Progetto Big Data:
Analisi di recensioni su Hotel di lusso

Mattia Corigliano mat. 264444,
Paolo Costa mat. 264586

13 January 2025

1 Dataset

Perché Analizzare Questo Dataset?

Nel competitivo mondo dell'ospitalità di lusso, **capire i desideri e le esigenze dei clienti** è la chiave per eccellere. Questo dataset, derivato da *Booking.com*, offre una straordinaria opportunità per esplorare a fondo il panorama degli hotel di lusso in Europa. Con **515.000 recensioni** dettagliate lasciate da ospiti di tutto il mondo, possiamo scoprire ciò che rende un soggiorno davvero *indimenticabile*. Le recensioni coprono **1.493 hotel di lusso** distribuiti in tutta Europa, fornendo un quadro completo delle esperienze degli ospiti, dalle più memorabili alle meno soddisfacenti. Questo dataset è una miniera d'oro per chi vuole approfondire l'esperienza nel settore dell'ospitalità.

Cosa Contiene il Dataset?

Questo dataset non è solo un elenco di recensioni; è una finestra sul mondo delle preferenze dei viaggiatori. Include:

- **Nome** e **indirizzo** degli hotel, per comprendere la distribuzione geografica.
- Recensioni **positive e negative**, per analizzare i punti di forza e debolezza.
- La **nazionalità dei recensori**, per identificare trend culturali e regionali.
- I **punteggi** assegnati, utili per confronti quantitativi.
- **Tag e tempi di recensione**, per studiare i motivi e i contesti delle valutazioni.

Quali Analisi Possiamo Fare?

Questo dataset permette molte possibilità di analisi tra cui:

- **Sentiment Analysis:** Per capire quali parole e frasi influenzano maggiormente le valutazioni degli ospiti
- **Correlazioni:** Esamina del legame tra la nazionalità dei recensori e i punteggi assegnati per svelare preferenze culturali.
- **Clustering e Raccomandazioni:** Raggruppando gli hotel in base a caratteristiche comuni per capire le esigenze specifiche dei clienti.
- **Visualizzazioni Informative:** Trasformare i dati in mappe interattive e grafici accattivanti per comunicare i risultati in modo chiaro e coinvolgente.

Perché È Importante?

Ogni ospite ha una storia da raccontare, e ogni recensione fa capire l'esperienza vissuta. Questo dataset ci permette di:

- Supportare gli albergatori nel comprendere meglio i bisogni dei propri ospiti e nel prendere decisioni strategiche.

- Ottimizzare i servizi offerti dagli hotel, migliorando l'accoglienza e la fidelizzazione.
- Offrire ai viaggiatori un'esperienza personalizzata e in linea con le loro aspettative.

Grazie a questa analisi, possiamo contribuire a trasformare un semplice soggiorno in un'esperienza da ricordare per sempre.

Per scoprirne di Più

Consulta il dataset su Kaggle cliccando qui.

2 Implementazione

L'idea del progetto era permettere, attraverso un semplice sito web, la visualizzazione, l'interazione e l'analisi di query sul dataset in esame. Data l'elevata dimensione del dataset in termini di volume e varietà di dati, la scelta del framework per l'elaborazione è ricaduta su Apache Spark, un sistema di elaborazione open source distribuito, utilizzato in genere con i carichi di lavoro per i Big Data. Per query analitiche rapide, utilizza la cache in memoria e l'esecuzione ottimizzata delle query su dati di qualsiasi dimensione.
L'implementazione del framework avviene nel linguaggio Python, supportato dal motore di esecuzione delle analisi, tramite PySpark, l'API di Python per Apache Spark. Con PySpark, è possibile scrivere comandi in Python e simili a SQL per manipolare e analizzare dati in un ambiente di elaborazione distribuito. Utilizzando PySpark, i data scientist possono manipolare dati, costruire pipeline di machine learning e ottimizzare modelli. Il sistema ha un flusso di esecuzione che parte dalla definizione della **SparkSession**, che rappresenta il punto di ingresso per l'utilizzo di Spark.

```
def get_spark_session():
return SparkSession.builder
    .master("local[*]")
    .appName("Hotel Review Statistics")
    .config("spark.hadoop.io.native.lib.available", "false")
    .config("spark.driver.extraJavaOptions", "-Djava.security.manager=allow")
    .config("spark.executor.extraJavaOptions", "-Djava.security.manager=allow")
    .config("spark.driver.memory", "8g")
    .config("spark.executor.memory", "8g")
    .config("spark.local.dir", "/tmp/spark-temp")
    .config("spark.executor.instances", "1")
    .config("spark.sql.execution.arrow.pyspark.enabled", "true")
    .config("spark.sql.execution.arrow.maxRecordsPerBatch", "1000")
    .getOrCreate()
```

Tramite questa è possibile generare il **DataFrame**, una struttura dati simile a una tabella in un database relazionale, contenente i dati derivati dal dataset. Inoltre, attraverso la SparkSession è possibile anche generare query in formato SQL che verranno ottimizzate dal motore interno. Tramite il modulo **SparkSQL** di Spark, che fornisce un'interfaccia per eseguire SQL su DataFrame, è stato possibile sviluppare, implementare e testare le diverse query utili all'analisi e alla visualizzazione dei dati. La libreria *pyspark.sql.functions* di SparkSQL in Python facilita l'esecuzione di query sui DataFrame, permettendo operazioni come selezioni, filtraggi e aggregazioni in modo efficiente. Una volta sviluppato il backend della nostra applicazione di data analysis e data visualization, si è passati al frontend, scegliendo il framework più adatto a questo ambiente. Dunque la dashboard è stata sviluppata utilizzando **Streamlit**, una libreria open-source in Python che facilita la creazione di applicazioni web interattive per progetti di data science e machine learning. Streamlit consente di trasformare script Python in applicazioni web intuitive senza la necessità di competenze avanzate in sviluppo web, rendendo più agevole la visualizzazione e l'analisi dei dati. Un esempio significativo dell'utilizzo di Streamlit in applicazioni di data analysis è illustrato nel paper intitolato *A Streamlit Tool For Airbnb Data Analysis And Visualization Insights* [2]. Questo studio dimostra come Streamlit possa essere impiegato per costruire dashboard dinamiche e interattive, facilitando l'analisi di dati complessi in modo accessibile ed efficiente. L'implementazione di Streamlit in questo contesto offre numerosi vantaggi, tra cui:

- **Rapidità di sviluppo**: Streamlit consente di creare applicazioni web funzionali con poche righe di codice, accelerando il processo di sviluppo.
- **Interattività**: Gli utenti possono interagire con i dati attraverso widget intuitivi, migliorando l'esperienza di analisi.
- **Integrazione con l'ecosistema Python**: La compatibilità con librerie come Pandas, Matplotlib e Plotly facilita l'implementazione di funzionalità avanzate di data analysis e visualizzazione.

Inoltre, l'architettura di Streamlit supporta l'aggiornamento in tempo reale dei dati, garantendo che le informazioni presentate siano sempre aggiornate. Questa caratteristica è particolarmente utile in scenari in cui i dati sono soggetti a frequenti variazioni. In sintesi, l'utilizzo di Streamlit per l'implementazione della dashboard ha permesso di creare un'interfaccia utente interattiva e dinamica, facilitando l'analisi dei dati attraverso pagine dedicate a diverse query, e migliorando l'efficienza complessiva del processo analitico.

3 Preprocessing

Prima di poter manipolare i dati e studiarli per ottenerne delle statistiche interessanti è stato necessario effettuare delle modifiche al dataset originale.
Innanzitutto è stato definito lo schema del dataset per evitare che l'inferenza effettuata da SparkSession.read.csv() risultasse errata.
Sono state corrette poi le colonne:

- **days_since_review**: sono stati rimossi le stringhe "day" e "days" accanto ai numeri e convertito la colonna in intero per renderla utilizzabile come valore numerico;

```
reviews = spark.read.csv("data/Hotel_Reviews.csv", schema=schema, header=True)

#days_since_review column has string values, so we need to convert it to integer
reviews = reviews\
    .withColumn("days_since_review", regexp_replace("days_since_review", " days| day", "")\
    .cast(IntegerType()))
```

- **Review_Date**: la formattazione delle date non seguiva lo standard anno-mese-giorno, per cui è stata convertita anche per facilitarne l'ordinamento.

```
#Review_Date column has string values and bad format, so we need to convert it
reviews = reviews.withColumn("Review_Date", to_date("Review_Date", "M/d/yyyy"))
```

- **Negative_Review, Positive_Review**: l'autore del dataset ha suddiviso la recensione in positiva e negativa inserendo rispettivamente "No Positive" e "No Negative" qualora le rispettive parti fossero mancanti. Tuttavia questi valori sono inutili e perciò sono stati eliminati.

```
#fixing empty postive and negative reviews rows by
#erasing respectively "No Negative" and "No Positive" values
reviews = reviews.withColumn("Negative_Review", when(col("Negative_Review")=="No Negative", "")\
    .otherwise(col("Negative_Review")))
reviews = reviews.withColumn("Positive_Review", when(col("Positive_Review")=="No Positive", "")\
    .otherwise(col("Positive_Review")))
```

- **Tags**: la colonna era costituita da un array in formato di stringa perciò è stato convertito in un array effettivo.

```
#Tags colum converted in array
reviews = reviews.withColumn("Tags_Cleaned", regexp_replace(col("Tags"), r"[\[\]']", ""))
reviews = reviews.withColumn("Tags_Array", split(col("Tags_Cleaned"), ",\s*"))\
    .drop("Tags_Cleaned", "Tags")
reviews = reviews.withColumn("Tags_Array", expr("transform(Tags_Array, x -> trim(x))"))
```

Sono stati poi conteggiati i valori nulli ed è emerso che per alcuni hotel (per un totale di 3268 righe) mancavano latitudine e longitudine all'interno delle rispettive colonne (importanti per la visualizzazione su mappa). Per questo motivo sono stati ricercati gli hotel ed aggiornate le informazioni mancanti:

```
cols = [col(c) for c in reviews.columns]
filter_expr = reduce(lambda a, b: a | b.isNull(), cols[1:], cols[0].isNull())

all_null_values = reviews.filter(filter_expr)
print("all null valuses count: "+str(all_null_values.count()))


my_list = [("Mercure Paris Gare Montparnasse", 48.839752, 2.323791),
        ("Holiday Inn Paris Montmartre", 48.889212, 2.333239),
        ("Maison Albar Hotel Paris Op ra Diamond", 48.8753896, 2.3233823),
        ("NH Collection Barcelona Podium", 41.3918780, 2.1779369),
        ("City Hotel Deutschmeister", 48.2210163, 16.3666115),
        ("Cordial Theaterhotel Wien", 48.2097133, 16.3514418),
        ("Fleming s Selection Hotel Wien City", 48.2095233, 16.3535636),
        ("Hotel Park Villa", 48.2336527, 16.3458807),
        ("Hotel Daniel Vienna", 48.1889864, 16.3837793),
        ("Renaissance Barcelona Hotel", 41.3928700, 2.1673869),
        ("Roomz Vienna", 48.1868055, 16.4205255),
        ("Austria Trend Hotel Schloss Wilhelminenberg Wien", 48.2197338, 16.2855277),
        ("Hotel Advance", 41.3834277, 2.1629281),
        ("Derag Livinghotel Kaiser Franz Joseph Vienna", 48.2461265, 16.3412116),
        ("Hotel City Central", 48.2137371, 16.3799296),
        ("Hotel Atlanta", 48.2205819, 16.3557971),
        ("Hotel Pension Baron am Schottentor", 48.2168996, 16.3599271)]

for hotel in my_list:
    reviews = reviews.withColumn("lat", when(col("Hotel_Name") == hotel[0], hotel[1])
                        .otherwise(col("lat")))
                     .withColumn("lng", when(col("Hotel_Name") == hotel[0], hotel[2])
                        .otherwise(col("lng")))


reviews.write.save("data.parquet", format="parquet")
```

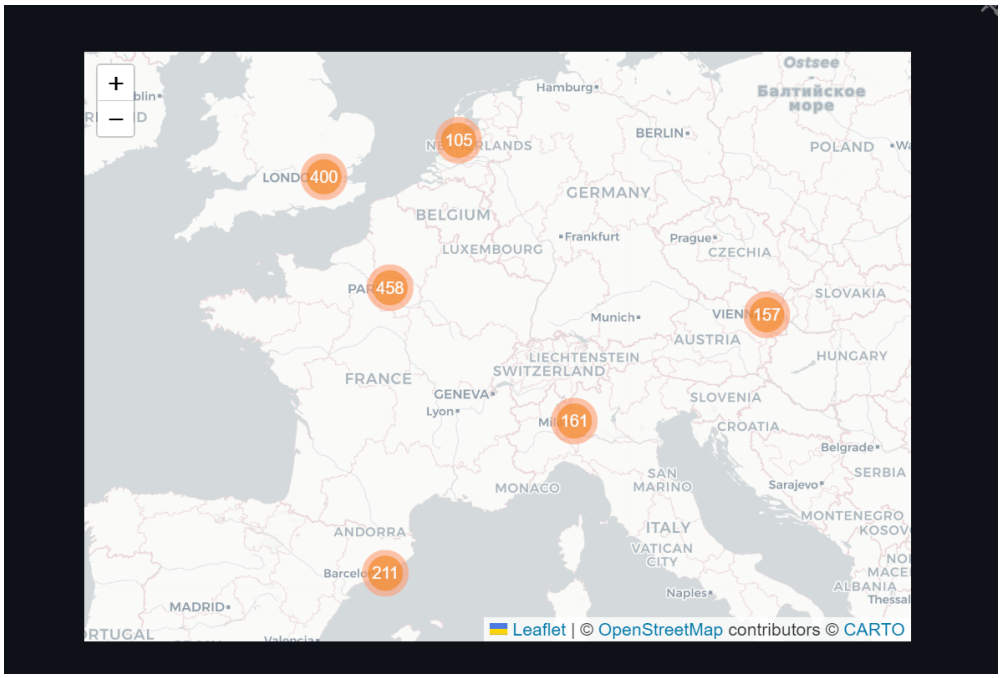
Infine sono stati salvati i dati in un file parquet, un formato column-based scelto per migliorare, rispetto al csv originario, le performance durante il caricamento del file, esecuzione di query e compressione dei dati su disco.

4 Dashboard di visualizzazione dei dati

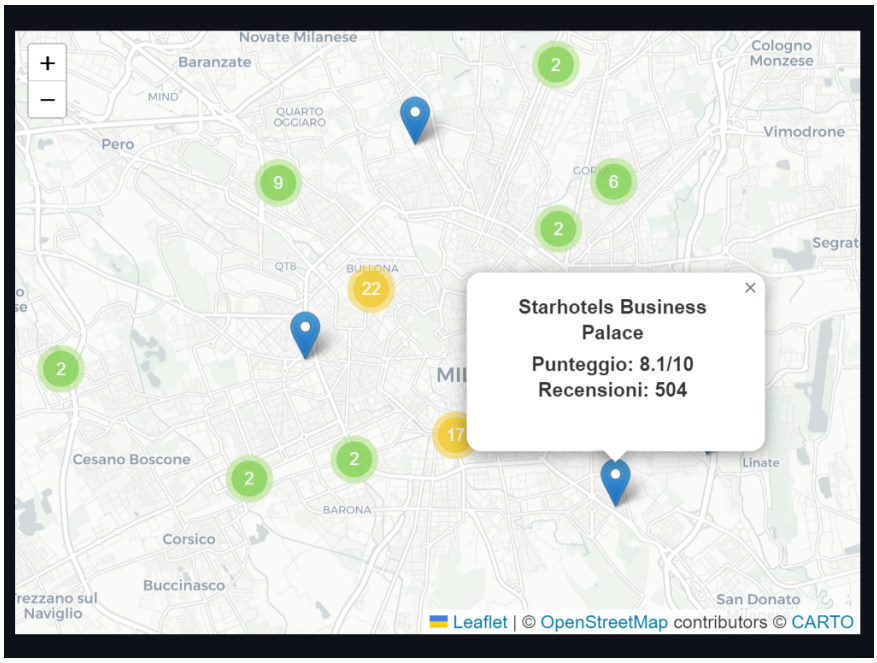
In questa sezione, analizzeremo le principali query implementate nella dashboard. Ogni query è stata progettata per estrarre e visualizzare informazioni rilevanti dal dataset, fornendo insight dettagliati e personalizzabili. Esamineremo come queste query consentano di rispondere a domande specifiche e supportino l'analisi interattiva dei dati, ottimizzando il processo decisionale.
La dashboard comprende diverse pagine, ciascuna dedicata a specifiche query, permettendo agli utenti di esplorare e analizzare i dati in modo strutturato e approfondito.

Homepage

A primo impatto risulta fin da subito visibile una ***mappa***: rappresenta gli Hotel in esame attraverso dei ping sull'indirizzo di quest'ultimi. A mappa ingrandita, i ping si uniscono andando a mantenere il numero di Hotel in quella area.



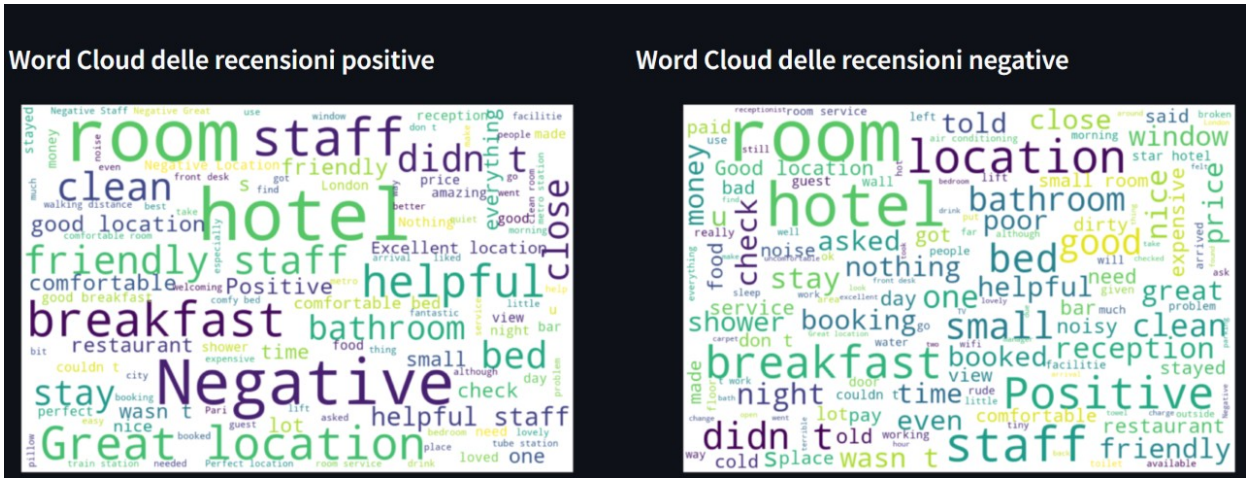
Ingrandendo nella posizione di un Hotel e cliccando il corrispettivo ping dell'Hotel è possibile visualizzare: Nome dell'Hotel, punteggio ed il numero di recensioni.



Ottenuta dalla seguente query:

```
return spark_session.sql("SELECT Hotel_Name, count(*) as Reviews,
                         first(Average_Score) as Average_Score,
                         first(lat) as lat, first(lng) as lng
                         FROM Hotel_Reviews GROUP BY Hotel_Name").toPandas()
```

Successivamente è presenta la ***word cloud***, ovvero la rappresentazione visiva delle parole più frequenti nelle recensioni, in cui la dimensione di ciascuna parola è proporzionale alla sua frequenza o importanza. Data la divisione nel dataset della review in 2 colonne differenti: *Negative_Review* e *Positive_Review*, sono presenti le rispettive nuvolette:



La query utilizzata per eseguere questa funzione è:

```
dfRet=reviews.withColumn("Review",
                         F.concat(F.col("Positive_Review"),
                         F.lit(" "),
                         F.col("Negative_Review")))
return dfRet.select("Review", "Reviewer_Score")
```

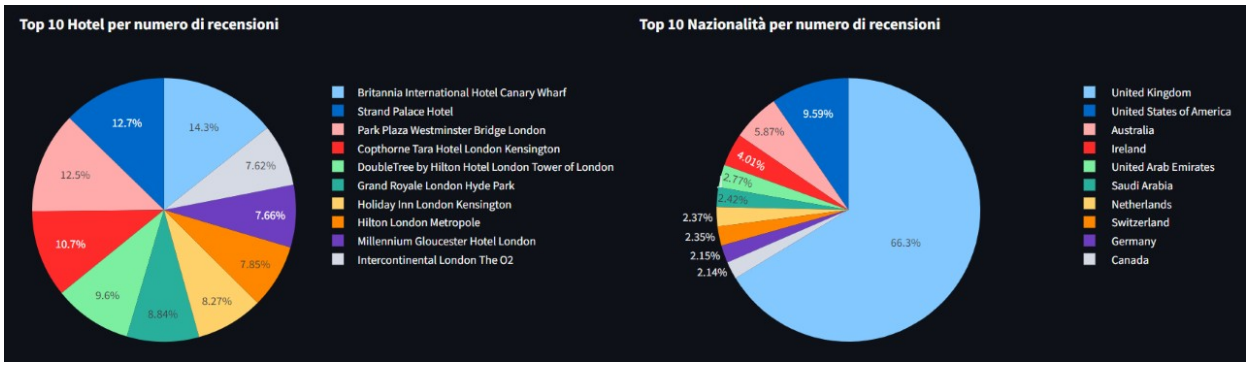
Seguono dei ***grafici a torta***, così da rendere chiare le proporzioni tra le diverse categorie. Grazie a questi grafici, è possibile identificare rapidamente la distribuzione dei dati e le relazioni tra i diversi elementi del dataset in modo intuitivo e immediato. Avremo 4 grafici per 4 query differenti ovvero:

- Top 10 Hotel per numero di recensioni,

```
df = df.groupby(dfCopia.Hotel_Name).count() \
        .withColumnRenamed("count","Totale recensioni") \
        .sort("Totale recensioni", ascending=False) \
        .select("Hotel_Name", "Totale recensioni")
```

- Top 10 Nazionalità per numero di recensioni

```
df = (df.groupby(dfCopia.Reviewer_Nationality).count()
       .withColumnRenamed("count", "Totale recensioni")
       .sort("Totale recensioni", ascending=False)) \
.select("Reviewer_Nationality", "Totale recensioni")
```

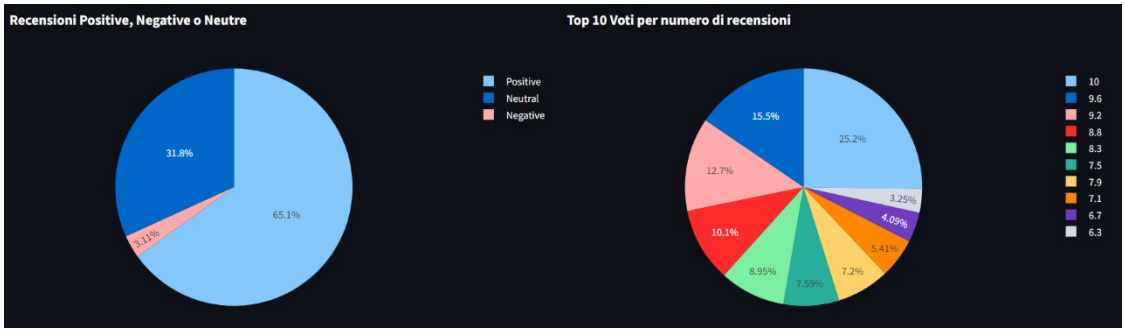


- Recensioni Positive, Negative o Neutre

```
df = df.withColumn(
    'Review_Type',
    F.when(dfCopia['Reviewer_Score'] >= 8, 'Positive')
    .when(dfCopia['Reviewer_Score'] <= 4.5, 'Negative')
    .otherwise('Neutral'))
df= df.groupBy('Review_Type').agg(
    F.count('*').alias('Totale recensioni'))
```

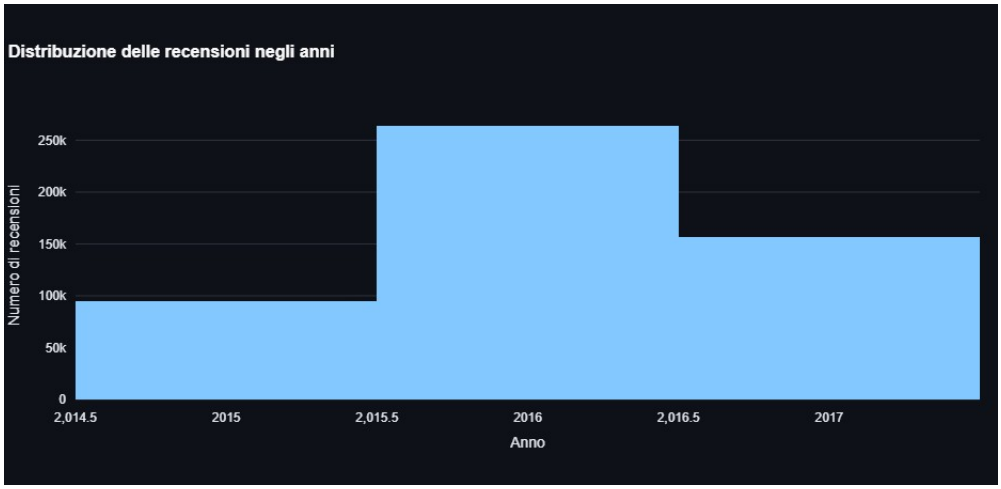
- Top 10 Voti per numero di recensioni

```
df = (df.groupby(dfCopia.Reviewer_Score).count()
          .sort("count", ascending=False)
          .withColumnRenamed("count", "Totale recensioni"))
```



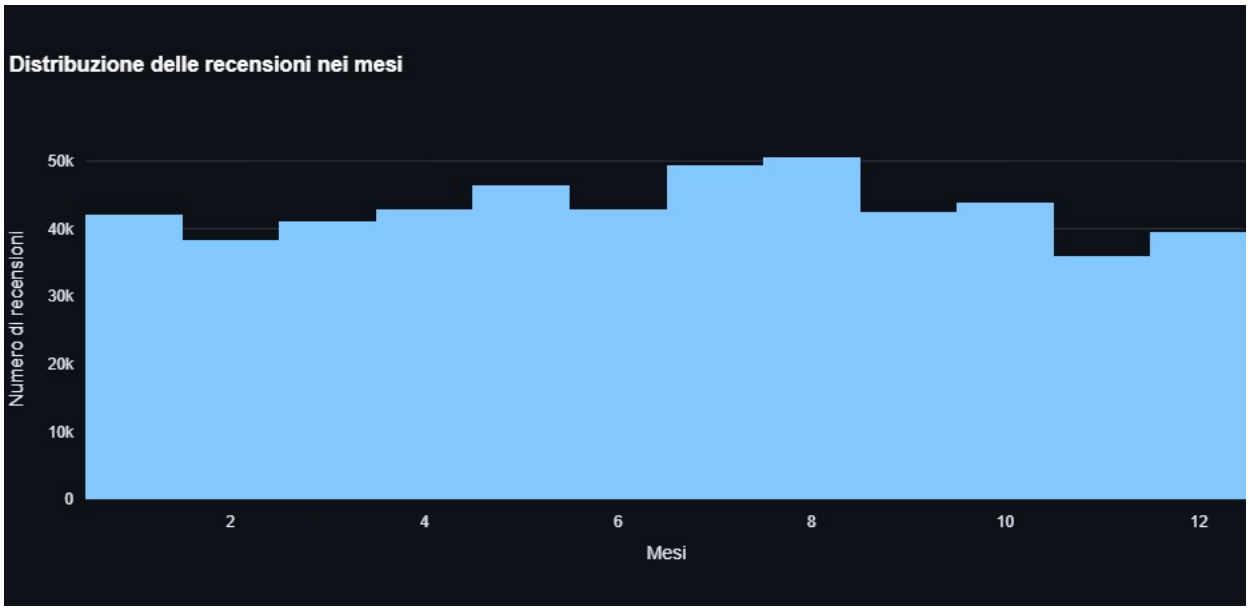
Ed infine gli ***istogrammi***, che permettono di osservare la frequenza di occorrenza di diverse fasce di valori, evidenziando pattern e tendenze nei dati. Avremo: Dalla seguente query l'istogramma in figura:

```
if hotel is not None:
    dfUtils = reviews.filter(reviews.Hotel_Name == hotel)
else:
    dfUtils = reviews

res = dfUtils.groupBy(F.year("Review_Date").alias("Year"))
        .agg(F.count("*").alias("Review_Count"))
        .orderBy("Year")
```



Dal grafico possiamo osservare che il numero di recensioni per anno presenta un picco significativo nel 2016, seguito da una diminuzione nell'anno successivo. Questo potrebbe indicare un aumento di popolarità o visibilità delle strutture nel 2016, seguito da una stabilizzazione o un calo delle recensioni negli anni successivi. Purtroppo non possiamo notare molto dell'andamento poiché le recensioni fanno riferimento soltanto a 3 anni specifici.
Mentre dalla seguente query avremo un differente istogramma:

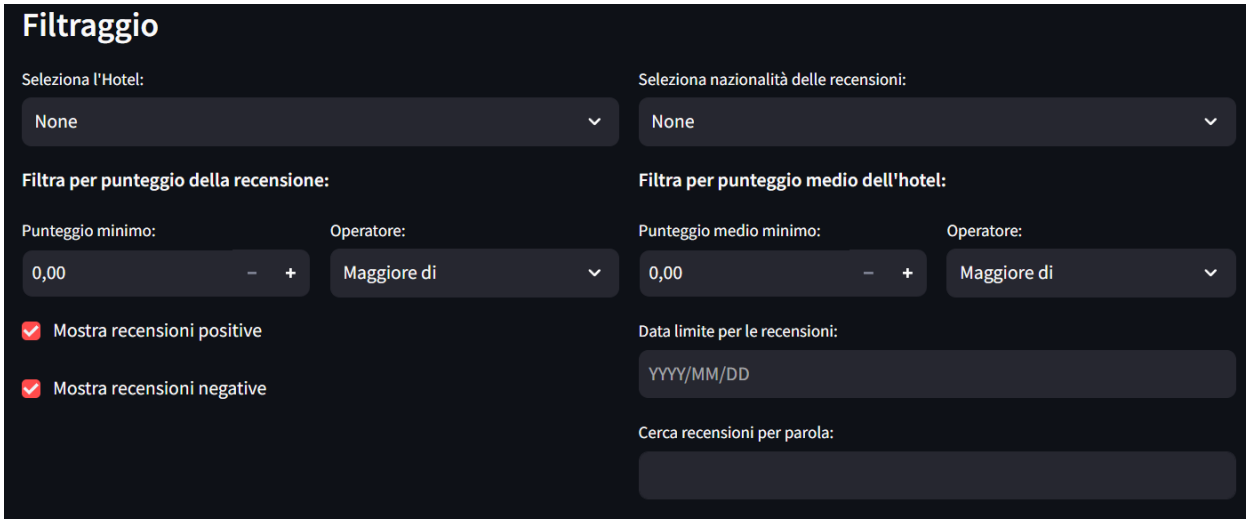
```
res = dfUtils.groupBy(F.month("Review_Date").alias("Month"))
        .agg(F.count("*").alias("Review_Count"))
        .orderBy("Month")
```



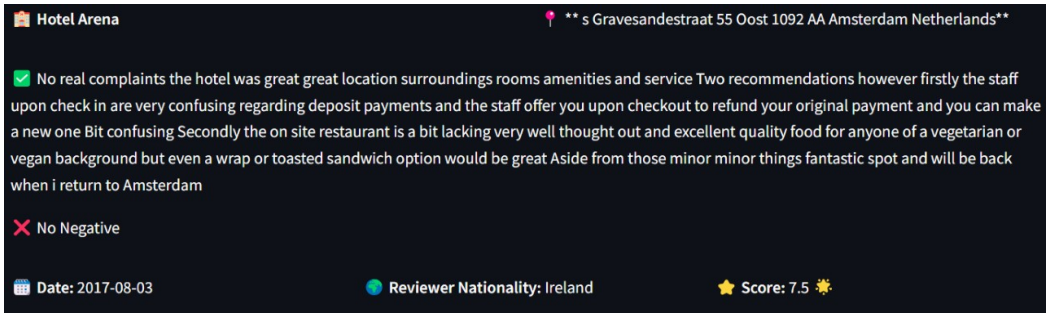
Osserviamo la distribuzione del numero di recensioni nei diversi mesi dell'anno per tutti gli anni. Si nota una certa uniformità nel numero di recensioni durante l'anno, con variazioni non troppo marcate. Tuttavia, alcuni mesi sembrano leggermente più ricchi di recensioni (ad esempio il mese 8), mentre altri mesi (come il mese 11) mostrano un lieve calo. Questo potrebbe indicare una stagionalità moderata, probabilmente legata ai periodi di vacanza o alta stagione per gli hotel.

Filtro Recensioni

Questa sezione consente di applicare una serie di filtri per visualizzare recensioni specifiche relative agli hotel. È possibile selezionare un hotel, una nazionalità del recensore, e impostare filtri sui punteggi delle recensioni e sui punteggi medi degli hotel. Inoltre, è previsto un filtro per visualizzare recensioni positive e/o negative, nonché la possibilità di restringere la ricerca alle recensioni inserite in una determinata data.



I risultati ottenuti vengono presentati in un formato tabellare che include informazioni dettagliate per ciascuna recensione. Ecco un'esempio di visualizzazione:



Filtraggio gestito come unica query:

```
#FILTRAGGIO PER NOME
if filter_Name is not None:
    dfCopia=dfCopia.filter(dfCopia.Hotel_Name == filter_Name)

#FILTRAGGIO PER VOTO RECENSIONE
if filter_Rev_Score is not None:
    if filter_Rev_Score[1]==">":
        dfCopia= dfCopia.filter(dfCopia.Reviewer_Score >= filter_Rev_Score[0])
    elif filter_Rev_Score[1]=="<":
        dfCopia= dfCopia.filter(dfCopia.Reviewer_Score <= filter_Rev_Score[0])
    #else sarebbe errore da verificare

#FILTRAGGIO PER VOTO MEDIO HOTEL
```

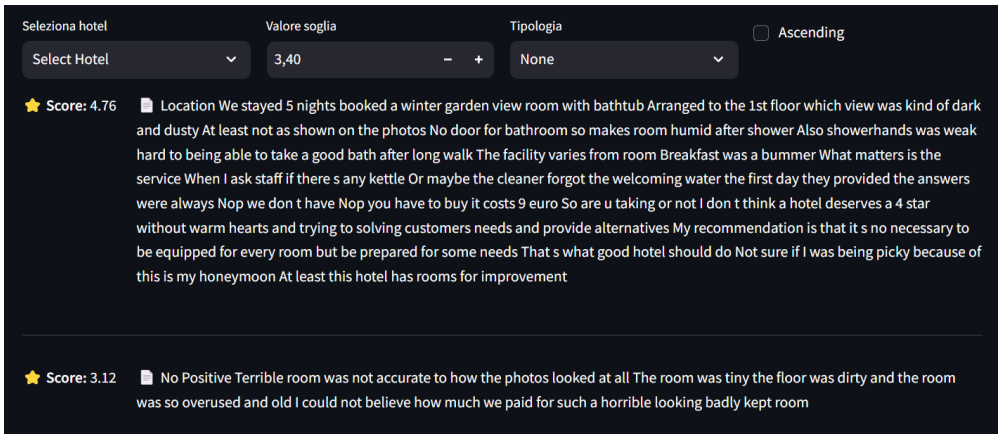
```
if filter_Avg_Score is not None:
    if filter_Avg_Score[1]==">":
        dfCopia= dfCopia.filter(dfCopia.Reviewer_Score >= filter_Avg_Score[0])
    elif filter_Avg_Score[1]=="<":
        dfCopia= dfCopia.filter(dfCopia.Reviewer_Score <= filter_Avg_Score[0])
    #else sarebbe errore da verificare

#FILTRAGGIO NAZIONALITà REVIEWER
if filter_Nationality_Rev is not None:
    dfCopia= dfCopia.filter(dfCopia.Reviewer_Nationality == filter_Nationality_Rev)

#FILTRAGGIO SE DEVE O MENO CONTENERE LA PARTE NEGATIVA
if filter_Negative_Rev is False:
    dfCopia = dfCopia.drop(dfCopia.Negative_Review)

#FILTRAGGIO SE DEVE O MENO CONTENERE LA PARTE POSITIVA
if filter_Positive_Rev is False:
    dfCopia = dfCopia.drop(dfCopia.Positive_Review)

#FILTRAGGIO PER DATA
if filter_Date is not None:
    dfCopia = dfCopia.filter(dfCopia.Review_Date <= filter_Date)

if word_to_search is not None:
    dfCopia= filter_reviews_by_word(dfCopia, word_to_search)
```

Recensioni Controverse

Questa pagina è dedicata esclusivamente all'identificazione delle recensioni più controverse. Utilizzando i criteri di differenza significativa tra il punteggio di una recensione e la media dell'hotel, questa sezione permette di evidenziare feedback che si discostano notevolmente dalle valutazioni generali. La soglia di differenza, inizialmente impostata a 3,4 punti, può essere regolata per adattarsi meglio alle specifiche esigenze dell'analisi. Attraverso la seguente funzione, avremo i risultati voluti.

```
df = spark_session.sql(f'''SELECT Positive_Review,
                        Hotel_Name,
                        Negative_Review,
                        (Average_Score-Reviewer_Score-3.4)+
                        (Review_Total_Negative_Word_Counts+
                        Review_Total_Positive_Word_Counts)/50 as Score FROM
                        Hotel_Reviews WHERE abs(Average_Score-Reviewer_Score) > {param}''')
if hotel is not None:
    df = df.filter(df.Hotel_Name == hotel)
if positive is not None:
    if filter:
        df = df.filter(df.Score > 0)
    else:
        df = df.filter(df.Score < 0)

return df.withColumn("Score", F.abs(df.Score))
    .orderBy("Score", ascending=False)
    .drop("Hotel_Name")\
    .withColumn("Review",
        F.concat(F.col("Positive_Review"),
        F.lit(" "),
        F.col("Negative_Review")))\
    .drop("Positive_Review", "Negative_Review")
```

Con la seguente dashboard:



Sentiment Analysis

Questa pagina consente di analizzare automaticamente il sentiment delle recensioni. Inserendo una recensione in inglese, il modello di sentiment analysis identifica e classifica il sentimento espresso come positivo o negativo. Grazie a sofisticati algoritmi di elaborazione del linguaggio naturale, il sistema offre risultati rapidi e accurati, semplificando il processo di analisi delle recensioni. L'inserimento della propria recensione avviene in questo apposito box:

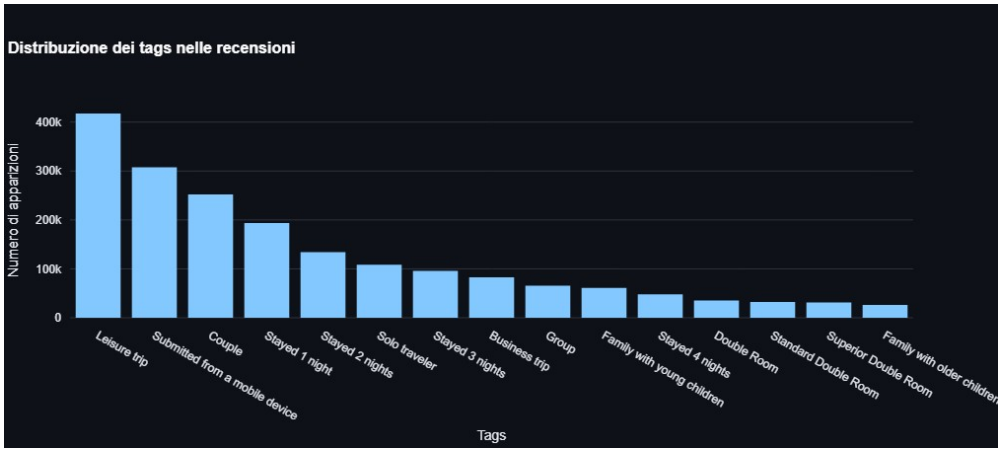
Che farà poi apparire la risposta del modello sviluppato.

Query

Questa pagina fornisce un'analisi interattiva delle recensioni degli hotel di lusso attraverso diverse visualizzazioni:

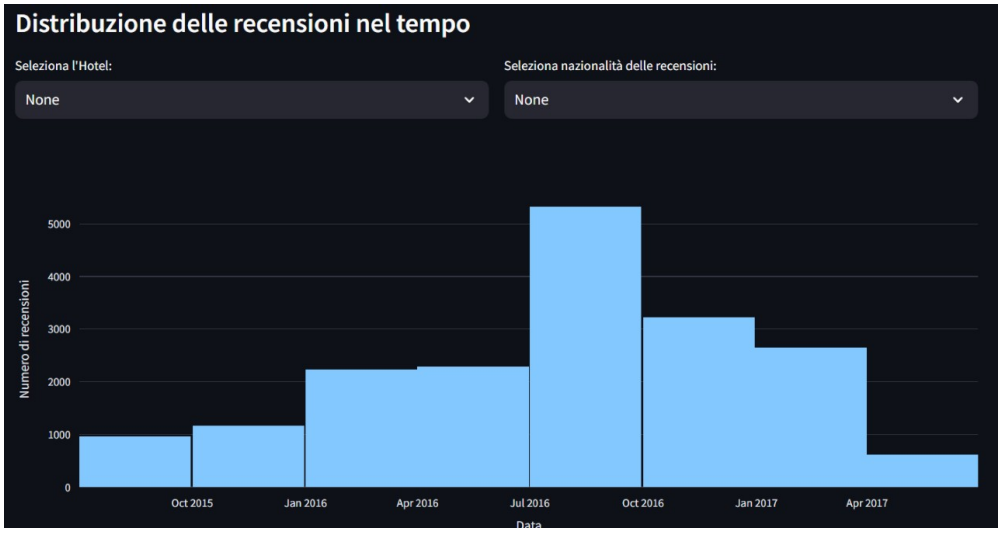
- Distribuzione dei tag più frequenti nelle recensioni.

```
#explode separa ogni elemento dell'array in una nuova riga
df_exploded = dfNew.withColumn("Tag", F.explode(F.col("Tags_Array")))
tag_count = df_exploded.groupBy("Tag").count().orderBy("count", ascending=False)
```



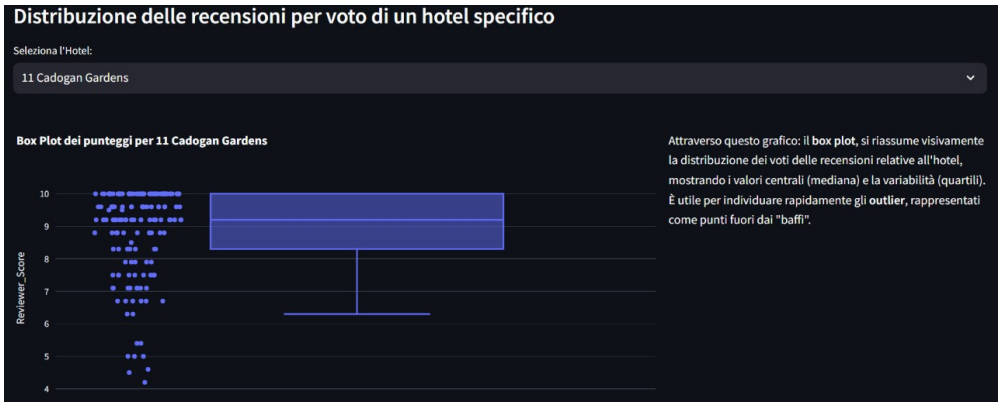
- Andamento temporale delle recensioni, filtrabile per hotel e nazionalità.

```
dfCount= reviews
if Hotel_Name is not None:
    dfCount = dfCount.filter(dfCount.Hotel_Name == Hotel_Name)
elif Nationality is not None:
    dfCount = dfCount.filter(dfCount.Reviewer_Nationality == Nationality)
dfCount=dfCount.groupby(dfCount.Review_Date).count()
               .withColumnRenamed("count","Rev_For_Date")
return dfCount.select("Rev_For_Date","Review_Date")
```



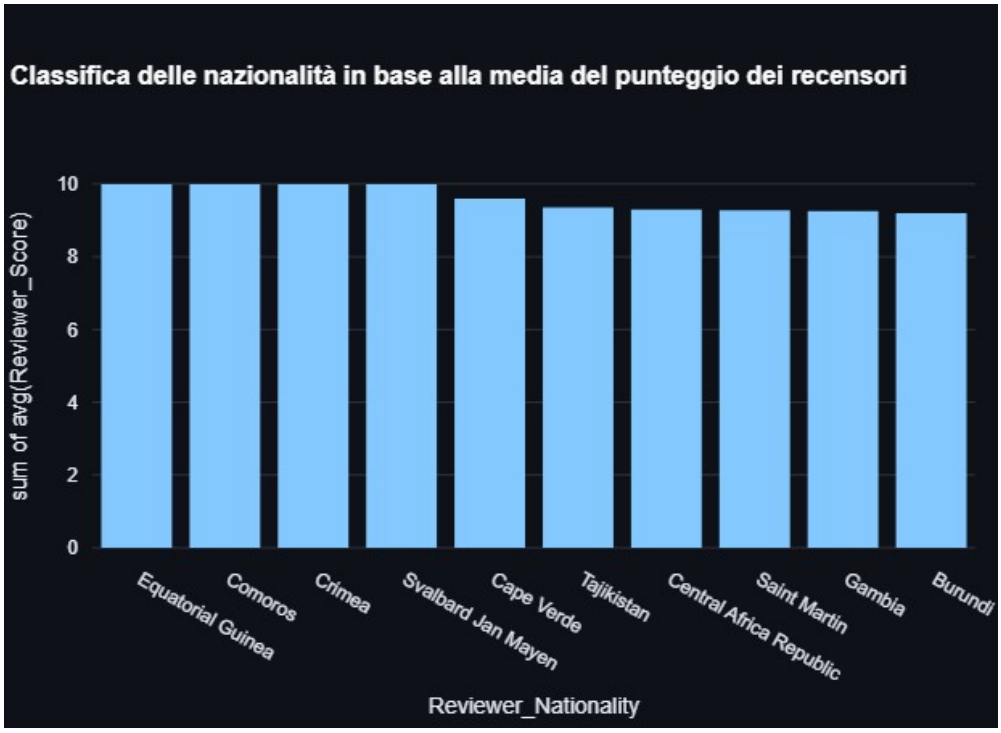
- Box plot per esplorare la distribuzione dei punteggi per hotel.

```
dfRet= reviews
    if Hotel_Name is not None:
    dfRet = dfRet.filter(dfRet.Hotel_Name == Hotel_Name)
return dfRet.select("Hotel_Name", "Reviewer_Score")
```



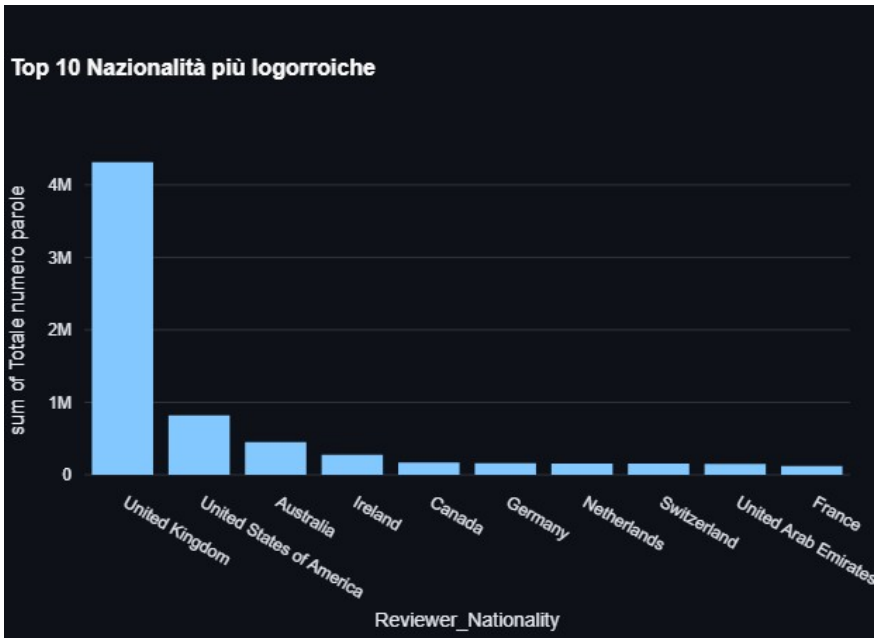
- Classifica delle nazionalità in base alla media del punteggio dei recensori.

```
df = (df.groupby(dfCopia.Reviewer_Nationality)
            .avg("Reviewer_Score")
           .sort("avg(Reviewer_Score)", ascending=False)) \
    .select("Reviewer_Nationality", "avg(Reviewer_Score)")
```



- Top 10 Nazionalità più logorroiche

```
df = df.groupby("Reviewer_Nationality") \
    .agg(
    F.sum("Review_Total_Positive_Word_Counts").alias("Total_Positive"),
    F.sum("Review_Total_Negative_Word_Counts").alias("Total_Negative"),
    F.sum("Reviewer_Score").alias("Num Review"),
) \
    .withColumn("Totale numero parole",
    F.col("Total_Positive") + F.col("Total_Negative")/ F.col("Num Review")) \
    .sort("Totale numero parole", ascending=False) \
    .select("Reviewer_Nationality", "Totale numero parole")
```

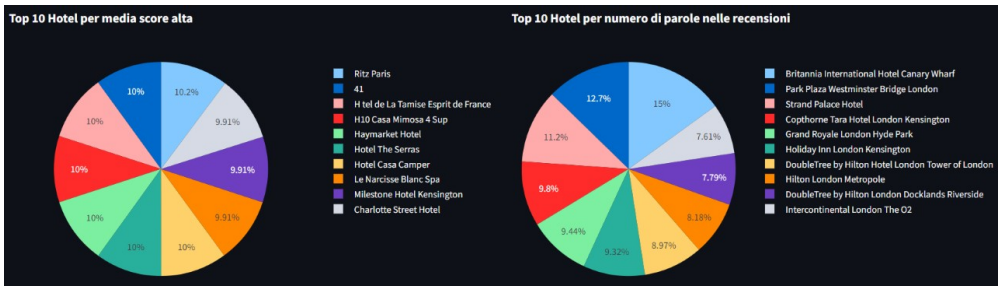


- Top 10 Hotel per media score alta

```
df = df.groupby(dfCopia.Hotel_Name)
.avg("Average_Score")
.sort("avg(Average_Score)", ascending=False)
.withColumnRenamed("avg(Average_Score)", "Totale recensioni")
.select("Hotel_Name", "Totale recensioni")
```

- Top 10 Hotel per numero di parole nelle recensioni

```
df = (df.groupby(dfCopia.Hotel_Name)
            .agg(
        F.sum("Review_Total_Positive_Word_Counts").alias("Total_Positive"),
        F.sum("Review_Total_Negative_Word_Counts").alias("Total_Negative")
    )
    .withColumn("Totale numero parole",
                F.col("Total_Positive") + F.col("Total_Negative"))
    .sort("Totale numero parole", ascending=False)
    .select("Hotel_Name", "Totale numero parole"))
```

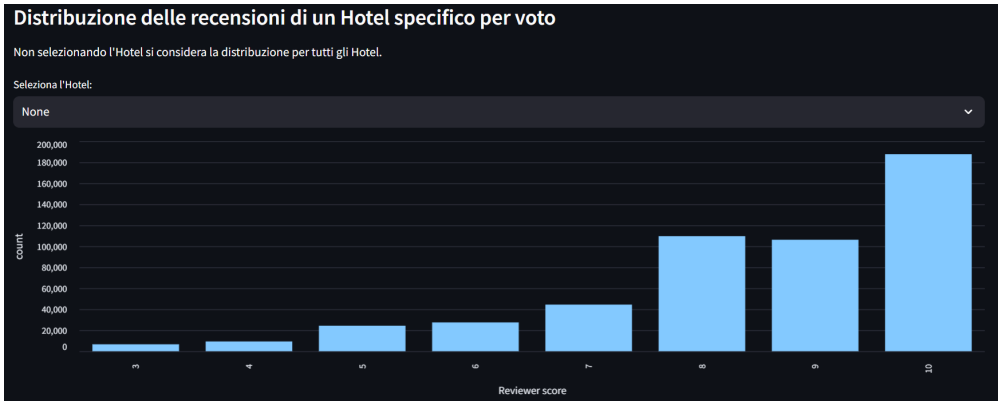


- Andamento dei punteggi medi delle recensioni nel tempo.

```
res = dfUtils.groupBy(F.year("Review_Date").alias("Year"),
                      F.month("Review_Date").alias("Month"))
            .agg(F.count("*").alias("Review_Count"))
            .orderBy("Year", "Month")
```

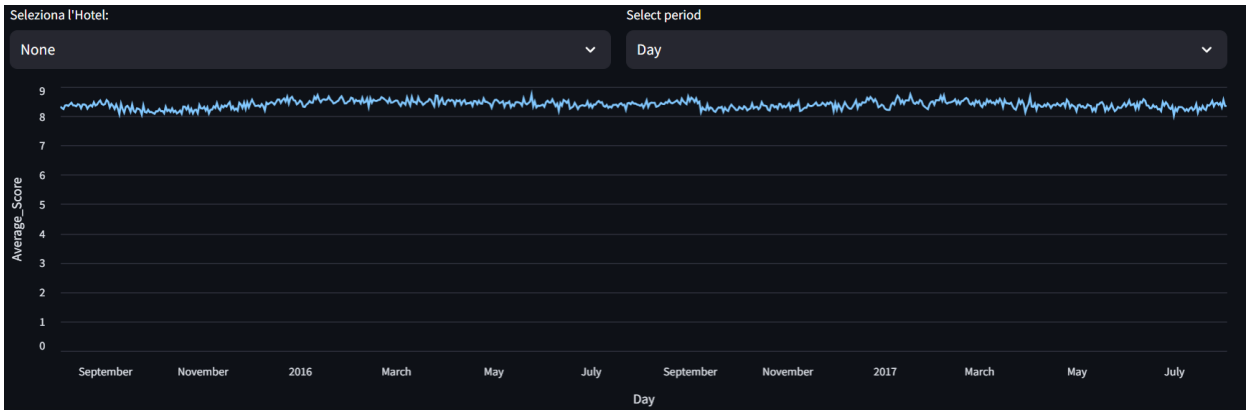
- Distribuzione delle recensioni di un Hotel specifico per voto

```
df = spark_session.sql(f"SELECT Reviewer_Score, Hotel_Name FROM Hotel_Reviews")
        .withColumn("Reviewer_Score", F.round(F.col("Reviewer_Score"), 0))
if hotel is not None:
    df = df.filter(df.Hotel_Name == hotel)
return df.groupBy("Reviewer_Score").count().orderBy("Reviewer_Score")
```



- Andamento del voto delle recensioni di un Hotel specifico per periodo di tempo, dove non selezionando l'Hotel si considera l'andamento per tutti gli Hotel.

```
df = spark_session.sql(f"SELECT Review_Date, Hotel_Name, Reviewer_Score
                         FROM Hotel_Reviews")

if hotel is not None:
    df = df.filter(df.Hotel_Name == hotel)

if period == "Day":
    return df.groupBy("Review_Date")
             .agg(F.avg("Reviewer_Score").alias("Average_Score"))
             .orderBy("Review_Date").withColumnRenamed("Review_Date", "Day")

elif period == "Month":
    return df.withColumn("Month", F.date_format(F.col("Review_Date"), "yyyy-MM"))          .g
             .orderBy("Month")
elif period == "Year":
    return df.withColumn("Year", F.date_format(F.col("Review_Date"), "yyyy"))
             .groupBy("Year").agg(F.avg("Reviewer_Score").alias("Average_Score"))
             .orderBy("Year")
```



5 Sentiment analysis

La sentiment analysis è una tecnica di elaborazione del linguaggio naturale (NLP, Natural Language Processing) utilizzata per identificare e classificare automaticamente le emozioni, opinioni o atteggiamenti espressi in un testo. In ambito turistico, e in particolare per le recensioni di hotel, questa tecnologia offre strumenti potenti per analizzare i feedback dei clienti e trarre informazioni utili per migliorare la qualità dei servizi offerti.
L'algoritmo utilizzato ha come scopo la classificazione della recensione come positiva o negativa e per farlo sono stati addestrati e confrontati due modelli differenti: un Support Vector Machine ed un Regressore Lineare.
Il primo è stato scelto perché generalmente i dati testuali sono particolarmente adatti alla classificazione con SVM (Support Vector Machine) a causa della natura sparsa del testo, in cui poche caratteristiche sono irrilevanti, ma tendono a essere correlate tra loro e generalmente organizzate in categorie linearmente separabili [1].
La regressione logistica è stata scelta perché generalmente indicata nei casi di classificazione binaria ed è anche utilizzata per l'analisi di testo. Poiché il testo non è direttamente utilizzabile dagli algoritmi di machine learning sono state effettuate delle modifiche al dataframe.

5.1 Undersampling

Innanzitutto le colonne "Positive_Review" e "Negative_Review" sono state riunite nella stessa colonna "Review":

```
dataframe = reviews.select("Positive_Review", "Negative_Review", "Reviewer_Score")
dataframe = dataframe\
    .withColumn("Review", F.concat(dataframe.Positive_Review, F.lit(" "), dataframe.Negative_Review))
    .drop("Positive_Review", "Negative_Review")
```

Osservando i punteggi è stato notato che circa il 90% delle recensioni aveva una valutazione maggiore della metà (5) e non sono presenti valutazioni inferiori a 3.
Inoltre, generalmente, valutazioni intorno al 6-7 corrispondono ad aspetti negativi che fanno scendere la valutazione rispetto a una recensione positiva.
Per questi fattori ogni recensione al di sotto del 7 è stata considerata negativa, assegnandogli uno 0, mentre alle altre è stato accostato un uno, binarizzando così i voti.
Oltre a ciò, le valutazioni positive sono state ridotte da circa 430.000 a 80.000, per rendere le classi omogenee in quantità.

```
dataframe = dataframe.withColumn("Reviewer_Score", F.when(dataframe.Reviewer_Score >= 7, 1)\
    .otherwise(0)).withColumnRenamed("Reviewer_Score", "label")

dataframe.groupBy("label").count().show()

# Separate classes and undersample positive class (the majority)
positive_df = dataframe.filter(dataframe.label == 1)
negative_df = dataframe.filter(dataframe.label == 0)


fraction = negative_df.count() / positive_df.count()
positive_sampled = positive_df.sample(withReplacement=False, fraction=fraction)

df = positive_sampled.union(negative_df)
```

5.2 Feature extraction

Una volta pre-processato il dataset è necessario estrarre le feature, ovvero ottenere valori numerici utilizzabili per addestrare il modello. Seguendo processi tipici del natural language processing [3], sono stati effettuati:

- **rimozione delle *stop-word***: per stop-word si intendono parole comuni che non contengono valore semantico utile all'analisi del teso e anzi renderebbero le varie recensioni troppo simili tra loro portando eventualmente ad errori di inferenza.
- **calcolo dell'*TF-IDF***: un metodo classico in letteratura per dare un peso ad ogni parola è quello di calcolare la frequenza di un termine all'interno del testo e rapportarlo alla frequenza in tutti gli altri documenti (le recensioni). In questo caso, parole come "hotel", molto comuni nelle recensioni, avranno un peso minore rispetto ad altre meno comuni e probabilmente più determinanti nella distinzione tra recensione positiva e negativa.

In PySpark è possibile definire una pipeline di oggetti per attuare le trasformazioni sopra annunciate e successivamente addestrare il modello con i dati preparati:

```
tokenizer = Tokenizer(inputCol="Review", outputCol="words")
remover = StopWordsRemover(inputCol="words", outputCol="filtered")
hashingTF = HashingTF(inputCol="filtered", outputCol="rawFeatures")
idf = IDF(inputCol="rawFeatures", outputCol="features")
eval = BinaryClassificationEvaluator(labelCol="label", metricName="areaUnderROC")

log_reg = LogisticRegression(featuresCol="features", labelCol="label", maxIter=20)

svm = LinearSVC(featuresCol="features", labelCol="label")

pipeline_svm = Pipeline(stages=[tokenizer, remover, hashingTF, idf, svm])
pipeline_lr = Pipeline(stages=[tokenizer, remover, hashingTF, idf, log_reg])
```

5.3 Cross-Validation

Per evitare di inserire manualmente i parametri utili per il tuning dei modelli è stata utilizzata la cross-validation per scegliere tra alcuni valori tipici dei seguenti parametri:

- *numFeatures*: rappresenta il numero di bucket nella tabella hash utilizzata per il calcolo del tf-idf. Valori molto alti diminuiscono la possibilità di collisioni ma aumentano il consumo di memoria, mentre valori molto bassi implicano maggior numero di collisioni.
- *regParam*: il parametro di regolarizzazione del modello. Ad un valore maggiore corrisponde un over-fitting minore ma allo stesso tempo un minore adattamento del modello ai dati.
- *elasticNetParam*: parametro che gestisce l'interpolazione tra la regolarizzazione Ridge e Lasso.

```
grid_lr = ParamGridBuilder()\
        .addGrid(hashingTF.numFeatures, [1000, 10000, 100000])\
        .addGrid(log_reg.regParam, [1, 0.1, 0.01])\
        .addGrid(log_reg.elasticNetParam, [0.0, 0.5])\
        .build()

grid_svm = ParamGridBuilder()\
        .addGrid(hashingTF.numFeatures, [1000, 10000, 100000])\
        .addGrid(svm.regParam, [1, 1, 0.1, 0.01])\
        .build()

cross_val_svm = CrossValidator(estimator=pipeline_svm,
                           estimatorParamMaps=grid_svm,
                           evaluator=eval,
                           numFolds=5)

cv_model_svm = cross_val_svm.fit(df)
```

```
bestModel = cv_model_svm.bestModel.stages[-1]
print("Best Parameters: ", bestModel.explainParams())

avgMetrics = cv_model_svm.avgMetrics
print("Cross-Validation Metrics: ", avgMetrics)

cross_val_lr = CrossValidator(estimator=pipeline_lr,
                           estimatorParamMaps=grid_lr,
                           evaluator=eval,
                           numFolds=5)

cv_model_lr = cross_val_lr.fit(df)

bestModel = cv_model_lr.bestModel.stages[-1]
print("Best Parameters: ", bestModel.explainParams())

avgMetrics = cv_model_lr.avgMetrics
print("Cross-Validation Metrics: ", avgMetrics)
```

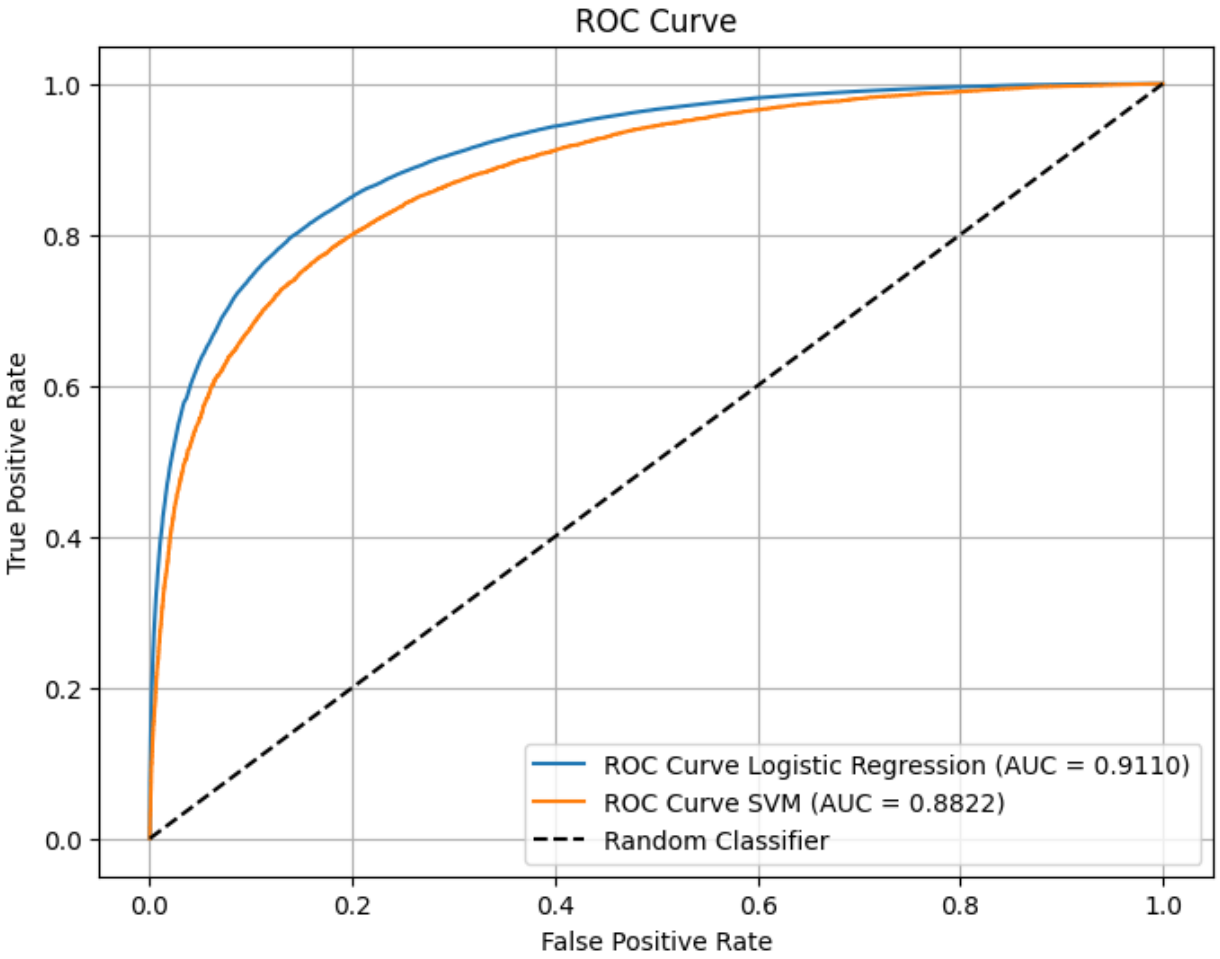
I migliori parametri risultano essere

- *numFeatures*: 10000;
- *regParam - log reg*: 0.1;
- *regParam - svm*: 0.1.
- *elasticNetParam*: 0.0.

5.4 Confronto

Una volta trovati i migliori parametri per entrambi i modelli essi sono stati confrontati per determinare il modello finale, utilizzando come metrica l'area della curva Receiver Operating Characteristic.

```
...
train, test = df.randomSplit([0.8, 0.2], seed=42)

pipeline_lr = Pipeline(stages=[tokenizer, remover, hashingTF, idf, log_reg])

pipeline_model_lr = pipeline_lr.fit(train)

pipeline_svm = Pipeline(stages=[tokenizer, remover, hashingTF, idf, svm])

pipeline_model_svm = pipeline_svm.fit(train)

roc_auc_lr = eval.evaluate(pipeline_model_lr.transform(test))
print(f"ROC-AUC for log reg: {roc_auc_lr}")

transformed = pipeline_model_svm.transform(test)
predictions = transformed.select("label", "rawPrediction").toPandas()
roc_auc_svm = eval.evaluate(transformed)
print(f"ROC-AUC for svm: {roc_auc_svm}")

lr_model = pipeline_model_lr.stages[-1]

# Get the ROC data for log reg
roc = lr_model.summary.roc.toPandas()
# Get the ROC data for svm
fpr_svm, tpr_svm, thres = roc_curve(y_true=predictions["label"],
                                    y_score=predictions["rawPrediction"].apply(lambda x: x[1]))

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(roc['FPR'], roc['TPR'],
        label="ROC Curve Logistic Regression (AUC = {:.4f})".format(lr_model.summary.areaUnderROC))
plt.plot(fpr_svm, tpr_svm, label="ROC Curve SVM (AUC = {:.4f})".format(roc_auc_svm))
plt.plot([0, 1], [0, 1], 'k--', label="Random Classifier")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc="best")
plt.grid()
plt.show()
```



La regressione logistica è risultata più performante rispetto al SVM e perciò è stata scelta all'interno dell'applicazione.

References

[1] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. "Sentiment analysis algorithms and applications: A survey". In: *Ain Shams Engineering Journal* 5.4 (2014), pp. 1093–1113. ISSN: 2090-4479. DOI: https://doi.org/10.1016/j.asej.2014.04.011. URL: https://www.sciencedirect.com/science/article/pii/S2090447914000550.

[2] Samuel Solomon, D. Rajinigirinath, and S. P. Audline. "Dynamic Dashboard For Airbnb Data Analysis Using Streamlit". In: *ResearchGate* (2024). URL: https://www.researchgate.net/publication/387364494_Dynamic_Dashboard_For_Airbnb_Data_Analysis_Using_Streamlit.

[3] Mayur Wankhade, Annavarapu Chandra Sekhara Rao, and Chaitanya Kulkarni. "A survey on sentiment analysis methods, applications, and challenges". In: *Artificial Intelligence Review* 55.7 (Oct. 2022), pp. 5731–5780. ISSN: 1573-7462. DOI: 10.1007/s10462-022-10144-1. URL: https://doi.org/10.1007/s10462-022-10144-1.