cpu".
Grazie all'operatore Multicast, abbiamo eseguito in parallelo due aggregazioni: una per calcolare le vendite totali di CPU, raggruppate per anno e regione, e l'altra per sommare le vendite di computer appartenenti alla stessa serie di CPU.
I due output, una volta ordinati, sono stati processati dall'operatore Merge Join, che ha combinato i dati aggregati, correlando le vendite più alte di CPU per regione e anno con le vendite totali di computer della stessa serie. Infine, con l'utilizzo di Derived Column, è stata creata una nuova colonna per calcolare la percentuale di vendite di ogni CPU rispetto alle vendite totali nella stessa serie.
I risultati finali sono stati salvati in un file flat, che include gli ID dei computer con le vendite di CPU più alte e la percentuale di vendite calcolata.

4. Costruire un Datacube

Per questa task abbiamo utilizzato il pacchetto SSAS di Visual Studio per costruire il nostro data cube. I primi due passaggi fondamentali sono stati definire l'origine dati e la vista dati. Una volta completati questi step, siamo passati alla creazione delle dimensioni, che includono: Cpu, Ram, Gpu, Time e Geography.
In particolare, per le ultime due sono state definite le seguenti gerarchie:

Figura 6: Gerarchia Time

Figura 5: Gerarchia Geography

Nella tabella Time, abbiamo aggiunto una nuova colonna chiamata ‘cat_month’, che associa i nomi dei mesi ai valori numerici della colonna ‘month’, garantendo una corrispondenza precisa tra mese numerico e mese categorico.

Completata la costruzione del cubo, abbiamo poi modificato le proprietà degli attributi ‘ram_sales_currency’, ‘cpu_sales_currency’ e ‘gpu_sales_currency’, impostando l’aggregazione su "somma" e il tipo di dato su "double", in modo da ottimizzare il calcolo delle vendite.

5. Query MDX

Questa task richiedeva la creazione di una query MDX: “mostrare i primi 5 brand di CPU, RAM e GPU in base alle vendite medie mensili, suddivise per ogni regione in Europa”. A causa di alcuni problemi riscontrati durante l'esecuzione, non è stato possibile ottenere un unico output. Pertanto, abbiamo suddiviso le query per questa task come segue:

```mdx
 WITH
-- Calcola le vendite medie mensili per RAM
MEMBER [Measures].[Monthly Avg Ram Sales] AS
    Avg(
        Descendants([Time].[time_Gerarchia].[Year], [Time].[time_Gerarchia].[Cat Month]),
        [Measures].[Ram Sales Currency]
    )

-- Set per le regioni europee
SET [EuropeanRegions] AS
    Descendants([Geography].[geo_Gerarchia].[Continent].&[Europe], [Geography].[geo_Gerarchia].[Region])

-- Top 5 marchi RAM per ciascuna regione specifica
SET [TopRamBrandsPerRegion] AS
    NONEMPTY(
        Generate(
            [EuropeanRegions],
            TopCount(
                NONEMPTY(
                    [Ram].[Ram Brand].Members,
                    ([Measures].[Monthly Avg Ram Sales], [Geography].[geo_Gerarchia].CurrentMember)
                ),
                6,
                [Measures].[Monthly Avg Ram Sales]
            )
        )
    )

SELECT
-- Righe: Regioni europee con i Top 5 marchi RAM locali
NONEMPTY(
    Generate(
        [EuropeanRegions],
        TopCount(
            NONEMPTY(
                CrossJoin(
                    [Ram].[Ram Brand].Members,
                    [Geography].[geo_Gerarchia].CurrentMember
                ),
                [Measures].[Monthly Avg Ram Sales]
            ),
            6,
            [Measures].[Monthly Avg Ram Sales]
        )
    )
) ON ROWS,

-- Colonne: Vendite medie mensili per i Top 5 marchi RAM locali
[Measures].[Monthly Avg Ram Sales] ON COLUMNS

FROM
   [Cubo_giovinazzi_stira]
```

Figura 7: Codice Query MDX per Ram

		Monthly Avg Ram Sales
All	analucia	112400.4275
G.SKILL	analucia	46930.8525
CORSAIR	analucia	33460.3375
KINGSTON	analucia	17984.6379166667
CRUCIAL	analucia	4706.74
TEAM GROUP	analucia	2842.7925
All	aragon	40612.0815384615
G.SKILL	aragon	21316.8784615385
CORSAIR	aragon	10008.9407692308
KINGSTON	aragon	7898.00384615384
CRUCIAL	aragon	1312.153
MUSHKIN	aragon	943.5
All	asturleon	37721.7984615385
G.SKILL	asturleon	19861.7853846154
CORSAIR	asturleon	8857.21153846154
KINGSTON	asturleon	7426.83769230769
CRUCIAL	asturleon	1331.20416666667
MUSHKIN	asturleon	1062.16333333333

Figura 8: Output query per Ram

		Monthly Avg Cpu Sales
All	analucia	1295324.9925
INTEL	analucia	1286456.3475
AMD	analucia	21284.748
All	aragon	71549.75
INTEL	aragon	59918.1369230769
AMD	aragon	11631.6130769231
All	asturleon	88455.49
INTEL	asturleon	80125.7876923077
AMD	asturleon	8329.70230769231
All	baden-wuttemberg	1341167.5488
INTEL	baden-wuttemberg	1305699.3832
AMD	baden-wuttemberg	40304.7336363636

Figura 10: Output Query per Cpu

```mdx
   WITH
MEMBER [Measures].[Monthly Avg Cpu Sales] AS
    Avg(
        Descendants([Time].[time_Gerarchia].[Year], [Time].[time_Gerarchia].[Cat Month]),
        [Measures].[Cpu Sales Currency]
    )

-- Set per le regioni europee
SET [EuropeanRegions] AS
    Descendants([Geography].[geo_Gerarchia].[Continent].&[Europe], [Geography].[geo_Gerarchia].[Region])

SET [TopRamBrandsPerRegion] AS
    NONEMPTY(
        Generate(
            [EuropeanRegions],
            TopCount(
                NONEMPTY(
                    [Cpu].[Cpu Brand].Members,
                    ([Measures].[Monthly Avg Cpu Sales], [Geography].[geo_Gerarchia].CurrentMember)
                ),
                6,
                [Measures].[Monthly Avg Cpu Sales]
            )
        )
    )

SELECT

NONEMPTY(
    Generate(
        [EuropeanRegions],
        TopCount(
            NONEMPTY(
                CrossJoin(
                    [Cpu].[Cpu Brand].Members,
                    [Geography].[geo_Gerarchia].CurrentMember
                ),
                [Measures].[Monthly Avg Cpu Sales]
            ),
            6,
            [Measures].[Monthly Avg Cpu Sales]
        )
    )
) ON ROWS,

[Measures].[Monthly Avg Cpu Sales] ON COLUMNS

FROM
   [Cubo_giovinazzi_stira]
```

Figura 9: Codice Query MDX per Cpu

```
    WITH

MEMBER [Measures].[Monthly Avg Gpu Sales] AS
    Avg(
        Descendants([Time].[time_Gerarchia].[Year], [Time].[time_Gerarchia].[Cat Month]),
        [Measures].[Gpu Sales Currency]
    )

-- Set per le regioni europee
SET [EuropeanRegions] AS
    Descendants([Geography].[geo_Gerarchia].[Continent].&[Europe], [Geography].[geo_Gerarchia].[Region])


SET [TopRamBrandsPerRegion] AS
    NONEMPTY(
        Generate(
            [EuropeanRegions],
            TopCount(
                NONEMPTY(
                    [Cpu].[Cpu Brand].Members,
                    ([Measures].[Monthly Avg Gpu Sales], [Geography].[geo_Gerarchia].CurrentMember)
                ),
                6,
                [Measures].[Monthly Avg Gpu Sales]
            )
        )
    )

SELECT

NONEMPTY(
    Generate(
        [EuropeanRegions],
        TopCount(
            NONEMPTY(
                CrossJoin(
                    [Gpu].[Gpu Brand].Members,
                    [Geography].[geo_Gerarchia].CurrentMember
                ),
                [Measures].[Monthly Avg Gpu Sales]
            ),
            6,
            [Measures].[Monthly Avg Gpu Sales]
        )
    )
) ON ROWS,

[Measures].[Monthly Avg Gpu Sales] ON COLUMNS

FROM
   [Cubo_giovinazzi_stira]
```

Figura 11: Codice Query MDX per Gpu

		Monthly Avg Gpu Sales
All	analucia	625009.0025
PNY	analucia	460501.067083333
Gigabyte	analucia	42200.9295833333
EVGA	analucia	29701.8083333333
MSI	analucia	29140.175
Aorus	analucia	27201.96625
All	aragon	94248.0130769231
Gigabyte	aragon	24024.0733333333
PNY	aragon	23082.823
Asus	aragon	18254.1063636364
MSI	aragon	17177.909
Aorus	aragon	12895.075
All	asturleon	93931.4461538462
Gigabyte	asturleon	29341.8825
Aorus	asturleon	23896.031
PNY	asturleon	22157.2566666667
Asus	asturleon	18400.0216666667
MSI	asturleon	13225.8522222222

Figura 12: Output query per Gpu

6. Creare un dashboard

L'obiettivo di questa visualizzazione è fornire un'analisi comparativa delle vendite di computer in diverse aree geografiche nel corso di tre anni consecutivi, mostrando sia la distribuzione geografica delle vendite sia il cambiamento delle vendite in valuta locale tra le varie regioni. Questo consente di identificare tendenze e cambiamenti nelle vendite in relazione al tempo e alla posizione geografica.

Un primo dato interessante è che le vendite sono significativamente più alte nel 2016 e nel 2017, mentre nel 2018 si riducono quasi della metà. In questo intervallo di tempo, l'Europa registra un numero notevole di vendite, seguita da America e Oceania. In particolare, la regione con il maggior volume di vendite nel 2016 e nel 2017 è "Prairie Provinces" nel Canada Occidentale, mentre nel 2018 la leadership passa a "Baden-Württemberg" in Germania.

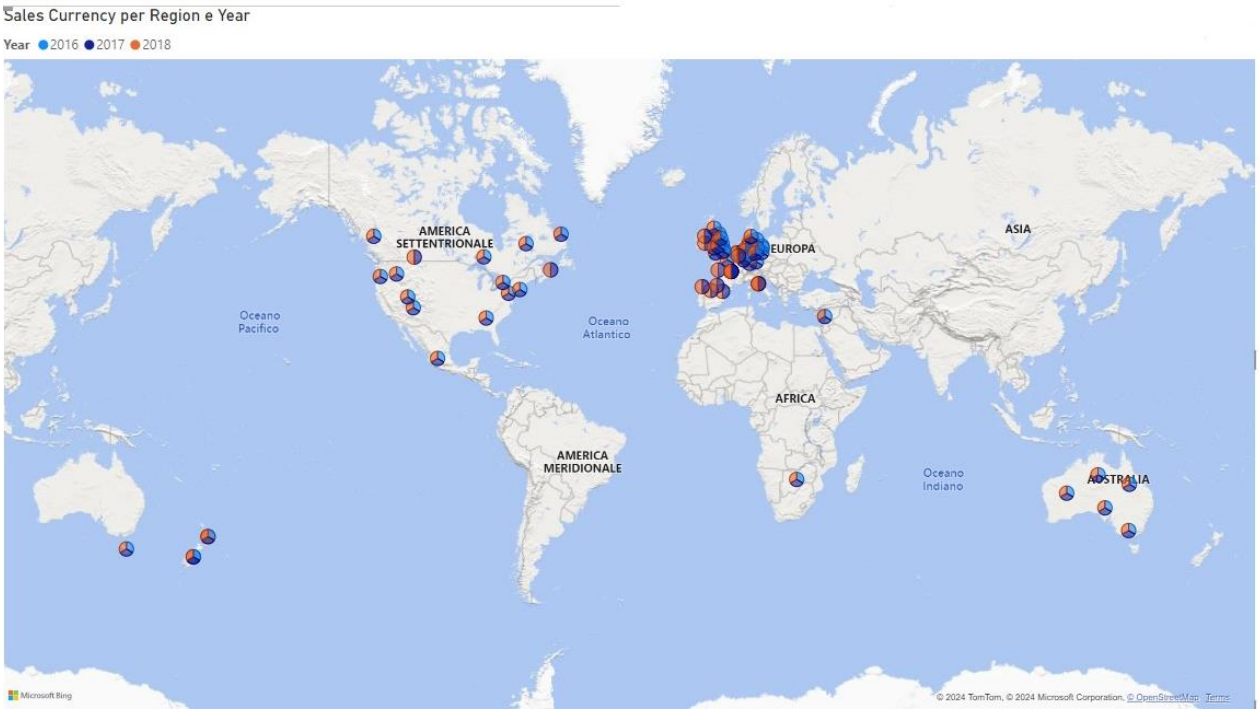

Figura 13: un'analisi comparativa delle vendite di computer in diverse aree geografiche nel corso di tre anni

Region	2016	2017	2018
analucia	19.233.325,68	26.217.914,14	3.334.465,60
aragon		1.911.997,90	771.335,30
asturleon		2.115.943,55	745.472,05
atlantic provinces		419.969,13	2.398.203,27
baden-wuttemberg	20.401.792,42	25.448.744,40	10.009.486,08
bavaria	19.063.674,63	26.655.614,81	3.662.013,24
berlin	18.887.812,12	25.379.668,18	3.830.209,15
brandenburg	20.372.939,37	24.975.204,36	2.786.081,28
bremen	19.698.685,89	25.060.622,74	3.187.366,60
british columbia	21.594.376,14	27.274.299,11	8.032.713,37
brussels	5.602.561,57	4.347.275,36	1.736.841,07
castilla		2.030.857,55	850.818,96
center italy		1.477.152,78	2.618.260,04
connaught		4.562.009,14	1.141.873,88
east england	11.380.909,83	11.933.773,19	4.970.446,56
east midlands	9.034.501,25	11.210.236,19	3.483.068,78
flanders		279.355,80	1.010.543,94
galicia		1.999.740,34	737.948,52
hamburg	19.111.858,98	25.273.869,15	3.976.342,19
heart of france		1.827.845,58	787.323,15
hessen	18.775.966,76	25.699.322,42	3.598.987,84
le midi france		2.578.538,04	647.649,78
leinster		1.285.254,42	2.940.513,37
london	7.715.529,58	12.944.301,05	3.444.457,86
lower saxony	19.118.152,12	25.947.152,24	3.135.505,18
mecklenburg-vorpommern	19.429.952,02	27.065.197,80	2.756.929,90
mid-atalntic	12.444.207,02	17.165.875,21	5.202.838,51
mid-west usa	13.527.900,42	15.091.822,44	6.085.365,96

Tabella 3: Output analisi comparativa