SISTEMA DI ANALISI PER IL TURISMO CON AFFITTI BREVI NELLA REGIONE PUGLIA

Esame: Ingegneria della conoscenza

Progetto sviluppato da: Michele Matteucci

Matricola: 76044

Mail Uniba: m.matteucci5@studenti.uniba.it

Mail Personale: michele.matteucci02@gmail.com

Link GitHub: https://github.com/MattMic02/Progetto-ICON.git

Sommario

CAP 0: INTRODUZIONE

L'obiettivo di questo progetto è condurre un'analisi approfondita sugli affitti a lungo e a breve termine in Puglia, con particolare attenzione agli annunci presenti sulla piattaforma Airbnb. Operando sui dati a disposizione, gli obiettivi specifici del progetto sono i seguenti:

Assegnazione di una Valutazione Complessiva agli Annunci:

L'obiettivo è sviluppare un sistema di valutazione integrata per gli annunci di case vacanza e altre strutture presenti su Airbnb. Questo sistema non si baserà unicamente sulle recensioni lasciate dai clienti, ma prenderà in considerazione una più ampia gamma di dati, inclusi elementi quantitativi e qualitativi. L'intento è fornire una valutazione più oggettiva e completa che rifletta accuratamente la qualità complessiva dell'annuncio, considerando informazioni che sono invisibili o nascoste agli utenti.

Clustering degli Annunci:

Si prevede di eseguire un'analisi di clustering sugli annunci, ponendo particolare enfasi su variabili quali il costo, la posizione geografica, la tipologia di struttura, i servizi offerti, e altre caratteristiche rilevanti. L'obiettivo di questa analisi è suggerire agli utenti appartamenti o strutture con caratteristiche simili a quelle che hanno già prenotato e valutato positivamente. Questo sistema di raccomandazione potrebbe migliorare significativamente l'esperienza utente, rendendo più facile trovare soluzioni abitative in linea con le proprie preferenze. (Considerazioni sull'efficacia dei risultati ottenuti ai fini di raggiungere l'obiettivo discusso nel CAP 3).

Acquisizione di Nuove Conoscenze dal Web:

Il progetto mira anche a raccogliere e integrare informazioni aggiuntive disponibili online, associandole agli appartamenti presenti nelle piattaforme di affitto. Incrociando i dati esistenti con quelli reperiti dal web, sarà possibile arricchire gli annunci con dettagli più precisi e concreti, migliorando la qualità e l'accuratezza delle informazioni fornite agli utenti. Questo processo permetterà di estrarre nuova conoscenza, rendendo gli annunci più informativi e, di conseguenza, più attraenti per i potenziali clienti.

Il **dataset** di partenza utilizzato per questo progetto è stato scaricato dalla piattaforma Kaggle e può essere consultato al seguente link: Airbnb Italy Dataset. Ho scelto di lavorare specificamente sul subset del dataset relativo agli annunci presenti nella regione Puglia, con dati aggiornati fino a settembre 2022.
Come descritto nella scheda "contenuto" del dataset su Kaggle (tradotta con Google Translate):
"Per ogni area geografica esiste una cartella dedicata che contiene questi file:

1. **Elenchi**: questo file CSV include tutte le informazioni necessarie per scoprire di più sugli host, sulla disponibilità geografica, sulle metriche necessarie per fare previsioni e trarre conclusioni con l'aiuto di alcuni strumenti di visualizzazione.

2. **Recensione**: questo file contiene tutte le recensioni elencate in base all'ID dell'ospite e dell'Host; quindi, può essere utilizzato per alcuni progetti di analisi del testo.
3. **Quartieri**: questo è un file JSON utile per tracciare mappe di grandi dimensioni."

Per il progetto, ho preferito concentrarmi esclusivamente sul dataset denominato "Elenchi" (*Listings.csv*). Questo file contiene una vasta gamma di informazioni sugli annunci, compresi dettagli sugli host, sulla localizzazione geografica delle proprietà, nonché su varie metriche utili per le analisi e le previsioni.
È importante notare che, sebbene il dataset "Recensioni" (Reviews.csv) offra dati potenzialmente preziosi per progetti di analisi del linguaggio naturale (NLP), come l'elaborazione e l'analisi dei feedback testuali degli utenti, non è stato utilizzato in questa fase del progetto. Tuttavia, l'integrazione delle recensioni con tecniche di Natural Language Processing (NLP) rappresenta un possibile sviluppo futuro del progetto, che potrebbe arricchire ulteriormente le analisi condotte.

Riporto di seguito una descrizione più dettagliata sul contenuto del dataset originale.

1. **id:**
   Descrizione: Identificatore univoco dell'annuncio su Airbnb.
   Tipo di dato: Numero intero.
2. **listing_url:**
   Descrizione: URL dell'annuncio su Airbnb.
   Tipo di dato: Link a pagina Web.
3. **scrape_id:**
   Descrizione: ID dello scrape, utilizzato per identificare la sessione di raccolta dati.
   Tipo di dato: Numero intero.
4. **last_scraped:**
   Descrizione: Data in cui i dati sono stati raccolti per l'ultima volta.
   Tipo di dato: Data rappresentata come stringa (YYYY-MM-DD).
5. **source:**
   Descrizione: Fonte dei dati (in genere sarà 'airbnb').
   Tipo di dato: Stringa.
6. **name:**
   Descrizione: Nome o titolo dell'annuncio.
   Tipo di dato: Stringa.
7. **description:**
   Descrizione: Descrizione dell'annuncio fornita dall'host.
   Tipo di dato: Stringa.
8. **neighborhood_overview:**
   Descrizione: Descrizione del quartiere fornita dall'host.
   Tipo di dato: Stringa.
9. **picture_url:**
   Descrizione: URL dell'immagine principale dell'annuncio.
   Tipo di dato: Link a pagina Web.

10. **host_id:**
    Descrizione: Identificatore univoco dell'host.
    Tipo di dato: Numero intero.
11. **host_url:**
    Descrizione: URL del profilo dell'host su Airbnb.
    Tipo di dato: Link a pagina Web.
12. **host_name:**
    Descrizione: Nome dell'host.
    Tipo di dato: Stringa.
13. **host_since:**
    Descrizione: Data da cui l'host è attivo su Airbnb.
    Tipo di dato: Data rappresentata come stringa (YYYY-MM-DD).
14. **host_location:**
    Descrizione: Posizione dichiarata dall'host (ad esempio, città o regione).
    Tipo di dato: Stringa.
15. **host_about:**
    Descrizione: Descrizione personale fornita dall'host.
    Tipo di dato: Stringa.
16. **host_response_time:**
    Descrizione: Tempo medio di risposta dell'host (ad esempio, "within an hour").
    Tipo di dato: Stringa.
17. **host_response_rate:**
    Descrizione: Percentuale di risposte dell'host (ad esempio, "90%").
    Tipo di dato: Numero intero.
18. **host_acceptance_rate:**
    Descrizione: Percentuale di richieste di prenotazione accettate dall'host.
    Tipo di dato: Numero intero.
19. **host_is_superhost:**
    Descrizione: Indica se l'host è un Superhost ("t" per vero, "f" per falso).
    Tipo di dato: Booleano.
20. **host_thumbnail_url:**
    Descrizione: URL della foto del profilo dell'host.
    Tipo di dato: Link a pagina Web.
21. **host_picture_url:**
    Descrizione: URL della foto completa del profilo dell'host.
    Tipo di dato: Link a pagina Web.
22. **host_neighbourhood:**
    Descrizione: Quartiere specifico dichiarato dall'host.
    Tipo di dato: Stringa.
23. **host_listings_count:**
    Descrizione: Numero totale di annunci attivi gestiti dall'host.
    Tipo di dato: Numero intero.
24. **host_total_listings_count:**
    Descrizione: Numero totale di annunci dell'host, compresi quelli non più attivi.

Tipo di dato: Numero intero.

25. **host_verifications:**
    Descrizione: Metodi di verifica del profilo dell'host (ad esempio, "email, phone").
    Tipo di dato: Stringa
26. **host_has_profile_pic:**
    Descrizione: Indica se l'host ha una foto del profilo ("t" per vero, "f" per falso).
    Tipo di dato: Booleano.
27. **host_identity_verified:**
    Descrizione: Indica se l'identità dell'host è stata verificata ("t" per vero, "f" per falso).
    Tipo di dato: Booleano.
28. **neighbourhood**:
    Descrizione: Quartiere dell'annuncio, secondo l'host.
    Tipo di dato: Stringa.
29. **neighbourhood**_cleansed:
    Descrizione: Quartiere ufficiale dell'annuncio secondo Airbnb.
    Tipo di dato: Stringa.
30. **neighbourhood**_group_cleansed:
    Descrizione: Raggruppamento ufficiale dei quartieri, se disponibile.
    Tipo di dato: Stringa.
31. **latitude**:
    Descrizione: Latitudine dell'annuncio.
    Tipo di dato: Numero intero.
32. **longitude**:
    Descrizione: Longitudine dell'annuncio.
    Tipo di dato: Numero intero.
33. **property_type:**
    Descrizione: Tipo di proprietà (ad esempio, "Appartamento", "Casa").
    Tipo di dato: Stringa.
34. **room_type:**
    Descrizione: Tipo di stanza offerta (ad esempio, "Intera casa/appartamento", "Stanza privata").
    Tipo di dato: Stringa.
35. **accommodates:**
    Descrizione: Numero massimo di ospiti che la proprietà può ospitare.
    Tipo di dato: Numero intero.
36. **bathrooms:**
    Descrizione: Numero di bagni. (Questo campo ha valori nulli in tutto il dataset).
    Tipo di dato: Numero intero.
37. **bathrooms_text:**
    Descrizione: Descrizione testuale dei bagni (ad esempio, "1 bagno privato").
    Tipo di dato: Stringa.
38. **bedrooms**:
    Descrizione: Numero di camere da letto.
    Tipo di dato: Numero intero.

39. **beds**:
    Descrizione: Numero di letti.
    Tipo di dato: Numero intero.
40. **amenities:**
    Descrizione: Lista dei servizi offerti dalla proprietà (ad esempio, "Wi-Fi, Cucina, TV").
    Tipo di dato: Stringa.
41. price:
    Descrizione: Prezzo per notte (formattato come stringa, di solito con un simbolo di valuta).
    Tipo di dato: Numero intero.
42. **minimum_nights:**
    Descrizione: Numero minimo di notti richieste per la prenotazione.
    Tipo di dato: Numero intero.
43. **maximum_nights:**
    Descrizione: Numero massimo di notti per cui è possibile prenotare la proprietà.
    Tipo di dato: Numero intero.
44. **minimum_minimum_nights:**
    Descrizione: Il valore minimo del campo minimum_nights tra tutte le proprietà dell'host.
    Tipo di dato: Numero intero.
45. **maximum_minimum_nights:**
    Descrizione: Il valore massimo del campo minimum_nights tra tutte le proprietà dell'host.
    Tipo di dato: Numero intero.
46. **minimum_maximum_nights:**
    Descrizione: Il valore minimo del campo maximum_nights tra tutte le proprietà dell'host.
    Tipo di dato: Numero intero.
47. **maximum_maximum_nights:**
    Descrizione: Il valore massimo del campo maximum_nights tra tutte le proprietà dell'host.
    Tipo di dato: Numero intero.
48. **minimum_nights_avg_ntm:**
    Descrizione: Valore medio del numero minimo di notti richiesto nelle proprietà dell'host.
    Tipo di dato: Numero intero.
49. **maximum_nights_avg_ntm:**
    Descrizione: Valore medio del numero massimo di notti richiesto nelle proprietà dell'host.
    Tipo di dato: Numero intero.
50. **calendar_updated:**
    Descrizione: Data dell'ultimo aggiornamento del calendario. (Questo campo ha solo valori nulli).
    Tipo di dato: Data rappresentata come stringa (YYYY-MM-DD).

51. **has_availability:**
    Descrizione: Indica se la proprietà ha disponibilità ("t" per vero, "f" per falso).
    Tipo di dato: Booleano.
52. **availability_30:**
    Descrizione: Numero di giorni disponibili nei prossimi 30 giorni.
    Tipo di dato: Numero intero.
53. **availability_60:**
    Descrizione: Numero di giorni disponibili nei prossimi 60 giorni.
    Tipo di dato: Numero intero.
54. **availability_90:**
    Descrizione: Numero di giorni disponibili nei prossimi 90 giorni.
    Tipo di dato: Numero intero.
55. **availability_365:**
    Descrizione: Numero di giorni disponibili nei prossimi 365 giorni.
    Tipo di dato: Numero intero.
56. **calendar_last_scraped:**
    Descrizione: Data dell'ultimo scrape del calendario.
    Tipo di dato: Data rappresentata come stringa (YYYY-MM-DD).
57. **number_of_reviews:**
    Descrizione: Numero totale di recensioni ricevute dall'annuncio.
    Tipo di dato: Numero intero.
58. **number_of_reviews_ltm:**
    Descrizione: Numero di recensioni ricevute negli ultimi 12 mesi.
    Tipo di dato: Numero intero.
59. **number_of_reviews_l30d:**
    Descrizione: Numero di recensioni ricevute negli ultimi 30 giorni.
    Tipo di dato: Numero intero.
60. **first_review:**
    Descrizione: Data della prima recensione ricevuta dall'annuncio.
    Tipo di dato: Data rappresentata come stringa (YYYY-MM-DD).
61. **last_review:**
    Descrizione: Data dell'ultima recensione ricevuta dall'annuncio.
    Tipo di dato: Data rappresentata come stringa (YYYY-MM-DD).
62. **review_scores_rating:**
    Descrizione: Punteggio medio delle recensioni.
    Tipo di dato: Numero decimale.
63. **review_scores_accuracy:**
    Descrizione: Punteggio di accuratezza delle recensioni.
    Tipo di dato: Numero decimale.
64. **review_scores_cleanliness:**
    Descrizione: Punteggio di pulizia delle recensioni.
    Tipo di dato: Numero decimale.
65. **review_scores_checkin:**
    Descrizione: Punteggio di check-in delle recensioni.

Tipo di dato: Numero decimale.

66. **review_scores_communication:**
    Descrizione: Punteggio di comunicazione delle recensioni.
    Tipo di dato: Numero decimale.
67. **review_scores_location:**
    Descrizione: Punteggio della posizione nelle recensioni.
    Tipo di dato: Numero decimale.
68. **review_scores_value:**
    Descrizione: Punteggio di valore delle recensioni (rapporto qualità/prezzo).
    Tipo di dato: Numero decimale.
69. **license:**
    Descrizione: Licenza o numero di registrazione, se richiesto dalla legge locale.
    Tipo di dato: Numero intero.
70. **instant_bookable:**
    Descrizione: Indica se la proprietà è prenotabile immediatamente senza approvazione ("t" per vero, "f" per falso).
    Tipo di dato: Booleano.
71. **calculated_host_listings_count:**
    Descrizione: Numero di annunci attivi dell'host.
    Tipo di dato: Numero intero.
72. **calculated_host_listings_count_entire_homes:**
    Descrizione: Numero di annunci attivi dell'host che sono intere case o appartamenti.
    Tipo di dato: Numero intero.
73. **calculated_host_listings_count_private_rooms:**
    Descrizione: Numero di annunci attivi dell'host che sono stanze private.
    Tipo di dato: Numero intero.
74. **calculated_host_listings_count_shared_rooms:**
    Descrizione: Numero di annunci attivi dell'host che sono stanze condivise.
    Tipo di dato: Numero intero.
75. **reviews_per_month:**
    Descrizione: Media delle recensioni ricevute per mese.
    Tipo di dato: Numero decimale.

Per il raggiungimento di ciascun obiettivo del progetto, partiremo dall'analisi di questo dataset, effettuando una selezione mirata e una modellazione accurata delle informazioni disponibili (eliminazione di righe, eliminazione di colonne, standardizzazione e normalizzazione dei dati). Verranno analizzati solo le informazioni che si dimostrano rilevanti per gli obiettivi specifici, in modo da focalizzare l'attenzione su quegli aspetti che possono realmente influire sui risultati finali.

Funzioni di pre processing sui dati

Per ogni modulo è necessario eseguire delle operazioni sulle righe e sulle colonne del dataset. Le operazioni in generale sono rimozioni di determinate colonne o righe, in base a feature non necessarie o a valori che rendono le righe poco rilevanti. Normalizzazione di valori di alcune colonne, come ad esempio trasformazioni da stringhe ad interi, codifiche compatte di stringhe ecc.

Output e Dettagli Finali

- Le funzioni che si occupano del preprocessing del dataset sono clear_dataset_prolog, clear_dataset_unsupervised_learning e clear_dataset_supervised_learning.
- Durante il processo di pulizia e preparazione, vengono aggiunte colonne utili per identificare i record e vengono rimossi i dati ridondanti o non rilevanti per l'analisi.
- Il dataset risultante viene ottimizzato per essere utilizzato con linguaggio prolog, in algoritmi di apprendimento non supervisionato e supervisionato.

CAP 1: PROLOG

L'inclusione del linguaggio logico PROLOG nel progetto risponde alla necessità di estrarre nuove informazioni dagli annunci di appartamenti in Puglia. L'obiettivo è creare un indicatore avanzato che offra una valutazione complessiva degli annunci, superando l'approccio tradizionale basato esclusivamente sul punteggio delle recensioni (ad esempio, 1 - 5 stelle) e sulla semplice lettura dei commenti degli utenti. Questo nuovo strumento, basato su un'analisi più profonda e strutturata, permetterà agli utenti di effettuare scelte più consapevoli e mirate nella selezione delle strutture da prenotare.

1. **Pulizia delle stringhe**: Rimuove caratteri speciali, spazi superflui e sostituisce le lettere accentate con lettere non accentate in alcune colonne che contengono dati potenzialmente problematici. Questo passaggio è essenziale per prevenire errori che potrebbero compromettere l’accuratezza dell’analisi.
2. **Eliminazione delle colonne irrilevanti**: Vengono eliminate tutte le colonne non pertinenti all’analisi in corso, concentrandosi esclusivamente sui dati necessari e ignorando le informazioni superflue. Questo contribuisce a migliorare l’efficienza e la chiarezza del dataset.
3. **Rimozione delle righe con valori mancanti**: Vengono eliminate tutte le righe in cui il valore di alcune colonne specificate è mancante, per evitare risultati potenzialmente errati o fuorvianti. Lo stesso trattamento viene applicato a determinate colonne che si è scelto di mantenere avvalorate.

```python
def clear_dataset_prolog(data):
    # Applica la pulizia delle stringhe ad alcune colonne
    data['name'] = data['name'].apply(lambda x: clean_string(x) if pd.notna(x) else x)
    data['host_location'] = data['host_location'].apply(lambda x: clean_string(x) if pd.notna(x) else x)
    data['host_verifications'] = data['host_verifications'].apply(lambda x: clean_string(x) if pd.notna(x) else x)
    #data['neighbourhood_cleansed'] = data['neighbourhood_cleansed'].apply(lambda x: clean_string(x) if pd.notna(x) else x)
    data['amenities'] = data['amenities'].apply(lambda x: clean_string(x) if pd.notna(x) else x)

    # axis=1: Si riferisce all'asse delle colonne (l'asse delle x).
    # #Quando utilizzi axis=1, l'operazione viene eseguita sulle colonne.
    # Ad esempio, se usi drop(columns, axis=1), Pandas rimuove le colonne specificate.
    def clean_price_string(price_str):
        # Controlla se la stringa non è vuota o nulla
        if price_str and isinstance(price_str, str):
            # Rimuovi il simbolo del dollaro
            cleaned_str = price_str.replace("$", "").replace(",", "").strip()
            # Trova la posizione del punto (se esiste) e conserva solo la parte prima del punto
            if '.' in cleaned_str:
                cleaned_str = cleaned_str.split('.')[0]
            # Ritorna la stringa pulita
            return int(cleaned_str)
        return None
    data['price'] = data['price'].apply(lambda x: clean_price_string(x) if pd.notna(x) else x)

    # Rimozione delle colonne non rilevanti
    data = data.drop('id', axis=1)
    data = data.drop('host_id', axis=1)
    data = data.drop('host_picture_url', axis=1)
    data = data.drop('host_url', axis=1)
    data = data.drop('listing_url', axis=1)
    data = data.drop('picture_url', axis=1)
    data = data.drop('host_about', axis=1)
    data = data.drop('host_thumbnail_url', axis=1)
    data = data.drop('host_neighbourhood', axis=1)
    data = data.drop('neighborhood_overview', axis=1)
    data = data.drop('description', axis=1)
    data = data.drop('bathrooms', axis=1)
    #data = data.drop('amenities', axis=1)

    # Aggiunta colonna id con ID
    data.loc[:,'id']=range(1,(data.shape[0])+1)

    # Aggiunta colonna id con ID
    data.loc[:,'host_id']=range(1,(data.shape[0])+1)

    # Rimozione delle righe con valori mancanti in 'room_type'
    data = data.dropna(subset=['room_type'])

    # Rimozione delle righe con valori mancanti in 'price'
    data = data.dropna(subset=['price'])

    # Rimozione delle righe con valori mancanti in 'neighbourhood_cleansed'
    data = data.dropna(subset=['neighbourhood_cleansed'])

    # Rimozione delle righe con valori mancanti in 'host_since'
    data = data.dropna(subset=['host_since'])

    # Rimozione delle righe con valori mancanti in 'latitude'
    data = data.dropna(subset=['latitude'])

    # Rimozione delle righe con valori mancanti in 'longitude'
    data = data.dropna(subset=['longitude'])

    # Rimozione delle righe dove 'host_since' non corrisponde al pattern 'XXXX-XX-XX'
    data = data[data['host_since'].str.match(r'^\d{4}-\d{2}-\d{2}$', na=False)]

    # Ottieni il numero di righe
    num_rows = data.shape[0]
    print(f"Il numero di righe nel file CSV è: {num_rows}")

    return data
```

La regola Prolog che segue definisce un predicato host_indicator_value(HostScore, ID) che calcola un punteggio complessivo (HostScore) per un determinato host identificato da ID, basato su una serie di criteri e attributi associati all'host e alla proprietà in questione.

```
regola = """
(host_indicator_value(HostScore,ID) :- property(ID,_,_,_,HostSince,_,HostResponceTime,HostResponceRate,

    % 1. Calcola il punteggio HostSinceScore : 10 punti
    (HostSince = "no_host_since" ->
        HostSinceScore = 0
    ;
        % Estrai l'anno da HostSince
        sub_string(HostSince, 0, 4, _, YearString),
        atom_number(YearString, YearHostSince),
        YearDiff is 2022 - YearHostSince,
        HostSinceScore is min(YearDiff, 10)
    ),

    % 2. Calcola il punteggio HostResponseTimeScore : 5 punti
    (HostResponceTime = 'within an hour' -> HostResponseTimeScore = 5;
    HostResponceTime = 'within a few hours' -> HostResponseTimeScore = 4;
    HostResponceTime = 'within a day' -> HostResponseTimeScore = 3;
    HostResponceTime = 'a few days or more' -> HostResponseTimeScore = 1;
    HostResponceTime = 'no_response_time' -> HostResponseTimeScore = 0),

    % 3. Calcola il punteggio HostResponseRateScore : 10 punti
    (HostResponceRate = -1 -> HostResponseRateScore = 0;
    HostResponceRate = 100 -> HostResponseRateScore = 10;
    HostResponceRate >= 90, HostResponceRate < 100 -> HostResponseRateScore = 9;
    HostResponceRate >= 80, HostResponceRate < 90 -> HostResponseRateScore = 8;
    HostResponceRate >= 70, HostResponceRate < 80 -> HostResponseRateScore = 7;
    HostResponceRate >= 60, HostResponceRate < 70 -> HostResponseRateScore = 6;
    HostResponceRate >= 50, HostResponceRate < 60 -> HostResponseRateScore = 5;
    HostResponceRate >= 40, HostResponceRate < 50 -> HostResponseRateScore = 4;
    HostResponceRate >= 30, HostResponceRate < 40 -> HostResponseRateScore = 3;
    HostResponceRate >= 20, HostResponceRate < 30 -> HostResponseRateScore = 2;
    HostResponceRate >= 10, HostResponceRate < 20 -> HostResponseRateScore = 1;
    HostResponceRate < 10 -> HostResponseRateScore = 0),
```

```
    % 4. Calcola il punteggio HostAcceptanceRateScore : 10 punti
    (HostAcceptanceRate = -1 -> HostAcceptanceRateScore = 0;
    HostAcceptanceRate = 100 -> HostAcceptanceRateScore = 10;
    HostAcceptanceRate >= 90, HostAcceptanceRate <100 -> HostAcceptanceRateScore = 9;
    HostAcceptanceRate >= 80, HostAcceptanceRate <90 -> HostAcceptanceRateScore = 8;
    HostAcceptanceRate >= 70, HostAcceptanceRate <80 -> HostAcceptanceRateScore = 7;
    HostAcceptanceRate >= 60, HostAcceptanceRate <70 -> HostAcceptanceRateScore = 6;
    HostAcceptanceRate >= 50, HostAcceptanceRate <60 -> HostAcceptanceRateScore = 5;
    HostAcceptanceRate >= 40, HostAcceptanceRate <50 -> HostAcceptanceRateScore = 4;
    HostAcceptanceRate >= 30, HostAcceptanceRate <40 -> HostAcceptanceRateScore = 3;
    HostAcceptanceRate >= 20, HostAcceptanceRate <30 -> HostAcceptanceRateScore = 2;
    HostAcceptanceRate >= 10, HostAcceptanceRate <20 -> HostAcceptanceRateScore = 1;
    HostAcceptanceRate >= 0, HostAcceptanceRate <10 -> HostAcceptanceRateScore = 0),

    % 5. Calcola il punteggio HostIsSuperHostScore : 5 punti
    (HostIsSuperHost = 't' -> HostIsSuperHostScore = 5;
    HostIsSuperHost = 'f' -> HostIsSuperHostScore = 0;
    HostIsSuperHost = 'no' -> HostIsSuperHostScore = 0),

    % 6. Calcola il punteggio HostHasProfilePictureScore : 5 punti
    (HostHasProfilePicture = 't' -> HostHasProfilePictureScore = 5;
    HostHasProfilePicture = 'f' -> HostHasProfilePictureScore = 0;
    HostHasProfilePicture = 'no' -> HostHasProfilePictureScore = 0),

    % 7- Calcola il punteggio HostIdentifyVerifiedScore : 5 punti
    (HostIdentifyVerified = 't' -> HostIdentifyVerifiedScore = 5;
    HostIdentifyVerified = 'f' -> HostIdentifyVerifiedScore = 0;
    HostIdentifyVerified = 'no' -> HostIdentifyVerifiedScore = 0),
```

```
% 8. Calcola il punteggio LastReviewScore : 5 punti
% Controlla se LastReview è 'no_last_review'
(LastReview = 'no_last_review' -> LastReviewScore = 0;
% Altrimenti, estrai l'anno da LastReview
sub_string(LastReview, 0, 4, _, YearStringLR),
atom_number(YearStringLR, YearLastReview),
% Calcola la differenza di anni
YearDiffLR is 2022 - YearLastReview,
% Assegna il punteggio in base alla differenza di anni
(YearDiffLR = 0 -> LastReviewScore = 5;
YearDiffLR = 1 -> LastReviewScore = 4;
YearDiffLR = 2 -> LastReviewScore = 3;
YearDiffLR = 3 -> LastReviewScore = 2;
YearDiffLR = 4 -> LastReviewScore = 1;
YearDiffLR >= 5 -> LastReviewScore = 0)),

% 9. Calcola il punteggio ReviewScoreRatingScore : 10 punti
(ReviewScoreRating = -1.0 -> ReviewScoreRatingScore = 0;
ReviewScoreRating = 5.0 -> ReviewScoreRatingScore = 10;
ReviewScoreRating >= 4.0, ReviewScoreRating <5 -> ReviewScoreRatingScore = 8;
ReviewScoreRating >= 3.0, ReviewScoreRating <4 -> ReviewScoreRatingScore = 6;
ReviewScoreRating >= 2.0, ReviewScoreRating <3 -> ReviewScoreRatingScore = 4;
ReviewScoreRating >= 1.0, ReviewScoreRating <2 -> ReviewScoreRatingScore = 2;
ReviewScoreRating = 0.0 -> ReviewScoreRatingScore = 0),

% 10. Calcola il punteggio ReviewsPerMonthScore : 5 punti
(ReviewsPerMonth = -1.0 -> ReviewsPerMonthScore = 0;
ReviewsPerMonth >= 10.0 -> ReviewsPerMonthScore = 5;
ReviewsPerMonth >= 8.0, ReviewsPerMonth <10 -> ReviewsPerMonthScore = 4;
ReviewsPerMonth >= 6.0, ReviewsPerMonth <8 -> ReviewsPerMonthScore = 3;
ReviewsPerMonth >= 4.0, ReviewsPerMonth <6 -> ReviewsPerMonthScore = 2;
ReviewsPerMonth >= 1.0, ReviewsPerMonth <4 -> ReviewsPerMonthScore = 1;
ReviewsPerMonth >= 0.0, ReviewsPerMonth <1 -> ReviewsPerMonthScore = 0),
```

```
% 11. Calcola il punteggio IsntantBookableScore : 5 punti
(IstantBookable = 't' -> IsntantBookableScore = 5;
IstantBookable = 'f' -> IsntantBookableScore = 0;
IstantBookable = 'no' -> IsntantBookableScore = 0),

% 12. Calcola il punteggio di ReviewScoreAccuracyScore : 5 punti
(ReviewScoreAccuracy = -1.0 -> ReviewScoreAccuracyScore = 0;
ReviewScoreAccuracy = 5.0 -> ReviewScoreAccuracyScore = 5;
ReviewScoreAccuracy >= 4.0, ReviewScoreAccuracy <5.0 -> ReviewScoreAccuracyScore = 4;
ReviewScoreAccuracy >= 3.0, ReviewScoreAccuracy <4.0 -> ReviewScoreAccuracyScore = 3;
ReviewScoreAccuracy >= 2.0, ReviewScoreAccuracy <3.0 -> ReviewScoreAccuracyScore = 2;
ReviewScoreAccuracy >= 1.0, ReviewScoreAccuracy <2.0 -> ReviewScoreAccuracyScore = 1;
ReviewScoreAccuracy = 0.0 -> ReviewScoreAccuracyScore = 0),

% 13. Calcola il punteggio di NumbeOfReviewsScore : 5 punti
(NumberOfReviews = -1 -> NumbeOfReviewsScore = 0;
NumberOfReviews >= 500 -> NumbeOfReviewsScore = 5;
NumberOfReviews >= 400, NumberOfReviews <500 -> NumbeOfReviewsScore = 4;
NumberOfReviews >= 300, NumberOfReviews <400 -> NumbeOfReviewsScore = 3;
NumberOfReviews >= 200, NumberOfReviews <300 -> NumbeOfReviewsScore = 2;
NumberOfReviews >= 50, NumberOfReviews <200 -> NumbeOfReviewsScore = 1;
NumberOfReviews < 50 -> NumbeOfReviewsScore = 0),

% 14. Calcola il punteggio di NumbeOfReviewsLastYearScore : 5 punti
(NumberOfReviewsLastYear = -1 -> NumbeOfReviewsLastYearScore = 0;
NumberOfReviewsLastYear >= 50 -> NumbeOfReviewsLastYearScore = 5;
NumberOfReviewsLastYear >= 40, NumberOfReviewsLastYear <50 -> NumbeOfReviewsLastYearScore = 4;
NumberOfReviewsLastYear >= 30, NumberOfReviewsLastYear <40 -> NumbeOfReviewsLastYearScore = 3;
NumberOfReviewsLastYear >= 20, NumberOfReviewsLastYear <30 -> NumbeOfReviewsLastYearScore = 2;
NumberOfReviewsLastYear >= 10, NumberOfReviewsLastYear <20 -> NumbeOfReviewsLastYearScore = 1;
NumberOfReviewsLastYear < 10 -> NumbeOfReviewsLastYearScore = 0),
```

```
    % 15. Calcola il punteggio di BedScore : 5 punti
    HalfAccomodates is Accomodates / 2,
    (NumberOfBeds >= HalfAccomodates -> BedScore = 5;
    NumberOfBeds < HalfAccomodates -> BedScore = 2),

    % 16. Calcola il punteggio di BedRoomsScore : 5 punti
    (Accomodates > 0 ->
    RatioBedRooms is float(NumberOfBedRooms) / float(Accomodates),
    (RatioBedRooms >= 0.333 -> BedRoomsScore = 5;
    RatioBedRooms >= 0.25, RatioBedRooms < 0.333 -> BedRoomsScore = 4;
    RatioBedRooms >= 0.2, RatioBedRooms < 0.25 -> BedRoomsScore = 3;
    RatioBedRooms >= 0.167, RatioBedRooms < 0.2 -> BedRoomsScore = 2;
    RatioBedRooms >= 0.143, RatioBedRooms < 0.167 -> BedRoomsScore = 1;
    RatioBedRooms < 0.143 -> BedRoomsScore = 0);
    BedRoomsScore = 0  % Se Accomodates è 0, assegna 0 punti
),

    HostScore is HostSinceScore + HostResponseTimeScore + HostResponseRateScore + HostAcceptanceRateScore + HostIsSuperHostScore + HostHasProfilePictureScore + HostIdentifyVeri
).
"""
```

La regola è suddivisa in diverse parti, ognuna delle quali calcola un punteggio parziale per un attributo specifico. Vediamo nel dettaglio cosa fa ciascuna parte della regola:

1. *HostSinceScore* (10 punti):
   Descrizione: Assegna un punteggio basato sull'anno in cui l'host ha iniziato a ospitare (HostSince). Se l'anno non è specificato, il punteggio è 0. Altrimenti, viene calcolata la differenza tra il 2022 e l'anno specificato, con un massimo di 10 punti.
   Formula: HostSinceScore = min(YearDiff, 10).
2. *HostResponseTimeScore* (5 punti):
   Descrizione: Assegna un punteggio in base al tempo di risposta dell'host (HostResponceTime).
   - "within an hour" → 5 punti
   - "within a few hours" → 4 punti
   - "within a day" → 3 punti
   - "a few days or more" → 1 punto
   - Nessun tempo di risposta → 0 punti
3. *HostResponseRateScore* (10 punti):
   Descrizione: Assegna un punteggio in base al tasso di risposta dell'host (HostResponceRate), espresso in percentuale.
   - 100% → 10 punti
   - 90%-99% → 9 punti
   - 80%-89% → 8 punti
   - E così via fino a:
   - Meno del 10% → 0 punti
4. *HostAcceptanceRateScore* (10 punti):
   Descrizione: Assegna un punteggio in base al tasso di accettazione dell'host (HostAcceptanceRate), espresso in percentuale.
   - 100% → 10 punti
   - 90%-99% → 9 punti
   - 80%-89% → 8 punti
   - E così via fino a:

- Meno del 10% → 0 punti

5. *HostIsSuperHostScore* (5 punti):
   Descrizione: Assegna 5 punti se l'host è un Superhost (HostIsSuperHost), altrimenti 0.
6. *HostHasProfilePictureScore* (5 punti):
   Descrizione: Assegna 5 punti se l'host ha una foto del profilo (HostHasProfilePicture), altrimenti 0.
7. *HostIdentifyVerifiedScore* (5 punti):
   Descrizione: Assegna 5 punti se l'identità dell'host è verificata (HostIdentifyVerified), altrimenti 0.
8. *LastReviewScore* (5 punti):
   Descrizione: Assegna un punteggio basato sull'anno dell'ultima recensione (LastReview).
   - Nessuna recensione recente → 0 punti
   - Recente (2022) → 5 punti
   - Recensione più vecchia di 5 anni → 0 punti
9. *ReviewScoreRatingScore* (10 punti):
   Descrizione: Assegna un punteggio in base alla valutazione delle recensioni (ReviewScoreRating), che va da 0 a 5 stelle.
   - 5 stelle → 10 punti
   - 4-4.9 stelle → 8 punti
   - 3-3.9 stelle → 6 punti
   - E così via fino a:
   - 0 stelle → 0 punti
10. *ReviewsPerMonthScore* (5 punti):
    Descrizione: Assegna un punteggio in base al numero di recensioni ricevute al mese (ReviewsPerMonth).
    - 10 o più recensioni → 5 punti
    - 8-9.9 recensioni → 4 punti
    - E così via fino a:
    - Meno di 1 recensione → 0 punti
11. *IsntantBookableScore* (5 punti):
    Descrizione: Assegna 5 punti se l'host permette la prenotazione immediata (IstantBookable), altrimenti 0.
12. *ReviewScoreAccuracyScore* (5 punti):
    Descrizione: Assegna un punteggio in base alla precisione delle recensioni (ReviewScoreAccuracy), con una scala da 0 a 5 punti.
13. *NumbeOfReviewsScore* (5 punti):
    Descrizione: Assegna un punteggio in base al numero totale di recensioni (NumberOfReviews).
    - 500 o più recensioni → 5 punti

- 400-499 recensioni → 4 punti
- E così via fino a:
- Meno di 50 recensioni → 0 punti

14. *NumbeOfReviewsLastYearScore* (5 punti):
    Descrizione: Assegna un punteggio in base al numero di recensioni ricevute nell'ultimo anno (NumberOfReviewsLastYear).
    - 50 o più recensioni → 5 punti
    - 40-49 recensioni → 4 punti
    - E così via fino a:
    - Meno di 10 recensioni → 0 punti
15. *BedScore* (5 punti):
    Descrizione: Assegna un punteggio in base al numero di letti (NumberOfBeds) rispetto al numero di persone che possono essere ospitate (Accomodates). Se ci sono abbastanza letti, assegna 5 punti; altrimenti 2 punti.
16. *BedRoomsScore* (5 punti):
    Descrizione: Assegna un punteggio in base al rapporto tra il numero di camere da letto (NumberOfBedRooms) e il numero di persone che possono essere ospitate (Accomodates). Un rapporto maggiore assegna un punteggio più alto, fino a un massimo di 5 punti.

Punteggio finale (*HostScore*): (qui non era in grassetto prima)
Il punteggio finale (HostScore) è la somma di tutti i punteggi parziali calcolati sopra. Questo punteggio complessivo riflette una valutazione dell'host basata su vari parametri relativi alla loro attività e alla qualità dell'esperienza offerta.

AT	AU
host_valutation	
79	
63	
60	
73	
43	
73	
68	
25	
51	
43	
69	
17	

Questa regola PROLOG è un esempio di utilizzo PROLOG sia come interrogazione ad una base di conoscenza che elaborazione e calcolo matematico utilizzando delle funzioni.

CAP 2: WEB SEMANTICO

Il dataset utilizzato nel progetto, sebbene ricco di informazioni, non fornisce dettagli sufficienti riguardo al territorio in cui si trovano gli annunci. Per esempio, mentre latitudine e longitudine sono dati utili per localizzare esattamente un luogo sulla mappa, mancano altre informazioni che potrebbero arricchire l'analisi, come l'altitudine della località. L'altitudine è un dato rilevante, soprattutto in contesti turistici, poiché influisce notevolmente sull'attrattiva di una zona. Oltre all'altitudine, un altro dato che può fornire una visione più completa della realtà turistica di un luogo è la popolazione. Classificare un'area in base al numero di abitanti può rivelarsi cruciale per comprendere meglio il tipo di destinazione, che sia un piccolo borgo tranquillo o una metropoli.
Pertanto, l'obiettivo è arricchire il dataset classificando ogni località sia in base all'altitudine, sia in base alla popolazione. Per l'altitudine, l'obiettivo è suddividere i luoghi nelle seguenti categorie: Bassa pianura, Alta pianura, Bassa collina, Alta collina, Bassa montagna, Media montagna, Alta montagna. Parallelamente, per la popolazione, le categorie saranno: Piccolo borgo, Villaggio, Piccola città, Città, Grande città, Metropoli.
Queste informazioni sono ottenute per ogni città tramite una richiesta SPARQL a DBpedia, o in caso di mancanza di informazioni, a Wikidata. Se anche in questo caso non si ottengono i dati necessari, vengono assegnate delle etichette di default per garantire che ogni città nel dataset abbia comunque una classificazione.
Le nuove informazioni acquisite vengono poi integrate nel dataset originale tramite l'aggiunta di tre nuove colonne: **population, elevation, classification.**

Di seguito l'esempio di query SPARQL all'interno dell'ambiente python.

```python
# Funzione per eseguire query su DBpedia
def query_dbpedia(city_name):
    sparql = SPARQLWrapper("http://dbpedia.org/sparql")
    sparql.setQuery(f"""
        SELECT ?population ?elevation ?feature
        WHERE {{
            ?city rdfs:label "{city_name}"@en ;
                  dbo:populationTotal ?population ;
                  dbo:elevation ?elevation .
            OPTIONAL {{
                ?city dbo:type ?feature .
            }}
        }}
        LIMIT 1
    """)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()

    if results["results"]["bindings"]:
        result = results["results"]["bindings"][0]
        population = int(result.get("population", {}).get("value", -1))
        elevation = float(result.get("elevation", {}).get("value", -1.0))
        feature = result.get("feature", {}).get("value", "N/A")

        return population, elevation, feature
    else:
        return -1, -1, "Nessuna informazione"
```

```python
def query_wikidata(city_name):
    sparql = SPARQLWrapper("https://query.wikidata.org/sparql")
    sparql.setQuery(f"""
        SELECT ?population ?elevation ?typeLabel
        WHERE {{
            ?city rdfs:label "{city_name}"@it ;
                  wdt:P31 wd:Q515 ;  # Identifica la città
                  wdt:P1082 ?population ;  # Popolazione
                  wdt:P2044 ?elevation .  # Elevazione

            OPTIONAL {{
                ?city wdt:P206 ?location .  # P206: Luogo di ubicazione naturale
                ?location rdfs:label ?locationLabel .
                FILTER(LANG(?locationLabel) = "it")
            }}

            OPTIONAL {{
                ?city wdt:P279 ?type .  # Classificazione geografica
                ?type rdfs:label ?typeLabel .
                FILTER(LANG(?typeLabel) = "it")
            }}
        }}
        LIMIT 1
    """)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()

    if results["results"]["bindings"]:
        result = results["results"]["bindings"][0]
        population = int(result.get("population", {}).get("value", -1))
        elevation = float(result.get("elevation", {}).get("value", -1))
        type_label = result.get("typeLabel", {}).get("value", "N/A")

        return population, elevation, type_label
    else:
        return -1, -1, "Nessuna informazione"
```

CAP 3: APPRENDIMENTO NON SUPERVISIONATO

L'obiettivo di questo modulo è quello di catalogare le strutture presenti in un annuncio per suggerire all'utente delle strutture affini a quelle di suo interesse. Per raggiungere questo scopo, il progetto si concentra sull'utilizzo di specifiche feature numeriche come latitudine, longitudine, altitudine, popolazione, e prezzo. Oltre a queste, vengono considerate anche altre feature meno rilevanti (ma con lo stesso peso all'interno dell'algoritmo) e le feature relative ai servizi, le quali sono state ottenute tramite tecniche di feature engineering, in particolare con l'applicazione di one-hot encoding.
Questa è la funzione per il preprocessing del dataset:

```python
def clear_dataset_unsupervised_learning(data):

    data = data.drop('source', axis=1)
    data = data.drop('name', axis=1)
    data = data.drop('host_since', axis=1)
    data = data.drop('host_location', axis=1)
    data = data.drop('host_response_time', axis=1)
    data = data.drop('host_listings_count', axis=1)
    data = data.drop('host_total_listings_count', axis=1)
    data = data.drop('host_verifications', axis=1)
    data = data.drop('id', axis=1)
    data = data.drop('host_id', axis=1)
    data = data.drop('calculated_host_listings_count', axis=1)
    data = data.drop('availability_90', axis=1)
    data = data.drop('calculated_host_listings_count_entire_homes', axis=1)
    data = data.drop('reviews_per_month', axis=1)

    # Elimino le righe con valore altitutidine -1 perchè non hanno contenuto informativo
    data = data[data['elevation'] != -1]

    # Inizializza un dizionario vuoto per memorizzare i servizi unici e il loro conteggio
    amenities_count = {}

    # Funzione per aggiornare il dizionario con il conteggio dei servizi
    def update_amenities_count(amenities_str):
        if pd.notna(amenities_str):  # Controlla se il valore non è NaN
            # Dividi la stringa di servizi in singoli elementi
            amenities_list = amenities_str.split(", ")
            # Aggiorna il conteggio dei servizi nel dizionario
            for amenity in amenities_list:
                if amenity in amenities_count:
                    amenities_count[amenity] += 1
                else:
                    amenities_count[amenity] = 1

    # Applica la funzione a tutte le righe della colonna 'amenities'
    data['amenities'].apply(update_amenities_count)

    # Ordina il dizionario in base ai valori (conteggi) in ordine decrescente
    sorted_amenities_count = dict(sorted(amenities_count.items(), key=lambda item: item[1], reverse=True))
    x = 320
    # Seleziona i primi x servizi
    top_amenities = list(sorted_amenities_count.keys())[:x]
```

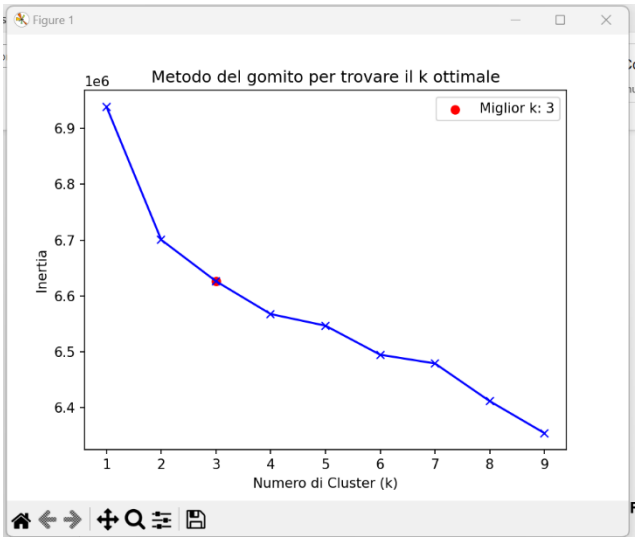
```python
    # Creare un DataFrame vuoto con colonne per ciascuna amenity
    amenities_df = pd.DataFrame(0, index=data.index, columns=top_amenities)

    # Funzione per aggiornare il DataFrame delle amenities
    def update_amenities_columns(row):
        if pd.notna(row['amenities']):
            amenities_list = row['amenities'].split(", ")
            for amenity in amenities_list:
                if amenity in top_amenities:
                    amenities_df.at[row.name, amenity] = 1

    # Applicare la funzione a ogni riga
    data.apply(update_amenities_columns, axis=1)

    # Concatenare il DataFrame delle amenities con il DataFrame originale
    data = pd.concat([data, amenities_df], axis=1)

    # Deframmentare il DataFrame
    data = data.copy()

    print("Numero di servizi unici:", len(amenities_count))
    print(len(amenities_count))

    def update_region_columns(data):
        # Lista delle regioni
        regions = ['Bari', 'Foggia', 'Lecce', 'Taranto', 'Brindisi', 'Barletta-Andria-Trani']

        # Aggiungi una colonna per ciascuna regione ed inizializzala a 0
        for region in regions:
            data[region] = 0

        # Funzione per aggiornare le colonne delle regioni
        def update_region(row):
            if row['neighbourhood_group_cleansed'] in regions:
                row[row['neighbourhood_group_cleansed']] = 1
            return row

        # Applica la funzione per aggiornare le colonne delle regioni
        data = data.apply(update_region, axis=1)

        return data

    data = update_region_columns(data)

    # Rimuove la colonna 'amenities' dal DataFrame in quanto è stata sostituita da colonne binarie per i servizi
    data = data.drop('amenities', axis=1)
```

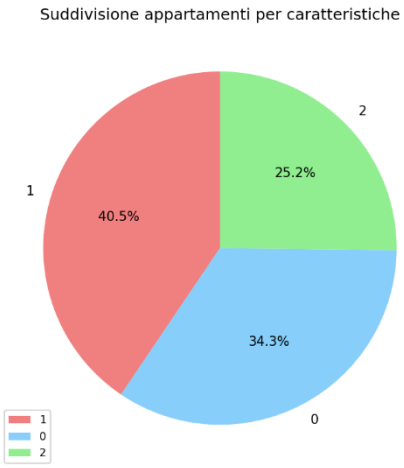
```python
    # Elimino tutte le righe con valori mancanti
    data = data.dropna()
    return data

def clean_percentage_string(percentage_str):
    # Controlla se la stringa non è vuota o nulla
    if percentage_str and isinstance(percentage_str, str):
        # Rimuovi il simbolo '%' e converti il valore a intero
        return int(percentage_str.replace('%', '').strip())
    return -1

def clear_dataset_supervised_learning(data):
    # Applicazione su una colonna specifica del DataFrame
    data['host_response_rate'] = data['host_response_rate'].apply(lambda x: clean_percentage_string(x) if pd.notna(x) else x)
    data['host_acceptance_rate'] = data['host_acceptance_rate'].apply(lambda x: clean_percentage_string(x) if pd.notna(x) else x)
    data = data[data['host_response_rate'] != -1]
    data = data[data['host_acceptance_rate'] != -1]

    # Conversione delle colonne booleane in valori binari
    data['host_is_superhost'] = data['host_is_superhost'].apply(lambda x: 1 if x == 't' else 0)
    data['host_has_profile_pic'] = data['host_has_profile_pic'].apply(lambda x: 1 if x == 't' else 0)
    data['host_identity_verified'] = data['host_identity_verified'].apply(lambda x: 1 if x == 't' else 0)
    data['has_availability'] = data['host_identity_verified'].apply(lambda x: 1 if x == 't' else 0)
    data['instant_bookable'] = data['instant_bookable'].apply(lambda x: 1 if x == 't' else 0)

    data = data.drop('first_review', axis=1)
    data = data.drop('last_review', axis=1)
    data = data.drop('neighbourhood_cleansed', axis=1)
    data = data.drop('neighbourhood_group_cleansed', axis=1)
    data = data.drop('classification', axis=1)
    # FARE FEATURE ENGINEERING SU QUESTE COLONNE
    # Crea colonne dummy per 'property_type' e 'room_type' con valori 0 e 1
    property_type_dummies = pd.get_dummies(data['property_type'], prefix='property_type').astype(int)
    room_type_dummies = pd.get_dummies(data['room_type'], prefix='room_type').astype(int)

    # Unisci le colonne dummy al dataset originale
    data = pd.concat([data, property_type_dummies, room_type_dummies], axis=1)

    # Rimuovi le colonne originali 'property_type' e 'room_type' se non sono più necessarie
    data = data.drop('property_type', axis=1)
    data = data.drop('room_type', axis=1)

    # Handle missing values
    data.fillna(data.mean(), inplace=True)

    return data
```

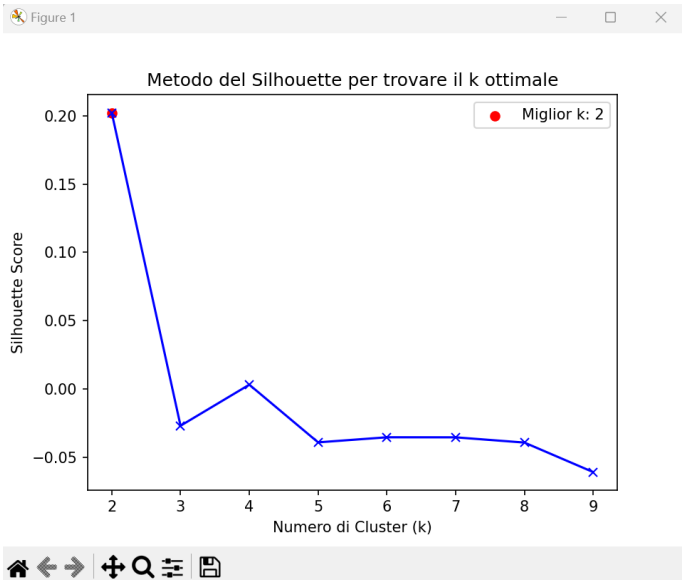
Dopo aver pulito il dataset i risultati preliminari hanno evidenziato la presenza di cluster di dimensioni piuttosto elevate. Utilizzando sia la regola del gomito sia l'analisi del coefficiente di silhouette, ho identificato un numero ottimale di cluster compreso tra k=3 e k=6. Tuttavia, alcuni cluster contengono oltre 10.000 elementi. Considerando questo, ho ritenuto più opportuno utilizzare il clustering solo come un passo preliminare per ridurre lo spazio di ricerca per applicare un algoritmo di calcolo delle similarità e migliorarne le prestazioni.
Eseguo quindi come mostrato l'algoritmo Kmeans per due valori di k ottenuti sia dalla regola del gomito sia dalla regola della silhouette.

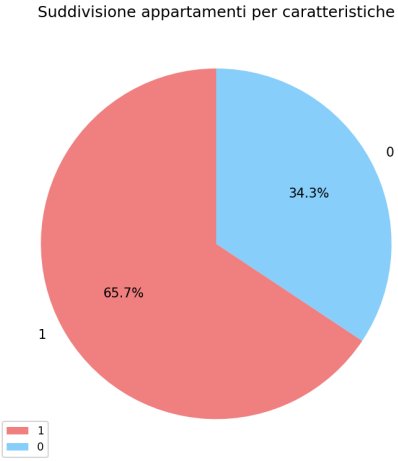
```python
#Funzione che esegue il kmeans sul dataset e restituisce le etichette e i centroidi
def calcolaCluster(dataSet, metodo):

    numerical_features = dataSet.select_dtypes(include=[np.number])
    #Standardizzazione delle feature numeriche
    scaler = StandardScaler()
    numerical_features_scaled = scaler.fit_transform(numerical_features)

    # Stampa una sola volta tutti i nomi delle colonne numeriche
    print("Campi numerici utilizzati per il clustering:", ', '.join(numerical_features))

    if metodo == 'gomito':
        k = regolaGomito(numerical_features_scaled)
    elif metodo == 'silhouette':
        k = regolaSilhouette(numerical_features_scaled)
    else:
        raise ValueError("Metodo non valido. Usa 'gomito' o 'silhouette'.")


    km = KMeans(n_clusters=k,n_init=10,init='k-means++')
    km = km.fit(numerical_features_scaled)
    etichette = km.labels_
    centroidi = km.cluster_centers_
    return etichette, centroidi
```

Qui di seguito i risultati ottenuti e una rappresentazione grafica della distribuzione dei cluster



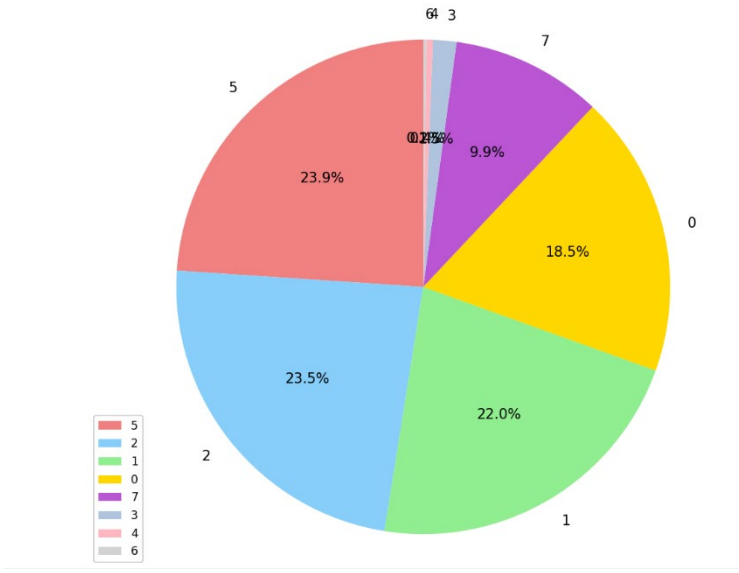






Riflessione sulla Curva del Metodo del Gomito

Il grafico del metodo del gomito mostra come l'inertia diminuisce con l'aumentare del numero di cluster (k). Sebbene ci sia un punto di gomito a k = 3, la curva continua a scendere in modo significativo fino a k = 8. Questo suggerisce che ulteriori suddivisioni dei dati continuano a ridurre l'inertia, sebbene a un tasso decrescente rispetto al punto con k = 2.



Il grafico a torta mostra la distribuzione degli annunci di Airbnb in 9 cluster distinti. Osservando la distribuzione, notiamo che alcuni cluster sono ancora molto grandi, mentre altri sono relativamente piccoli. In particolare, i cluster 0, 1, 2, 5 e 7 rappresentano una porzione significativa degli annunci e necessitano di ulteriori analisi per fornire suggerimenti personalizzati agli utenti. Questo significa utilizzare un algoritmo di similarità all'interno di ciascun cluster per identificare le strutture più affini a quelle in cui gli utenti si sono trovati bene.

Invece, i cluster 3, 4 e 6 sono molto piccoli e probabilmente rappresentano categorie di annunci di nicchia. Questi cluster possono essere utilizzati direttamente per offrire strutture altamente specifiche agli utenti senza ulteriori suddivisioni.

Il clustering con k = 9 ha permesso di identificare una suddivisione più dettagliata degli annunci di Airbnb. Mentre i cluster grandi richiedono un'analisi approfondita e ulteriori elaborazioni per suggerire strutture affini, i cluster di nicchia possono essere utilizzati direttamente per fornire raccomandazioni altamente specifiche.

CAP. 4 APPRENDIMENTO SUPERVISIONATO

I modelli di regressione usati sono: **RandomForestRegressor**, **Ridge** e **Lasso**. L'obiettivo è stato quello di trovare i migliori iperparametri per ciascun modello e confrontarne le prestazioni utilizzando la tecnica di **K-Fold Cross-Validation** e le metriche di errore come il **Mean Absolute Error (MAE)**, il **Mean Squared Error (MSE)** e il **R2**.

Modelli Utilizzati

1. **RandomForestRegressor**
   Il **RandomForestRegressor** è un modello di ensemble che utilizza più alberi decisionali per migliorare la previsione e ridurre la varianza del modello. Ogni albero nella foresta viene addestrato su un diverso sottoinsieme di dati, un processo noto come "bagging". Di seguito sono riportati gli iperparametri chiave considerati nel tuning del modello:
   - **n_estimators**: Il numero di alberi nella foresta.
   - **max_depth**: La profondità massima degli alberi.
   - **min_samples_split**: Il numero minimo di campioni richiesti per suddividere un nodo.
   - **min_samples_leaf**: Il numero minimo di campioni richiesti per essere in un nodo foglia.
   - **RandomForest__bootstrap:** determina se il campionamento con ripetizione (bootstrap sampling) deve essere utilizzato quando si costruiscono gli alberi della foresta.
2. **Ridge Regression**
   La regressione Ridge è un'estensione della regressione lineare che aggiunge una penalizzazione all'equazione di regressione per prevenire l'overfitting, utilizzando un termine di regolarizzazione L2.
   - **alpha**: Il parametro di regolarizzazione che controlla l'intensità della penalizzazione.
   - **solver**: Il metodo di risoluzione utilizzato dall'algoritmo di ottimizzazione.

3. **Lasso Regression**
   Il modello di regressione Lasso introduce una penalizzazione L1, che può ridurre a zero i coefficienti di alcuni feature, risultando in una selezione automatica delle feature.
   - **alpha**: Il parametro di regolarizzazione per il Lasso.
   - **max_iter**: Il numero massimo di iterazioni per il solver.

Metodo di Selezione degli Iperparametri

Ho utilizzato il metodo della **Grid Search con Cross Validation** per trovare i migliori iperparametri per ciascun modello. Questo processo coinvolge:

- **Grid Search**: Creazione di una griglia di combinazioni di iperparametri da testare.
- **Cross Validation**: Valutazione delle prestazioni del modello suddividendo il dataset in più fold (in questo caso, 10 fold ripetuti 5 volte).

Il processo è stato implementato utilizzando la funzione GridSearchCV della libreria **scikit-learn**. La metrica di valutazione utilizzata durante la cross-validation è stata il **Mean Squared Error (MSE)**, misurato in valore negativo per conformità con le convenzioni di scikit-learn.

Iperparametri Considerati

RandomForestRegressor

- **n_estimators**: [50, 100, 200]
  Numero di alberi nella foresta.
- **max_depth**: [10, 20, 30, None]
  Profondità massima degli alberi.
- **min_samples_split**: [2, 5, 10]
  Numero minimo di campioni per suddividere un nodo.
- **min_samples_leaf**: [1, 2, 4]
  Numero minimo di campioni in un nodo foglia.
- **bootstrap**: [True, False]
  Se usare il campionamento con sostituzione.

Ridge Regression

- **alpha**: [0.01, 0.1, 1, 10, 100, 1000]
  Intensità della regolarizzazione.
- **solver**: ['auto', 'svd', 'cholesky', 'saga', 'lsqr']
  Algoritmo di ottimizzazione.

Lasso Regression

- **alpha**: [0.001, 0.01, 0.1, 1, 10, 100, 1000]
  Intensità della regolarizzazione.
- **max_iter**: [500, 1000, 2000, 5000, 10000]
  Numero massimo di iterazioni del solver.

Gli iperparametri restituiti sono:

Random Forest

n_estimators	100
max_depth	None
min_samples_split	5
min_samples_leaf	2
bootstrap	True

RIDGE

alpha	10
solver	‘auto’

LASSO

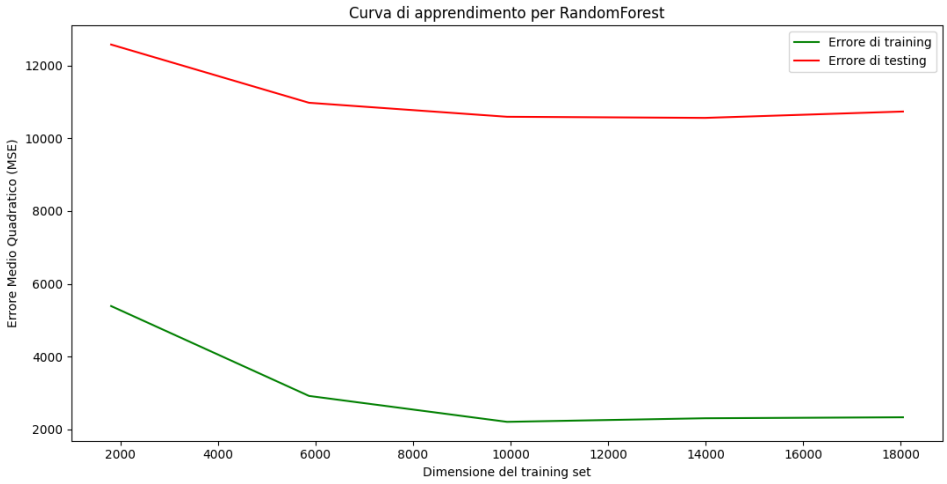
alpha	5
max_iter	2

Confronto tra Ridge e Lasso

- **Ridge Regression:**
  - Riduce l'ampiezza dei coefficienti, ma non li azzera.
  - È utile quando tutte le feature sono rilevanti, ma alcune devono essere ridotte.
- **Lasso Regression:**
  - Può ridurre i coefficienti a zero, eliminando le feature meno rilevanti.
  - È utile quando si sospetta che molti feature siano irrilevanti.

Metriche Utilizzate

1. **Mean Absolute Error (MAE)**:
   - **Descrizione**: Rappresenta la media degli errori assoluti tra i valori predetti e quelli reali. È utile per interpretazioni dirette della magnitudine dell'errore.
2. **Mean Squared Error (MSE)**:
   - **Descrizione**: La media degli errori al quadrato tra i valori predetti e quelli reali. Penalizza maggiormente gli errori più grandi rispetto al MAE.
3. **Coefficient of Determination R2R^2R2**:
   - **Descrizione**: Indica quanto bene il modello riesce a spiegare la variabilità dei dati. Un valore vicino a 1 indica una buona spiegazione dei dati.



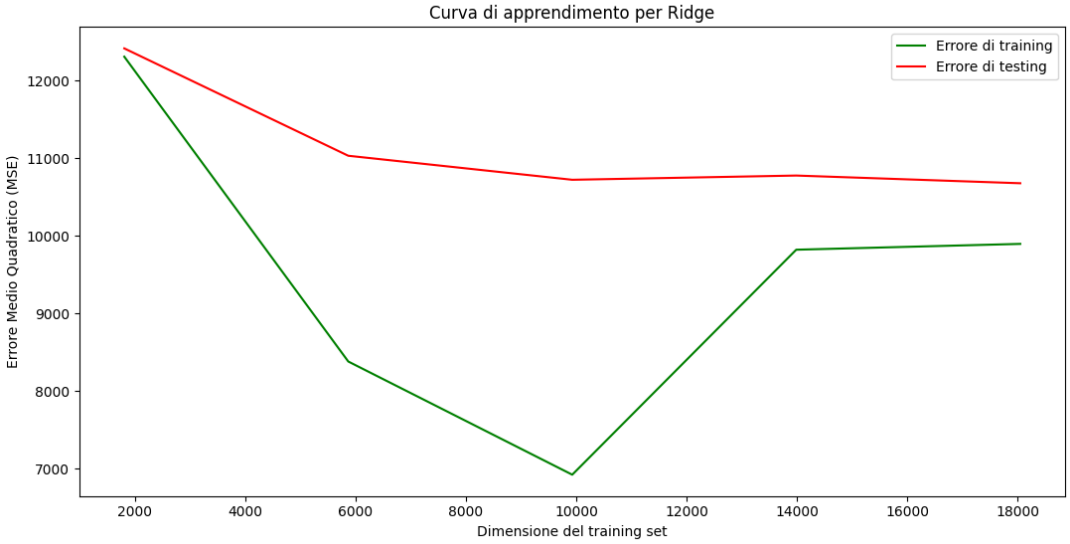
Interpretazione del Grafico

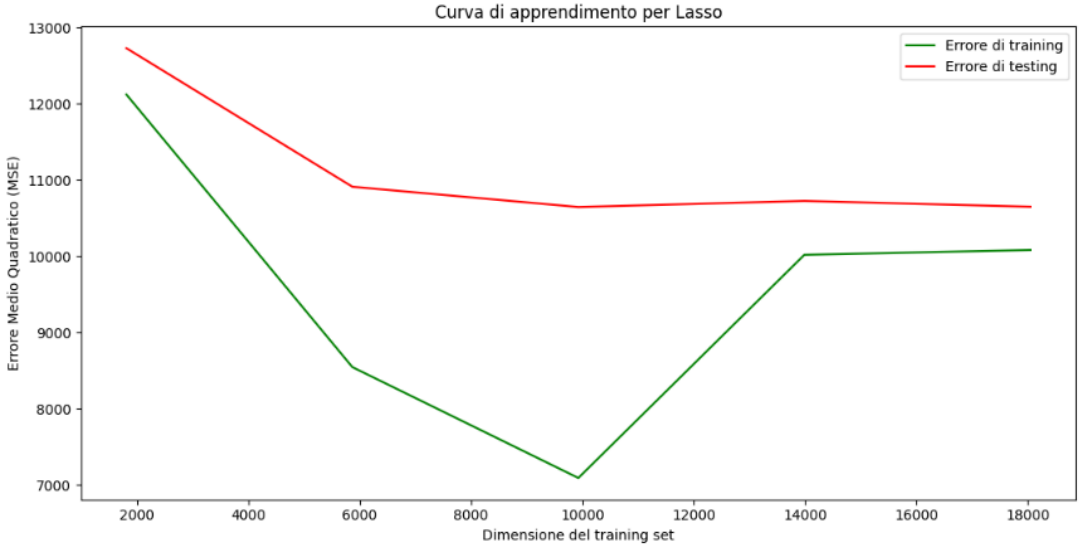
1. **Errore di Training**:
   - All'inizio, con un numero ridotto di dati di training, l'errore di training è relativamente alto.
   - Man mano che la dimensione del set di training aumenta, l'errore di training diminuisce rapidamente.
   - Dopo un certo punto, l'errore di training si stabilizza, indicando che il modello si è adattato bene ai dati di training e ulteriori dati di training non migliorano significativamente l'errore.
2. **Errore di Testing**:
   - L'errore di testing è inizialmente molto alto, indicando che il modello non generalizza bene sui nuovi dati.
   - Con l'aumento della dimensione del set di training, l'errore di testing diminuisce, mostrando che il modello sta imparando a generalizzare meglio.
   - Tuttavia, dopo un certo punto, l'errore di testing si stabilizza o addirittura inizia a salire leggermente, suggerendo che il modello ha raggiunto la sua capacità massima di generalizzazione con i dati disponibili.

Conclusioni

- **Underfitting all'inizio**: Con pochi dati di training, il modello è in underfitting, ovvero non è in grado di catturare le complessità nei dati, come indicato dall'alto errore di training e testing.
- **Apprendimento**: Man mano che si aggiungono più dati di training, il modello migliora sia in termini di errore di training che di testing, indicando che sta imparando a rappresentare meglio la relazione tra le feature e l'output.
- **Stabilizzazione**: Dopo una certa dimensione del set di training, l'errore di testing si stabilizza, suggerendo che l'aggiunta di ulteriori dati di training non porta a significativi miglioramenti nella performance del modello.

- **Capacità del Modello**: Il punto in cui le curve si stabilizzano indica la capacità del modello di Random Forest con i dati forniti. Un aumento dell'errore di testing dopo questo punto potrebbe indicare che il modello sta iniziando a overfittare i dati di training.





Confronto tra Lasso Regression e Ridge Regression

- **Errore di Training**:
  - Entrambi i modelli mostrano una diminuzione dell'errore di training con l'aumento dei dati di training fino a un certo punto, seguita da un leggero aumento.
  - Tuttavia, il modello Lasso mostra un minimo più pronunciato intorno ai 10.000 esempi, mentre nel modello Ridge l'aumento dell'errore di training è meno pronunciato.
- **Errore di Testing**:
  - L'errore di testing diminuisce in entrambi i modelli con l'aumento dei dati di training.

- Nel modello Lasso, l'errore di testing si stabilizza intorno ai 10.000 esempi, simile al modello Ridge.
- La stabilizzazione dell'errore di testing è più evidente nel modello Lasso rispetto al modello Ridge, suggerendo che il modello Lasso potrebbe avere una maggiore capacità di generalizzazione con i dati forniti.

Conclusioni

- **Underfitting all'inizio**: Con pochi dati di training, entrambi i modelli sono in underfitting, come indicato dall'alto errore di training e testing.
- **Apprendimento**: Man mano che si aggiungono più dati di training, entrambi i modelli migliorano sia in termini di errore di training che di testing, indicando che stanno imparando a rappresentare meglio la relazione tra le feature e l'output.
- **Sovra-adattamento del training set**: Dopo un certo punto, entrambi i modelli mostrano un leggero aumento dell'errore di training, suggerendo un sovra-adattamento ai dati di training.
- **Stabilizzazione**: L'errore di testing si stabilizza in entrambi i modelli dopo una certa dimensione del set di training, indicando che l'aggiunta di ulteriori dati di training non porta a significativi miglioramenti nella performance del modello.
- **Capacità del Modello**: Entrambi i modelli mostrano una capacità di generalizzazione simile, ma il modello Lasso sembra avere una maggiore stabilità nell'errore di testing.

In sintesi, mentre entrambi i modelli mostrano tendenze simili nelle curve di apprendimento, il modello Lasso sembra avere una maggiore capacità di generalizzazione con i dati disponibili, come indicato dalla stabilizzazione dell'errore di testing.

CAP. 5 ESTENSIONI E SVILUPPI FUTURI.

Considerando che il dataset scaricato dalla piattaforma **Kaggle** fa parte di un gruppo di dataset contenente anche dati specifici per regioni come **Sicilia** e **Trentino**, il codice potrebbe essere facilmente adattato per analizzare queste aree. Effettuando piccole modifiche ai parametri e alla selezione dei dati, si potrebbe estendere l'analisi per confrontare le performance delle strutture in diverse regioni italiane. Questo permetterebbe un confronto interessante tra aree turisticamente differenti in termini di attrattività e prezzi medi.
Inoltre, sono presenti dati anche per specifiche **province** all'interno di alcune regioni italiane. Un passo successivo potrebbe essere quello di focalizzarsi su analisi a livello provinciale, che potrebbe rivelare tendenze locali significative.

1. Richiesta di Dati Nazionali e Aggiornati

Un'idea importante per estendere il progetto sarebbe quella di **contattare direttamente Airbnb** e richiedere un rilascio di un dataset nazionale, aggiornato al 2024, contenente informazioni su tutte le regioni italiane. Questo permetterebbe di avere un dataset molto più completo e aggiornato, migliorando la capacità predittiva dei modelli e aumentando la rilevanza dei risultati. Avere accesso a dati su scala nazionale offrirebbe la possibilità di costruire modelli capaci di **generalizzare meglio** su tutto il territorio italiano, nonché di esplorare ulteriori correlazioni tra prezzi e altre caratteristiche regionali o provinciali.

2. Ampliamento del Dataset per Migliorare l'Apprendimento Supervisionato

Per migliorare l'**apprendimento supervisionato**, si potrebbe lavorare con un **dataset più grande** che includa più esempi e un maggior numero di feature. Avere più esempi migliora la capacità dei modelli di apprendere le relazioni complesse tra le variabili e riduce il rischio di overfitting.
In alternativa, si potrebbe prendere una porzione di dataset per ogni regione e **creare un dataset unico e più ampio**. Questo permetterebbe di costruire un modello in grado di **generalizzare meglio** e fornire previsioni più accurate sui prezzi in diverse aree geografiche. Includere feature relative a diverse regioni potrebbe aiutare il modello a catturare dinamiche economiche locali che altrimenti andrebbero perse.

3. Analisi delle Feature e Studio delle Correlazioni

Un'altra area di approfondimento potrebbe essere lo studio di quali feature influenzano maggiormente il costo delle strutture. Oltre alle feature già presenti nel dataset, si potrebbe prendere in considerazione l'inclusione di dati aggiuntivi non presenti, come la **vicinanza al centro città** o ad altri punti di interesse. Per esempio, calcolare la **distanza dal centro città** utilizzando le coordinate di latitudine e longitudine delle strutture rispetto a quelle del centro potrebbe rivelarsi un'informazione cruciale per determinare il prezzo.
Si potrebbe pensare di calcolare in automatico questa distanza e aggiungerla come nuova feature nel dataset, migliorando così le performance del modello. Un'altra variabile che potrebbe essere aggiunta è la **popolarità turistica** di una specifica area o la **stagionalità** delle prenotazioni.

4. Apprendimento Supervisato con Modelli Più Complessi

In termini di **modelli di apprendimento supervisionato**, oltre a Ridge, Lasso e Random Forest, si potrebbero testare modelli più avanzati come:

- **Gradient Boosting Machines (GBM)**, inclusi **XGBoost**, **LightGBM** e **CatBoost**, che spesso forniscono risultati eccellenti nelle competizioni di machine learning.
- **Reti neurali profonde** (Deep Learning), soprattutto in presenza di dataset molto ampi e feature complesse. Le reti neurali sono in grado di catturare relazioni non lineari tra le feature e di generare modelli molto potenti.

L'uso di tecniche come **feature importance** o **SHAP (SHapley Additive exPlanations)** aiuterebbe a interpretare meglio i modelli complessi e a identificare quali variabili contribuiscono maggiormente alle predizioni dei prezzi.

5. Apprendimento Non Supervisionato e Raccomandazioni

Per quanto riguarda l'**apprendimento non supervisionato**, i cluster generati potrebbero essere utilizzati per ridurre lo spazio di ricerca degli algoritmi di raccomandazione. Ad esempio, una volta identificati i cluster, questi potrebbero essere usati per suggerire strutture simili agli utenti su piattaforme di prenotazione come Airbnb, basandosi sui loro gusti e preferenze. Questo approccio ridurrebbe anche l'impatto computazionale degli algoritmi, ottimizzando l'uso delle risorse e riducendo il consumo energetico.