Università della Calabria

Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistemistica

Corso di Laurea Magistrale in Ingegneria Informatica

Elaborato Finale di Modelli e Tecniche Per Big Data

“Recensioni di Hotel”

Monea Stefano Francesco
***Matricola**: 256942*
Prisco Ivan
***Matricola**:252320*

Prof. Trunfio Paolo

Prof. Marozzo Fabrizio

Anno Accademico 2023 / 2024

1. Analisi Del Dataset

Il dataset oggetto di studio rappresenta un vasto insieme di dati ottenuti mediante scraping dal sito www.Booking.com, comprendente 515 mila recensioni di clienti e relative valutazioni per un totale di 1493 hotel di lusso situati in Europa. Tutte le informazioni presenti nel file sono già di dominio pubblico, con l'originale proprietà dei dati attribuita a Booking.

Oltre alle dettagliate recensioni e valutazioni, il dataset offre un contesto informativo completo. Tra le informazioni incluse vi sono l'indirizzo degli hotel, la data delle recensioni, il punteggio medio calcolato sulla base degli ultimi commenti nell'anno precedente, il nome dell'hotel, la nazionalità del revisore, le recensioni negative e positive con il conteggio delle parole, il punteggio assegnato dal revisore basato sulla propria esperienza, il numero totale di recensioni precedentemente fornite dal revisore, il totale delle recensioni valide dell'hotel, le etichette assegnate dal revisore, la durata in giorni tra la data della recensione e la data di estrazione, ed infine le coordinate geografiche (comprendenti latitudine e longitudine) dell'hotel.

Dettagliatamente il file in questione (circa 233 MB), dati strutturati in formato CSV (Comma-Separated Values), è costituito da precisamente 515738 record con 17 features di seguito riportati:

Feature	Significato
Hotel_Address	Indirizzo dell'Hotel.
Review_Date	Data in cui il revisore ha pubblicato la recensione corrispondente.
Avarage_Score	Punteggio medio dell'Hotel, calcolato in base all'ultimo commento nell'ultimo anno.
Hotel_Name	Nome dell'Hotel.
Reviewer_Nationality	Nazionalità del revisore.
Negative_Review	Recensione negativa che il revisore ha dato all'Hotel.
Review_Total_Negative_Word_Counts	Numero totale di parole nella recensione negativa.
Positive_Review	Recensione positiva che il revisore ha dato all'hotel.
Review_Total_Positive_Word_Counts	Numero totale di parole nella recensione positiva.
Reviewer_Score	Punteggio che il revisore ha assegnato all'hotel, basato sulla sua esperienza.
Total_Number_of_Reviews_Reviewer_Has_Given	Numero di recensioni che il revisore ha dato in passato.
Total_Number_of_Reviews	Numero totale di recensioni dell'hotel.
Tags	Etichette che il revisore ha dato all'hotel.
days_since_review	Durata tra la data della recensione e la data di estrazione del dato.
Additional_Number_of_Scoring	Ci sono anche alcuni ospiti che hanno semplicemente assegnato un punteggio al servizio anziché una recensione. Questo numero indica il punteggio aggiuntivo assegnato dal revisore.
lat	Latitudine dell'hotel.
lng	Longitudine dell'hotel.

Qualora il revisore non dovesse rilasciare una recensione negativa, nella colonna targata “Negative_Review” comparirà un valore pari a “No Negative”. Per quanto riguarda invece “Positive_Review”, in assenza di una recensione comparirà un valore pari a “No Positive”.

Durante l’analisi dei dati, si è riscontrato che una parte di record presentava dati mancanti, in particolare:

- 523 istanze non prevedevano la presenza della nazionalità del recensore;
- 3268 istanze non prevedevano le coordinate geografiche (latitudine e longitudine) dell’hotel.

Si è scelto subito di rimuovere le prime 523 istanze danneggiate in quanto impossibilitati a ottenere la nazionalità del recensore. Mentre per le restanti 3268 si è cercato di verificare se nel dataset fossero presenti delle recensioni dello stesso hotel nella quale fossero presenti le coordinate geografiche, in modo tale da recuperare le informazioni mancanti, giungendo alla conclusione che tutte le istanze dello stesso hotel, non prevedevano le coordinate geografiche comportandone una conseguente eliminazione.

Eseguendo queste righe di codice otteniamo delle informazioni relative al dataset:

```
if __name__ == "__main__":
    spark = SparkBuilder("appProva")
    dataset = spark.dataset.toPandas()
    print(dataset.info())
```

```
Data columns (total 21 columns):
 #   Column                                      Non-Null Count   Dtype
---  ------                                      --------------   -----
 0   Hotel_Address                               515738 non-null  object
 1   Additional_Number_of_Scoring                515738 non-null  int32
 2   Review_Date                                 515738 non-null  object
 3   Average_Score                               515738 non-null  float32
 4   Hotel_Name                                  515738 non-null  object
 5   Reviewer_Nationality                        515738 non-null  object
 6   Negative_Review                             515738 non-null  object
 7   Review_Total_Negative_Word_Counts           515738 non-null  int32
 8   Total_Number_of_Reviews                     515738 non-null  int32
 9   Positive_Review                             515738 non-null  object
 10  Review_Total_Positive_Word_Counts           515738 non-null  int32
 11  Total_Number_of_Reviews_Reviewer_Has_Given  515738 non-null  int32
 12  Reviewer_Score                              515738 non-null  float32
 13  Tags                                        515738 non-null  object
 14  days_since_review                           515738 non-null  int32
 15  lat                                         512470 non-null  float32
 16  lng                                         512470 non-null  float32
 17  Country_Hotel                               515738 non-null  object
 18  City_Hotel                                  515738 non-null  object
 19  Review_Year                                 515738 non-null  int32
 20  Review_Month                                515738 non-null  int32
```

Figura 1:"Descrizione del dataset pre-operazioni preliminari"

Possiamo subito notare che in corrispondenza delle colonne “lat” e “lng” sono presenti dei valori mancanti in quanto contano 512470 istanze rispetto alle 515738 delle restanti colonne. Inoltre, eseguendo le seguenti righe di codice:

```
if __name__ == "__main__":
    spark = SparkBuilder("appProva")
    dataset = spark.dataset.filter(col("Reviewer_Nationality") == " ")
    dataset.show()
```

```
+--------------------+---------------------------+-----------+-------------+--------------------+--------------------+--------------------+-------------------------------+
|       Hotel_Address|Additional_Number_of_Scoring|Review_Date|Average_Score|          Hotel_Name|Reviewer_Nationality|     Negative_Review|Review_Total_Negative_Word_Counts|
+--------------------+---------------------------+-----------+-------------+--------------------+--------------------+--------------------+-------------------------------+
|1 3 Queens Garden...|                       1058| 2017-03-19|          7.7|The Park Grand Lo...|                    |         No Negative|                              0|
|1 3 Queens Garden...|                       1058| 2016-11-10|          7.7|The Park Grand Lo...|                    | The room was inc...|                             15|
|1 3 Queens Garden...|                       1058| 2016-07-12|          7.7|The Park Grand Lo...|                    | Room size Servic...|                              6|
|1 Addington Stree...|                       1322| 2016-11-12|          8.4|Park Plaza County...|                    | Breakfast could ...|                              7|
|1 Inverness Terra...|                       1274| 2016-11-17|          7.7|Grand Royale Lond...|                    | Room service was...|                             13|
|1 Inverness Terra...|                       1274| 2016-10-24|          7.7|Grand Royale Lond...|                    | Rooms are very o...|                             44|
|1 Inverness Terra...|                       1274| 2016-10-22|          7.7|Grand Royale Lond...|                    | The account of m...|                             36|
|1 Inverness Terra...|                       1274| 2017-02-17|          7.7|Grand Royale Lond...|                    | WI FII NOT WORKE...|                              7|
|1 Inverness Terra...|                       1274| 2017-01-27|          7.7|Grand Royale Lond...|                    | No free breakfas...|                              7|
|1 Inverness Terra...|                       1274| 2016-10-14|          7.7|Grand Royale Lond...|                    | Room quite small...|                             14|
|1 Rue Du G n ral ...|                         89| 2017-06-14|          8.2| Gardette Park Hotel|                    | I called down fo...|                            110|
|1 Shortlands Hamm...|                        704| 2017-06-18|          8.3| Novotel London West|                    | The booking was ...|                            129|
|1 Shortlands Hamm...|                        704| 2017-02-27|          8.3| Novotel London West|                    | Late night early...|                             17|
|1 Shortlands Hamm...|                        704| 2017-02-12|          8.3| Novotel London West|                    |         No Negative|                              0|
|1 Shortlands Hamm...|                        704| 2017-02-11|          8.3| Novotel London West|                    | The Quality of t...|                              7|
|10 Berners Street...|                         85| 2016-12-25|          9.1|  The London EDITION|                    | Consirege he rec...|                             47|
|10 Carlisle Stree...|                        329| 2016-11-11|          9.0|     The Nadler Soho|                    |            The bed |                              4|
|100 110 Euston Ro...|                        728| 2017-01-10|          8.9|Pullman London St...|                    |         No Negative|                              0|
|100 110 Euston Ro...|                        728| 2016-11-01|          8.9|Pullman London St...|                    | Elevators take t...|                              5|
|100 Queen s Gate ...|                        541| 2017-03-25|          8.0|Doubletree by Hil...|                    | poor organisatio...|                              6|
+--------------------+---------------------------+-----------+-------------+--------------------+--------------------+--------------------+-------------------------------+
only showing top 20 rows
```

Possiamo chiaramente osservare che siano presenti record nella quale manca la nazionalità del recensore.

In seguito a delle operazioni preliminari, che verranno spiegate successivamente, otteniamo il seguente risultato:

```
Data columns (total 21 columns):
 #   Column                                      Non-Null Count   Dtype
---  ------                                      --------------   -----
 0   Hotel_Address                               511950 non-null  object
 1   Additional_Number_of_Scoring                511950 non-null  int32
 2   Review_Date                                 511950 non-null  object
 3   Average_Score                               511950 non-null  float32
 4   Hotel_Name                                  511950 non-null  object
 5   Reviewer_Nationality                        511950 non-null  object
 6   Negative_Review                             511950 non-null  object
 7   Review_Total_Negative_Word_Counts           511950 non-null  int32
 8   Total_Number_of_Reviews                     511950 non-null  int32
 9   Positive_Review                             511950 non-null  object
 10  Review_Total_Positive_Word_Counts           511950 non-null  int32
 11  Total_Number_of_Reviews_Reviewer_Has_Given  511950 non-null  int32
 12  Reviewer_Score                              511950 non-null  float32
 13  Tags                                        511950 non-null  object
 14  days_since_review                           511950 non-null  int32
 15  lat                                         511950 non-null  float32
 16  lng                                         511950 non-null  float32
 17  Country_Hotel                               511950 non-null  object
 18  City_Hotel                                  511950 non-null  object
 19  Review_Year                                 511950 non-null  int32
 20  Review_Month                                511950 non-null  int32
```

Figura 2:"Descrizione del dataset post-operazioni preliminari"

Notiamo che nella figura 2 il numero di istanze per tutte le colonne è lo stesso.

2. Struttura dell'Applicazione

L'intera applicazione è stata realizzata in Python. La scelta di tale linguaggio di programmazione è stata motivata da diverse ragioni, in particolare:

- *Sintassi*: Python è noto possedere una sintassi chiara e intuitiva, che lo rende accessibile anche a coloro che non hanno una vasta esperienza di programmazione;
- *Librerie e Framework*: Python vanta una vasta gamma di librerie specializzate per l'analisi dei dati, come NumPy, SciPy, Scikit-learn e molte altre. Tali librerie forniscono strumenti potenti per l'elaborazione e manipolazione dei dati per ottenere, anche in maniera semplice, dei risultati sofisticati;

Nel contesto dell'analisi dei big data, l'efficace elaborazione e l'analisi di grandi volumi di dati sono spesso cruciali per ottenere informazioni significative e prendere decisioni corrette. In questo scenario, PySpark emerge come una potente soluzione per l'elaborazione distribuita dei dati, consentendo agli sviluppatori di scrivere codice Python ad alte prestazioni per analizzare grandi datataset.

Pyspark è una libreria Python per l'integrazione con Apache Spark, un framework di elaborazione progettato per l'analisi di dati su larga scala. Difatti uno dei suoi vantaggi è l'elaborazione dei dati in memoria centrale, in quanto il dataset viene caricato in RAM in strutture quali Dataframe, RDD o Dataset, strutture dati resilienti sulla quale vengono eseguite varie interrogazioni. Utilizzare la RAM, a patto di averne abbastanza, fa sì che si può effettuare una sola lettura dei dati e successivamente fare qualsiasi tipo di interrogazione direttamente in memoria centrale.

Quanto detto in precedenza è anche uno dei motivi principali per la quale si è scelto Apache Spark come modello per l'elaborazione dei dati, in quanto se avessimo usato un framework come Hadoop, basato su un approccio MapReduce, avrebbe comportato una serie di letture e scritture sul HDFS, con un conseguente elevato costo computazionale.

La problematica principale rimane quella per la visualizzazione dei risultati ottenuti, con la necessità di creare strumenti user-friendly ed intuitivi per la visualizzazione dei dati. In risposta a questa esigenza, si è sfruttato una delle precedenti citate caratteristiche di Python, ossia *Librerie e Framework*. Streamlit si distingue come un framework Python, che mette a disposizione vari componenti per rendere semplificata la realizzazione di un applicazione web per la visualizzazione dei dati.

Streamlit è un framework open-source, progettato per semplificare la creazione di applicazioni web, dando la possibilità agli sviluppatori di creare un frontend senza dover ricorrere all'uso di codice HTML, CSS o JavaScript. In questo modo la priorità degli sviluppatori diventa la logica dell'applicazione e sulla manipolazione dei dati, anziché sulla complessa gestione della parte visiva e grafica.

Di seguito viene riportato un diagramma molto semplificato di quello che rappresenta il backend dell'applicazione:

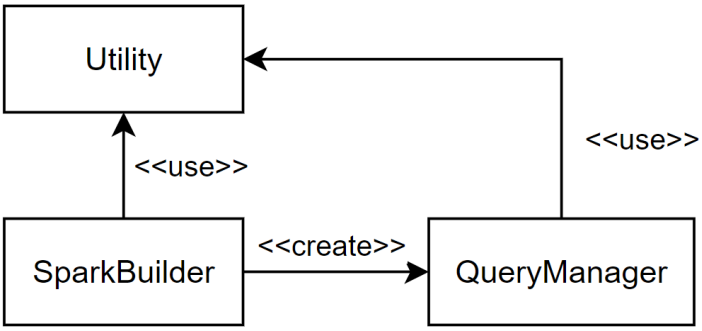


Figura 3: Diagramma rappresentativo backend dell'applicazione

In particolare, avremo tre classi:

- **SparkBuilder:** riveste un ruolo fondamentale in quanto è responsabile di istanziare una sessione Spark, inoltre tra le sue variabili ne è presente una denominata "query", istanza della classe QueryManager;
- **QueryManager:** rappresenta un involucro dove al suo interno sono state definite tutte le query, la quale verranno successivamente approfondite;
- **Utility:** è una classe di utilità che possiede dei metodi generici, usata da entrambi le classi sopra citate.

Nel dettaglio:

```
class SparkBuilder:

    def __init__(self, appname):
        self.spark = (SparkSession.builder.master("local[*]").
                      appName(appname).getOrCreate())
        self.dataset = self.spark.read.csv(PATH_DS, header=True, inferSchema=True)
        self.castDataset()
```

Figura 4:"Costruttore della classe SparkBuilder"

Il costruttore della classe SparkBuilder ha come parametro una stringa "*appname*", che ovviamente, rappresenta il nome dell'applicazione. Successivamente definiamo una variabile d'istanza "*dataset*", la quale caricherà il dataset localizzato nella variabile "*PATH_DS*". Successivamente viene richiamato il metodo castDataset().

Quest'ultimo metodo sostanzialmente va ad effettuare un cast a determinate colonne del dataset, per esempio convertiamo le colonne rappresentati le coordinate geografiche (latitudine e longitudine) in un float.

```
1 usage
def castDataset(self):
    df = self.dataset
    #Effettuaiamo il cast di qualche colonna
    df = df.withColumn( colName: "Additional_Number_of_Scoring", df["Additional_Number_of_Scoring"].cast(IntegerType()))
    df = df.withColumn( colName: "Review_Date", to_date(df["Review_Date"], format: "M/d/yyyy"))
    df = df.withColumn( colName: "Average_Score", df["Average_Score"].cast(FloatType()))
    df = df.withColumn( colName: "Review_Total_Negative_Word_Counts", df["Review_Total_Negative_Word_Counts"] \
                        .cast(IntegerType()))
    df = df.withColumn( colName: "Total_Number_of_Reviews", df["Total_Number_of_Reviews"].cast(IntegerType()))
    df = df.withColumn( colName: "Review_Total_Positive_Word_Counts", df["Review_Total_Positive_Word_Counts"] \
                        .cast(IntegerType()))
    df = df.withColumn( colName: "Total_Number_of_Reviews_Reviewer_Has_Given",
                        df["Total_Number_of_Reviews_Reviewer_Has_Given"].cast(IntegerType()))
    df = df.withColumn( colName: "Reviewer_Score", df["Reviewer_Score"].cast(FloatType()))
    df = df.withColumn( colName: "days_since_review",
                        regexp_extract(col("days_since_review"), pattern: r"(\d+)", idx: 1).cast(IntegerType()))

    df = df.withColumn( colName: "lat", df["lat"].cast(FloatType()))
    df = df.withColumn( colName: "lng", df["lng"].cast(FloatType()))

    #Convertiamo i tag in un array di stringhe
    df = df.withColumn( colName: "Tags", regexp_replace(col("Tags"), pattern: "[\[\]']", replacement: ""))
    df = df.withColumn( colName: "Tags", split(col("Tags"), pattern: ", "))
    df = df.withColumn( colName: "Tags", expr("transform(Tags, x -> trim(x))"))
```

Figura 5:"Prima parte del metodo castDataset"

Dalla figura 6 è possibile osservare l'aggiunta di quattro nuove colonne, scelta che è stata presa per semplificare le query svolte. Rispettivamente:

- "Country_Hotel": che rappresenta la nazionalità dove risiede l'hotel;
- "City_Hotel": che rappresenta la città dove risiede l'hotel;
- "Review_Year": che rappresenta l'anno di pubblicazione della recensione;
- "Review_Month": che rappresenta il mese della pubblicazione della recensione.

```
# Aggiunta di quattro colonne per facilitare le query:
#Country_Hotel rappresenta la nazionalità dell'hotel
df = df.withColumn( colName: "Country_Hotel",
                    regexp_extract(col("Hotel_Address"), pattern: r'(United\s+Kingdom|\b[A-Z][a-z]+)$', idx: 1))

udf_estraiCitta = udf(estraiCitta, StringType())
#City_Hotel rappresenta la città dove è ubicato l'hotel
df = df.withColumn( colName: "City_Hotel", udf_estraiCitta( *args: col("Hotel_Address"), col("Country_Hotel")))
#Review_Year rappresenta l'anno di pubblicazione della recensione
df = df.withColumn( colName: "Review_Year", year(df["Review_Date"]))
#Review_Month rappresenta il mese di pubblicazione della recensione
df = df.withColumn( colName: "Review_Month", month(df["Review_Date"]))
```

Figura 6: "Seconda parte del metodo castDataset()"

Le informazioni relative all'hotel sono state estratte dalla colonna "Hotel_Address" che rappresenta l'indirizzo dell'hotel, per poter estrarre tali informazioni si è fatto uso però, di una funzione UDF, acronimo di User Defined Function.

In PySpark, un UDF è una funzione definita dall'utente, che consente agli sviluppatori di applicare operazioni personalizzate o complesse ai dati di un DataFrame o di una colonna. Gli UDF consentono agli sviluppatori di estendere le funzionalità di PySpark, consentendo loro di eseguire operazioni che non sono disponibili direttamente tramite le funzioni built-in.

Le UDF in PySpark possono essere create utilizzando la funzione *udf()* del modulo *pyspark.sql.functions*, oppure utilizzando un annotazione decorator in Python. Invece per quanto

riguarda le informazioni relative alle date della recensione, sono state estratte dalla rispettiva colonna “Review_Date”.

Inoltre possiamo notare l’uso del metodo “*cache()*”. Esso è stato utilizzato per velocizzare il funzionamento dell'applicazione, in quanto il suo scopo è quello di anticipare il calcolo del contenuto rendendolo disponibile in memoria per gli usi futuri. Ciò va ad alterare il comportamento tipico della struttura caratterizzata da lazy evaluation, principio che prevede che ogni trasformazione operata su un dataframe sia effettivamente computata solo in seguito al richiamo di un'azione, per risparmiare uso di risorse nei momenti in cui non ve ne è necessità.

Infine, sempre nella figura 6, si può osservare che il dataset viene filtrato, questo perché, come detto precedentemente, durante un’attenta analisi del dataset si è riscontrato che determinati record erano danneggiati, comportandone un’eliminazione.

Per quanto riguarda il frontend dell’applicazione, come già accennato, è stato realizzato mediante il framework Streamlit. La ragione principale dell’uso di tale framework è la sua semplicità nella progettazione e realizzazione di applicazione web, anche grazie alla sua compatibilità con diverse librerie e componenti, fornite sempre dallo stesso framework.

Un diagramma molto semplificato dell’applicazione risulta essere dunque:

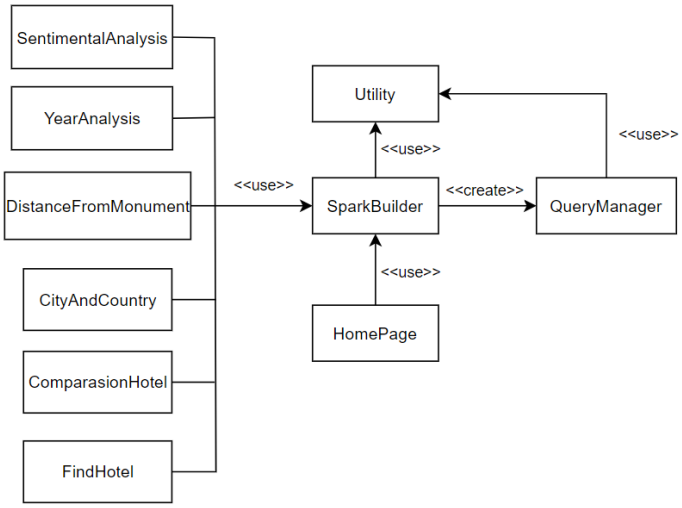


Figura 7:"Diagramma figurativo del backend e frontend"

In particolare, abbiamo che il frontend è costituito rispettivamente da sette pagine, la quale mostrano i risultati dell’uso di query aggregate. La navigazione tra le pagine avviene mediante l’uso di un menu laterale presente a sinistra, nella quale ogni pagina rappresenta un file python differente, questa è anche la motivazione del perché nella figura 7 sono stati realizzati come blocchi a sé stanti.

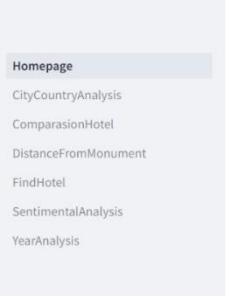


Figura 7:"Menù laterale dell'applicazione"

3. Analisi delle Query

In questa sezione verranno analizzate tutte le query ritenute più rilevanti, sviluppate all'interno della classe "QueryManager".

3.1 Query 1 – Informazioni Degli Hotel (*getCityHotelInformation*)

La seguente query ha il compito di restituire le informazioni degli hotel raggruppati per le città. In particolare, verrà restituito un dataframe contenente per ogni città il numero totale delle recensioni, il numero degli Hotel presenti sul territorio, la valutazione media di tutti gli hotel e la quantità di recensione negative e positive.

```
def cityHotelInformation(self):
    df = self.spark.dataset

    all_info = df.groupby("City_Hotel").agg(
        count("*").alias("Total_Reviews"),
        countDistinct("Hotel_Name").alias("Number_Hotel"),
        avg("Average_Score").alias("Average_Score"),
        sum(when((col("Negative_Review").like("No Negative")) |
(col("Negative_Review").like("Nothing")),
                 0).otherwise(1)).alias("TotalN"),
        sum(when((col("Positive_Review").like("No Positive")) |
(col("Positive_Review").like("Nothing")),
                 0).otherwise(1)).alias("TotalP"),
    )
    return all_info
```

Figura 8:"Codice del metodo getCityHotelInformation"

Le informazioni restituite verranno mostrate mediante l'uso di diagrammi e tabelle, tutti interattivi fornendo la possibilità all'utente l'interazione con essi. In particolare, avremo che:

- Il numero di recensioni totali viene mostrato mediante un diagramma a barre, inoltre ponendo il cursore sopra ogni barra si ottiene il numero specifico associato ad ogni città.

Figura 9:"Grafico a barre query 1"

- Il numero di hotel presenti in ogni città e il punteggio medio sono mostrati rispettivamente mediante un diagramma a torta e un diagramma a barre orizzontale.

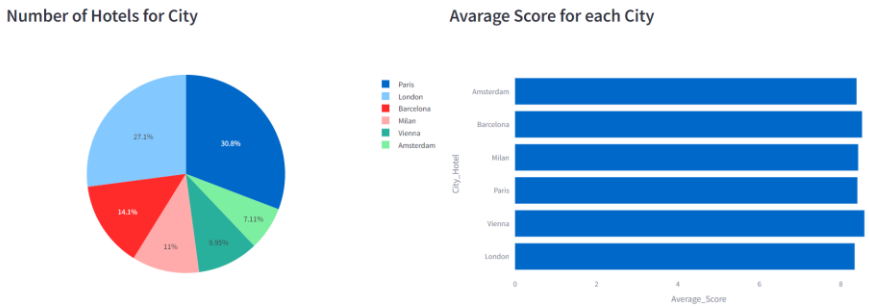


Figura 10:"Diagrammi per i risultati prodotti dalla query 1"

- Il numero totale di recensioni positive e negative invece viene rappresentato mediante un diagramma a barre dove viene usato un colore differente per rappresentare le due quantità.

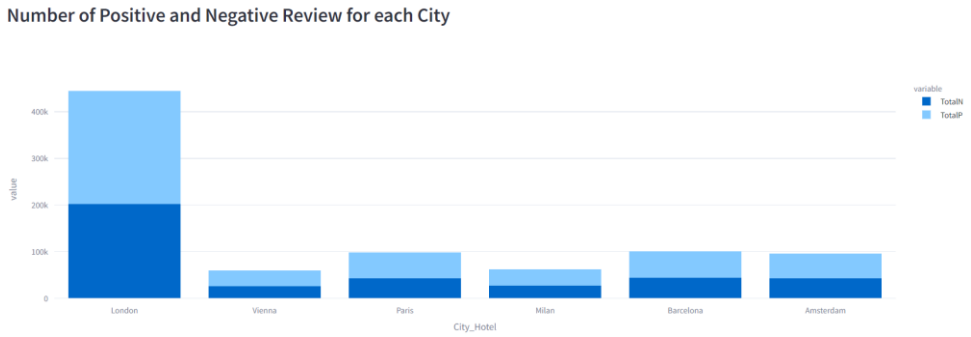


Figura 11:"Ultimo diagramma per i risultati prodotti dalla query 1"

I risultati ottenuti possono essere utilizzati per identificare quale città possiede più hotel, con un punteggio medio più alto e con un numero di recensione positive maggiore.

3.2 Query 2 – Parola più e meno usata per città (*mostLeastUsedWordsByCity*)

La seconda query ha l'obiettivo di restituire la parola e la corrispettiva frequenza della parola più usata per ogni città. In particolare, avremo che il risultato sarà composto da due dataframe differenti, una contenete per ogni città la parola più usata, mentre il restante restituisce per ogni città la parola meno usata.

Come prima azione si va a concatenare le colonne "negative_review" e "positive_review" in un'unica colonna denominata "words", successivamente si effettua un'operazione di "explode" in questo modo otteniamo le singole parole, il tutto per ogni città. Successivamente si va ad effettuare un raggruppamento per ogni città e per la parola, effettuandone un conteggio. Per ottenere il risultato si istanziano due nuovi dataframe, ottenuti effettuando un raggruppamento per città e selezionano rispettivamente la minima e la massima frequenza per ogni parola. Infine, prima di restituirli viene effettuata una inner join, inoltre per quanto riguarda il minimo, durante l'esecuzione della query, si è riscontrato l'esistenza di molte parole con frequenza minima con la conseguenza scelta di selezionare solamente la prima.

```
def mostLeastUsedWordsByCity(self):
    df = self.spark.dataset
    stop_words = set(stopwords.words('english'))

    new_df = df.select(col("City_Hotel"), explode(
        split(lower(concat(col("Negative_Review"), col("Positive_Review"))),
"\s+")).alias("Words"))

    new_df = new_df.filter(~(new_df["Words"].isin(stop_words)) &
(new_df["Words"] != ""))

    word_frequency = new_df.groupBy("City_Hotel",
"Words").agg(count("*").alias("Frequency"))

    maxword = word_frequency.groupBy("City_Hotel").agg(max("Fre-
quency").alias("Frequency"))

    maxword = (maxword.join(word_frequency,["City_Hotel", "Frequency"], how="in-
ner"))

    minword = word_frequency.groupBy("City_Hotel").agg(min("Fre-
quency").alias("Frequency"))

    minword = (minword.join(word_frequency, ["City_Hotel", "Frequency"],
how="inner")
    .groupBy("City_Hotel").agg(
        first("Words").alias("Words"),
        first("Frequency").alias("Frequency")
    ))

    return maxword, minword
```

Figura 12:"Codice query mostLeastUsedWordsByCity"

Per avere una corretta visualizzazione delle parole più usate, si è deciso di rimuovere le stop-word dal dataframe, tale operazione è resa possibile grazie alla libreria NLTK (Natural Language Toolkit), utilizzata per l'analisi simbolica e statistica nel campo dell'elaborazione del linguaggio naturale principalmente in lingua inglese. Entrambi i dataset vengono mostrati sottoforma di tabella, dando quindi la possibilità all'utente di vedere il contenuto dei corrispettivi dataset.

Most Used Word

	City_Hotel	Frequency	Words
0	Amsterdam	33756	room
1	Paris	34687	room
2	London	173180	room
3	Barcelona	32281	room
4	Vienna	20118	room
5	Milan	19523	room

Least Used Word

	City_Hotel	Words	Frequency
0	Amsterdam	bobby	1
1	Barcelona	etiquette	1
2	London	resplendent	1
3	Milan	defeated	1
4	Paris	accolades	1
5	Vienna	exessive	1

Figura 13:"Tabelle per visualizzare i risultati"

Questa query viene usata per capire la tendenza della parola più e meno usata nelle diverse città.

3.3 Query 3 – Statistiche degli Hotel (*hotelStatistic*)

Questa query viene usata per restituire un dataframe contenente per ogni hotel diverse informazioni utili per delle analisi. In particolare, il risultato che restituisce è un dataframe che per ogni hotel contiene rispettivamente: il totale delle recensioni, il numero di recensioni positive e negative, il

valore massimo, il minimo e la media dei voti che sono stati attribuiti dai recensori, la latitudine e la longitudine ed infine la media del punteggio addizionale fornito sempre dai recensori.

```
def hotelStatistics(self):
    df = self.spark.dataset

    res = df.groupBy("Hotel_Name").agg(
        count("*").alias("Total_Reviews"),
        sum(when(col("Positive_Review") != "No Positive",
1).otherwise(0)).alias("Total_Positive_Reviews"),
        sum(when(col("Negative_Review") != "No Negative",
1).otherwise(0)).alias("Total_Negative_Reviews"),
        max("Reviewer_Score").alias("Max_Reviewer_Score"),
        min("Reviewer_Score").alias("Min_Reviewer_Score"),
        avg("Reviewer_Score").alias("Avg_Reviewer_Score"),
        first("lat").alias("Latitude"),
        first("lng").alias("Longitude"),

avg("Additional_Number_of_Scoring").alias("Avg_Additional_Number_of_Scoring")
    )

    return res
```

Figura 14:"Codice della query hotelStatistic"

Il risultato viene ottenuto andando ad aggregare il dataframe per i nomi degli hotel e calcolato tutto quello sopra citato mediante le corrispettive funzioni di count, max, min ecc.

La query viene chiamata da un'altra query denominata *getH1H2statistic(nome_hotel1,nome_hotel2)*, che riceve come parametro il nome di due hotel ed effettua un filtraggio al dataset ottenuto con la query descritta precedentemente. Ciò è motivato dal fatto che nell'applicazione è presente una pagina che dà all'utente la possibilità di confrontare le statistiche di due hotel.

In particolare, abbiamo che le informazioni vengono quindi visualizzati attraverso diverse metriche, ad esempio le informazioni relative ai punteggi medi vengono così visualizzate:

Figura 15:"Visualizzazione dei risultati per la query hotelStatistic"

3.4 Query 4 – Recensione più e meno lunga (*longestShortestReview*)

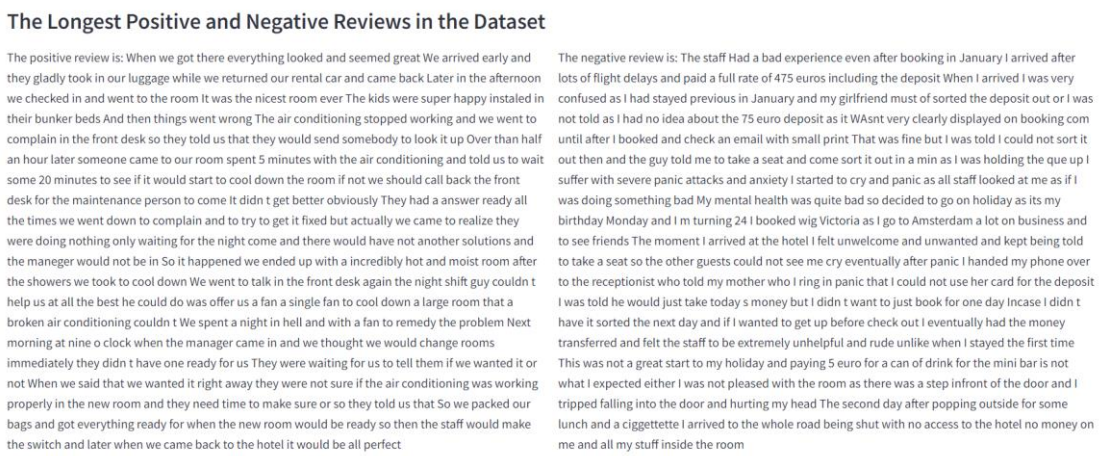
La quarta query ha l'obiettivo di restituire rispettivamente la recensione positiva e negativa più lunga e più corta dell'intero dataset oggetto di studio. In particolare, avremo che la funzione restituisce quattro dataframe contenenti rispettivamente: la recensione positiva più lunga, la recensione negativa più lunga, la recensione positiva più corta e la recensione negativa più corta.

Questo risultato è reso possibile sostanzialmente grazie alle colonne “Review_Total_Negative_Word_Counts” e “Review_Total_Positive_Word_Counts”, colonne che contengono il numero di parole delle recensioni positive e negative, in quanto si è confrontato le recensioni su quest’ultime due colonne sopra citate.

```
def longestShortestReviews(self):
    df = self.spark.dataset

    max_positive_count = df.selectExpr("MAX(Review_Total_Positive_Word_Counts)
as Max_Positive_Count") \
        .collect()[0]["Max_Positive_Count"]
    max_reviews_positive = df.filter((col("Review_Total_Positive_Word_Counts")
== max_positive_count)) \
        .select("Positive_Review", "Review_Total_Positive_Word_Counts")

    max_negative_count = df.selectExpr("MAX(Review_Total_Negative_Word_Counts)
as Max_Negative_Count") \
        .collect()[0]["Max_Negative_Count"]
    max_negative_review = df.filter(col("Review_Total_Negative_Word_Counts") ==
max_negative_count) \
        .select("Negative_Review", "Review_Total_Negative_Word_Counts")

    min_positive_count = df.selectExpr("MIN(Review_Total_Positive_Word_Counts)
as Min_Positive_Count") \
        .collect()[0]["Min_Positive_Count"]
    min_reviews_positive = df.filter((col("Review_Total_Positive_Word_Counts")
== min_positive_count)) \
        .select("Positive_Review", "Review_Total_Positive_Word_Counts")

    min_negative_count = df.selectExpr("MIN(Review_Total_Negative_Word_Counts)
as Min_Negative_Count") \
        .collect()[0]["Min_Negative_Count"]
    min_negative_review = df.filter(col("Review_Total_Negative_Word_Counts") ==
min_negative_count) \
        .select("Negative_Review", "Review_Total_Negative_Word_Counts")

    return max_reviews_positive, max_negative_review, min_reviews_positive,
min_negative_review
```

Figura 16:"Codice della query longestShortestReview"

Il risultato viene mostrato mediante quattro colonne, dove le prime due possiederanno le recensioni più lunghe, mentre le restanti due rappresenteranno le recensioni più brevi.



The Longest Positive and Negative Reviews in the Dataset

The positive review is: When we got there everything looked and seemed great We arrived early and they gladly took in our luggage while we returned our rental car and came back Later in the afternoon we checked in and went to the room It was the nicest room ever The kids were super happy instaled in their bunker beds And then things went wrong The air conditioning stopped working and we went to complain in the front desk so they told us that they would send somebody to look it up Over than half an hour later someone came to our room spent 5 minutes with the air conditioning and told us to wait some 20 minutes to see if it would start to cool down the room if not we should call back the front desk for the maintenance person to come It didn t get better obviously They had a answer ready all the times we went down to complain and to try to get it fixed but actually we came to realize they were doing nothing only waiting for the night come and there would have not another solutions and the maneger would not be in So it happened we ended up with a incredibly hot and moist room after the showers we took to cool down We went to talk in the front desk again the night shift guy couldn t help us at all the best he could do was offer us a fan a single fan to cool down a large room that a broken air conditioning couldn t We spent a night in hell and with a fan to remedy the problem Next morning at nine o clock when the manager came in and we thought we would change rooms immediately they didn t have one ready for us They were waiting for us to tell them if we wanted it or not When we said that we wanted it right away they were not sure if the air conditioning was working properly in the new room and they need time to make sure or so they told us that So we packed our bags and got everything ready for when the new room would be ready so then the staff would make the switch and later when we came back to the hotel it would be all perfect

The negative review is: The staff Had a bad experience even after booking in January I arrived after lots of flight delays and paid a full rate of 475 euros including the deposit When I arrived I was very confused as I had stayed previous in January and my girlfriend must of sorted the deposit out or I was not told as I had no idea about the 75 euro deposit as it WAsnt very clearly displayed on booking com until after I booked and check an email with small print That was fine but I was told I could not sort it out then and the guy told me to take a seat and come sort it out in a min as I was holding the que up I suffer with severe panic attacks and anxiety I started to cry and panic as all staff looked at me as if I was doing something bad My mental health was quite bad so decided to go on holiday as its my birthday Monday and I m turning 24 I booked wig Victoria as I go to Amsterdam a lot on business and to see friends The moment I arrived at the hotel I felt unwelcome and unwanted and kept being told to take a seat so the other guests could not see me cry eventually after panic I handed my phone over to the receptionist who told my mother who I ring in panic that I could not use her card for the deposit I was told he would just take today s money but I didn t want to just book for one day Incase I didn t have it sorted the next day and if I wanted to get up before check out I eventually had the money transferred and felt the staff to be extremely unhelpful and rude unlike when I stayed the first time This was not a great start to my holiday and paying 5 euro for a can of drink for the mini bar is not what I expected either I was not pleased with the room as there was a step infront of the door and I tripped falling into the door and hurting my head The second day after popping outside for some lunch and a ciggettette I arrived to the whole road being shut with no access to the hotel no money on me and all my stuff inside the room

Figura 17:"Visualizzazione dei risultati della query longestShortestReview"

3.5 Query 5 – Tag più e meno usati (*mostAndLeastTagUsed*)

La quinta query ha l'obiettivo di restituire un dataframe contenente per ogni tag presente nel dataset originale la frequenza di utilizzo, in questo modo riordinandolo in maniera decrescente in base alla frequenza avremo ottenuto i tag più/meno usati.

```
def mostAndLeastTagUsed(self):
    df = self.spark.dataset

    df_tags = df.select(explode("Tags").alias("word"))

    frequenza_tag = df_tags.groupBy("word").count()

    frequenza_tag = frequenza_tag.orderBy("count", ascending=False)

    return frequenza_tag
```

Figura 18:"Codice della query mostAndLeastTagUsed"

Il risultato è reso possibile grazie ad aver trasformato nelle operazioni preliminari, il tipo della colonna "Tags" in un array di stringhe, in questo modo mediante la funzione build-in di Pyspark, "explode", otteniamo i singoli tags potendo salvare ogni singolo tag nella colonna denominata "word". A questo punto quello che si è fatto successivamente non è stato altro che un word-count, ossia per ogni tag ne è stata calcolata la frequenza, disponendo un ordine decrescente all'interno del dataset risultate.

Per la visualizzazione in questo caso si è fatto uso di un diagramma delle parole, di seguito riportato:

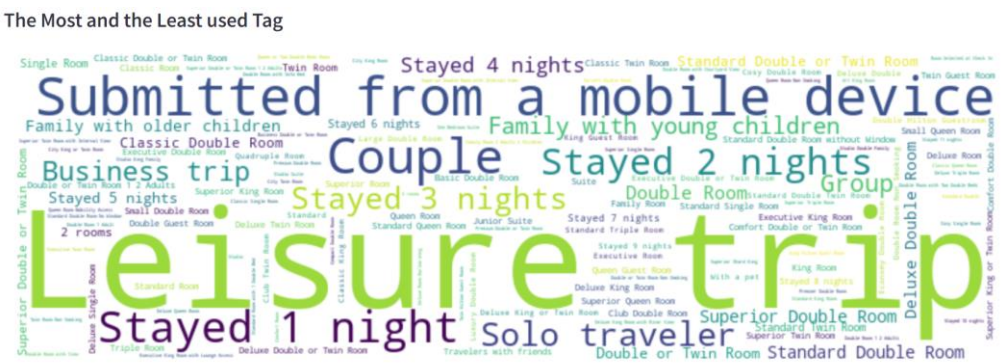


Figura 19:"Visualizzazione del risultato della query mostAndLeastTagUsed"

Le parole che presentano un font più grande rappresentano i tag più frequenti, man mano che la dimensione del font diminuisce, la frequenza tende a diminuire. La query viene utilizzata per capire intuitivamente quale tag è stato il più usato.

3.6 Query 6 – L'ultima recensione postata per un hotel (*getWhenLastReviewWasPostedOfHotel*)

La query seguente si pone l'obiettivo di restituire quando è stata pubblicata l'ultima recensione negativa e positiva. In particolare, per ogni hotel viene restituita qual è l'ultima recensione positiva e negativa, chiaramente se ci dovessero essere più di una recensione, si considera di prendere la prima. Il risultato è stato possibile grazie all'uso della colonna "days_since_review", in quanto grazie alle operazioni preliminari è stato trasformato da un valore per esempio pari a "30 days", in un intero "30", in questo modo ha semplificato l'esecuzione di tale query.

```
def getWhenLastReviewWasPostedOfHotel(self):
    df = self.spark.dataset

    m_df =
df.groupBy("Hotel_Name").agg(min(col("days_since_review")).alias("min_days_since
_review"))
    m_df = m_df.withColumnRenamed("Hotel_Name", "Name")

    joined_df = df.join(m_df, (df["Hotel_Name"] == m_df["Name"]) & (
            df["days_since_review"] == m_df["min_days_since_review"]), "inner")

    reviews = joined_df.select("Hotel_Name", "days_since_review",
"Positive_Review", "Negative_Review").orderBy(
        "days_since_review")

    recensioni = reviews.groupBy("Hotel_Name").agg(
        {"days_since_review": "first", "Positive_Review": "first",
"Negative_Review": "first"})

    return recensioni
```

Figura 20:"Codice dell'interrogazione getWhenLastReviewWasPostedOfHotel"

La query appena descritta viene richiamata dentro una query denominata "*getLastPositiveNegativeReviews(hotel_name1, hotel_name2)",* che riceve come parametro due nomi di hotel. Questo perché tale query viene eseguita in una sezione dell'applicazione dedicata al confronto fra hotel. Essa viene mostrata per intero, suddividendo la recensione positiva e negativa in due colonne differenti, riportando anche quanti giorni fa è stata postata.

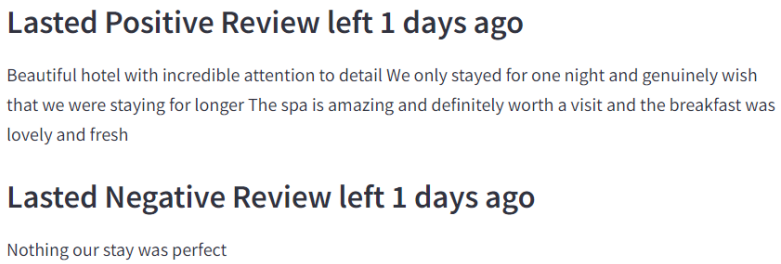


Figura 21:"Visualizzazione della query getWhenLastReviewWasPostedOfHotel"

Il risultato risulta essere utile per capire l''ultima esperienza di una persona negli hotel, quali si sta effettuando il confronto.

3.7 Query 7 – Distribuzione della media delle parole negative e positive per mese e anno (*averageNegativeAndPositiveWordsForMonthAndYear*)

La settima query è progettata per calcolare la media delle parole negative e positive per mese e anno, delle recensioni presenti nel dataset. Nel dettaglio abbiamo che il risultato restituito è costituito da due dataframe i quali rappresentano rispettivamente la media delle parole positive e la media delle parole negative. I due risultati sono ottenuti effettuando un raggruppamento per "Review_Year" e "Review_Month", colonne che rispettivamente indicano l'anno e il mese di pubblicazione della recensione. Il DataFrame viene raggruppato per ogni anno e mese e successivamente, viene utilizzata la funzione avg, ossia la funzione di media, su "Review_Total_Negative_Word_Counts" per calcolare la media delle parole negative all'interno di ciascun gruppo, stessa cosa per "Review_Total_Positive_Word_Counts".

```
def averageNegativeAndPositveWordsForMonthAndYear(self):
    df = self.spark.dataset

    average_negative_words = (df.groupBy("Review_Year", "Review_Month").agg(
        avg("Review_Total_Negative_Word_Counts").alias("Avarage"))
                              .orderBy("Review_Year", "Review_Month"))

    average_positive_words = (df.groupBy("Review_Year", "Review_Month").agg(
        avg("Review_Total_Positive_Word_Counts").alias("Avarage"))
                              .orderBy("Review_Year", "Review_Month"))

    return average_positive_words, average_negative_words
```

Figura 22:"Codice della query averageNegativeAndPositiveWordsForMonthAndYear"

Il risultato viene mostrato mediante un diagramma a linee, un esempio è riportato di seguito:

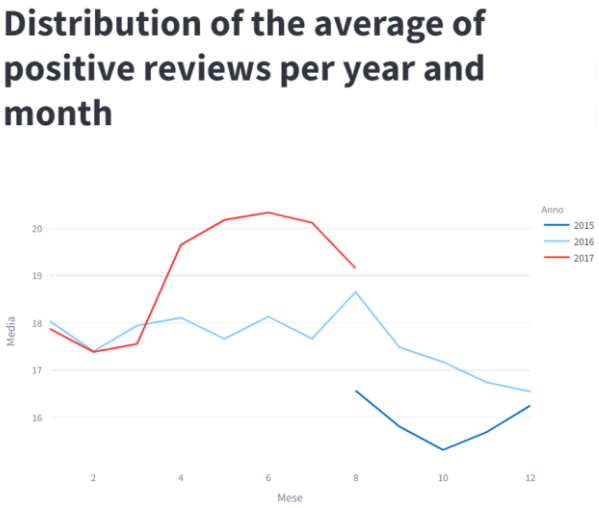


Figura 23:"Diagramma per la query averageNegativeAndPositiveWordsForMonthAndYear"

Questa query è utile per ottenere una panoramica della distribuzione media delle parole negative e positive nelle recensioni nel tempo, suddivisa per mese e anno, in questo modo l'utente può determinare se vi è una tendenza crescente o decrescente nell'uso di più parole positive o negative.

3.8 Query 8 – Parola più e meno usata per anno (*getMostAndLeastUsedWordPerYear*)

Questa funzione denominata getMostAndLeastUsedWordPerYear è progettata, come dice chiaramente il nome stesso, per ottenere la parola più e meno usata per ogni anno presente nel dataset. Nel dettaglio avremo che viene creato un nuovo DataFrame, che include una colonna aggiuntiva chiamata "Full_Review". Questa colonna combina le recensioni negative e positive in una singola stringa utilizzando la funzione concat, separandole con uno spazio utilizzando lit(" "). Per l'estrazione delle parole viene usata la funzione split, in questo modo si divide ogni recensione completa in parole. La funzione lower viene utilizzata per convertire tutte le parole in minuscolo, e quindi la funzione explode viene utilizzata per "splittare" le parole in righe distinte, mantenendo l'associazione con l'anno della recensione.

```
def getMostAndLeastUsedWordPerYear(self):
    df = self.spark.dataset
    stop_words = set(stopwords.words('english'))

    df = df.withColumn("Full_Review", concat(col("Negative_Review"), col("Posi-
tive_Review")))

    m_df = df.select("Review_Year", explode(split(lower("Full_Review"),
"\s+")).alias("Words"))

    m_df = m_df.filter(~(m_df["Words"].isin(stop_words)) & (m_df["Words"] !=
""))

    word_counts_per_year = m_df.groupBy("Review_Year", "Words").count()

    word_most_used_per_year = (word_counts_per_year.orderBy("Re-
view_Year",word_counts_per_year["count"].desc())
                               .groupby("Review_Year").agg({'Words': 'first'}))
    word_least_used_per_year = (word_counts_per_year.orderBy("Review_Year",
word_counts_per_year["count"])
                                .groupby("Review_Year").agg({'Words': 'first'}))

    return word_most_used_per_year, word_least_used_per_year
```

Figura 24:"Codice di getMostAndLeastUsedWordPerYear"

Le parole estratte vengono quindi raggruppate per anno e conteggiate utilizzando il metodo groupBy e count. Questo fornisce il numero di volte che ogni parola appare in ogni anno. I risultati del conteggio delle parole vengono ordinati sia per anno che per conteggio delle parole. Utilizzando la funzione orderBy, viene determinata la parola più utilizzata (word_most_used_per_year) e la meno utilizzata (word_least_used_per_year) per ogni anno. Infine, la funzione restituisce due DataFrame separati, uno contenente le parole più utilizzate per ogni anno e l'altro contenente le parole meno utilizzate per ogni anno.

Dal punto di vista visivo, all'utente viene mostrato mediante una tabella, così come riportato di seguito:

The most used word by year

	Review_Year	first(Words)
0	2015	room
1	2016	room
2	2017	room

The least used word by year

	Review_Year	first(Words)
0	2015	ened
1	2016	moderisation
2	2017	200euros

Figura 25:"Visualizzazione risultati query getMostAndLeastUsedWordPerYear"

Questa query può essere utile per identificare trend nell'uso delle parole nel tempo, fornendo informazioni sul linguaggio e sulle tendenze di comunicazione nei feedback o recensioni.

3.9 Query 9 – Correlazione tra recensioni e stagioni (*getCorrelationBetweenReviewAndSeason*)

La nona query è progettata per calcolare la correlazione tra le recensioni e le stagioni dell'anno. In particolare, avremo che viene creato un nuovo dataframe che include una colonna aggiuntiva chiamata "Season". Questa colonna assegna a ciascuna recensione una stagione dell'anno in base al mese della recensione. Ad esempio, se il mese della recensione è compreso tra marzo e maggio, le

viene assegnata la stagione "Spring". Le recensioni vengono quindi raggruppate per stagione utilizzando il metodo groupBy. Per ogni stagione, vengono calcolati due valori statistici:

- Il numero totale di recensioni (*Total*), calcolato utilizzando la funzione di aggregazione count.
- La valutazione media (*AScore*), calcolata utilizzando la funzione di aggregazione avg sul campo "Average_Score", che rappresenta la media del punteggio del hotel dopo aver ottenuto delle recensioni.

Successivamente i risultati vengono ordinati per stagione in ordine cronologico ed infine viene restituito il dataset risultate.

```
def getCorrelationBetweenReviewAndSeason(self):
    df = self.spark.dataset
    df = df.withColumn("Season",
                       when((df["Review_Month"] >= 3) & (df["Review_Month"] <=
5), "Spring")
                       .when((df["Review_Month"] >= 6) & (df["Review_Month"] <=
8), "Summer")
                       .when((df["Review_Month"] >= 9) & (df["Review_Month"] <=
11), "Autumn")
                       .otherwise("Winter"))

    res = df.groupBy("Season").agg(
        count("*").alias("Total"),
        avg("Average_Score").alias("AScore")
    ).orderBy("Season")

    return res
```

Figura 26:"Codice della query getCorrelationBetweenReviewAndSeason"

All'utente, per poter facilitare anche la comprensione, viene mostrato mediante due grafici a barre, nella quale in uno vengono mostrati il numero di recensioni totali per ogni stagione mentre nel secondo viene mostrato il punteggio medio per ogni stagione.

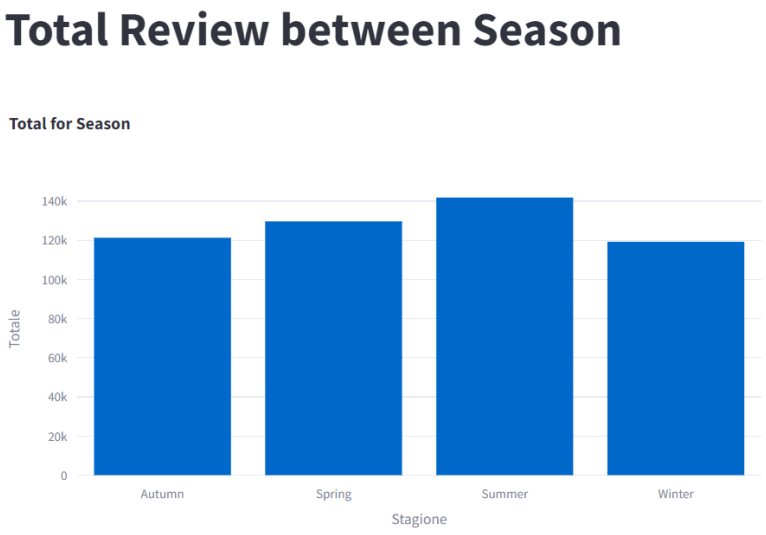


Figura 27:"Visualizzazione parziale dei risultati di getCorrelationBetweenReviewAndSeason"

L'utilità della seguente query è per esaminare se esiste una correlazione tra la stagione dell'anno e le recensioni degli utenti. Ad esempio, potrebbe rivelare se le recensioni sono più positive in determinate stagioni rispetto ad altre, o se il numero di recensioni varia significativamente tra le stagioni.

3.10 Query 10 - I top N tag per Hotel (*getFirstNTagMorePopularofHotel*)

La decima query è stata progettata per ottenere i primi N tag più popolari per un hotel specifico. Il numero dei tag a cui si è interessati ed il nome dell'hotel, vengono passati come parametri. Essa ricalca un po' la logica della quinta query (Tag più e meno usati - *mostAndLeastTagUsed*).

```
def getFirstNTagMorePopular(self, hotel_name, n):
    df = self.spark.dataset

    hotel_df = df.filter(col("Hotel_Name") == hotel_name)

    exploded_tags = hotel_df.select(explode(col("Tags")).alias("Tag"))

    tag_count = exploded_tags.groupBy("Tag").agg(count("*").alias("Tag_Count"))

    top_n_tags = tag_count.orderBy(col("Tag_Count").desc()).limit(n)\
        if n >= tag_count.count() else
tag_count.orderBy(col("Tag_Count").desc())

    return top_n_tags
```

Figura 28:"Codice di getFirstNTagMorePopularofHotel"

In particolare, avremo che il dataframe viene filtrato per il nome dell'hotel, in questo modo nel dataframe risultate otteniamo solamente le istanze corrispondenti all'hotel d'interesse. La colonna dei tag viene "espansa" utilizzando la funzione explode. Questo significa che, se un hotel ha più tag associati, ogni tag verrà posto in una riga separata, mantenendo l'associazione con l'hotel. I tag espansi vengono quindi raggruppati per tag univoci e il conteggio di ciascun tag viene calcolato utilizzando la funzione groupBy e count.

I risultati del conteggio dei tag vengono ordinati in ordine decrescente in base alla frequenza di occorrenza utilizzando il metodo orderBy. Viene selezionato il numero desiderato di tag più popolari, limitato da *limit(n)* se il numero richiesto (ossia *n*) è minore o uguale al numero totale di tag distinti. In caso contrario, vengono restituiti tutti i tag distinti.

All'utente viene mostrato mediante una tabella, contenente il nome del tag con la corrispettiva frequenza, in questo modo si ha una chiara visione del risultato.

Top 5 Tags

	Tag	Tag_Count
0	Solo traveler	49
1	Leisure trip	40
2	Submitted from a mobile device	38
3	Double Room	31
4	Single Room	31

Figura 29:"Visualizzazione del risultato della query getFirstNTagMorePopularofHotel"

Tale interrogazione può essere utilizzata per identificare i tag più comuni associati a un hotel. Queste informazioni possono essere utilizzate per comprendere meglio le caratteristiche o le esperienze più menzionate relative a un hotel specifico.

3.11 Query 11 – Hotel con la più e meno alta differenza tra recensioni positive e negative (*hotelsWithMaxMinReviewDifference*)

L'interrogazione nominata *hotelsWithMaxMinReviewDifference* è progettata per trovare gli hotel con la massima e minima differenza tra il numero di recensioni positive e negative all'interno del dataset oggetto di studio. Per ottenere tale calcolo della differenza, si è effettuato un raggruppamento per il nome dell'hotel e vengono calcolate rispettivamente due quantità:

1. La somma delle recensioni positive, contando il numero di righe dove la colonna "Positive_Review" non è uguale a "No Positive";
2. La somma delle recensioni negative, contando il numero di righe dove la colonna "Negative_Review" non è uguale a "No Negative".

Il valore assoluto di tale differenza rappresenta la differenza tra il numero di recensioni positive e negative per ogni hotel.

```
def hotelsWithMaxMinReviewDifference(self):
    df = self.spark.dataset

    review_difference = df.groupBy("Hotel_Name").agg(
        abs(
            sum(when(col("Positive_Review") != "No Positive", 1).otherwise(0))
            -
            sum(when(col("Negative_Review") != "No Negative",
1).otherwise(0))).alias("Review_Difference")
    )

    max_difference =
review_difference.orderBy(col("Review_Difference").desc()).first()
    min_difference =
review_difference.orderBy(col("Review_Difference").asc()).first()

    return max_difference, min_difference
```

Figura 30:"Codice dell'interrogazione hotelsWithMaxMinReviewDifference"

Utilizzando il metodo orderBy e first, vengono identificati l'hotel con la massima e minima differenza di recensioni. Questo viene fatto ordinando il DataFrame risultante per la colonna "Review_Difference" in ordine decrescente per trovare la massima differenza e in ordine crescente per trovare la minima differenza.

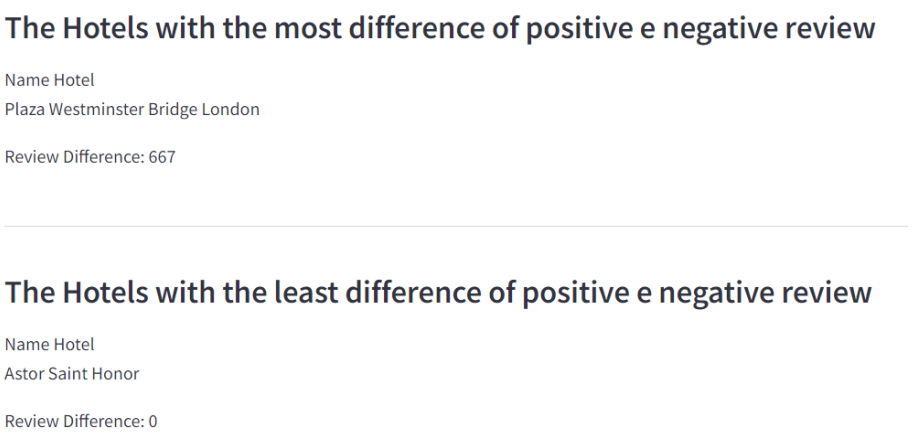


Figura 31:"Visualizzazione dei risultati della query hotelsWithMaxMinReviewDifference"

Tale interrogazione può essere utilizzata per identificare gli hotel che hanno generato tante opinioni contrastanti tra i recensori, evidenziando potenziali aree di miglioramento, di forza o di svantaggio per ciascuna struttura.

3.12 Query 12 - Punteggio medio per una data parola (*averageRatingByKeyword*)

La penultima query è stata realizzata per calcolare la media delle valutazioni per le recensioni che contengono una parola chiave, specificata all'interno delle recensioni. Per realizzare tale interrogazione si è innanzitutto filtrato il dataframe, nella quale vengono selezionate solo le recensioni che contengono la parola chiave specificata. Questo viene fatto cercando la presenza della parola chiave sia nella colonna "Negative_Review" che nella colonna "Positive_Review".

Per le recensioni che contengono la parola chiave, viene calcolata la media dei corrispettivi hotel utilizzando la funzione di aggregazione avg su "Average_Score". Infine, viene restituito il dataframe contenente la media degli hotel per le recensioni che contengono la parola chiave.

```
def averageRatingByKeyword(self, keyword):
    df = self.spark.dataset

    keyword_reviews = df.filter(
        (col("Negative_Review").contains(keyword)) |
(col("Positive_Review").contains(keyword)))

    if keyword_reviews.count() == 0:
        return None, None

    avg_rating_by_keyword =
keyword_reviews.agg(avg("Average_Score").alias("Average_Rating"))

    hotel_w_max_score_review =
keyword_reviews.orderBy(col("Average_Score").desc()).limit(1)

    return avg_rating_by_keyword, hotel_w_max_score_review.collect()
```

Figura 32:"Codice della query averageRatingByKeyword"

Questa funzione può essere utilizzata per valutare la percezione degli utenti su un determinato argomento o tema, identificato dalla parola chiave specificata. Ad esempio, se la parola chiave è "pulizia", la funzione calcolerà la media degli hotel, tale per cui alcune delle recensioni hanno menzionano la pulizia dell'hotel.

Quello che viene riportato di seguito è quello che viene mostrato all'utente, in particolare avrà uno spazio a disposizione per digitare la keyword specifica e successivamente viene restituito il punteggio medio degli hotel che possiedono quella parola nelle recensioni ed inoltre restituisce anche l'hotel con punteggio più alto.

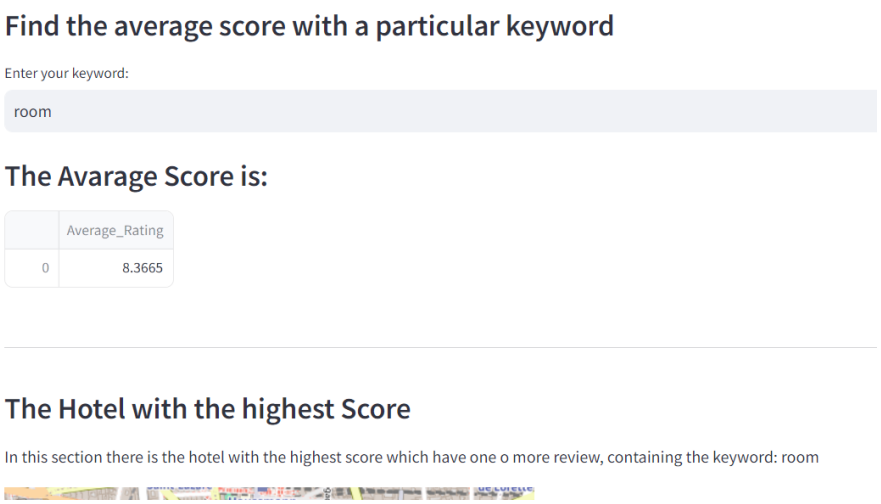


Figura 33:"Visualizzazione parziale del risultato della query averageRatingByKeyword"

3.13 Query 13 – Distanza fra Hotel (*hotelsWithinDistance*)

L'ultima query si pone l'obiettivo di trovare gli hotel che si trovano entro una determinata distanza (in chilometri) da una posizione specificata, utilizzando le coordinate di latitudine e longitudine.

Viene utilizzata una funzione esterna *haversine* per calcolare la distanza in chilometri tra la posizione fornita (latitudine e longitudine) e le coordinate di latitudine e longitudine di ogni hotel nel dataset. Questo calcolo viene eseguito per ogni riga del DataFrame utilizzando la funzione withColumn, e i risultati vengono memorizzati nella nuova colonna "distance_km".

```
def hotelsWithinDistance(self, latitude, longitude, distance_km):
    df = self.hotelStatistics()

    hotel_stats = self.hotelStatistics()

    haversine_udf = udf(haversine, FloatType())

    latitude_col = lit(latitude).cast(FloatType())
    longitude_col = lit(longitude).cast(FloatType())

    hotel_stats = hotel_stats.withColumn("distance_km",
        haversine_udf(latitude_col, longitude_col, col("Latitude"),
col("Longitude")))

    hotels_within_distance = (hotel_stats.filter(col("distance_km") <=
distance_km)
                              .select("Hotel_Name", "Latitude",
"Longitude","distance_km"))

    return hotels_within_distance
```

Figura 34:"Codice della query hotelsWithinDistance"

Utilizzando il metodo filter, vengono selezionati solo gli hotel che si trovano entro la distanza specificata (in chilometri) dalla posizione fornita. Questo viene fatto confrontando il valore della colonna "distance_km" con la distanza specificata. Vengono quindi selezionate solo le colonne "Hotel_Name", "lat" e "lng" per mantenere solo le informazioni essenziali sugli hotel che soddisfano il criterio di distanza.

Per il calcolo della distanza viene utilizzata la formula di Haversine, che permette di determinare la distanza sulla circonferenza massima tra due punti su una sfera, dati i loro valori di longitudine e latitudine.

```
def haversine(lat1, lon1, lat2, lon2):
    R = 6371.0

    lat1 = math.radians(lat1)
    lon1 = math.radians(lon1)
    lat2 = math.radians(lat2)
    lon2 = math.radians(lon2)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    # Formula di Haversine
    a = math.sin(dlat / 2) ** 2 + math.cos(lat1) * math.cos(lat2) *
math.sin(dlon / 2) ** 2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    distance = R * c

    return distance
```

Figura 35:"Codice della funzione haversine"

Questa funzione può essere utilizzata per identificare gli hotel che si trovano entro una determinata distanza da una posizione specifica, ad esempio, dal luogo di interesse di un utente o da un punto di riferimento.

4. Query di supporto

Per quanto riguarda questa parte invece vengono descritte tutte le query di supporto, con la finalità di ottenere un risultato molto velocemente, utilizzate spesso anche nelle interrogazioni più importanti sopra citate.

4.1 QueryS1 – Lista delle nazioni ove è ubicato un hotel (*getCountryHotel*)

L'obiettivo è quello di ottenere una lista di tutte le nazioni ove è situato un hotel presente nel dataset oggetto di studio.

```
def getCountryHotel(self):
    return self.spark.dataset.
select(col("Country_Hotel").alias("Country_Hotel")).distinct().collect()
```

Figura 36:"Codice della query getCountryHotel"

4.2 QueryS2 – Lista delle città ove è ubicato un hotel (*getCityHotel*)

La seconda query di supporto è stata realizzata per ottenere la città ove è situato un hotel.

```
def getCityHotel(self):
    return self.spark.dataset.
select(col("City_Hotel").alias("City_Hotel")).distinct().collect()
```

Figura 37:"Codice della query getCityHotel"

4.3 QueryS3 – Lista degli Hotel per nazione (*getHotelsbyCountry*)

L'obiettivo della terza query è quella di restituire tutti gli hotel presenti in una nazione passata come input.

```
def getHotelsByCountry(self, country_name):
    return self.spark.dataset.
filter(col("Country_Hotel").like(country_name)).select(col("Hotel_Name"))
.distinct().collect()
```

Figura 38:"Codice della query getHotelsbyCountry"

4.4 QueryS4 – Lista degli Hotel per città (*getHotelsByCity*)

Stessa finalità della query precedente ma usata in questo caso con la città.

```
def getHotelsByCity(self, cityname):
    return self.spark.dataset.filter(col("City_Hotel").like(cityname)).select(
        col("Hotel_Name")).distinct().collect()
```

Figura 39:"Codice della query getHotelsByCity"

4.5 QueryS5 – Coordinate geografiche per ogni hotel (*getlatlong*)

L'obiettivo di tale query è quello di ottenere le coordinate geografiche (latitudine e longitudine) di ogni hotel presente nel dataset oggetto di studio.

```
def getlatlong(self):
    return self.spark.dataset.groupBy("Hotel_Name").agg({"lng": "first", "lat":
"first"})
```

Figura 40:"Codice della query getlatlong"

4.6 QueryS6 – Numero di hotel differenti (*getNumOfHotel*)

La sesta query di supporto ha il compito di ottenere il numero degli hotel distinti presenti nel dataset.

```
def getNumOfHotel(self):
    return self.spark.dataset.select(col("Hotel_Name")).distinct().count()
```

Figura 41:"Codice della query getNumOfHotel"

4.7 QueryS7 – Nazionalità dei recensori (*getReviewerNationality*)

Tale interrogazione ha la finalità di restituire un dataframe contenente tutte le nazionalità dei vari rencesori, che hanno lasciato almeno una recensione.

```
def getReviewerNationality(self):
    df = self.spark.dataset
    return df.select(col("Reviewer_Nationality")).distinct()
```

Figura 42:"Codice della query getReviewerNationality"

4.8 QueryS8 – Frequenza delle Parole (*wordsFrequency*)

Questa query restituisce un dataframe contenente per ogni parola la corrispettiva frequenza.

```
def wordsFrequency(self):
    df = self.spark.dataset

    words = df.select(
        explode(
            split(
                lower(concat(col("Negative_Review"), lit(" "),
col("Positive_Review"))), "\s+")).alias("word"))

    return words.groupby("word").agg(
        count("*").alias("Frequency")
    )
```

Figura 43:"Codice della query wordsFrequency"

4.9 QueryS9 – Frequenza massima e minima per le parole (*maxMinFrequency*)

Si occupa di restituire due dataframe contenente nel primo le parole più frequenti, nel secondo le parole meno frequenti. Richiama l'interrogazione sopra descritta.

```
def maxMinFrequency(self):
    df = self.wordsFrequency()
    max_word = df.orderBy(col("Frequency").desc()).first()
    min_word = df.orderBy(col("Frequency").asc()).first()
    return max_word, min_word
```

Figura 44:"Codice della query maxMinFrequency"

4.10 QueryS10 – Statistiche di due hotel (*getH1H2statistic*)

La decima query restituisce due dataframe, contenente le informazioni degli hotel, il cui nome viene passato in input.

```
def getH1H2statistic(self, hotelname1, hotelname2):
    df = self.hotelStatistics()
    if hotelname1 == hotelname2:
        h1_stats = df.filter(col("Hotel_Name") == hotelname1).first().asDict()
        return h1_stats,h1_stats
    else:
        h1_stats = df.filter(col("Hotel_Name") == hotelname1).first().asDict()
        h2_stats = df.filter(col("Hotel_Name") == hotelname2).first().asDict()
        return h1_stats, h2_stats
```

Figura 45:"Codice della query getH1H2statistic"

4.11 QueryS11 – Quando è stata postata l'ultima recensione positiva e negativa (*getLastPositiveNegativeReviews*)

La query numero 11 di supporto ha come obiettivo di restituire l'ultima recensione positiva e negativa di due hotel, il cui nome è passato in input. Essa richiama l'interrogazione *getWhenLastReviewWasPostedOfHotel()*, precedentemente spiegata.

```
def getLastPositiveNegativeReviews(self, hotel_name, hotel_name2):
    df = self.getWhenLastReviewWasPostedOfHotel()
    df.rename(columns={'first(days_since_review)': 'DSR'}, inplace=True)
    df.rename(columns={'first(Positive_Review)': 'Positive'}, inplace=True)
    df.rename(columns={'first(Negative_Review)': 'Negative'}, inplace=True)
    return df[df['Hotel_Name'] == hotel_name], df[df['Hotel_Name'] ==
hotel_name2]
```

Figura 46:"Codice della query getLastPositiveNegativeReviews"

4.12 QueryS12 – Totale delle nazionalità dei recensori per hotel (*getNumberOfDifferentReviewerNationality*)

Essa restituisce il totale delle nazioni dei recensori.

```
def getNumberOfDifferentReviewerNationality(self):
    df = self.spark.dataset
    numNationality = df.groupBy("City_Hotel") \

.agg(countDistinct("Reviewer_Nationality").alias("Different_Nationality")) \
        .orderBy("City_Hotel")

    return numNationality
```

Figura 47:"Codice della query getNumberOfDifferentReviewerNationality "

4.13 QueryS13 – Distribuzioni delle valutazioni per anno e mese (*getValutationByYearAMonth*)

```
def getValutationByYearAMonth(self):
    df = self.spark.dataset

    average_score_per_year =
df.groupBy("Review_Year").avg("Average_Score").orderBy("Review_Year")
    average_score_per_month = (df.groupBy("Review_Year",
"Review_Month").avg("Average_Score")
                               .orderBy("Review_Year","Review_Month"))
    return average_score_per_year, average_score_per_month
```

Figura 48:"Codice della query getValutationByYearAMonth"

Ha come obiettivo di restituire le distribuzioni dei punteggi medi per mese e anno.

4.14 QueryS14 – Distribuzione del totale delle recensioni per mese e anno (getTotalReviewByYearAMonth)

L'ultima query di supporto si pone l'obiettivo di restituire due dataframe contenenti le distribuzioni del totale delle recensioni per mese e anno.

```
def getTotalReviewByYearAMonth(self):
    df = self.spark.dataset

    reviews_count_per_year =
df.groupBy("Review_Year").count().orderBy("Review_Year")

    reviews_count_per_month = df.groupBy("Review_Year",
"Review_Month").count().orderBy("Review_Year",

"Review_Month")
    return reviews_count_per_year, reviews_count_per_month
```

Figura 49:"Codice della query getTotalReviewByYearAMonth"

5. Analisi Sentimentale

Per fornire un potenziamento aggiuntivo all'applicazione è stata eseguita un'analisi sentimentale, una tecnica computazionale che mira a determinare il *sentiment* o il tono emotivo associato a un particolare testo, frase o documento. Questa tecnica è ampiamente utilizzata in una varietà di contesti, tra cui il monitoraggio delle opinioni degli utenti sui social media, l'analisi delle recensioni dei clienti, la valutazione della soddisfazione del cliente e molto altro.

Negli ultimi anni, con la crescente disponibilità di dati testuali online e lo sviluppo di algoritmi di apprendimento automatico, l'analisi del sentiment è diventata sempre più importante per le aziende e gli individui che desiderano comprendere meglio le opinioni e i sentimenti del pubblico.

In questo caso l'obiettivo è stato quello di estrarre le prime N feature più importanti (N è un parametro specificato dall'utente) delle recensioni che hanno rappresentato il dataset oggetto di studio.

In particolare una volta inizializzata una istanza della classe SparkBuilder(_) si utilizza un metodo denominato *getDatasetForClassification*, di seguito riportato:

```
def getDatasetForClassification(self):
    df = self.spark.dataset
    review_p = df.select(col("Positive_Review").alias("Review"))
    new_df_p = review_p.withColumn("Sentiment", lit(1))
    new_df_p = new_df_p.filter(~col("Review").like("No Positive"))

    review_n = df.select(col("Negative_Review").alias("Review"))
    new_df_n = review_n.withColumn("Sentiment", lit(0))
    new_df_n = new_df_n.filter(~col("Review").like("No Negative"))

    dataset = new_df_p.union(new_df_n)

    return dataset
```

Figura 50:"Codice della funzione getDatasetForClassification"

L'obiettivo di tale metodo è quello di restituire un unico dataframe costituito da tutte le recensioni correttamente etichettate, avremo che alle recensioni estratte dalla colonna "Positive_Review" è stato associato una colonna contenente il valore unitario 1, corrispondente all'etichetta "Positivo", in maniera duale è stato associato alle recensioni estratte dalla colonna "Negative_Review" una colonna unitaria contenente il valore 0, rappresentante dell'etichetta "Negativo".

Infine, il dataframe risultate sarà dato dall'unione dei due dataframe corrispondenti. Da sottolineare che entrambi i dataframe, uno relativo alle recensioni positive e l'altro alle recensioni negative, sono stati filtrati eliminando le corrispettive righe in cui compariva "No Positive" o "No Negative", questo perché potrebbero falsare il modello di classificazione che verrà introdotto a breve.

Una volta ottenuto il dataframe, costituito da circa 861412 record, pronto per la classificazione, si estraggono in due colonne i rispettivi *feature*, nel nostro caso le recensioni, e la *variabile di target*, nel nostro caso la colonna "sentiment" che contiene un valore compreso tra 0 ed 1.

Utilizzando la libreria scikit-learn si è fatto uso della funzione di split, in questo modo abbiamo suddiviso in nostro dataframe in dataset per l'addestramento e per il test assegnando ad ognuno il 50%.

A questo punto inizializziamo un oggetto della classe *TfidfVectorized()*, questo perché il testo può essere rappresentato mediante una rappresentazione numerica, cosi da usare tale testo come input per i modelli di machine learning. Per ottenere questo risultato usiamo una trasformazione TF-IDF (acronimo di Term Frequency-Inverse Document Frequency). Successivamente è stato creato un oggetto RandomForestClassifier, passandogli in input:

- n_estimator: che indica il numero di alberi decisionali che compongono il RandomForest. Aumentare il numero di alberi può migliorare la capacità di generalizzazione del modello. Tuttavia, aumentare troppo il numero di alberi può aumentare il tempo di addestramento e l'uso della memoria oltre che a comportare un rischio di overfitting.
- Random_state: Viene impostato il seed per il generatore di numeri casuali, impostarlo ad un numero fisso garantisce che il modello addestrato riproduce sempre gli stessi risultati casuali. Questo è utile per scopi di debug e confronto tra modelli. Se non viene specificato, il Random Forest utilizzerà un seed diverso ad ogni esecuzione, producendo risultati casuali diversi.

Il Random Forest è un tipo di algoritmo di apprendimento supervisionato utilizzato per problemi di classificazione e regressione. Il funzionamento è molto semplice, vengono creati diversi alberi decisionali durante il processo di addestramento. Ogni albero viene addestrato su un sottoinsieme casuale dei dati di addestramento e utilizza un sottoinsieme casuale delle feature disponibili. Ogni albero decisionale genera una previsione indipendente dagli altri alberi. Tutte queste decisioni prese dai diversi alberi vengono combinate attraverso un processo di voto o media.

Una volta addestrato il modello sul dataset di addestramento e testato sul dataset di test, viene di seguito riportato il report:

Accuracy		0.9407608304443634	
Classification	***Precision***	***Recall***	***F1-Score***
0	0.92	0.95	0.93
1	0.96	0.93	0.95

I parametri rispettivamente identificano:

- Precision: è definita come il numero di predizioni corrette per una classe diviso per il numero totale di predizioni fatte per quella classe. In altre parole, è la proporzione di predizioni corrette rispetto a tutte le predizioni positive fatte dal modello per una classe specifica. Nel nostro caso la precisione per la classe 1 è 0.96, il che significa che il 96% delle recensioni classificate come positive dal modello erano effettivamente positive.
- Recall: è definito come il numero di predizioni corrette per una classe diviso per il numero totale di istanze appartenenti a quella classe. In altre parole, è la proporzione di predizioni corrette rispetto a tutte le istanze effettivamente positive nella popolazione. Ad esempio, il recall per la classe 0 è 0.95 il che significa che il 95% di tutte le recensioni negative presenti nel dataset sono state correttamente identificate dal modello.
- F1-Score: è la media armonica tra la precisione e il recall. Fornisce un singolo valore che combina precisione e recall in una sola metrica. È particolarmente utile quando si desidera trovare un equilibrio tra precisione e recall.

A questo punto, si è sfruttato il modello addestrato anche per fornire una classificazione di nuove recensioni mai viste prima. Infine, le migliori cinque feature ritenute più importanti dal classificatore sono riportate di seguito:

Feature	***Importance***
location	0.06922660534926725
staff	0.04191059769262169
not	0.022302883674323447
great	0.02079988278918077
and	0.020707596493084797

Si è scelto l'uso del RandomForest per condurre l'analisi sentimentale perché spesso i dati sono rappresentati da un formato non strutturato, come il testo, e tale modello di apprendimento risulta essere molto robusto a dati non strutturati. Inoltre, all'interno dei testi può esserci la presenza di feature non lineari, come per esempio la combinazione di parole o frasi che determinano il sentimento complessivo, anche in queste circostanze il Random Forest rappresenta essere un ottimo candidato. Infine, la tecnica Random Forest è meno incline all'overfitting rispetto ad altri modelli complessi, come le reti neurali profonde. Tutto il processo di addestramento può essere parallelizzato facilmente su più core o cluster, il che lo rende adatto per l'analisi su grandi dataset.

Un esempio visivo della classificazione, sopra citata, all'interno dell'applicazione è riportato di seguito: