



**Università della Calabria**

Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistemistica

---

Corso di Laurea Magistrale in Ingegneria Informatica

## **Relazione progetto Big Data**

Docenti:

Prof. Paolo Trunfio

Prof. Fabrizio Marozzo

Studenti:

**Matteo GRECO,**

**matr. 252238**

**Vincenzo PRESTA,**

**matr. 252290**

---

Anno Accademico 2024/2025

# Sommario

<b>1. Pre-Processing .....</b>	<b>3</b>
<b>1.1 Analisi del Dataset .....</b>	<b>3</b>
<b>1.2 Preparazione Dataset.....</b>	<b>4</b>
1.2.1 Cast colonne .....	4
1.2.2 Pulizia Dataset .....	4
1.2.3 Recupero informazioni geografiche .....	5
<b>2. Struttura del backend.....</b>	<b>7</b>
<b>2.1 SparkBuilder .....</b>	<b>9</b>
<b>2.2 QueryManager .....</b>	<b>9</b>
<b>2.3 RoBERTa_Sentiment.....</b>	<b>10</b>
2.2.2 Pre elaborazione del testo .....	10
2.2.3 Analisi del sentiment delle recensioni .....	12
2.2.3 Predizione del Sentiment con RoBERTa .....	13
2.2.3 Analisi del sentiment per un hotel .....	14
<b>2.4 SeasonSentimentAnalysis.....</b>	<b>15</b>
2.4.1 Costruttore .....	15
2.4.2 Pre-elaborazione del Sentiment .....	15
2.4.3 VADER .....	16
<b>2.5 DeepSeek Summary.....</b>	<b>16</b>
2.5.1 Inizializzazione della classe .....	16
2.5.2 Individuare le recensioni .....	17
2.5.3 Creazione summary.....	17
<b>2.6 utils .....</b>	<b>18</b>
2.6.1 Correzione degli indirizzi e Geocoding .....	18
2.6.2 Individuazione coordinate.....	19
2.6.3 Visualizzazione dei Trend di Sentiment e Recensioni .....	20
2.6.4 Calcolo delle distanze tra hotel.....	21
2.6.5 Identificazione di avverbi e aggettivi .....	22
<b>2.7. Struttura finale dell'applicazione.....</b>	<b>22</b>
<b>3. Analisi delle query.....</b>	<b>23</b>
<b>3.1 Analisi degli aggettivi e degli avverbi .....</b>	<b>23</b>
<b>3.2 Analisi dell'influenza dei tag sui punteggi delle recensioni .....</b>	<b>25</b>
<b>3.3 Analisi della lunghezza recensioni in funzione dello score.....</b>	<b>26</b>
<b>3.4 Analisi della reputazione di un Hotel .....</b>	<b>27</b>
<b>3.5 Seasonal Sentiment Analysis .....</b>	<b>29</b>
<b>3.6 Individuazione delle recensioni anomale .....</b>	<b>30</b>
<b>3.7 Trend Mensile .....</b>	<b>31</b>
<b>3.8 Statistiche generali degli hotel.....</b>	<b>32</b>
<b>3.9 Analisi del sentiment (RoBERTa).....</b>	<b>33</b>
<b>3.10 Creazione del summary delle recensioni di un hotel.....</b>	<b>34</b>
<b>3.11 Ottenerre gli hotel di una città in base ai tag .....</b>	<b>35</b>

<b>3.12 Ricerca degli hotel vicini ad una posizione geografica .....</b>	<b>35</b>
<b>3.13 Confronto andamento mensile degli Hotel .....</b>	<b>37</b>
<b>3.14 Tag più utilizzati .....</b>	<b>38</b>
<b>3.15 Conteggio recensioni per ogni nazione .....</b>	<b>38</b>

# 1. Pre-Processing

## 1.1 Analisi del Dataset

Il dataset in esame contiene 515 mila recensioni di clienti su hotel situati in Europa ottenute dal sito [www.booking.com](http://www.booking.com).

Il dataset, in particolare, contiene 17 colonne riportate nella tabella sottostante insieme alla relativa spiegazione.

Colonna	Spiegazione
Hotel_Address	Indirizzo dell'hotel.
Review_Date	Data in cui il recensore ha postato la recensione.
Average_Score	Punteggio medio dell'hotel, calcolato sulla base dell'ultimo commento dell'ultimo anno.
Hotel_Name	Nome dell'hotel.
Reviewer_Nationality	Nazionalità del recensore.
Negative_Review	Recensione negativa. “No negative” nel caso in cui non sia presente.
Review_Total_Negative_Word_Counts	Numero di parole della recensione negativa.
Positive_Review	Recensione positiva. “No positive” nel caso in cui non sia presente.
Review_Total_Positive_Word_Counts	Numero di parole della recensione positiva.
Reviewer_Score	Punteggio assegnato all'hotel dal recensore.
Total_Number_of_Reviews_Reviewer_Has_Given	Numero di recensioni che il recensore ha dato in passato.
Total_Number_of_Reviews	Numero totale di recensione ricevute dall'hotel.
Tags	Etichette che il recensore ha assegnato all'hotel.
Days_since_review	Numero di giorni tra la pubblicazione della recensione e l'estrazione dei dati.
Additional_Number_of_Scoring	Numero di recensioni valide senza recensione.
lat	Latitudine dell'hotel.
lng	Longitudine dell'hotel.

Dalle analisi preliminari si è notato:

- 523 istanze con valori mancanti nella colonna “Reviewer\_Nationality”, cioè non è presente la nazionalità del recensore.
- 3268 istanze con valori mancati nelle colonne “lat” e “lng”, cioè non sono presenti le coordinate geografiche dell’hotel a cui è stata assegnata la recensione.

## 1.2 Preparazione Dataset

Prima di poter sfruttare il dataset per l'estrazione di informazioni sono state necessarie operazioni di preparazione del dataset riportate nei paragrafi sottostanti.

### 1.2.1 Cast colonne

La prima operazione effettuata è stata il casting delle colonne, in particolare:

- Gli attributi contenenti conteggi sono stati convertiti in interi.
- Gli attributi contenenti medie o punteggi sono stati convertiti in float.
- L’attributo contenente la data è stato convertito nel tipo Data di PySpark.
- L’attributo “tags” è stato convertito in una lista di stringhe.

In seguito a tale operazione:

```
|-- Hotel_Address: string (nullable = true)
|-- Additional_Number_of_Scorings: integer (nullable = true)
|-- Review_Date: date (nullable = true)
|-- Average_Score: float (nullable = true)
|-- Hotel_Name: string (nullable = true)
|-- Reviewer_Nationality: string (nullable = true)
|-- Negative_Review: string (nullable = true)
|-- Review_Total_Negative_Word_Counts: integer (nullable = true)
|-- Total_Number_of_Reviews: integer (nullable = true)
|-- Positive_Review: string (nullable = true)
|-- Review_Total_Positive_Word_Counts: integer (nullable = true)
|-- Total_Number_of_Reviews_Reviewer_Has_Given: integer (nullable = true)
|-- Reviewer_Score: float (nullable = true)
|-- Tags: array (nullable = true)
|   |-- element: string (containsNull = false)
|-- days_since_review: integer (nullable = true)
|-- lat: float (nullable = true)
|-- lng: float (nullable = true)
```

### 1.2.2 Pulizia Dataset

Come riportato in [1.1](#) nel dataset sono presenti 523 istanze in cui la nazionalità del recensore non è presente. Si è deciso di eliminare tali istanze.

```
{'Hotel_Address': 0,
 'Hotel_Name': 0,
 'Negative_Review': 849,
 'Positive_Review': 183,
 'Reviewer_Nationality': 523}
```

L’immagine sopra riportata mostra il conteggio delle righe in cui gli attributi stringhe sono vuoti (ottenuta tramite la funzione `checkStrings()`).

Il fatto che vi sono anche righe in cui “Negative\_Review” e “Positive\_Review” sono vuoti non desta preoccupazione in quanto come riportato nella spiegazione del dataset, vi sono recensioni valide senza recensione testuale.

### 1.2.3 Recupero informazioni geografiche

```
Colonna: Hotel_Address, Non Null: 515215, Mancanti: 0
Colonna: Additional_Number_of_Scoring, Non Null: 515215, Mancanti: 0
Colonna: Review_Date, Non Null: 515215, Mancanti: 0
Colonna: Average_Score, Non Null: 515215, Mancanti: 0
Colonna: Hotel_Name, Non Null: 515215, Mancanti: 0
Colonna: Reviewer_Nationality, Non Null: 515215, Mancanti: 0
Colonna: Negative_Review, Non Null: 515215, Mancanti: 0
Colonna: Review_Total_Negative_Word_Counts, Non Null: 515215, Mancanti: 0
Colonna: Total_Number_of_Reviews, Non Null: 515215, Mancanti: 0
Colonna: Positive_Review, Non Null: 515215, Mancanti: 0
Colonna: Review_Total_Positive_Word_Counts, Non Null: 515215, Mancanti: 0
Colonna: Total_Number_of_Reviews_Reviewer_Has_Given, Non Null: 515215, Mancanti: 0
Colonna: Reviewer_Score, Non Null: 515215, Mancanti: 0
Colonna: Tags, Non Null: 515215, Mancanti: 0
Colonna: days_since_review, Non Null: 515215, Mancanti: 0
Colonna: lat, Non Null: 511950, Mancanti: 3265
Colonna: lng, Non Null: 511950, Mancanti: 3265
```

L'immagine sopra-riportata (output ottenuto attraverso la funzione *contaNulli()*) mostra come preannunciato in [1.1](#) l'assenza per 3265 istanze dei valori lat e long. Dall'immagine, inoltre, si può notare anche l'eliminazione delle 523 istanze precedentemente motivata.

Per tali in primis si è verificata l'eventuale presenza nel dataset di recensioni sullo stesso Hotel in modo da recuperare tali informazioni ma tale verifica non ha portato ad alcun risultato.

Si è deciso quindi di sfruttare le API di Open Street Map. In particolare, si sfrutta l'indirizzo a disposizione per ottenere le coordinate geografiche dell'Hotel.

Prima di sfruttare le API messe a disposizione, però, è stato necessario effettuare un controllo sulla correttezza degli indirizzi, in particolare, vista la rimozione dei caratteri unicode in seguito all'estrazione dei dati (come scritto sulla pagina Kaggle del dataset), alcuni caratteri fondamentali nell'identificazione di un indirizzo sono stati sostituiti con spazi rendendo l'indirizzo inutilizzabile.

Ovviamente tutto ciò è stato possibile grazie al numero ridotto di hotel sprovvisti di coordinate geografiche.

Se non fosse stata effettuata la rimozione dei caratteri unicode sarebbe stato immediato ottenere le coordinate geografiche senza richiedere lavori manuali.

La funzione utilizzata è la seguente:

```
def fillLatLnG(self):
    #Estrazione hotel sprovvisti di lat e lng: in totale sono 17
    df_reqfill = self.dataset.filter(col("lat").isNull() | col("lng").isNull()).select("hotel_address", "lat", "lng").distinct()
    #Definizione UDF
    get_lat_udf = udf(utils.get_lat, FloatType())
    get_lng_udf = udf(utils.get_lng, FloatType())
    #Riempio le 17 righe
    df_reqfill = df_reqfill \
        .withColumn("lat", get_lat_udf(col("Hotel_Address"))) \
        .withColumn("lng", get_lng_udf(col("Hotel_Address")))
    #Rename per comodità (drop successivo più facile)
    df_reqfill = df_reqfill.withColumnRenamed("lat", "lat_f").withColumnRenamed("lng", "lng_f")
    #Merge
    df_merged = self.dataset.join(df_reqfill, on="hotel_address", how="left")
    # Usa coalesce per riempire i valori nulli in "lat" e "lng" di df con quelli di df1
    df_merged = df_merged.withColumn(
        "lat",
        coalesce(self.dataset["lat"], df_reqfill["lat_f"]))
    .withColumn(
        "lng",
        coalesce(self.dataset["lng"], df_reqfill["lng_f"]))
    )
    # rimuovere le colonne duplicate di df1 ("lat" e "lng" di df1)
    ret = df_merged.drop("lat_f", "lng_f")
    return ret
```

Per ridurre il numero di chiamate alle API di Open Street Map è stato creato e riempito un dataframe contenente solo gli hotel per cui mancavano le informazioni. Dopodiché si è effettuata una merge delle informazioni ottenute con il dataframe di partenza. La funzione che richiama le API di Open Street Map è:

```
def get_coordinates_osm(address):
    try:
        address = fix_suspicious_spaces(address)
        url = f"https://nominatim.openstreetmap.org/search?q={quote(address)}&format=json&addressdetails=1&limit=1"
        headers = {'User-Agent': 'Geocoding Script'}
        response = requests.get(url, headers=headers)

        if response.status_code == 200:
            data = response.json()
            if data: # Se ci sono risultati
                location = data[0]
                lat = float(location['lat'])
                lon = float(location['lon'])
                return lat, lon
    except Exception as e:
        print(f"Errore durante il recupero delle coordinate: {e}")
    return None, None
```

In seguito alle operazioni di preparazione del dataset la situazione è la seguente:

```
Colonna: Hotel_Address, Non Null: 515215, Mancanti: 0
Colonna: Additional_Number_of_Scoring, Non Null: 515215, Mancanti: 0
Colonna: Review_Date, Non Null: 515215, Mancanti: 0
Colonna: Average_Score, Non Null: 515215, Mancanti: 0
Colonna: Hotel_Name, Non Null: 515215, Mancanti: 0
Colonna: Reviewer_Nationality, Non Null: 515215, Mancanti: 0
Colonna: Negative_Review, Non Null: 515215, Mancanti: 0
Colonna: Review_Total_Negative_Word_Counts, Non Null: 515215, Mancanti: 0
Colonna: Total_Number_of_Reviews, Non Null: 515215, Mancanti: 0
Colonna: Positive_Review, Non Null: 515215, Mancanti: 0
Colonna: Review_Total_Positive_Word_Counts, Non Null: 515215, Mancanti: 0
Colonna: Total_Number_of_Reviews_Reviewer_Has_Given, Non Null: 515215, Mancanti: 0
Colonna: Reviewer_Score, Non Null: 515215, Mancanti: 0
Colonna: Tags, Non Null: 515215, Mancanti: 0
Colonna: days_since_review, Non Null: 515215, Mancanti: 0
Colonna: lat, Non Null: 515215, Mancanti: 0
Colonna: lng, Non Null: 515215, Mancanti: 0
```

## 2. Struttura del backend

L'applicazione è stata interamente realizzata in Python.

Nell'ambito dell'analisi dei big data, la gestione efficiente di grandi moli di informazioni riveste un ruolo fondamentale per estrarre conoscenze utili e supportare processi decisionali informati. In questo contesto, PySpark rappresenta una soluzione avanzata per l'elaborazione distribuita dei dati, offrendo agli sviluppatori la possibilità di scrivere codice Python ad alte prestazioni per analizzare dataset di dimensioni considerevoli.

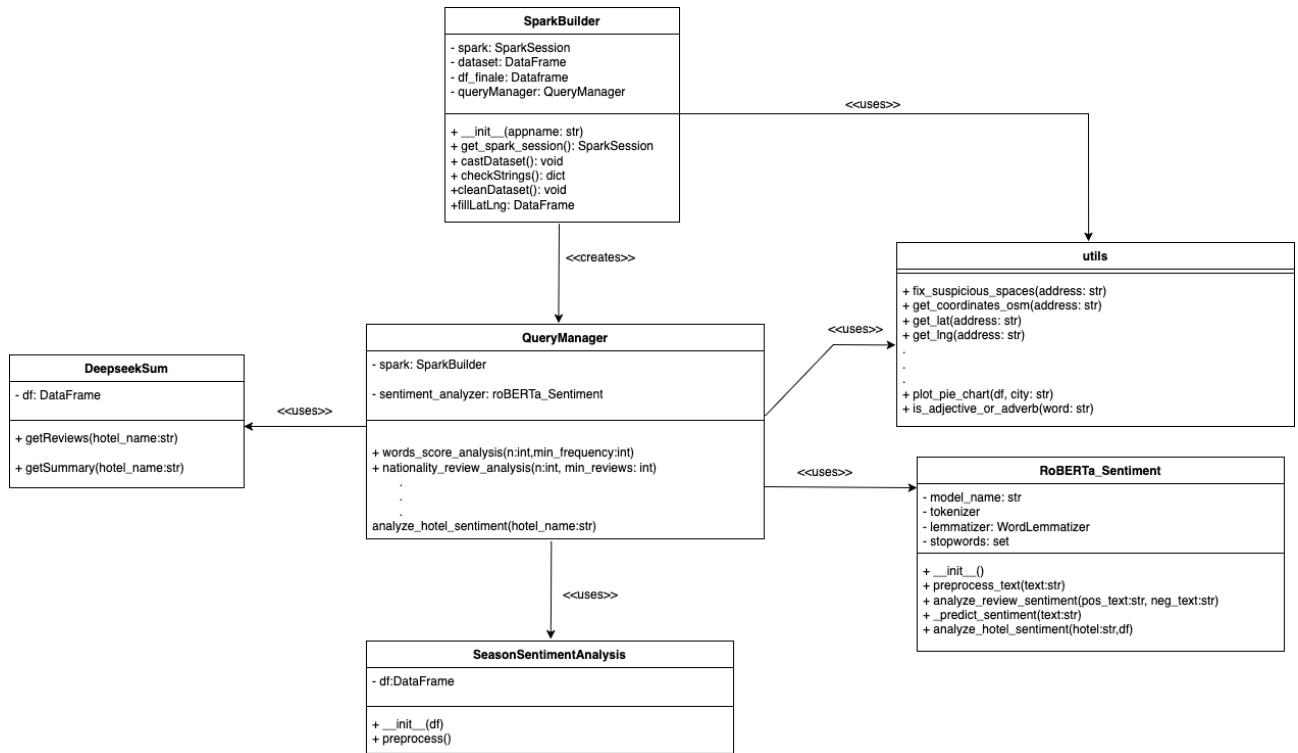
PySpark è una libreria che consente l'integrazione di Python con Apache Spark, un framework progettato per il trattamento di dati su larga scala. Uno dei suoi punti di forza risiede nell'elaborazione in-memory, che prevede il caricamento dei dati direttamente in RAM all'interno di strutture resilienti come RDD, DataFrame e Dataset. Questo approccio consente di ridurre i tempi di accesso ai dati, evitando ripetute operazioni di lettura e scrittura su disco e migliorando così l'efficienza computazionale, a condizione di disporre di una quantità di memoria sufficiente.

La scelta di Apache Spark per l'analisi dei dati si è rivelata strategica proprio per le sue capacità di ottimizzazione delle operazioni in memoria. A differenza di Hadoop, basato su MapReduce, Spark evita il continuo accesso al file system distribuito HDFS, riducendo significativamente il carico computazionale derivante dalla gestione di letture e scritture su disco.

Una delle principali sfide in un progetto di big data, tuttavia, riguarda la rappresentazione chiara e accessibile dei risultati ottenuti. Per garantire una visualizzazione efficace, è fondamentale sviluppare strumenti intuitivi e user-friendly che permettano di esplorare e comprendere i dati in modo agevole. Per soddisfare questa esigenza, si è scelto di sfruttare le ampie possibilità offerte dall'ecosistema Python, in particolare adottando Streamlit.

Streamlit è un framework open-source concepito per semplificare la creazione di applicazioni web interattive. Il suo principale vantaggio consiste nel permettere agli sviluppatori di realizzare interfacce grafiche senza la necessità di scrivere codice HTML, CSS o JavaScript, concentrandosi unicamente sulla logica dell'applicazione e sull'elaborazione dei dati. Grazie ai suoi componenti integrati, Streamlit facilita la costruzione di dashboard e strumenti di analisi visiva, rendendo più immediata l'interpretazione dei risultati da parte degli utenti finali.

Il backend dell'applicazione presenta la seguente struttura:



1 - Struttura del backend dell'applicazione

In particolare, si hanno 6 classi:

- **SparkBuilder**
- **QueryManager**
- **RoBERTa\_Sentiment**
- **DeepseekSum**
- **utils**

## 2.1 SparkBuilder

```
class SparkBuilder:  
    Tabnine | Edit | Test | Explain | Document  
    def __init__(self, appname: str):  
  
        self.spark = (SparkSession.builder  
            .master("local[*]")  
            .appName(appname)  
            .getOrCreate())  
  
        self.spark.sparkContext.setLogLevel("ERROR")  
  
        self.dataset = self.spark.read.csv(dataset_path, header=True, inferSchema=True)  
        self.castDataset()  
        self.cleanDataset()  
        self.df_finale = self.fillLatLng()  
  
        #Query Manager associato alla sessione Spark  
        self.queryManager = QueryManager(self)
```

Nel costruttore della classe `SparkBuilder`, inizialmente si carica il dataset attraverso la funzione `read_csv`, che va a caricare appunto il dataset situato nella variabile `dataset_path` opportunamente definita.

Su questo si effettua il preprocessing, come precedentemente mostrato.

## 2.2 QueryManager

La classe `QueryManager` si occupa di gestire tutta la logica legata alle query implementate.

```
def __init__(self, spark_builder: SparkBuilder):  
  
    self.spark = spark_builder  
    self.sentiment_analyzer = RoBERTa_Sentiment()  
    self.summaryLLM = SummaryLLM(self.spark.df_finale)
```

Il costruttore della classe si occupa di inizializzare l'istanza configurando due elementi fondamentali per l'analisi delle recensioni degli hotel.

In primo luogo, assegna a `self.spark` l'oggetto `spark_builder`, che rappresenta l'istanza di Spark necessaria per la gestione dei dati e l'esecuzione delle query. Questo consente di lavorare con grandi volumi di recensioni in modo efficiente.

In secondo luogo, inizializza un'istanza della classe `RoBERTa_Sentiment` e la assegna a `self.sentiment_analyzer`. Questo permette di eseguire l'analisi del sentiment sulle recensioni utilizzando il modello pre-addestrato RoBERTa, fornendo così un metodo strutturato per determinare il tono generale delle opinioni espresse dagli utenti. Infine, inizializza un'istanza della classe `SummaryLLM` in modo da poter eseguire il riassunto delle recensioni di un particolare hotel.

## 2.3 RoBERTa\_Sentiment

La classe RoBERTa\_Sentiment è progettata per analizzare il sentimento delle recensioni degli hotel utilizzando il modello di deep learning RoBERTa, specificamente la versione *cardiffnlp/twitter-roberta-base-sentiment*, addestrata per il sentiment analysis sui testi di Twitter. L'obiettivo principale di questa classe è classificare il tono delle recensioni degli utenti in positivo, negativo o neutrale, fornendo così una valutazione complessiva del sentimento degli ospiti per ciascun hotel.

```
class RoBERTa_Sentiment:  
    Tabnine | Edit | Test | Explain | Document  
    def __init__(self):  
        self.model_name = "cardiffnlp/twitter-roberta-base-sentiment"  
        self.tokenizer = AutoTokenizer.from_pretrained(self.model_name)  
        self.model = AutoModelForSequenceClassification.from_pretrained(self.model_name)  
        self.model.eval()  
  
        self.lemmatizer = WordNetLemmatizer()  
        self.stopwords = set(stopwords.words("english"))
```

Il costruttore inizializza la classe e carica il modello RoBERTa per l'analisi del sentimento. Durante questa fase:

- Viene definito il nome del modello da caricare.
- Viene inizializzato il **tokenizer**, che suddivide il testo in unità comprensibili dal modello.
- Viene caricato il modello stesso e impostato in modalità valutazione (eval), garantendo che non venga eseguito addestramento durante l'uso.
- Viene istanziato un **lemmatizzatore** di WordNet per la normalizzazione del testo.
- Viene creata una lista di **stopwords** per rimuovere parole irrilevanti dall'analisi del sentimento.

### 2.2.2 Pre elaborazione del testo

```
def preprocess_text(self, text):  
    """ pulizia, tokenizzazione, stopwords, lemmatizzazione."""  
    if not text or text.lower() in ["no negative", "no positive"]:  
        return ""  
  
    text = text.lower()  
    text = re.sub(r"http\S+|www\S+|https\S+", "", text) # Rimuove URL  
    text = re.sub(r"<.*?>", "", text) # Rimuove tag HTML  
    text = re.sub(r"[\^w\s]", "", text) # Rimuove punteggiatura  
    text = re.sub(r"\d+", "", text) # Rimuove numeri  
  
    tokens = word_tokenize(text)  
    tokens = [word for word in tokens if word not in self.stopwords]  
    tokens = [self.lemmatizer.lemmatize(word) for word in tokens]  
  
    return " ".join(tokens)
```

Il metodo *preprocess\_text(text)* è responsabile della pulizia del testo prima che venga analizzato. Questa funzione:

- Converte il testo in minuscolo per garantire coerenza.
- Rimuove tutti gli elementi che non contribuiscono al significato semantico della recensione.
- Tokenizza il testo, suddividendolo in parole singole.
- Elimina le **stopwords** per focalizzarsi solo sulle parole significative.
- Lemmatizza i termini, riducendoli alla loro forma base, migliorando così l'accuratezza dell'analisi.

### 2.2.3 Analisi del sentiment delle recensioni

```
def analyze_review_sentiment(self, positive_text, negative_text):
    """Combina il sentimento della recensione positiva e negativa per restituire un'unica valutazione finale."""

    strong_words = {"all", "everything", "always"} # Parole che indicano giudizi forti
    neutral_negative_words = {"nothing", "none", "no negative"} # Nessun problema
    neutral_positive_words = {"no positive"} # Nessun aspetto positivo

    positive_text = positive_text.strip().lower()
    negative_text = negative_text.strip().lower()

    if negative_text in neutral_negative_words:
        negative_text = ""

    if positive_text in neutral_positive_words:
        positive_text = ""

    contains_strong_positive = any(word in positive_text.split() for word in strong_words)
    contains_strong_negative = any(word in negative_text.split() for word in strong_words)

    if contains_strong_negative and not contains_strong_positive:
        return "negative"

    if contains_strong_positive and not contains_strong_negative:
        return "positive"

    # Se solo una parte è significativa, usa solo quella
    if positive_text and not negative_text:
        return self._predict_sentiment(positive_text)
    if negative_text and not positive_text:
        return self._predict_sentiment(negative_text)

    # Se entrambe le parti contengono testo utile, calcola il sentimento combinato
    pos_sentiment = self._predict_sentiment(positive_text)
    neg_sentiment = self._predict_sentiment(negative_text)

    # Mapping del sentimento in valori numerici
    sentiment_map = {"negative": -1, "neutral": 0, "positive": 1}
    pos_score = sentiment_map[pos_sentiment]
    neg_score = sentiment_map[neg_sentiment]

    # Calcolo dei pesi in base alla lunghezza del testo
    pos_length = len(positive_text) if positive_text else 1
    neg_length = len(negative_text) if negative_text else 1
    total_length = pos_length + neg_length

    pos_weight = pos_length / total_length
    neg_weight = neg_length / total_length

    # Media ponderata del sentimento
    combined_score = (pos_score * pos_weight) + (neg_score * neg_weight)

    if combined_score > 0.3:
        return "positive"
    elif combined_score < -0.3:
        return "negative"
    else:
        return "neutral"
```

Il metodo `analyze_review_sentiment` è uno degli elementi chiave della classe. Riceve come input una recensione suddivisa in una parte positiva e una parte negativa, come previsto dal dataset. L'algoritmo segue una logica specifica per determinare il sentimento complessivo:

1. Identifica parole particolarmente **forti** (es. "all", "everything", "always") che possono amplificare il tono positivo o negativo della recensione.

2. Riconosce le espressioni **neutre** (es. "no negative", "nothing") che indicano l'assenza di aspetti negativi o positivi e le ignora.
3. Se solo una delle due parti contiene testo significativo, il sentiment viene calcolato esclusivamente su quella.
4. Se entrambe le parti sono presenti, viene eseguita un'analisi separata su ciascuna, e il risultato finale viene ottenuto attraverso una media ponderata, basata sulla lunghezza dei testi positivo e negativo.

### 2.2.3 Predizione del Sentiment con RoBERTa

```
def _predict_sentiment(self, text):
    clean_text = self.preprocess_text(text)
    if not text.strip():
        return "neutral"

    inputs = self.tokenizer(clean_text, return_tensors="pt", truncation=True, padding=True, max_length=512)

    with torch.no_grad():
        outputs = self.model(**inputs)

    predicted_class = torch.argmax(outputs.logits, dim=-1).item()
    labels = {0: "negative", 1: "neutral", 2: "positive"}

    return labels[predicted_class]
```

Il metodo *\_predict\_sentiment* è una funzione interna che utilizza il modello RoBERTa per assegnare un'etichetta di sentiment al testo. Dopo aver pulito il testo con *preprocess\_text*, la funzione:

- Tokenizza il testo e lo converte in input tensoriale per RoBERTa.
- Esegue la predizione con il modello senza aggiornare i pesi (*torch.no\_grad*).
- Estrae la classe più probabile (negativo, neutrale o positivo) e restituisce l'etichetta corrispondente.

### 2.2.3 Analisi del sentiment per un hotel

```
def analyze_hotel_sentiment(self, hotel_name, df):

    hotel_reviews = df.filter(col("Hotel_Name") == hotel_name).select("Positive_Review", "Negative_Review").collect()

    if not hotel_reviews:
        return "Nessuna recensione"

    sentiment_scores = []
    sentiment_map = {"negative": -1, "neutral": 0, "positive": 1}

    for row in hotel_reviews:
        print(f"analizzando: {row}")
        positive_text = row["Positive_Review"].strip()
        negative_text = row["Negative_Review"].strip()

        # Calcola il sentiment della recensione utilizzando il metodo per la singola recensione
        review_sentiment = self.analyze_review_sentiment(positive_text, negative_text)

        print(f"sentiment = {review_sentiment}")

        # Mappa il sentiment in valori numerici e accumula
        sentiment_scores.append(sentiment_map[review_sentiment])

    avg_score = sum(sentiment_scores) / len(sentiment_scores)

    # Determina il sentiment complessivo basato sulla media
    print(avg_score)

    if avg_score > 0.2 and avg_score < 0.4:
        return "Piuttosto Positivo"
    elif avg_score >= 0.4:
        return "Molto Positivo"
    elif avg_score < -0.2 and avg_score > - 0.3:
        return "Piuttosto Negativo"
    elif avg_score <= -0.4:
        return "Molto Negativo"
    else:
        return "Neutrale"
```

Il metodo `analyze_hotel_sentiment` consente di calcolare il sentiment medio di tutte le recensioni di un hotel. Questa funzione:

1. Filtra il DataFrame per ottenere solo le recensioni relative all'hotel specificato.
2. Per ogni recensione, calcola il sentiment combinato tra la parte positiva e negativa utilizzando `analyze_review_sentiment()`.
3. Converte le valutazioni testuali in **valori numerici** per poter effettuare una media complessiva.
4. Determina il sentiment generale dell'hotel basandosi sul punteggio medio.

## 2.4 SeasonSentimentAnalysis

La classe `SeasonSentimentAnalysis` è progettata per analizzare il sentiment delle recensioni degli hotel e mettere in relazione il punteggio ottenuto con la stagione dell'anno in cui la recensione è stata pubblicata. Questo permette di individuare eventuali variazioni stagionali nel livello di soddisfazione dei clienti, fornendo così un'analisi più dettagliata sull'andamento del sentiment nel tempo.

### 2.4.1 Costruttore

```
class SeasonSentimentAnalysis:  
    Tabnine | Edit | Test | Explain | Document  
    def __init__(self, df):  
        """  
        Inizializza la classe con il DataFrame.  
        """  
        self.df = df  
        ensure_nltk_resources()
```

Il costruttore inizializza la classe con un DataFrame Spark contenente le recensioni degli hotel. Durante l'inizializzazione, viene anche richiamata la funzione `ensure_nltk_resources()`, che verifica la presenza delle risorse necessarie per l'analisi del sentiment tramite VADER (Valence Aware Dictionary and sEntiment Reasoner). Se il lessico richiesto non è disponibile, viene automaticamente scaricato.

### 2.4.2 Pre-elaborazione del Sentiment

```
def preprocess(self):  
    """  
    Calcola il sentiment per recensioni positive e negative  
    e aggiunge la stagione per ogni recensione.  
    """  
    # Calcola il sentiment per recensioni positive e negative  
    self.df = self.df.withColumn("Total_Review", concat(col("Positive_Review"), col("Negative_Review"))))  
    self.df = self.df.withColumn("Net_Sentiment", sentiment_udf(col("Total_Review")))  
  
    # Determina la stagione in base al mese  
    self.df = self.df.withColumn(  
        "Season",  
        when((month(col("Review_Date")).isin(12, 1, 2)), "Winter") # Inverno  
            .when((month(col("Review_Date")).isin(3, 4, 5)), "Spring") # Primavera  
            .when((month(col("Review_Date")).isin(6, 7, 8)), "Summer") # Estate  
            .when((month(col("Review_Date")).isin(9, 10, 11)), "Autumn") # Autunno  
    )  
    return self.df
```

Il metodo `preprocess()` è responsabile della preparazione dei dati per l'analisi, eseguendo diverse operazioni chiave sul DataFrame:

#### 1. Calcolo del Sentiment:

Invece di analizzare separatamente il sentiment delle recensioni positive e negative, la funzione concatena entrambe le parti in un'unica colonna `Total_Review`.

Viene quindi utilizzata la User Defined Function (UDF) *sentiment\_udf*, che applica il modello VADER a *Total\_Review* per ottenere un punteggio di sentiment netto (Net\_Sentiment).

Il valore restituito da VADER è un punteggio numerico che varia da -1 (molto negativo) a +1 (molto positivo), con 0 che rappresenta un sentiment neutrale.

## 2. Assegnazione della Stagione:

Utilizzando la funzione `month(col("Review_Date"))`, viene estratto il mese di pubblicazione della recensione.

In base al mese, la recensione viene classificata in una delle quattro stagioni dell'anno.

### 2.4.3 VADER

Per l'analisi del sentiment viene utilizzato **VADER**, un modello NLP ottimizzato per testi brevi, come recensioni o post sui social media. Il modello assegna un punteggio di polarità basato su un dizionario predefinito di parole con valori di sentiment associati. La funzione *calculate\_sentiment* applica VADER e restituisce il valore numerico corrispondente alla recensione.

## 2.5 DeepSeek Summary

La classe `SummaryLLM` realizzata nel file *DeepSeekSum* è progettata per realizzare il summary delle recensioni (sia positive che negative) di uno specifico hotel sfruttando il large language model *DeepSeek* in particolare “*deepseek-r1:1.5b*” (modello DeepSeek con 1.5 miliardi di parametri).

### 2.5.1 Inizializzazione della classe

```
def __init__(self, dataframe):  
    self.df = dataframe
```

Nel costruttore viene passato un dataframe di Spark contenente le recensioni dell'hotel e viene memorizzato come attributo di classe.

## 2.5.2 Individuare le recensioni

```
def getReviews(self, hotel_name):
    reviews_df = self.df.filter((col("Hotel_Name") == hotel_name) &
        (col("Positive_Review") != "No Positive") &
        (col("Negative_Review") != "No Negative")).orderBy(col
        ("Total_Number_of_Reviews_Reviewer_Has_Given")).select("Negative_Review", "Positive_Review").
        limit(50)

    reviews_Pandas = reviews_df.toPandas()
    reviews = ""
    for _, row in reviews_Pandas.iterrows():
        reviews += f"{row['Positive_Review']} ; {row['Negative_Review']}. "
    return reviews
```

Tale funzione filtra le recensioni per un determinato hotel escludendo le recensioni senza contenuto effettivo.

Le recensioni vengono ordinate in base al numero totale di recensioni lasciate dall'utente in quanto si può ipotizzare una maggiore affidabilità dei recensori più esperti.

Vengono in particolare considerate le prime 50 recensioni.

Il dataframe Spark viene convertito in un dataframe Pandas in modo da poter combinare le recensioni in un'unica stringa.

## 2.5.3 Creazione summary

```
def getSummary(self, hotel_name):
    reviews = self.getReviews(hotel_name)
    response = ollama.chat(model='deepseek-r1:1.5b', messages=[
    {
        'role': 'user',
        'content':f'Give me a detailed and elegant summary of the following text:{reviews}',
    },
    ])
    summary = response['message']['content']
    thinking = "<think>" + summary.split('<think>')[1].split('</think>')[0] + "</think>"
    return summary.replace(thinking, "")
```

In primis vengono recuperate le recensioni chiamando la funzione sopra descritta.

Si da il testo in pasto al modello chiedendo un riassunto dettagliato ed elegante.

Il modello utilizzato, DeepSeek, restituisce anche una parte di "*think*" che viene eliminata dalla stringa restituita.

## 2.6 utils

La classe `utils` è una classe contenente metodi di utilità che verranno elencati di seguito.

### 2.6.1 Correzione degli indirizzi e Geocoding

```
def fix_suspicious_spaces(address):
    corrections = {
        "Damr mont": "Damrémont",
        "Mont martre": "Montmartre",
        "St Ger main": "St Germain",
        "P pini re": "Pépinière",
        "arr": "Arrondissement",
        "Ga t": "Gaité",
        "Bail n": "Bailen",
        "Gr nentorgasse 30 09": "30 Grünentorgasse",
        "Landstra er G rtel 5 03": "5 Landstrasser Guertel",
        "Landstra e": "Landstraße",
        "Hasenauerstra e 12 19": "12 Hasenauerstrasse",
        "D bling": "Döbling",
        "Josefst dter Stra e 22 08": "22 Josefstdter Straße",
        "Josefst dter Stra e 10 12 08": "10 Josefstdter Straße",
        "10 12 08 Josefstadt": "Josefstadt",
        "Paragonstra e 1 11": "1 Paragonstrasse",
        "Sieveringer Stra e 4 19": "4 Sieveringer Straße Lower Sievering",
        "Pau Clar s": "Pau Claris",
        "Sep lveda": "Sepulveda",
        "Savoyenstra e 2 16": "2 Savoyenstraße",
        "W hringer Stra e 33 35 09": "33-35 Währinger Straße",
        "W hringer Stra e 12 09": "12 Währinger Straße",
        "Taborstra e 8 A 02": "8 Taborstraße"
    }
    for wrong, correct in corrections.items():
        if wrong in address:
            address = address.replace(wrong, correct)
    return address
```

La funzione `fix_suspicious_spaces(address)` si occupa di correggere errori comuni negli indirizzi, come spazi errati o nomi di strade scritti in modo incompleto. Questo è particolarmente utile quando si lavora con dati non standardizzati provenienti da fonti diverse.

## 2.6.2 Individuazione coordinate

```
def get_coordinates_osm(address):
    """
    Ottiene latitudine e longitudine da OpenStreetMap (Nominatim) dato un indirizzo.
    """
    try:
        address = normalize_address(address)
        address = fix_suspicious_spaces(address)

        url = f"https://nominatim.openstreetmap.org/search?q={quote(address)}&format=json&addressdetails=1&limit=1"
        headers = {'User-Agent': 'Geocoding Script'}
        response = requests.get(url, headers=headers)

        if response.status_code == 200:
            data = response.json()
            if data: # Se ci sono risultati
                location = data[0]
                lat = float(location['lat'])
                lon = float(location['lon'])
                return lat, lon
    except Exception as e:
        print(f"Errore durante il recupero delle coordinate: {e}")
    return None, None
```

Per ottenere le coordinate geografiche di un hotel, il modulo utilizza la funzione `get_coordinates_osm`, che interroga il servizio OpenStreetMap (Nominatim) per ottenere latitudine e longitudine corrispondenti a un dato indirizzo. Se l'indirizzo non è valido o non restituisce un risultato, la funzione gestisce l'errore e restituisce `None`.

Per comodità, vengono fornite due funzioni di supporto:

- `get_lat(address)`, che estrae solo la latitudine.
- `get_lng(address)`, che estrae solo la longitudine.

### 2.6.3 Visualizzazione dei Trend di Sentiment e Recensioni

Il modulo fornisce due funzioni per la visualizzazione dei dati:

#### 1. *graficoTrend*

```
def graficoTrend(dataframe, single : bool):
    trend_pandas = dataframe.toPandas()
    # Impostare stile del grafico
    sns.set(style="whitegrid")

    # Creare il grafico per ogni hotel
    plt.figure(figsize=(12, 6))
    for hotel in trend_pandas["Hotel_Name"].unique():
        # Filtrare i dati per ogni hotel
        hotel_data = trend_pandas[trend_pandas["Hotel_Name"] == hotel]

        # Creare il grafico a linee per l'hotel
        sns.lineplot(data=hotel_data, x="YearMonth", y="Average_Score", marker="o", label=hotel)

    # Migliorare l'estetica del grafico
    if single:
        plt.title("Trend dello Score Medio per Mese", fontsize=16)
    else:
        plt.title("Trend dello Score Medio per Mese degli hotel vicini", fontsize=16)
    plt.xlabel("Anno/Mese", fontsize=12)
    plt.ylabel("Punteggio Medio", fontsize=12)
    plt.xticks(rotation=45)
    plt.legend(title="Hotel", bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.grid(True, linestyle="--", alpha=0.7)
    plt.tight_layout()

    return plt
```

Questa funzione genera un grafico a linee per mostrare l'andamento del punteggio medio degli hotel nel tempo.

- Se il parametro `single` è impostato su `True`, viene generato il trend per un singolo hotel.
- Altrimenti, il grafico include anche gli hotel nelle vicinanze, facilitando il confronto tra strutture simili.

## 2. `plot_pie_chart(df, city)`

Questa funzione crea un **grafico a torta** che rappresenta la distribuzione dei tag più usati nelle recensioni di una città specifica.

```
def plot_pie_chart(df, city):
    """
    Genera un pie chart per mostrare la distribuzione dei tag in una città.
    """
    # Filtra il DataFrame per la città selezionata
    city_df = df.filter(col("Hotel_Address").contains(city))

    # Espandi i tag (ARRAY<STRING> → righe singole)
    tags_df = city_df.select(explode(col("tags")).alias("tag"))

    # Conta i tag
    tag_counts = tags_df.groupBy("tag").agg(count("*").alias("count")).orderBy(desc("count"))

    # Converti in Pandas per Plotly
    tag_counts_pd = tag_counts.toPandas()

    top_10 = tag_counts_pd[:10]

    others_count = tag_counts_pd[10:]["count"].sum()

    if others_count > 0:
        top_10 = top_10.append({"tag": "Other", "count": others_count}, ignore_index=True)
        top_10 = pd.concat([top_10, pd.DataFrame([{"tag": "Other", "count": others_count}])], ignore_index=True)

    fig = px.pie(top_10, values="count", names="tag", title=f"Distribuzione dei tag per {city}")
    return fig
```

- Il DataFrame viene filtrato per la città desiderata.
- I tag vengono contati e ordinati per frequenza.
- Viene visualizzato un grafico interattivo con **Plotly**, che include i 10 tag più frequenti e un'aggregazione per gli altri.

### 2.6.4 Calcolo delle distanze tra hotel

```
def haversine(lat1, lon1, lat2, lon2):
    R = 6371
    lat1, lon1, lat2, lon2 = map(math.radians, [lat1, lon1, lat2, lon2]) # Converti in radianti
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = math.sin(dlat / 2) ** 2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon / 2) ** 2
    return 2 * R * math.asin(math.sqrt(a)) * 1000 # Converti in metri
```

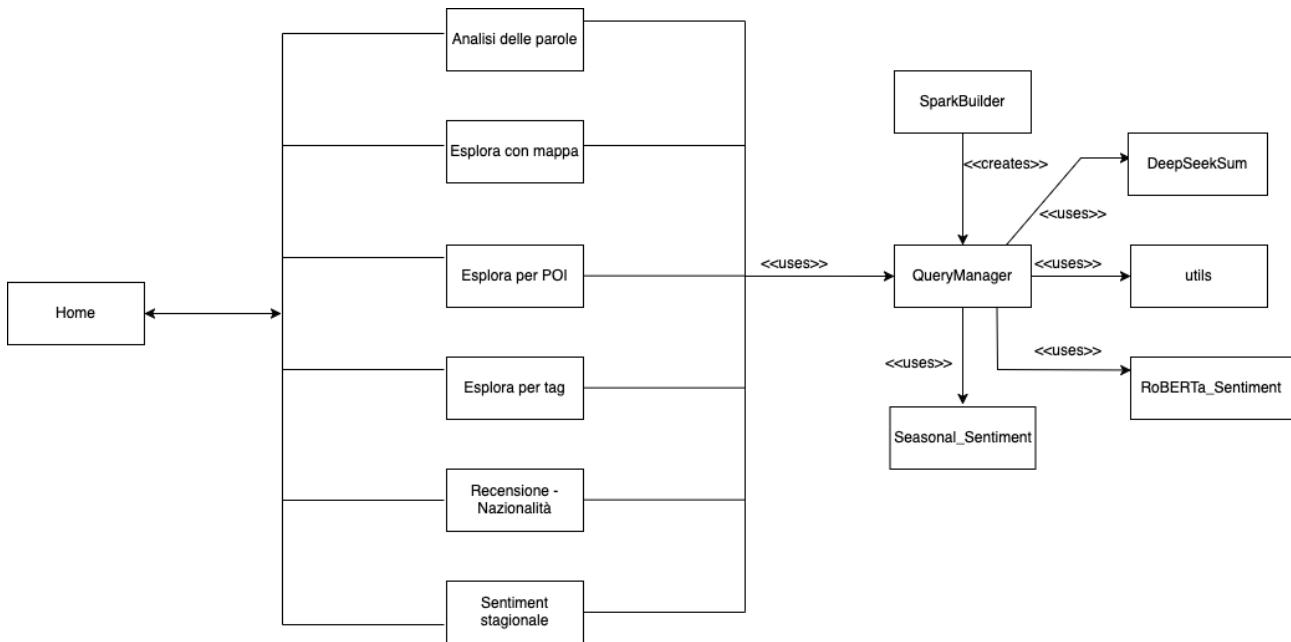
La funzione `haversine(lat1, lon1, lat2, lon2)` implementa la **formula dell'Haversine**, che calcola la distanza tra due punti sulla superficie terrestre misurata in metri. Questa funzione è utile per determinare la vicinanza tra hotel o per altre analisi geospatiali.

## 2.6.5 Identificazione di avverbi e aggettivi

```
def is_adjective_or_adverb(word):
    """
    Determina se una parola è un aggettivo (a) o un avverbio (r) utilizzando WordNet.
    """
    synsets = wordnet.synsets(word)
    if not synsets:
        return False
    return any(s.pos() == 'a' or s.pos()=='r' for s in synsets)
```

Infine, la funzione *is\_adjective\_or\_adverb(word)* sfrutta il lessico di WordNet per determinare se una parola è un aggettivo o un avverbio. Questa funzionalità è utile per l'analisi semantica delle recensioni, ad esempio per individuare parole chiave che influenzano il sentimento, come si vedrà in seguito.

## 2.7. Struttura finale dell'applicazione



2- Struttura finale dell'applicazione (semplicificata)

È riportata la struttura finale (semplicificata) dell'applicazione.

In particolare, nella parte frontend si hanno sei pagine di esplorazione e una pagina Home che funge da landing page.

### 3. Analisi delle query

#### 3.1 Analisi degli aggettivi e degli avverbi

Questa query è progettata per analizzare quali parole, in particolare avverbi e aggettivi, sono indicatori di punteggi alti o bassi.

L'obiettivo è identificare i termini che influenzano maggiormente la valutazione media degli utenti, escludendo quelli con una frequenza inferiore ad una soglia.

```
def words_score_analysis(self, min_frequency=1000):
    is_adjective_or_adverb_udf = udf(utils.is_adjective_or_adverb, BooleanType())
    df = self.spark.df_finale

    #Aggettivi e avverbi nelle recensioni positive
    positive_words = df.select(
        col("Reviewer_Score"),
        explode(split(col("Positive_Review"), r"\s+")).alias("word")
    ).filter(col("word") != "") # Rimuove parole vuote
    positive_words = positive_words.withColumn("word", lower(col("word")))
    positive_words_filtered = positive_words.filter(is_adjective_or_adverb_udf(col("word")))
    positive_word_scores = positive_words_filtered.groupBy("word") \
        .agg(
            avg("Reviewer_Score").alias("avg_score"),
            count("word").alias("word_count")
        ) \
        .filter(col("word_count") >= min_frequency) \
        .orderBy(desc("avg_score"))
    #Aggettivi e avverbi nelle recensioni negative
    negative_words = df.select(
        col("Reviewer_Score"),
        explode(split(col("Negative_Review"), r"\s+")).alias("word")
    ).filter(col("word") != "") # Rimuove parole vuote
    negative_words = negative_words.withColumn("word", lower(col("word")))
    negative_words_filtered = negative_words.filter(is_adjective_or_adverb_udf(col("word")))
    negative_word_scores = negative_words_filtered.groupBy("word") \
        .agg(
            avg("Reviewer_Score").alias("avg_score"),
            count("word").alias("word_count")
        ) \
        .filter(col("word_count") >= min_frequency) \
        .orderBy("avg_score") # Ordina dal punteggio più basso
    return positive_word_scores, negative_word_scores
```

Vengono estratte le parole sia delle recensioni positive che da quelle negative. Si applica una funzione UDF (is\_adjective\_or\_adverb\_udf) per selezionare solo gli aggettivi e gli avverbi.

Si sfrutta una funzione definita nel file *utils* (precedentemente trattata) che sfrutta Wordnet per identificare se la parola è un aggettivo o un avverbio.

Si calcola il punteggio medio delle recensioni in cui la parola compare e si calcola frequenza; in particolare, la frequenza è utile per considerare solo le parole con un numero di occorrenze maggiore di una certa soglia.

Per recensioni positive si ordina in modo decrescente rispetto al punteggio medio, quindi, i punteggi più alti sono in cima.

Per le recensioni negative, invece, le parole sono ordinate in modo crescente rispetto al punteggio medio, quindi, parole con punteggi più bassi sono in cima.

In output vengono restituiti due DataFrame che contengono rispettivamente gli aggettivi (e avverbi) che compaiono nelle recensioni positive con il loro punteggio medio associato, e gli aggettivi (e avverbi) che compaiono nelle recensioni negative con il loro punteggio medio associato.

Questa query viene utilizzata nella pagina *Analisi delle parole*:

1. Confronto top parole positive con top parole negative:

 Top parole positive

	word	avg_score	word_count
0	unforgettable	9.6966	118
1	highly	9.646	3,886
2	defiantly	9.6082	257
3	friendliest	9.6017	175
4	definitely	9.5678	9,968
5	soon	9.5637	1,022
6	thoroughly	9.5603	519
7	phenomenal	9.5518	170
8	unbelievable	9.55	206
9	pure	9.5187	166
...	...	...	...

 Top parole negative

	word	avg_score	word_count
0	unclean	5.2596	245
1	impolite	5.2801	161
2	worst	5.3786	2,002
3	incompetent	5.4257	140
4	dirty	5.6326	6,201
5	unhelpful	5.6735	1,245
6	western	5.7459	183
7	firstly	5.8313	131
8	unprofessional	5.8396	493
9	unfriendly	5.8809	1,273
...	...	...	...

2. Distribuzioni parole positive e distribuzioni parole negative:

 Distribuzione parole positive



 Distribuzione parole negative

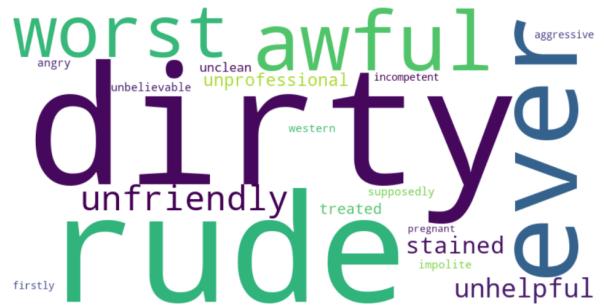


3. Word cloud parole positive e word cloud parole negative:

Word Cloud delle parole positive



Word Cloud delle parole negative



Nelle immagini riportate è stato limitato il numero di parole a 20 (è possibile modificarlo portandolo a 50 o a 100).

### 3.2 Analisi dell'influenza dei tag sui punteggi delle recensioni

```
def tag_influence_analysis(self, min_count=1000):
    df = self.spark.df_finale

    # Esplodi la colonna Tags in righe individuali
    exploded_tags = df.select(
        col("Reviewer_Score"),
        explode(col("Tags")).alias("tag")
    )
    # Calcola la media del punteggio e il conteggio per ciascun tag
    tag_scores_desc = exploded_tags.groupBy("tag") \
        .agg(
            avg("Reviewer_Score").alias("avg_score"),
            count("*").alias("tag_count")
        ) \
        .filter(col("tag_count") >= min_count).orderBy(desc("avg_score"))

    return tag_scores_desc
```

Questa query ha l'obiettivo di individuare quali tag sono correlati a punteggi più alti o più bassi considerando solo quelli con una frequenza minima predefinita.

La colonna *Tags* contiene liste di tags per ogni recensione, viene utilizzata quindi la funzione *explode* per trasformare ogni tag in una riga separata.

Per ogni tag, quindi, vengono calcolati:

- Punteggio medio delle recensioni in cui compare.
- Numero di volte in cui il tag appare.

Tali informazioni vengono mostrate nella pagina *Esplora per tag*:

Tag con score migliore:

	tag	avg_score	tag_count
0	Deluxe Double	9.1304	1630
1	Standard	9.1234	1229
2	Classic King Room	8.9908	1316
3	Large Double Room	8.9864	1192
4	Luxury Double Room	8.9741	1149
5	Superior King Room	8.9328	4245
6	Superior King or Twin Room	8.9244	1829
7	King Guest Room	8.9047	1630
8	Superior Double Room with Internal View	8.8475	1112
9	Classic Double or Twin Room	8.7847	6101

Tag con score peggiore:

	tag	avg_score	tag_count
0	Standard Double Room without Window	7.0301	2423
1	Double Hilton Guestroom	7.6900	1295
2	Double Room Non Smoking	7.7729	1257
3	Deluxe Single Room	7.8225	3281
4	Standard Single Room	7.8880	4607
5	Quadruple Room	7.9453	1156
6	Business trip	7.9732	82594
7	Standard Twin Room	8.0219	9732
8	Single Room	8.0322	9652
9	Basic Double Room	8.0675	2849

In particolare, si considera come frequenza minima un valore pari a mille.

### 3.3 Analisi della lunghezza recensioni in funzione dello score

```
def review_length_analysis(self):
    df = self.spark.df_finale.filter((col("Review_Total_Positive_Word_Counts") > 0) | (col("Review_Total_Negative_Word_Counts") > 0))

    # Calcolo della lunghezza media delle recensioni positive e negative per punteggio
    review_length = df.groupBy("Reviewer_Score") \
        .agg(
            avg("Review_Total_Positive_Word_Counts").alias("avg_positive_length"),
            avg("Review_Total_Negative_Word_Counts").alias("avg_negative_length")
        ).orderBy("Reviewer_Score")

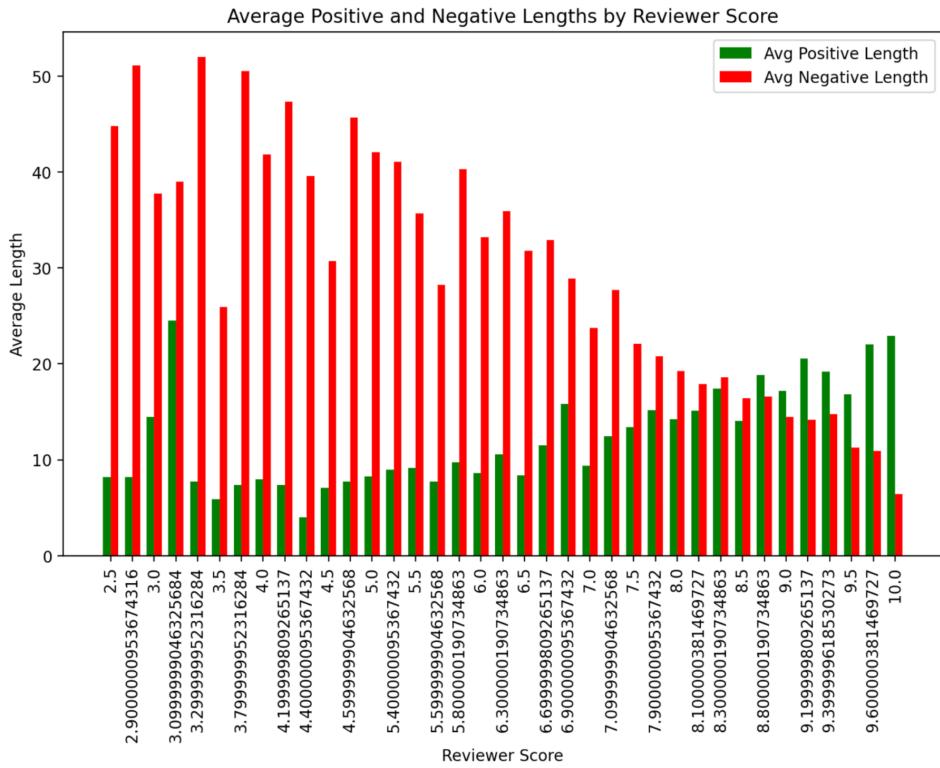
    return review_length
```

Tale funzione analizza la relazione tra la lunghezza delle recensioni (positive e negative) e il punteggio assegnato.  
L'obiettivo è comprendere la correlazione tra il numero di parole utilizzate e il punteggio assegnato.

Per ogni punteggio viene calcolata la lunghezza media delle recensioni positive e negative.

I risultati, per comodità, sono ordinati in base al punteggio in ordine crescente.

Viene mostrato un grafico con le informazioni ottenute nella pagina *Analisi delle parole*:



In particolare, si può notare come il numero di parole nella parte negativa della recensione è sempre maggiore rispetto al numero di parole presenti nella parte positiva; un'inversione di tale comportamento la si ha solamente per voti estremamente alti, cioè per voti strettamente maggiori di 8.5.

### 3.4 Analisi della reputazione di un Hotel

```

def reputation_analysis_single(self, hotel_name, recent_reviews=30):
    df = self.spark.df_finale.filter(col("Hotel_Name") == hotel_name)

    if df.count() == 0:
        print(f"Nessuna recensione trovata per l'hotel: {hotel_name}")
        return None

    # Calcolo del punteggio medio storico per l'hotel specificato
    avg_historical_window = Window.partitionBy("Hotel_Name")
    df = df.withColumn(
        "avg_historical_score", avg("Reviewer_Score").over(avg_historical_window)
    )

    # Calcolo del punteggio medio delle recensioni recenti (finestra di N recensioni)
    recent_reviews_window = Window.partitionBy("Hotel_Name").orderBy(desc("Review_Date")).rowsBetween(0,
    recent_reviews - 1)
    df = df.withColumn(
        "avg_recent_score", avg("Reviewer_Score").over(recent_reviews_window)
    )

    # Estrazione di una singola riga con i dati aggregati
    df_aggregated = df.groupBy("Hotel_Name").agg(
        first("avg_historical_score").alias("avg_historical_score"),
        first("avg_recent_score").alias("avg_recent_score")
    )

    # Calcolo della differenza tra il punteggio recente e quello storico
    df_aggregated = df_aggregated.withColumn(
        "score_difference", col("avg_recent_score") - col("avg_historical_score")
    )
    return df_aggregated

```

Questa funzione confronta il punteggio medio storico dell'hotel con quello delle recensioni più recenti, in particolare, con le ultime 30.

Il dataset viene filtrato per selezionare solo le recensioni relative all'hotel indicato.

Si utilizza una finestra di aggregazione per calcolare il punteggio medio su tutte le recensioni dell'hotel.

Si definisce una finestra temporale che considera solo le recensioni recenti e si calcola il punteggio medio su di esse.

Si crea una tabella con:

- Punteggio medio storico.
- Punteggio medio recensioni recenti.
- Differenza tra i due punteggi (questo permette di determinare se l'hotel sta migliorando o peggiorando)

Un utilizzo è presente nella pagina *Esplora con mappa*:

Confronto ultime recensioni con la media per Idea Hotel Milano San Siro:

Average Score	Differenza
<b>6.58237</b>	<b>-0.2857</b>
Average Score recente	L'hotel sta peggiorando
<b>6.29667</b>	

### 3.5 Seasonal Sentiment Analysis

La funzione *seasonal\_sentiment\_analysis* analizza l'andamento del sentiment medio delle recensioni in base alla stagione dell'anno, sfruttando il modello VADER per l'analisi del sentiment.

```
def seasonal_sentiment_analysis(self):
    """
    Analizza come il sentimento medio delle recensioni varia in base alla stagione dell'anno.
    Utilizza VADER → rule based e lessico
    """
    sentiment_analysis = SeasonSentimentAnalysis(self.spark.df_finale)

    # Preprocessa i dati per calcolare il sentimento e la stagione
    df_preprocessed = sentiment_analysis.preprocess()

    # Aggrega il sentimento per hotel e stagione
    seasonal_sentiment = df_preprocessed.groupBy("Hotel_Name", "Hotel_Address", "Season").agg(
        avg("Net_Sentiment").alias("avg_sentiment"),
        avg("Reviewer_Score").alias("avg_reviewer_score"),
        count("*").alias("review_count")
    ).orderBy("Hotel_Name", "Season")

    return seasonal_sentiment
```

Il metodo si basa sulla classe SeasonSentimentAnalysis descritta in precedenza.

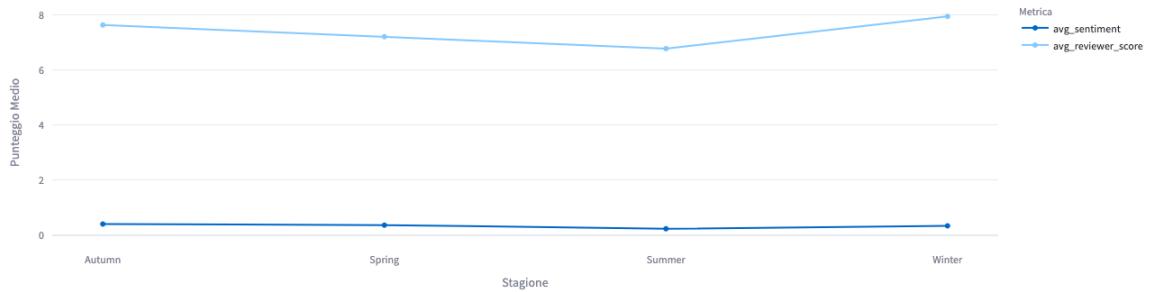
La funzione calcola poi il sentimento medio (Net\_Sentiment) di ogni recensione e assegna a ciascuna una stagione.

Una volta che i dati sono stati processati, vengono aggregati per hotel e stagione e si calcolano:

- Sentiment medio delle recensioni, che rappresenta il tono generale delle opinioni degli utenti in ogni stagione.
- Punteggio medio assegnato dagli utenti
- Numero di recensioni per stagione, per analizzare il volume di feedback ricevuti nei diversi periodi dell'anno.

In particolare, questa funzione ha una pagina dedicata, ovvero “Sentiment Stagionale” di cui si mostra un estratto:

### SENTIMENT E PUNTEGGIO PER STAGIONE - BEST WESTERN SERAPHINE KENSINGTON OLYMPIA



### SENTIMENT COMPLESSIVO DELLE RECENSIONI: 😐 Neutrale

#### DATI AGGREGATI DELLE RECENSIONI

	Hotel_Name	Hotel_Address	Season	avg_sentiment	avg_reviewer_score	review_count
0	Best Western Seraphine Kensington Olympia	225 High Street Kensington Kensington and Chelsea London W8 6SA United Kingdom	Autumn	0.3994	7.625	84
1	Best Western Seraphine Kensington Olympia	225 High Street Kensington Kensington and Chelsea London W8 6SA United Kingdom	Spring	0.3578	7.1972	107
2	Best Western Seraphine Kensington Olympia	225 High Street Kensington Kensington and Chelsea London W8 6SA United Kingdom	Summer	0.2245	6.7684	76
3	Best Western Seraphine Kensington Olympia	225 High Street Kensington Kensington and Chelsea London W8 6SA United Kingdom	Winter	0.328	7.9374	107

💡 Insight: L'hotel Best Western Seraphine Kensington Olympia ha un sentiment medio di 0.33 e un punteggio utenti di 7.38 durante l'anno.

In particolare, si seleziona una città, un hotel e si visualizzano le statistiche.

### 3.6 Individuazione delle recensioni anomale

La funzione *anomaly\_detection* ha lo scopo di individuare recensioni anomale, ovvero punteggi significativamente più alti o più bassi rispetto alla media dell'hotel.

```
def anomaly_detection(self, hotelName):
    df = self.spark.df_finale

    # Calcola la media e deviazione standard del punteggio per ogni hotel
    hotel_stats = df.groupBy("Hotel_Name").agg(
        avg("Reviewer_Score").alias("Avg_Score"),
        stddev("Reviewer_Score").alias("Std_Dev_Score")
    )

    # Unisci il trend generale al dataframe originale
    df_with_stats = df.join(hotel_stats, on="Hotel_Name")

    # Identifica recensioni estremamente positive o negative rispetto al trend
    extreme_reviews = df_with_stats.filter(
        abs(col("Reviewer_Score") - col("Avg_Score")) > (2 * col("Std_Dev_Score"))
    )

    df_fin = extreme_reviews.filter(col("Hotel_Name") == hotelName).select(
        "Reviewer_Score", "Avg_Score", "Positive_Review", "Negative_Review"
    ).orderBy(asc("Reviewer_Score"))

    return df_fin
```

Si inizia calcolando per ogni hotel la media e la deviazione standard del punteggio delle recensioni. Questi valori rappresentano il trend generale delle valutazioni per ciascuna struttura.

Successivamente questi dati vengono uniti al DataFrame principale tramite una join, in modo che ogni recensione abbia accesso ai valori medi dell'hotel a cui appartiene.

Le recensioni anomale vengono identificate filtrando i punteggi che si discostano rispetto alla media di più di  $2 * \sigma$ .

Questa condizione serve a individuare recensioni estremamente positive o negative, cioè quelle che si discostano significativamente dal comportamento generale degli utenti.

Infine, si filtrano le anomalie e si restituisce un DataFrame contenente il punteggio della recensione anomala, il punteggio medio dell'hotel e le parti (positiva e negativa) della recensione.

In particolare, questa funzione viene utilizzata all'interno della pagina “Esplora con mappa”, in cui dopo aver selezionato un hotel, viene mostrata la seguente tabella:

Recensioni anomale rispetto alla media per Sheraton Diana Majestic:

	Reviewer_Score	Avg_Score	Positive_Review	Negative_Review
0	2.5	7.6229	appertivo	Our things got stolen from the bags when they were packed and left in the room before we checked out
1	2.5	7.6229	Nice building and interiors	Terrible service Staff has no idea what they're doing
2	3	7.6229	No Positive	Very poor room and sure there were bugs in bed both myself and partner were badly affected
3	3.3	7.6229	No Positive	Poor service

### 3.7 Trend Mensile

La funzione *trend\_mensile* analizza l'andamento del punteggio medio delle recensioni nel tempo per un determinato hotel, restituendo un grafico della variazione mensile.

```
def trend_mensile(self, hotelName):
    df_trend = self.spark.df_finale.filter(col("Hotel_Name") == hotelName)

    #Creazione colonna "YearMonth" che contiene l'anno e il mese
    df_trend = df_trend.withColumn("YearMonth", date_format(col("Review_Date"), "yyyy-MM"))

    # Aggregare per "Hotel_Name" e "YearMonth" e calcolare la media degli score
    trend_df = df_trend.groupBy("Hotel_Name", "YearMonth").agg(
        avg("Reviewer_Score").alias("Average_Score")
    ).orderBy("Hotel_Name", "YearMonth")

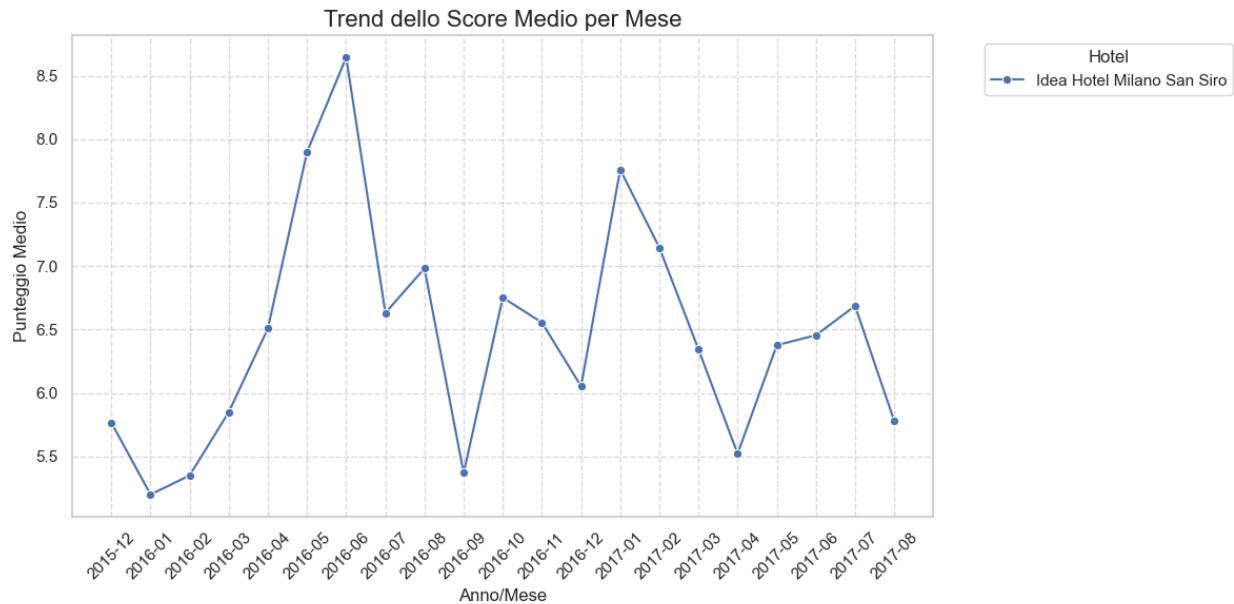
    return utils.graficoTrend(trend_df, True)
```

Inizialmente si opera filtrando i dati per il nome dell'hotel specificato, creando una nuova colonna “Year\_Month” che aggrega la data della recensione al formato anno-mese (yyyy-MM). Successivamente si raggruppa per hotel e mese, calcolando la media dei punteggi delle recensioni. Infine si ordina a livello temporale.

Il risultato viene passato alla funzione *graficoTrend* presente in **utils** che converte il dataframe in Pandas e si utilizza Seaborn per creare un grafico a linee che mostra l'evoluzione del punteggio medio nel tempo.

In particolare, questa funzione viene utilizzata all'interno della pagina “Esplora con mappa”, in cui dopo aver selezionato un hotel, viene mostrato il seguente grafico:

Trend Mensile per Idea Hotel Milano San Siro:



### 3.8 Statistiche generali degli hotel

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

    res = df.groupBy("Hotel_Name").agg(
        count("*").alias("Total_Reviews"),
        sum(when(col("Reviewer_Score") >= 6, 1).otherwise(0)).alias("Total_Positive_Reviews"),
        sum(when(col("Reviewer_Score") < 6, 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")
    )
    return res
```

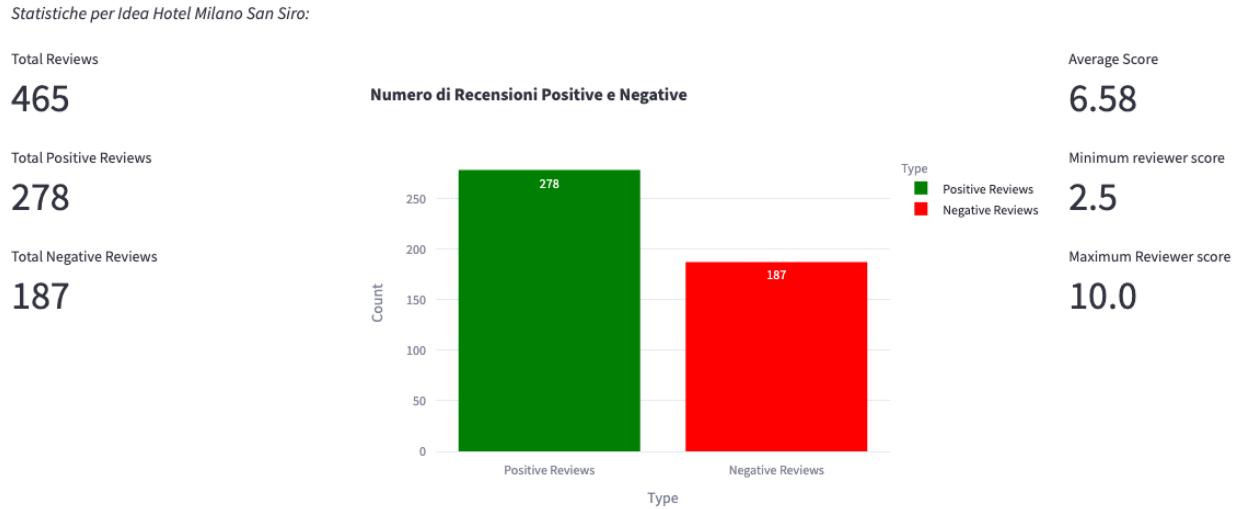
La funzione *hotelStatistics* ha l'obiettivo di aggregare e sintetizzare le principali statistiche sulle recensioni degli hotel presenti nel dataset.

Questa funzione restituisce un dataframe aggregato in cui, per ciascun hotel, vengono calcolati diversi indicatori chiave.

In particolare, tra le statistiche principali, vengono calcolati il numero totale di recensioni, il numero di recensioni positive e negative basandosi su una soglia di punteggio (positive se  $\geq 6$ , negative se  $< 6$ ), oltre ai punteggi massimo, minimo e medio assegnati dagli utenti. Inoltre,

vengono estratte le coordinate geografiche dell'hotel, assumendo che siano uniche per struttura.

Questa funzione viene utilizzata all'interno della pagina "Esplora con mappa", in cui dopo aver selezionato un hotel, vengono mostrate le seguenti statistiche:



### 3.9 Analisi del sentiment (RoBERTa)

La funzione `analyze_hotel_sentiment` è progettata per calcolare il sentiment medio delle recensioni di un hotel specifico, utilizzando il modello di deep learning RoBERTa per l'analisi del sentiment.

```
def analyze_hotel_sentiment(self, hotel_name):
    """Calcola il sentiment medio delle recensioni di un hotel."""
    print(f"Analizzando il sentiment di {hotel_name}")
    return self.sentiment_analyzer.analyze_hotel_sentiment(hotel_name, self.spark.df_finale)
```

Il metodo prende in input il nome dell'hotel e passa il dataset delle recensioni alla funzione `analyze_hotel_sentiment` della classe RoBERTa\_Sentiment, che si occupa dell'analisi. Questa classe sfrutta il modello pre-addestrato `cardiffnlp/twitter-roberta-base-sentiment`, specializzato per il sentiment analysis sui testi brevi, come descritto precedentemente.

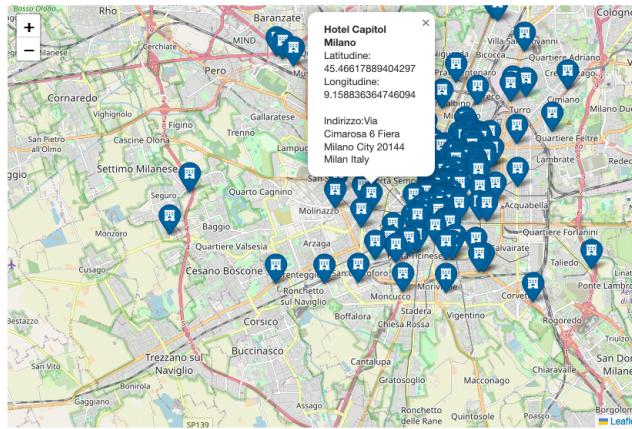
In breve, vengono prelevate tutte le recensioni dell'hotel estraendo separatamente la parte positiva e la parte negativa di ogni recensione. Ogni segmento viene elaborato tramite una funzione (`analyze_review_sentimen`) che utilizza appunto il modello RoBERTa per classificare il testo.

Si converte così il sentiment di ogni recensione in un valore numerico (positivo = 1, neutrale = 0, negativo = -1) e viene calcolata la media complessiva del sentiment dell'hotel. In base al valore medio ottenuto, il sentiment viene categorizzato in:

- Molto positivo: ( $\geq 0.4$ )

- Piuttosto positivo: ( $0.2 \leq score \leq 0.4$ )
- Neutrale: ( $-0.2 \leq score \leq 0.2$ )
- Piuttosto negativo: ( $-0.3 \leq score \leq -0.2$ )
- Molto negativo: ( $\leq 0.4$ )

La funzione viene utilizzata all'interno della pagina “Esplora con mappa”. In particolare, dopo aver selezionato un hotel si può vedere il sentimento medio delle recensioni di quest'ultimo:



## Informazioni

Sentiment Complessivo

Molto Positivo



### 3.10 Creazione del summary delle recensioni di un hotel

```
def sumReviews(self, hotel_name):
    return self.summaryLLM.getSummary(hotel_name)
```

La funzione è estremamente semplice in quanto richiama la classe *summaryLLM* descritta nel capitolo precedente.

Essa è utilizzata nella pagina *Esplora con mappa*:

The Hotel Boutique Duomo offers an exceptional stay for short breaks but is located a bit outside the city center. The friendly and helpful staff made your stay memorable, though some basic facilities like TV and coffee maker didn't work. The spacious room was cozy, and breakfast was top-notch with no issues. However, the location's proximity to the metro station isn't ideal, and noise from the road outside the window made it challenging at night. Despite this, overall, it wasn't a central spot.

The hotel lacked Wi-Fi, which could have been better for your comfort during the trip. The room was large, but the staff tried their best to make things easier. Unfortunately, there were some snags like a cold shower and poor water pressure in multiple rooms, especially when others were showering simultaneously.

After staying two nights, the guest faced a challenge with the hotel's air conditioning not working properly during your stay. Waiting for the shower to warm up was frustrating due to the lack of proper heating capacity. The staff tried their best but consistently couldn't make it easier, and the manager suggested further improvements could be worth trying.

The hotel has some positive aspects: modern design, clean rooms, and friendly staff. However, it lacks a central location and personal touches like a mini bar with no adjustable lights at night. Overall, while it had room for improvement, many users found the stay memorable and appreciated the comfort of the spacious room.

### 3.11 Ottenerne gli hotel di una città in base ai tag

```
def get_hotels_by_tag(self, city, tag):
    df_ret = self.spark.df_finale.filter(col("Hotel_Address").contains(city) & array_contains(col("Tags"),
tag)).groupBy("Hotel_Name", "Hotel_Address").agg(
    avg("Reviewer_Score").alias("avg_score")
).orderBy(desc("avg_score"))

    return df_ret
```

La funzione estrae le recensioni degli hotel che si trovano nella città specificata selezionando quelle che contengono il tag scelto.

Raggruppa i dati per nome dell'hotel (e indirizzo), calcola la media dei punteggi delle recensioni per ciascun hotel e li inserisce in una nuova colonna.

Un suo utilizzo è presente nella pagina *Esplora per tag* dove dopo aver selezionato la città, viene selezionato un tag per mostrare i migliori hotel (in base al punteggio medio):



#### Hotel ordinati secondo lo score medio

Tabella contenente gli hotel della città selezionata associati al tag selezionato, ordinati in modo decrescente.

	Hotel_Name	Hotel_Address	avg_score
0	Hotel Casa Camper	Elisabets 11 Ciutat Vella 08001 Barcelona Spain	9.7215
1	H10 Casa Mimosa 4 Sup	Pau Claris 179 Eixample 08037 Barcelona Spain	9.6620
2	Hotel The Serras	Passeig de Colom 9 Ciutat Vella 08002 Barcelona Spain	9.6157
3	Catalonia Magdalenes	Magdalenes 13 15 Ciutat Vella 08002 Barcelona Spain	9.5815
4	The Wittmore Adults Only	Riudarenes 7 Ciutat Vella 08002 Barcelona Spain	9.5354
5	Hotel Palace GL	Gran Via de les Corts Catalanes 668 Eixample 08010 Barcelona Spain	9.4955
6	Catalonia Square 4 Sup	Ronda Sant Pere 9 Eixample 08010 Barcelona Spain	9.4879
7	Mercer Hotel Barcelona	Dels Lledo 7 Ciutat Vella 08003 Barcelona Spain	9.4815
8	Aparthotel Arai 4 Superior	Avinyo 30 Ciutat Vella 08002 Barcelona Spain	9.4800
9	The One Barcelona GL	277 Carrer de Provena e Eixample 08037 Barcelona Spain	9.4653
10	Hotel DO Pla a Reial G L	Pla a Reial 1 Ciutat Vella 08001 Barcelona Spain	9.4604

### 3.12 Ricerca degli hotel vicini ad una posizione geografica

```
def get_nearby_hotels(self, user_lat, user_lng, max_distance):
    # Registra la funzione come UDF
    haversine_udf = udf(lambda lat, lng, user_lat, user_lng: utils.haversine(lat, lng, user_lat, user_lng))
    df_hotels = self.spark.df_finale
    return df_hotels.withColumn(
        "distance", haversine_udf(col("lat"), col("lng"), lit(user_lat), lit(user_lng)))
    .filter(col("distance") <= max_distance)
```

Tale funzione permette di trovare gli hotel più vicini ad una determinata posizione geografica (identificata dalla latitudine e dalla longitudine passata in input), calcolando la distanza tra la posizione e le coordinate dell'hotel.

Viene registrata una UDF con la funzione *haversine* di *utils.py* (precedentemente descritta) che permette di calcolare la distanza tra l'hotel e la posizione di interesse aggiungendo una nuova colonna “*distance*” al dataframe.

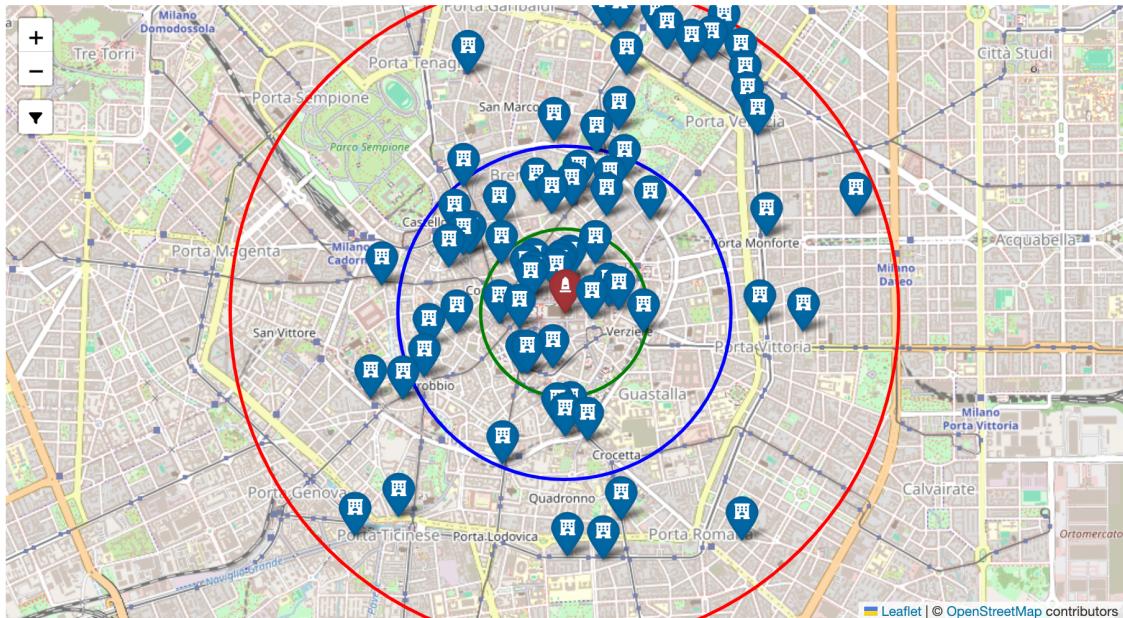
Infine, vengono mantenuti solo gli hotel che si trovano ad una distanza minore o uguale ad una distanza stabilita; quindi, viene restituito un DataFrame Spark con gli hotel che si trovano entro il raggio massimo specificato, includendo una colonna aggiuntiva “*distance*”.

Un suo utilizzo è presente nella pagina *Esplora per punto di interesse* dove scelta una città, scelto un punto di interesse (definiti in un file .csv), vengono mostrati gli hotel entro una certa distanza:

### Hotel vicini al punto di interesse

-  Cerchio Verde: Hotel entro 500 metri dal punto di interesse scelto.
-  Cerchio Blu: Hotel tra 500 e 1000 metri dal punto di interesse scelto.
-  Cerchio Rosso: Hotel tra 1000 e 2000 metri dal punto di interesse scelto.

 Filtra gli hotel per distanza direttamente sulla mappa:



### 3.13 Confronto andamento mensile degli Hotel

```
def trend_mensile_compare(self, dataframe):

    #Creazione colonna "YearMonth" che contiene l'anno e il mese
    df_trend = dataframe.withColumn("YearMonth", date_format(col("Review_Date"), "yyyy-MM"))

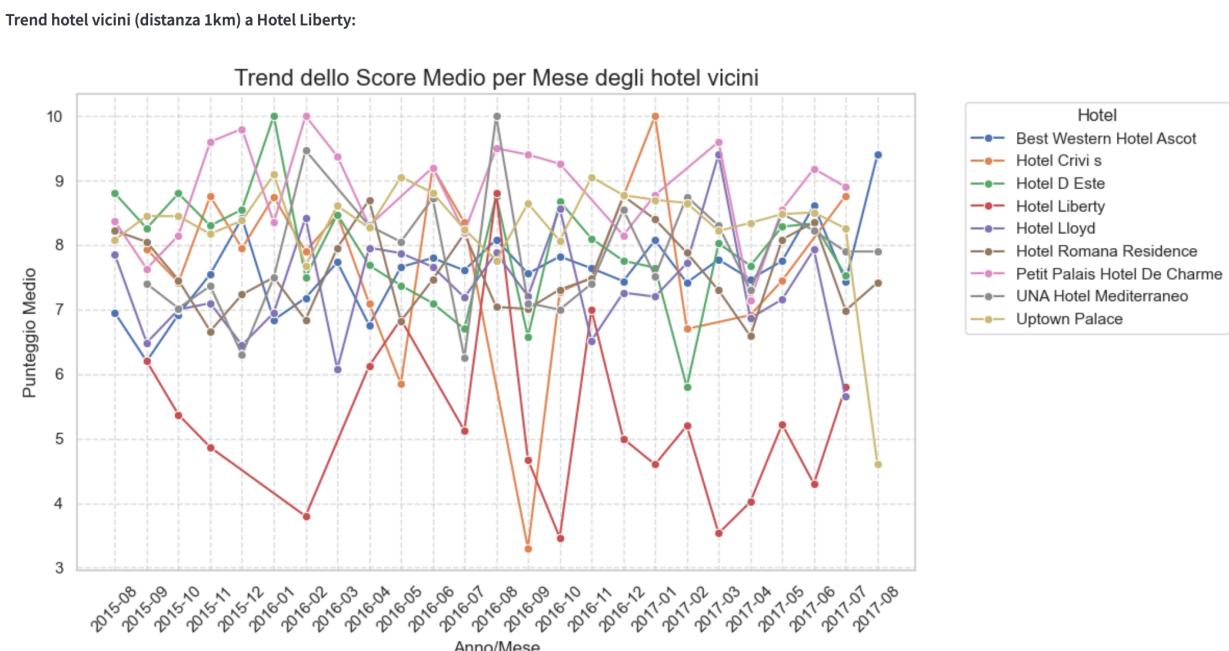
    # Aggregare per "Hotel_Name" e "YearMonth" e calcolare la media degli score
    trend_df = df_trend.groupBy("Hotel_Name", "YearMonth").agg(
        avg("Reviewer_Score").alias("Average_Score")
    ).orderBy("Hotel_Name", "YearMonth")

    return utils.graficoTrend(trend_df, False)
```

La funzione permette di analizzare l'andamento mensile degli hotel analizzando le recensioni contenute nel dataframe ricevuto in input.

Questa funzione è utilizzata, in particolare, per effettuare un confronto tra un hotel e i suoi vicini visualizzando un grafico in cui sono presenti i diversi andamenti dei punteggi medi.

Un suo utilizzo è presente nella pagina *Esplora con mappa*:



### 3.14 Tag più utilizzati

```
def get_most_used_tags(self):
    # Esplosione dell'array "tags" in valori singoli
    tags_df = self.spark.df_finale.select(explode(col("tags")).alias("tag"))
    # Contare la frequenza di ogni tag e ordinarli in ordine decrescente
    tags_count_df = tags_df.groupBy("tag").agg(count("*").alias("count")).orderBy(desc("count"))
    tags = tags_count_df.select("tag")

    return tags
```

Tale funzione permette di identificare i tag più utilizzati nelle recensioni degli hotel.

La colonna “Tags” contiene liste di stringhe (tag) per ogni recensione. Attraverso la funzione *explode* si trasforma ogni elemento dell’array in una nuova riga con una sola colonna.

Si raggruppano i dati per tag e si effettua un conteggio, ordinando in modo decrescente.

Un esempio di utilizzo è nella pagina *Esplora per tag* dove vi è la possibilità di scegliere il tag da sfruttare per un successivo filtro sugli Hotel.

### 3.15 Conteggio recensioni per ogni nazione

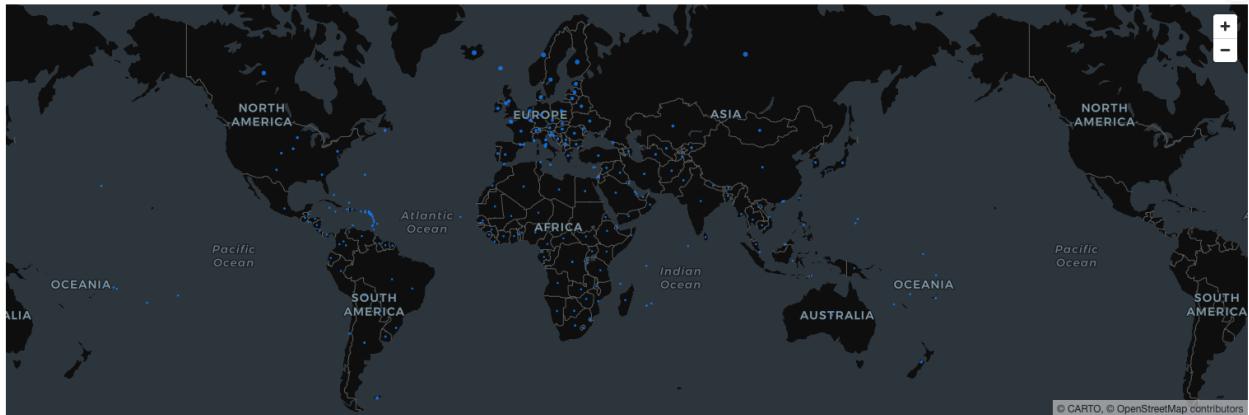
```
def nationality_review_analysis(self, min_reviews=2):

    df = self.spark.df_finale
    nationality_reviews = df.groupBy("Reviewer_Nationality") \
        .agg(
            count("*").alias("review_count")
        ).filter(col("review_count") >= min_reviews)
    return nationality_reviews
```

Tale funzione permette di ottenere un dataframe contenente “nazionalità” e “numero di recensioni”.

È utilizzata nella pagina *Recensione-Nazionalità* per ottenere la lista delle nazioni per cui esistono almeno un numero minimo di recensori di quella nazionalità e mostrare così una mappa:

**Mappa delle nazionalità dei recensori**



### 3.16 Filtrare recensioni per nazionalità

```
def get_reviews_for_nationality(self, nationality):
    # Filtra i dati per la nazionalità selezionata
    return self.spark.df_finale.filter(col("Reviewer_Nationality") == nationality).select("Hotel_Name",
    "Review_Date", "Reviewer_Nationality", "Reviewer_Score", "Negative_Review",
    "Review_Total_Negative_Word_Counts", "Positive_Review", "Review_Total_Positive_Word_Counts", "Tags", )
```

La funzione permette di filtrare le recensioni in base alla nazionalità del recensore.

Questa funzione, molto semplice, è utilizzata nella pagina *Recensione-Nazionalità* per ottenere:

#### Dati delle recensioni per nazionalità

Seleziona una nazionalità:

United Arab Emirates

#### Analisi per United Arab Emirates

Dati delle recensioni:

	Hotel_Name	Review_Date	Reviewer_Nationality	Reviewer_Score	Negative_Review	Review_Total_Negative_Word_Counts	Positive_Review
0	The Bloomsbury Hotel	2016-04-09	United Arab Emirates	10	The croissants were a little too hard and crusty	11	The welcome gift was a nice touch ar
1	The Bloomsbury Hotel	2016-03-01	United Arab Emirates	7.5	service at breakfast was appalling they said they would correct breakfast charges wh	33	the lobby is a very nice place for drin
2	The Bloomsbury Hotel	2015-09-15	United Arab Emirates	8.8	Very small closet pedi shower not available More Arabic channel TV	13	Location is near many destinations w
3	The Bloomsbury Hotel	2016-08-06	United Arab Emirates	9.6	No Negative	0	The staff were helpful The food was g
4	The Bloomsbury Hotel	2016-01-10	United Arab Emirates	10	Nothing	2	Everything
5	The Bloomsbury Hotel	2015-12-29	United Arab Emirates	10	Stay was too short	6	Fantastic location and good price
6	Saint Georges Hotel	2017-01-03	United Arab Emirates	5.8	The rooms were dirty and we found blood spots in the upper bed sheets	15	Only the location and the young guy
7	Saint Georges Hotel	2016-10-31	United Arab Emirates	7.1	Cleanliness and facility wise it is not a proper 4 star establishment	13	Location is great on Regent and Oxfo
8	Saint Georges Hotel	2016-06-21	United Arab Emirates	8.3	It is a little bit an old hotel it is not the type of hotel to enjoy its facilities but the locat	27	The location is perfect It is between r
9	Saint Georges Hotel	2017-03-13	United Arab Emirates	7.5	Wifi did not really work properly too slow	9	Location was fantastic