CASO DI STUDIO DEL CORSO IN

INGEGNERIA DELLA CONOSCENZA

Airbnb in NYC:

Strategie Smart per trovare gli airbnb migliori a New York

Autori:

Mauro Carlucci

E-mail: *m.carlucci66@studenti.uniba.it*

Matricola: *758284*

Paola Campaniello

E-mail: *p.campaniello@studenti.uniba.it*

Matricola: *758277*

Docente:

Prof. Nicola Fanizzi

INTRODUZIONE

Il nostro progetto si concentra sull'analisi approfondita dei prezzi degli affitti delle case e delle stanze di Airbnb a New York, con l'obiettivo di identificare le opportunità migliori.
Il nostro obiettivo è quello di utilizzare i dati esistenti per individuare il prezzo migliore per un alloggio date determinate caratteristiche.

 **GitHub Repository:** https://github.com/maurocarlu/StudioAirbnb_ICON.git

 **Kaggle dataset: https://www.kaggle.com/datasets/arianazmoudeh/airbnbopendata/data**

1. Analisi dei dati

La fase introduttiva consiste nel leggere il dataset e comprenderne la struttura, focalizzandosi sulla risoluzione di alcune problematiche legate alle celle vuote e l'individuazione di correlazioni tra la colonna target e le altre feature.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
import plotly.express as px
```

1.1 LETTURA E COMPRENSIONE DEL DATASET

Implementiamo la libreria pandas in modo da poter modellare il dataset.

```
[2]: airbnb = pd.read_csv('../dataset/Originali/Airbnb_Open_Data.csv')
     airbnb.head()
```

[2]:

	id	NAME	host id	host_identity_verified	host name	neighbourhood group	neighbourhood	lat	long	country	...
0	1001254	Clean & quiet apt home by the park	80014485718	unconfirmed	Madaline	Brooklyn	Kensington	40.64749	-73.97237	United States	...
1	1002102	Skylit Midtown Castle	52335172823	verified	Jenna	Manhattan	Midtown	40.75362	-73.98377	United States	...
2	1002403	THE VILLAGE OF HARLEM....NEW YORK !	78829239556	NaN	Elise	Manhattan	Harlem	40.80902	-73.94190	United States	...
3	1002755	NaN	85098326012	unconfirmed	Garry	Brooklyn	Clinton Hill	40.68514	-73.95976	United States	...
4	1003689	Entire Apt: Spacious Studio/Loft by central park	92037596077	verified	Lyndon	Manhattan	East Harlem	40.79851	-73.94399	United States	...

5 rows × 26 columns

[2]:

	service fee	minimum nights	number of reviews	last review	reviews per month	review rate number	calculated host listings count	availability 365	house_rules	license
0	$193	10.0	9.0	10/19/2021	0.21	4.0	6.0	286.0	Clean up and treat the home the way you'd like...	NaN
1	$28	30.0	45.0	5/21/2022	0.38	4.0	2.0	228.0	Pet friendly but please confirm with me if the...	NaN
2	$124	3.0	0.0	NaN	NaN	5.0	1.0	352.0	I encourage you to use my kitchen, cooking and...	NaN
3	$74	30.0	270.0	7/5/2019	4.64	4.0	1.0	322.0	NaN	NaN
4	$41	10.0	9.0	11/19/2018	0.10	3.0	1.0	289.0	Please no smoking in the house, porch or on th...	NaN

5 ro

```
[4]: airbnb.shape
```

```
[4]: (102599, 26)
```

```
[5]: airbnb.describe()
```

	id	host id	lat	long	Construction year	minimum nights	number of reviews	reviews per month	review rate number	calculated host listings count	availability 365
count	1.025990e+05	1.025990e+05	102591.000000	102591.000000	102385.000000	102190.000000	102416.000000	86720.000000	102273.000000	102280.000000	102151.000000
mean	2.914623e+07	4.925411e+10	40.728094	-73.949644	2012.487464	8.135845	27.483743	1.374022	3.279106	7.936605	141.133254
std	1.625751e+07	2.853900e+10	0.055857	0.049521	5.765556	30.553781	49.508954	1.746621	1.284657	32.218780	135.435024
min	1.001254e+06	1.236005e+08	40.499790	-74.249840	2003.000000	-1223.000000	0.000000	0.010000	1.000000	1.000000	-10.000000
25%	1.508581e+07	2.458333e+10	40.688740	-73.982580	2007.000000	2.000000	1.000000	0.220000	2.000000	1.000000	3.000000
50%	2.913660e+07	4.911774e+10	40.722290	-73.954440	2012.000000	3.000000	7.000000	0.740000	3.000000	1.000000	96.000000
75%	4.320120e+07	7.399650e+10	40.762760	-73.932350	2017.000000	5.000000	30.000000	2.000000	4.000000	2.000000	269.000000
max	5.736742e+07	9.876313e+10	40.916970	-73.705220	2022.000000	5645.000000	1024.000000	90.000000	5.000000	332.000000	3677.000000

```
[6]: airbnb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 102599 entries, 0 to 102598
Data columns (total 26 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   id                              102599 non-null  int64  
 1   NAME                            102349 non-null  object 
 2   host id                         102599 non-null  int64  
 3   host_identity_verified          102310 non-null  object 
 4   host name                       102193 non-null  object 
 5   neighbourhood group             102570 non-null  object 
 6   neighbourhood                   102583 non-null  object 
 7   lat                             102591 non-null  float64
 8   long                            102591 non-null  float64
 9   country                         102067 non-null  object 
 10  country code                    102468 non-null  object 
 11  instant_bookable                102494 non-null  object 
 12  cancellation_policy             102523 non-null  object 
 13  room type                       102599 non-null  object 
 14  Construction year               102385 non-null  float64
 15  price                           102352 non-null  object 
 16  service fee                     102326 non-null  object 
 17  minimum nights                  102190 non-null  float64
 18  number of reviews               102416 non-null  float64
 19  last review                     86706 non-null   object 
 20  reviews per month               86720 non-null   float64
 21  review rate number              102273 non-null  float64
 22  calculated host listings count  102280 non-null  float64
 23  availability 365                102151 non-null  float64
 24  house_rules                     50468 non-null   object 
 25  license                         2 non-null       object 
dtypes: float64(9), int64(2), object(15)
memory usage: 20.4+ MB
```

Individuiamo come target la colonna ‘price’, essendo una colonna numerica si tratta dunque di un task di regressione.

1.2 PRE-PROCESSING DEL DATASET

Dopo aver concluso la fase inziale e la fase successiva, relativa all'analisi strutturale del dataset, passiamo alla fase del pre-processing, nella quale andremo ad effettuare un cleaning dei dati.

```
: columns_to_drop = ['country', 'country code', 'NAME', 'license','host name', 'house_rules']
  airbnb = airbnb.drop(columns=columns_to_drop, errors='ignore')
  airbnb.head()
```

	id	host id	host_identity_verified	neighbourhood group	neighbourhood	lat	long	instant_bookable	cancellation_policy	room type
0	1001254	80014485718	unconfirmed	Brooklyn	Kensington	40.64749	-73.97237	False	strict	Private room
1	1002102	52335172823	verified	Manhattan	Midtown	40.75362	-73.98377	False	moderate	Entire home/apt
2	1002403	78829239556	NaN	Manhattan	Harlem	40.80902	-73.94190	True	flexible	Private room
3	1002755	85098326012	unconfirmed	Brooklyn	Clinton Hill	40.68514	-73.95976	True	moderate	Entire home/apt
4	1003689	92037596077	verified	Manhattan	East Harlem	40.79851	-73.94399	False	moderate	Entire home/apt

	Construction year	price	service fee	minimum nights	number of reviews	last review	reviews per month	review rate number	calculated host listings count	availability 365
0	2020.0	$966	$193	10.0	9.0	10/19/2021	0.21	4.0	6.0	286.0
1	2007.0	$142	$28	30.0	45.0	5/21/2022	0.38	4.0	2.0	228.0
2	2005.0	$620	$124	3.0	0.0	NaN	NaN	5.0	1.0	352.0
3	2005.0	$368	$74	30.0	270.0	7/5/2019	4.64	4.0	1.0	322.0
4	2009.0	$204	$41	10.0	9.0	11/19/2018	0.10	3.0	1.0	289.0

<u>Le colonne eliminate sono le seguenti:</u>

'country' e 'country_code' in quanto il dataset fa riferimento a tutti annunci sul territorio americano e dunque questi dati non servono per il nostro caso.

'NAME' In quanto essa rappresenta la descrizione dell'annuncio della casa/stanza in affitto e dunque, cioè non serve per la nostra analisi.

'license' in quanto tale colonna è sempre vuota

'host name' in quanto essa rappresenta il nome dell'host es: Mauro, Paola, Michele e dunque tale dato non è necessario per la nostra analisi

'house_rules' in quanto tale campo rappresenta le regole della casa es: Non fumare, non portare animali, no feste. Dunque, questo campo non ci è utile.

La metrica 'availability 365' indica il numero massimo di giorni in cui una stanza di un B&B può essere affittata. Durante l'analisi dei dati nel nostro dataset, abbiamo osservato la presenza di alcuni valori negativi o superiori a 365 associati a questa metrica. I valori superiori sono stati regolati a 365, rappresentando il limite massimo, mentre quelli negativi sono stati convertiti in valori positivi.

```
airbnb['availability 365'].describe()
```

```
count    101124.000000
mean        141.033642
std         135.410940
min         -10.000000
25%           3.000000
50%          96.000000
75%         268.000000
max        3677.000000
Name: availability 365, dtype: float64
```

```
airbnb['availability 365'] = np.where(airbnb['availability 365']<0, airbnb['availability 365']*-1, airbnb['availability 365'])
airbnb['availability 365'] = np.where(airbnb['availability 365']>365, 365, airbnb['availability 365'])
```

La metrica 'minimum nights' indica il numero minimo di giorni in cui una stanza di un B&B può essere affittata. Durante l'analisi dei dati nel nostro dataset, abbiamo osservato la presenza di alcuni valori negativi, valori nulli, o superiori a 365 associati a questa metrica. I valori superiori sono stati regolati a 365, rappresentando il limite massimo di giorni durante l'anno, quelli negativi sono stati convertiti in valori positivi, mentre i valori nulli, sono stati posti tutti a 1, dato che si parte da un valore minimo di 1

```
airbnb['minimum nights'].describe()
```

```
count    101172.000000
mean          8.103892
std          30.620932
min       -1223.000000
25%           2.000000
50%           3.000000
75%           5.000000
max        5645.000000
Name: minimum nights, dtype: float64
```

```
airbnb['minimum nights'] = np.where(airbnb['minimum nights']<0, airbnb['minimum nights']*-1, airbnb['minimum nights'])
airbnb['minimum nights'] = np.where(airbnb['minimum nights']>365, 365, airbnb['minimum nights'])
airbnb['minimum nights'] = airbnb['minimum nights'].fillna(1)
```

1.2.1 Gestione delle celle null per ogni colonna:

avendo le seguenti voci:

- *host_identity_verified*, ove all'interno delle colonne troveremo i valori: unconfirmed e confirmed, se il valore non è presente, avremo per certo *"unconfirmed";*

- *calculated host listings count*, tale voce rappresenta quanti immobili appartengono ad un determinato host, di conseguenza, sostituiamo i valori null col "*count"* di quante volte si presenta l'host id.

```
airbnb['host_identity_verified'] = airbnb['host_identity_verified'].fillna('unconfirmed')
airbnb['calculated host listings count'] = airbnb.groupby('host id')['calculated host listings count'].transform(lambda x: x.fillna(x.count()))
```

- *neighbourhood group* e *neighbourhood*, se una delle due celle è null, la riga viene eliminata, in quanto la moda non puo' essere utilizzata per sostituire il valore null, in quanto non potrebbe essere accurata;
- *lat e long*, i valori rappresentano latitudine e longitudine, dati che anche in questo caso, attraverso la media potremmo avere dei risultati non corretti, la miglior soluzione è eliminare le righe in cui sono null per evitare inaccuratezza;
- *instant_bookable*, anche in questo caso le righe in cui la cella è null sono state eliminate poiché non abbiamo la sicurezza che sia effettivamente prenotabile

- **Sostituisco i valori null di 'cancellation_policy' con la moda**

```
modal_value = airbnb['cancellation_policy'].mode()[0]
airbnb['cancellation_policy'].fillna(modal_value, inplace=True)
```

- **Sostituisco i valori null di *"Construction year"* con la media del quartiere corrispondente**

```
construction_year_mean = airbnb.groupby('neighbourhood group')['Construction year'].mean()
airbnb['Construction year'] = airbnb.apply(lambda row: construction_year_mean[row['neighbourhood group']]
                                if pd.isnull(row['Construction year'])
                                else row['Construction year'], axis=1)
airbnb['Construction year'] = airbnb['Construction year'].astype(int)
```

Breve Descrizione di Media e Mediana

Moda:

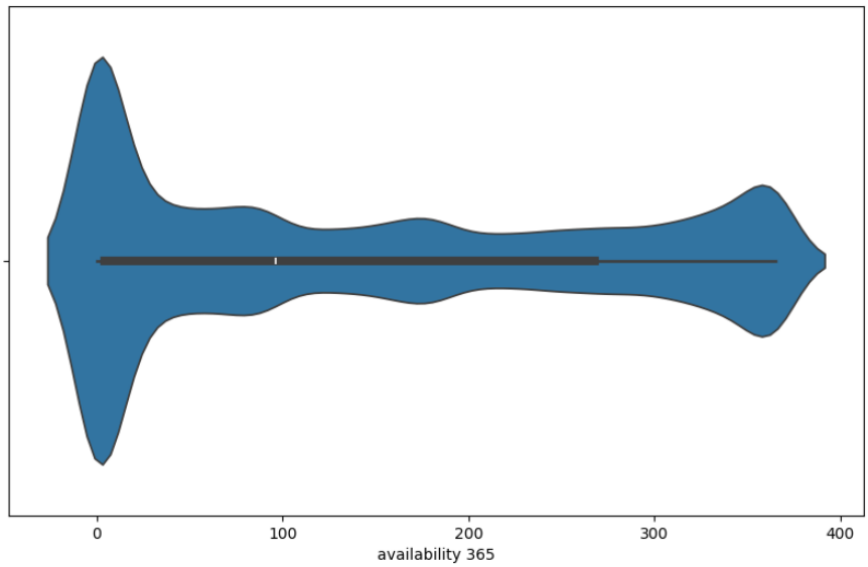
La moda è il valore che compare più frequentemente in un insieme di dati, inoltre, se un insieme di dati non ha alcun valore che si ripete, si dice che è senza moda o che ha una moda nulla.
Un insieme di dati può anche avere più di una moda se ci sono due o più valori che compaiono con la stessa massima frequenza.

Mediana:

La mediana è il valore centrale in un insieme di dati ordinati in modo crescente o decrescente, essa è meno influenzata da valori estremi rispetto alla media aritmetica, rendendola spesso una misura di tendenza centrale più robusta in presenza di dati anomali.

- **Verifico la distribuzione di "*number of reviews*"**

```
]: plt.figure(figsize=(10, 6))  # Imposta le dimensioni della figura
   sns.violinplot(x=airbnb['availability 365'])
   plt.show()
```



-*Descrizione del grafico*

Il grafico a violino è stato utilizzato in questo caso per visualizzare la distribuzione e la densità dei dati relativi alla disponibilità di 365 giorni.

-Dati rilevati a seguito del grafico:

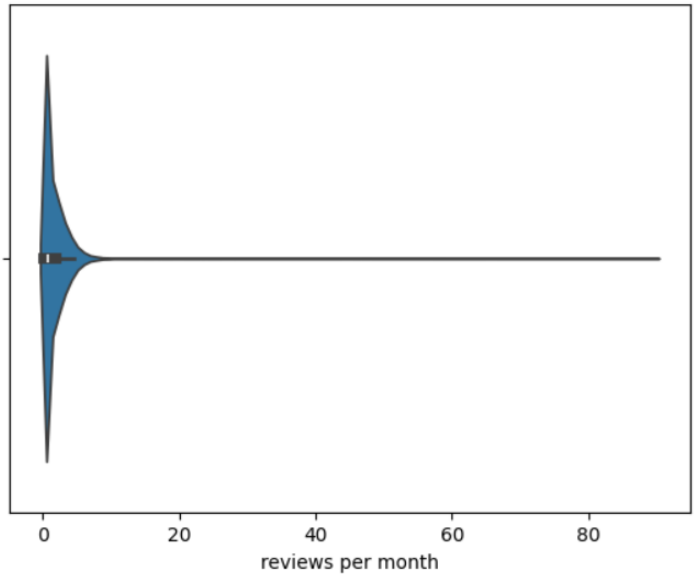
Attraverso il grafico notiamo che ci sia una maggiore densità di dati intorno a 0 e 365 nella disponibilità, indicando che molte proprietà sono disponibili per essere affittate per la maggior parte o per nessuna parte dell'anno.

- caratteristiche del grafico:

Il corpo del violino mostra la distribuzione dei dati. La larghezza del violino in un punto specifico rappresenta la densità della distribuzione in quella regione.
Se il violino è più largo in una certa area, significa che ci sono più dati concentrati in quella parte della distribuzione.
Possono essere inclusi punti o linee interni per indicare i valori individuali o la media, fornendo ulteriori dettagli sulla distribuzione.

- **Il seguente grafico rappresenta la distribuzione delle recensioni mensili per B&B**

```
sns.violinplot(x=airbnb['reviews per month'])
plt.show()
```

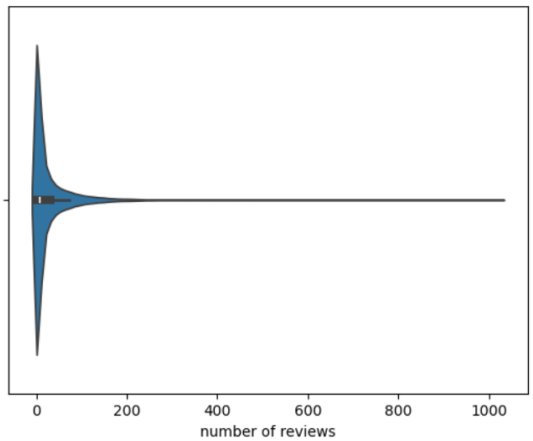


-Dati rilevati a seguito del grafico:

dal grafico, abbiamo rilevato una concentrazione di recensioni mensili in corrispondenza di valori bassi, questo indica appunto che avremo molte proprietà hanno un numero limitato di recensioni mensilmente, e poche proprietà avranno un alto numero alto di recensioni, quest'ultime sono indicate come proprietà eccezionali.

- **Nel grafico successivo, invece vedremo la distribuzione del numero di recensioni**

```
sns.violinplot(x=airbnb['number of reviews'])
plt.show()
```

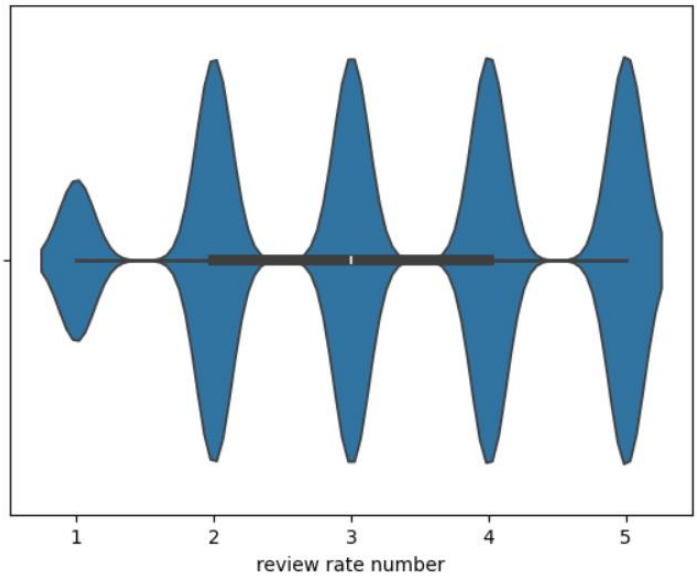


-Dati rilevati a seguito del grafico:

dal grafico, abbiamo rilevato una concentrazione di recensioni in corrispondenza di valori bassi, questo indica appunto che avremo molte proprietà aventi un numero limitato di recensioni, e poche proprietà avranno un alto numero di recensioni.

- **Il grafico a violino in questo caso rappresenta la distribuzione dei dati relativi ai numeri di valutazione delle recensioni**

```
sns.violinplot(x=airbnb['review rate number'])
plt.show()
```



-Dati rilevati a seguito del grafico:

dal grafico, abbiamo rilevato una distribuzione delle recensioni per ciascun livello di valutazione:
Ogni violino rappresenta un diverso livello di valutazione (da 1 a 5). La linea nera orizzontale all'interno di ciascun violino rappresenta la mediana delle recensioni per quel livello di valutazione, in questo caso ci dà un'idea del valore centrale delle recensioni.

- a seguito della visione dei grafici precedenti, abbiamo potuto osservare, che per i dati:
  number of reviews, *reviews per month*, *availability 365*, *review rate number*,
  è più conveniente effettuare la media, anziché la mediana;

```
airbnb['number of reviews'].fillna(airbnb['number of reviews'].median(), inplace=True)
airbnb['reviews per month'].fillna(airbnb['reviews per month'].median(), inplace=True)
airbnb['availability 365'].fillna(airbnb['availability 365'].median(), inplace=True)
airbnb['review rate number'].fillna(airbnb['review rate number'].mean(), inplace=True)
```

- **Per evitare inconsistenza, se la cella number of reviews è 0, pongo anche reviews per month a 0**

```
airbnb.loc[airbnb['number of reviews'] == 0, 'reviews per month'] = 0
```

- **L'unico elemento che rimane da verificare è *'last review'***

```
]: airbnb['last review'].describe()
```

```
]: count                           85662
   mean     2019-06-10 16:57:58.714014976
   min                2012-07-11 00:00:00
   25%                2018-10-27 00:00:00
   50%                2019-06-13 00:00:00
   75%                2019-07-05 00:00:00
   max                2058-06-16 00:00:00
   Name: last review, dtype: object
```

- **Poiché ci sono delle date errate, prima pongo le date posteriori al giorno attuale, poi sostituisco le celle vuote con la media**

```
: from datetime import datetime
  today = datetime.now().date()
  airbnb['last review'] = airbnb['last review'].apply(lambda x: today if pd.notna(x) and x.date() > today else x)
  airbnb['last review'] = pd.to_datetime(airbnb['last review'])
  mean_date = airbnb['last review'].mean().date()
  airbnb['last review'].fillna(mean_date,inplace=True)
  airbnb['last review'] = pd.to_datetime(airbnb['last review'])
```

```
]: airbnb.isnull().sum()
```

```
]: id                                0
   host id                          0
   host_identity_verified           0
   neighbourhood group              0
   neighbourhood                    0
   lat                              0
   long                             0
   instant_bookable                 0
   cancellation_policy              0
   room type                        0
   Construction year                0
   price                            0
   service fee                      0
   minimum nights                   0
   number of reviews                0
   last review                      0
   reviews per month                0
   review rate number               0
   calculated host listings count   0
   availability 365                 0
   dtype: int64
```

1.3 Individuazione di eventuali correlazioni

- **Voglio vedere i quartieri con più airbnb**

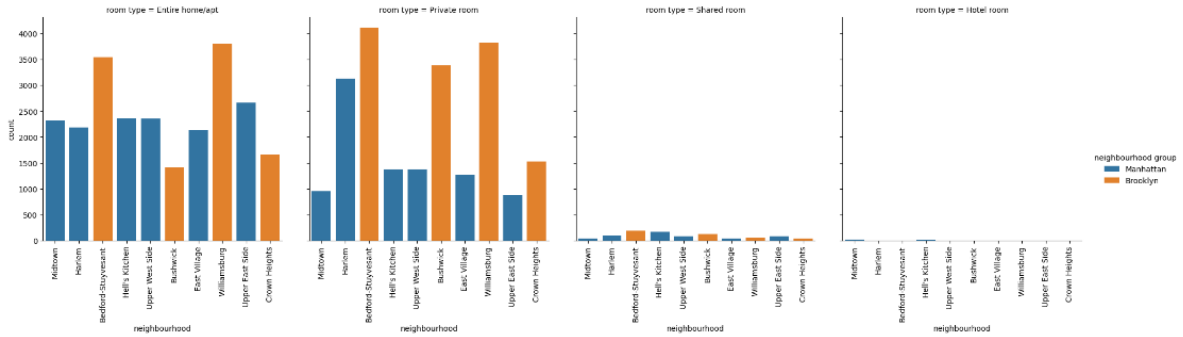
```
airbnb.neighbourhood.value_counts().head(10)
```

```
neighbourhood
Bedford-Stuyvesant    7846
Williamsburg          7682
Harlem                5407
Bushwick              4921
Hell's Kitchen        3935
Upper West Side       3824
Upper East Side       3634
East Village          3445
Midtown               3337
Crown Heights         3231
Name: count, dtype: int64
```

- **Per ogni neighbourhood group, verifico i tipi di stanza in ogni quartiere**

```
sub_7=airbnb.loc[airbnb['neighbourhood'].isin(['Bedford-Stuyvesant','Williamsburg','Harlem','Bushwick',
                'Hell\'s Kitchen','Upper West Side','Upper East Side','East Village','Crown Heights','Midtown'])]

viz_3=sns.catplot(x='neighbourhood', hue='neighbourhood group', col='room type', data=sub_7, kind='count')
viz_3.set_xticklabels(rotation=90)
```

```
<seaborn.axisgrid.FacetGrid at 0x218051c8ed0>
```

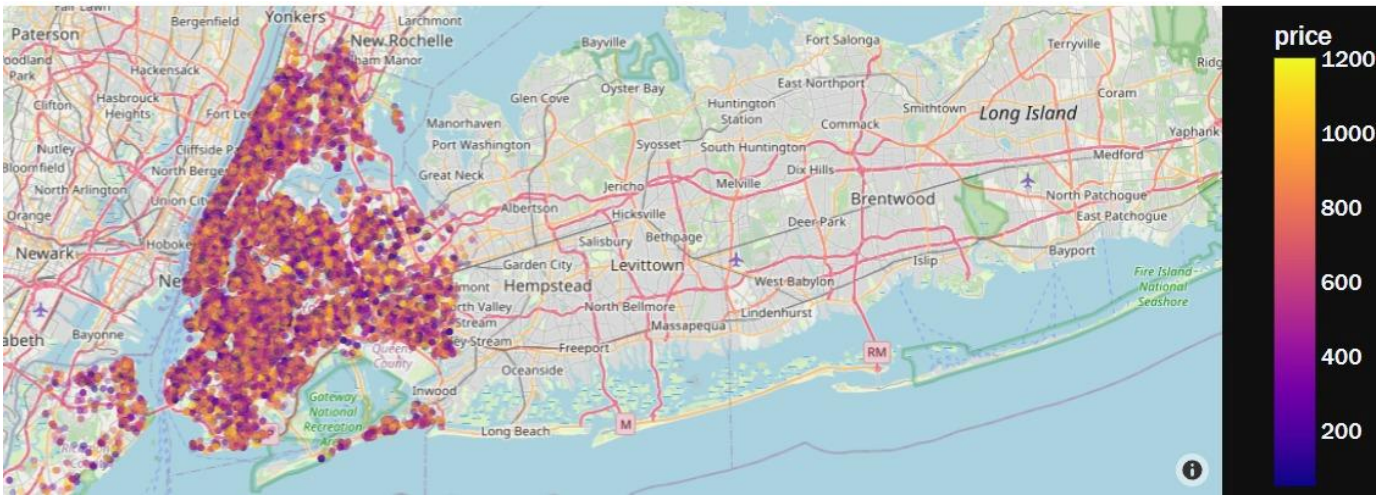


-Dati rilevati a seguito del grafico:

Da questo grafico emerge che c'è una quantità significativa in alcune zone come Midtown e Hell's Kitchen a Manhattan, e Williamsburg a Brooklyn, per quanto riguarda le stanze intere **(Entire home/apt),** le stanze private **(Private room)**, sono anch'esse abbondanti, ma comunque meno numerose rispetto alle stanze intere.
Possiamo invece notare che la categoria stanze condivise **(Shared room)**, ha meno opzioni disponibili in tutte le zone, inoltre come valori quasi nulli, abbiamo invece la categoria di stanze d'albergo **(Hotel room)**.

Rappresentazione delle posizioni geografiche di alloggi dei vari alloggi, includendo il punto sulla mappa, latitudine, longitudine, gruppo di quartiere, e prezzo:

```
fig = px.scatter_mapbox(airbnb,lat="lat",
            lon="long",
            opacity = 0.3,
            hover_name="neighbourhood group",
            hover_data=["neighbourhood group", "price"],
            color="price",
            color_discrete_sequence=px.colors.sequential.PuBuGn,
            template = "plotly_dark",
            zoom=10
            )
fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0},font = dict(size=17,family="Franklin Gothic"))
fig.show()
```

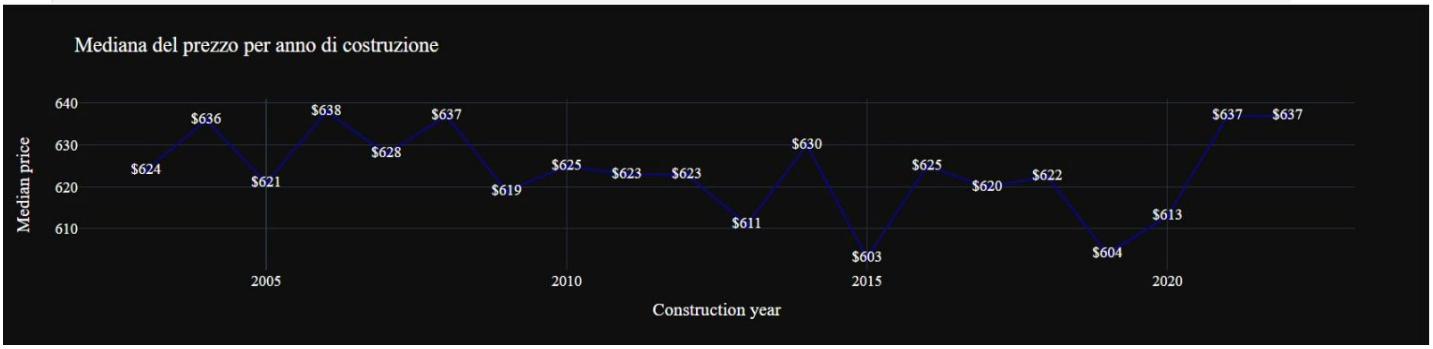


-Dati rilevati a seguito del grafico:

In questa mappa, i valori rappresentano i prezzi delle proprietà in diverse aree di New York.
La scala di colori a destra indica il range di prezzo da 200 a 1200, con colori differenti che corrispondono a diverse fasce di prezzo, all'interno della mappa vengono indicati anche nomi di luoghi e strade, rendendo facile identificare posizioni specifiche.
Si può dunque notare che la distribuzione dei prezzi è uniforme.

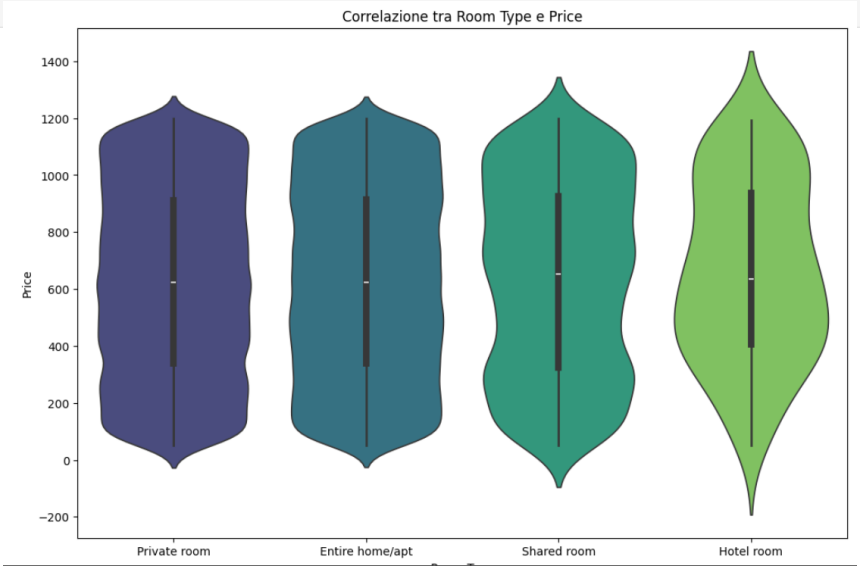
- **Vedo la relazione tra il prezzo e l'anno di costruzione**

```
price_per_year = airbnb.groupby('Construction year')['price'].median()

fig = px.line(price_per_year,
              x=price_per_year.index,
              y=price_per_year.values,
              labels={'x': 'Construction year', 'y': 'Median price'},
              text=['$' + str(int(i)) for i in price_per_year.values],
              title='Mediana del prezzo per anno di costruzione',
              color_discrete_sequence=px.colors.sequential.Plasma,  # Cambiato il colore
              template='plotly_dark'
)

fig.update_layout(font=dict(size=16, color='white', family='Avenir'))

fig.show()
```



Cerco una correlazione tra *'room type'* e *'price'*

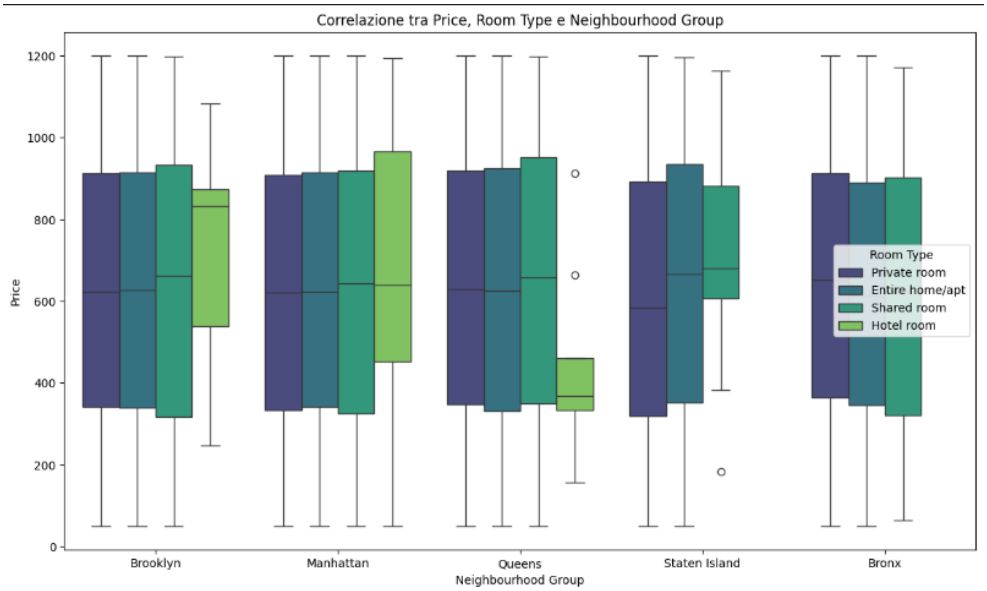
```
plt.figure(figsize=(12, 8))
sns.violinplot(x='room type', y='price', data=airbnb, palette='viridis', hue='room type', legend=False)
plt.title('Correlazione tra Room Type e Price')
plt.xlabel('Room Type')
plt.ylabel('Price')
plt.show()
```



Anche in questo caso è possibile individuare che il prezzo non cambia significativamente in base al tipo di room type.

- **Osservo la correlazione tra *price* e *room type* in base al quartiere**

```
plt.figure(figsize=(14, 8))
sns.boxplot(x='neighbourhood group', y='price', hue='room type', data=airbnb, palette='viridis')
plt.title('Correlazione tra Price, Room Type e Neighbourhood Group')
plt.xlabel('Neighbourhood Group')
plt.ylabel('Price')
plt.legend(title='Room Type')
plt.show()
```



-Dati rilevati a seguito del grafico:

Il grafico evidenzia come i prezzi delle stanze a New York variano in base al quartiere e al tipo di stanza. Si osserva che i prezzi per "*entire room"* e "*private room*" mantengono una sostanziale uniformità attraverso i diversi quartieri, mentre emerge una significativa variazione nei costi delle "*hotel room"* e "*shared room"* in relazione alla località.

- **Mostro la correlazione tra price e le variabili mancanti**

```
fig, axs = plt.subplots(1, 5, figsize=(20, 5))

# Scatter plot per 'minimum_nights' e 'price'
sns.scatterplot(x='minimum nights', y='price', data=airbnb, hue='instant_bookable', ax=axs[0])
axs[0].set_title('Minimum Nights vs Price')

# Bar plot per 'host_identity_verified' e 'price'
sns.barplot(x='host_identity_verified', y='price', data=airbnb, ax=axs[1])
axs[1].set_title('Host Identity Verified vs Price')

# Bar plot per 'cancellation_policy' e 'price'
sns.barplot(x='cancellation_policy', y='price', data=airbnb, ax=axs[2])
axs[2].set_title('Cancellation Policy vs Price')

# Scatter plot per 'number_of_reviews' e 'price'
sns.scatterplot(x='number of reviews', y='price', data=airbnb, hue='instant_bookable', ax=axs[3])
axs[3].set_title('Number of Reviews vs Price')

# Scatter plot per 'availability_365' e 'price'
sns.scatterplot(x='availability 365', y='price', data=airbnb, hue='instant_bookable', ax=axs[4])
axs[4].set_title('Availability 365 vs Price')

plt.tight_layout()
plt.show()
```

-Dati rilevati a seguito del grafico:

dai grafici appena analizzati, abbiamo rilevato:

Il grafico *"Minimum Nights vs Price"* mostra come il prezzo varia in base al numero minimo di notti richiesto per la prenotazione, anche in questo caso è abbastanza uniforme. Nel grafico abbiamo delle variabili "true" e "false" (definite da pallini arancio e blu), queste variabili rappresentano le varie disponibilità immediate o non immediate dei vari AirB&B. Come possiamo notare, è molto omogenea la correlazione tra le disponibilità immediate e le disponibilità che hanno un bisogno di accettazione da parte del rispettivo host, prima di portare a termine la pratica di prenotazione.

Nel sottografo *"Host Identity Verified vs Price",* vediamo come l'host verificato influisce sul prezzo. La verifica dell'identità dell'host potrebbe aumentare la fiducia dei potenziali ospiti e giustificare un prezzo leggermente superiore. Nel grafico vediamo l'omogeneità determinata dalle colonne dagli host verificati e host non ancora verificati.

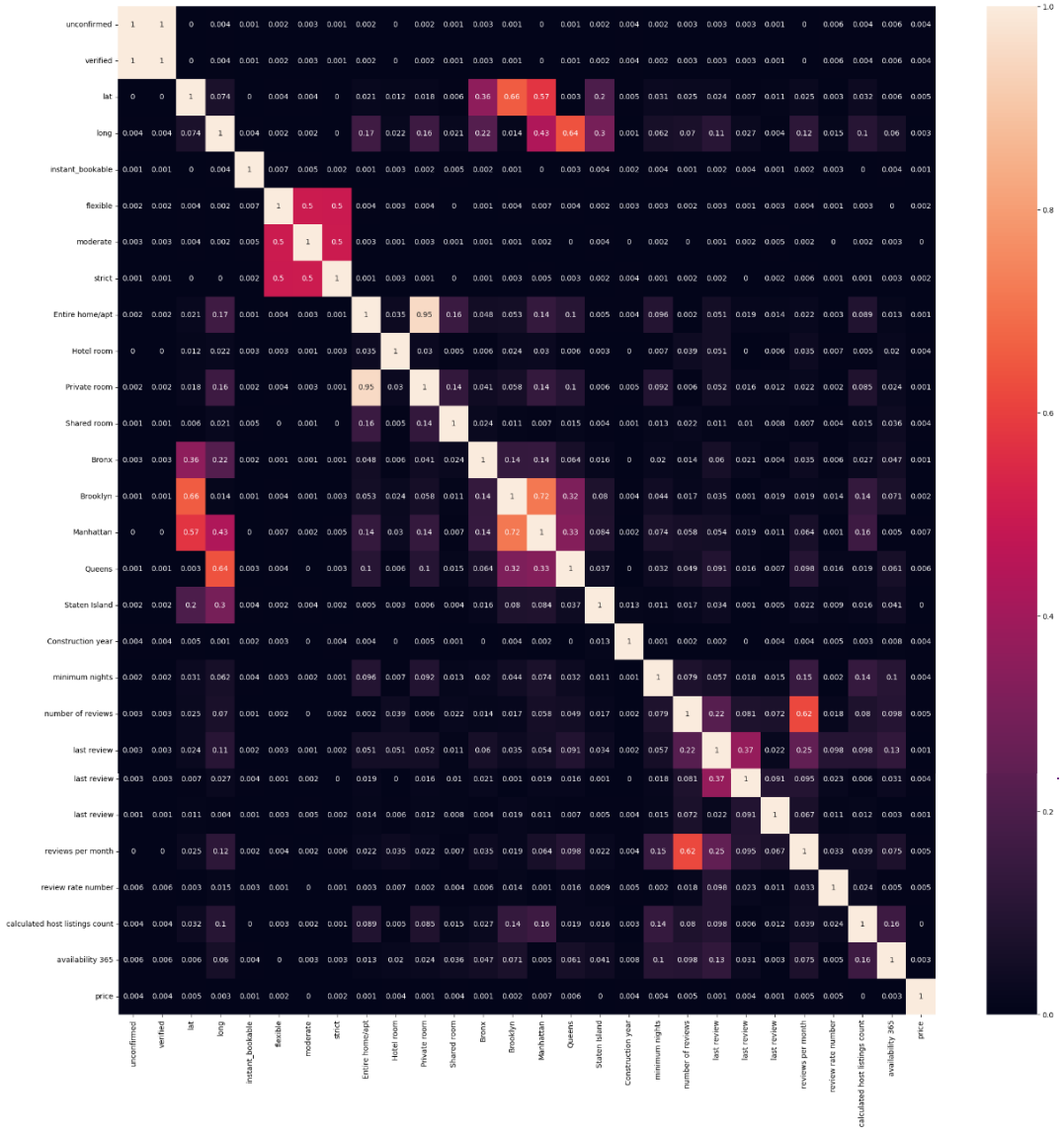
Nel sottografo *"Cancellation Policy vs Price",* osserviamo come diverse politiche di cancellazione impattano il prezzo. Questo è coerente con l'idea che una politica di cancellazione più flessibile possa rendere gli ospiti più disposti a prenotare, questo però porterebbe ad un maggiore costo.
Nel grafico, tuttavia, notiamo come non ci sia una tendenza del prezzo a cambiare in base alla politica di cancellazione.

Nel sottografo *"Number of Reviews vs Price",* vediamo come il prezzo è correlato al numero di recensioni ricevute. Anche in questo caso abbiamo delle variabili all'interno del grafico "true" e "false" (definite da pallini arancio e blu), notiamo la correlazione omogenea tra le varie disponibilità immediate, e non immediate che c'è nel prenotare un AirB&B molto recensito con un AirB&B molto conveniente.

Nel sottografo *"Availability 365 vs Price",* esaminiamo come la disponibilità dell'alloggio durante l'anno influisce sul prezzo. Il grafico presenta dei punti densamente impacchettati, i punti sono differenziati da colori e significati, in base alle varie disponibilità dei vari AirB&B a cui facciamo riferimento, il che indica una distribuzione omogenea delle disponibilità che sono bilanciate con le disponibilità che dovranno essere approvate successivamente da un host.

Utilizziamo una heatmap per individuare le colonne che hanno una maggiore correlazione con il prezzo.

```
top_neighbourhoods = airbnb['neighbourhood'].value_counts().nlargest(10).index

df = pd.concat([
    pd.get_dummies(airbnb.host_identity_verified).astype(int),
    airbnb.lat,
    airbnb.long,
    airbnb.instant_bookable.astype(int),
    pd.get_dummies(airbnb.cancellation_policy).astype(int),
    pd.get_dummies(airbnb['room type']).astype(int),
    pd.get_dummies(airbnb['neighbourhood group']).astype(int),
    #pd.get_dummies(airbnb['neighbourhood'][airbnb['neighbourhood'].isin(top_neighbourhoods)]).astype(int),
    airbnb['Construction year'],
    airbnb['minimum nights'],
    airbnb['number of reviews'],
    airbnb['last review'].dt.strftime('%Y').astype(int),
    airbnb['last review'].dt.strftime('%m').astype(int),
    airbnb['last review'].dt.strftime('%d').astype(int),
    airbnb['reviews per month'],
    airbnb['review rate number'],
    airbnb['calculated host listings count'],
    airbnb['availability 365'],
    airbnb.price
], axis = 1)

import matplotlib.pyplot as plt

sns.heatmap(df.corr().apply(lambda x: round(x.abs(), 3)), annot = True)

plt.rcParams["figure.figsize"] = (25,25)

plt.show()
```



In conclusione, possiamo affermare che nessuna feature ha una buona correlazione con la nostra feature target, il prezzo.

2 Apprendimento Supervisionato

Lo scopo di questo capitolo è quello di individuare il modello di predizione migliore per la predizione del prezzo di un alloggio.

2.1 *Train-Test split:*

L'insieme di esempi viene diviso in due parti: un training set e un test set.
Training Set: Questo è il set di dati su cui il modello viene effettivamente addestrato. Il modello usa questi dati per imparare e adattarsi in modo da poter fare previsioni o classificazioni accurate.

Test Set: Questo set di dati viene utilizzato per testare la precisione del modello. Non viene utilizzato durante la fase di addestramento. L'idea è di fornire al modello dati "nuovi" o "sconosciuti" e vedere quanto bene riesce a fare previsioni su questi dati.

La divisione tra training set e test set è un passo importante nel processo di apprendimento automatico perché ci permette di valutare quanto bene il nostro modello sarà in grado di performare.

```
from sklearn.model_selection import train_test_split
Y = df['price']
###v1 = df.drop('price', axis=1)
###v2  columns_to_drop = ['price', 'unconfirmed', 'verified', 'lat', 'long', 'last review', 'availability 365', 'flexible', 'moderate', 'strict']
columns_to_drop = ['price', 'unconfirmed', 'verified', 'lat', 'long', 'last review', 'availability 365', 'flexible', 'moderate', 'strict','instant_bookable','review rate
X = df.drop(columns=columns_to_drop, axis=1)
np.random.seed(0)
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.1, random_state = 100)
print(X.columns)
```

```
Index(['Entire home/apt', 'Hotel room', 'Private room', 'Shared room', 'Bronx',
       'Brooklyn', 'Manhattan', 'Queens', 'Staten Island',
       'number of reviews'],
      dtype='object')
```

Dopo aver eseguito numerose prove con diverse caratteristiche nel nostro codice, abbiamo scoperto che i migliori risultati sono stati ottenuti eliminando alcune colonne specifiche dal set di dati X. Questo è stato confermato anche dall'analisi della heatmap.

Ora, possiamo osservare un miglioramento nei risultati. Le colonne che presentavano correlazioni nulle o molto basse sono state eliminate

In confronto alle versioni precedenti che abbiamo commentato, l'eliminazione di queste colonne ha portato a un miglioramento significativo dei risultati.

2.2 Standardizzo i valori numerici

La standardizzazione coinvolge la trasformazione dei dati in modo che abbiano una media di zero e una deviazione standard di uno, assicura che tutte le variabili siano su una scala comune, permettendo al modello di apprendere in modo più efficace da ciascuna di esse.

In questo caso abbiamo utilizzato un **RobustScaler**, il quale: rimuove la mediana dai dati; scala i dati in base all'intervallo interquartile (IQR), che è l'intervallo tra il primo quartile e il terzo quartile.

La centratura e la scalatura avvengono indipendentemente su ogni caratteristica calcolando le statistiche rilevanti sui campioni nel set di addestramento, non è influenzato dagli outlier. Questo perché utilizza la mediana e l'IQR, sono statistiche robuste agli outlier, memorizza la mediana e l'IQR scalato per ogni caratteristica nel set di addestramento.

In molti casi, gli outlier possono influenzare negativamente la media e la varianza del campione. Pertanto, l'uso della mediana e dell'IQR spesso produce risultati migliori.

```
from sklearn.preprocessing import RobustScaler

rs = RobustScaler()
X_train = rs.fit_transform(X_train)
X_test = rs.transform(X_test)
```

2.3 *Applico la K-Fold-Cross-Validation*

La K-Fold Cross Validation è una tecnica di validazione incrociata.
Questa tecnica divide il nostro set di dati in 'K' parti o 'fold'. Poi, addestrando il modello su 'K-1' fold e testa le prestazioni del modello sul fold rimanente. Questo processo viene ripetuto 'K' volte, ogni volta con un fold diverso usato come set di test. Alla fine, ottieni 'K' diverse misure delle prestazioni del modello, che possono essere aggregate per ottenere una stima più robusta delle prestazioni del modello. Sulla base di questa suddivisione in K fold abbiamo applicato tutta una serie di metriche per poter valutare correttamente il modello

```
from sklearn.model_selection import KFold
### v1 kfold = KFold(n_splits=5, shuffle=True, random_state=42)
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
```

Com'è possibile notare dal codice abbiamo effettuato due prove, una con K-Fold a 5 e una a 10, e abbiamo ricevuto dei risultati migliori utilizzando 10 come numero di fold

2.4 *Funzione per testare i modelli:*

È stata scritta una funzione univoca per poter testare i modelli di predizione.

```python
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score, mean_squared_log_error, mean_absolute_error,mean_squared_error
import time

def tryModello(model, parameters, X_train, Y_train, X_test, Y_test, cv):
    mod = GridSearchCV(model, parameters, cv=cv, refit=True)
    mod.fit(X_train, Y_train)

    mod = mod.best_estimator_

    start_time = time.time()

    Y_pred_test = mod.predict(X_test)
    Y_pred_train = mod.predict(X_train)


    elapsed_time = time.time() - start_time
    print(f"\nTime elapsed: {round(elapsed_time, 3)} seconds")
    print("\nTrain metrics: ")
    print("Mean Squared log error train: ", round(mean_squared_log_error(Y_train, Y_pred_train), 4))
    print("Root Mean Squared Error train: ", round(np.sqrt(mean_squared_error(Y_train, Y_pred_train)), 4))
    print("Mean Absolute Error train: ", round(mean_absolute_error(Y_train, Y_pred_train), 4))
    print("R2 score pred train: ", round(r2_score(Y_train, Y_pred_train), 4))
    print("\nTest Metrics: ")
    print("Mean Squared log Error test: ", round(mean_squared_log_error(Y_test, Y_pred_test), 4))
    print("Root Mean Squared Error test: ", round(np.sqrt(mean_squared_error(Y_test, Y_pred_test)), 4))
    print("Mean Absolute Error test: ", round(mean_absolute_error(Y_test, Y_pred_test), 4))
    print("R2 score pred test: ", round(r2_score(Y_test, Y_pred_test), 4))

    # Differenza tra le metriche di allenamento e test per rilevare underfitting o overfitting
    train_test_difference = {
        "RMSE": np.sqrt(mean_squared_error(Y_train, Y_pred_train)) - np.sqrt(mean_squared_error(Y_test, Y_pred_test)),
        "MAE": mean_absolute_error(Y_train, Y_pred_train) - mean_absolute_error(Y_test, Y_pred_test),
        "R2": r2_score(Y_train, Y_pred_train) - r2_score(Y_test, Y_pred_test)
    }
    print("\nDifference between Train and Test metrics:")
    for metric, difference in train_test_difference.items():
        print(f"{metric}: {round(difference, 4)}")

    # Visualizzo il grafico
    fig = plt.figure(figsize=(8, 6))
    plt.scatter(Y_test, Y_pred_test)
    fig.suptitle('Y test vs Y predicted', fontsize=20)
    plt.xlabel('Y test', fontsize=18)
    plt.ylabel('Y predicted', fontsize=16)
    plt.show()
```

Questa funzione dati in input il modello, i parametri, i dati di addestramento e di test ed il numero di fold, esegue una gridsearch che implementa una ricerca esaustiva su specificati valori di parametri per un modello. L'obiettivo è trovare la combinazione di parametri che produce il miglior modello e che minimizzi l'errore.

Il codice esegue poi un'analisi comparativa tra le metriche di allenamento e di test. Questa analisi è realizzata calcolando la differenza tra le due, un processo noto come "train test difference". In particolare, le metriche di test vengono sottratte da quelle di allenamento. L'analisi di queste differenze ci permette di identificare eventuali problemi di underfitting o overfitting nel modello.

Se il modello è in **underfitting**, significa che non è riuscito a catturare adeguatamente i pattern nei dati di allenamento, risultando in prestazioni scarse sia sui dati di allenamento che di test.
Se il modello è in **overfitting**, significa che ha "imparato a memoria" i dati di allenamento, inclusi i loro rumori e anomalie. Di conseguenza, avrà ottime prestazioni sui dati di allenamento, ma prestazioni scarse sui dati di test, che non ha mai visto prima.

2.5 *Ricerca del modello di predizione più adatto.*

2.5.1 *Predittore randomico*

Un "predittore randomico" è un termine che fa riferimento ad un modello che non segue un criterio fisso o deterministico, ma genera previsioni che possono variare ogni volta che viene eseguito.

Utilizziamo un predittore randomico come baseline per i risultati dei modelli successivi, perché fornisce un punto di riferimento su cui si possono confrontare le prestazioni dei modelli più complessi. Se un modello non riesce a superare le prestazioni di un predittore randomico, allora probabilmente c'è qualcosa che non va nel modello o nei dati.

```python
# Genera n valori casuali con distribuzione normale nel range (50-1200)
mean_value = (1200 + 50) / 2
std_dev = (1200 - 50) / (2 * 3.29)  # Utilizzando 3.29 come fattore di scala
n = len(df)
y_rand = np.random.normal(loc=mean_value, scale=std_dev, size=n)

y_true = df['price']
rmse_random = np.sqrt(mean_squared_error(y_true, y_rand))
r2_random = r2_score(y_true, y_rand)

y_pred_random = np.random.normal(loc=mean_value, scale=std_dev, size=n)
rmse_pred_random = np.sqrt(mean_squared_error(y_true, y_pred_random))
r2_pred_random = r2_score(y_true, y_pred_random)

print(f"RMSE tra 'price' e valori casuali generati: {rmse_random}")
print(f"R2 tra 'price' e valori casuali generati: {r2_random}")

print(f"\nRMSE tra 'price' e predittore randomico: {rmse_pred_random}")
print(f"R2 tra 'price' e predittore randomico: {r2_pred_random}")
```

```
RMSE tra 'price' e valori casuali generati: 374.4893440380752
R2 tra 'price' e valori casuali generati: -0.2747834183804203

RMSE tra 'price' e predittore randomico: 375.12022759447046
R2 tra 'price' e predittore randomico: -0.279082164612325
```

2.5.2 *Regressione lineare*

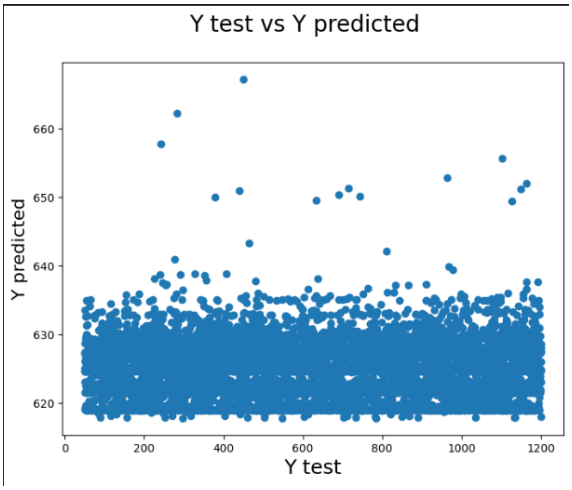
```python
from sklearn.linear_model import LinearRegression
parameters = {'fit_intercept': [True, False], 'copy_X': [True, False]}
tryModello(LinearRegression(), parameters, X_train, Y_train, X_test, Y_test, cv=kfold)
```

```
Time elapsed: 0.005 seconds

Train metrics:
Mean Squared log error train:  0.5817
Root Mean Squared Error train:  331.7164
Mean Absolute Error train:  286.8617
R2 score pred train:  0.0001

Test Metrics:
Mean Squared log Error test:  0.5618
Root Mean Squared Error test:  331.2133
Mean Absolute Error test:  286.7976
R2 score pred test:  -0.0002

Difference between Train and Test metrics:
RMSE: 0.5031
MAE: 0.0641
R2: 0.0004
```

Abbiamo i seguenti *parametri*:

fit_intercept: Questo parametro determina se si desidera calcolare l'intercetta per il modello. Se impostato su True, il modello calcolerà l'intercetta. Se impostato su False, il modello non calcolerà l'intercetta e assumerà che i tuoi dati siano già centrati.

copy_X: Questo parametro determina se i dati devono essere copiati prima di essere sovrascritti. Se impostato su True, i dati verranno copiati prima dell'elaborazione. Se impostato su False, i dati originali potrebbero essere sovrascritti durante l'elaborazione.



Possiamo notare che le prestazioni sono molto scarse.

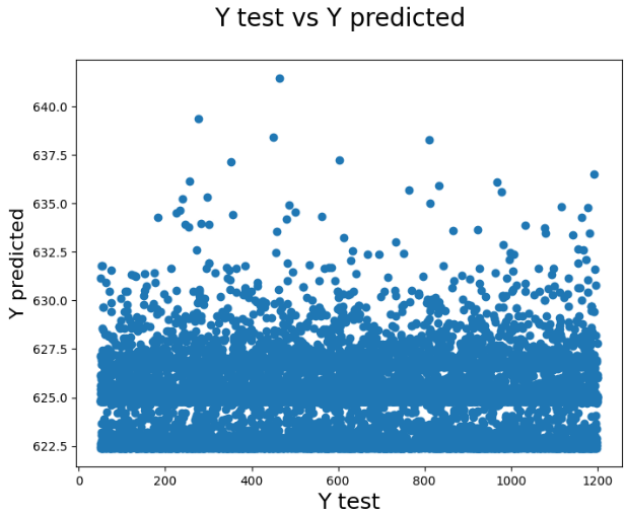
2.5.3 *Testo il modello Lasso*

```
from sklearn.linear_model import Lasso
parameters = {
    'alpha': [0.1, 0.5, 1.0, 2.0, 5.0],
    'fit_intercept': [True, False],
    'max_iter': [1000, 2000, 5000]
}

tryModello(Lasso(), parameters, X_train, Y_train, X_test, Y_test, cv=kfold)
```

```
Time elapsed: 0.003 seconds

Train metrics:
Mean Squared log error train:  0.5817
Root Mean Squared Error train:  331.7206
Mean Absolute Error train:  286.8677
R2 score pred train:  0.0001

Test Metrics:
Mean Squared log Error test:  0.5618
Root Mean Squared Error test:  331.2175
Mean Absolute Error test:  286.8034
R2 score pred test:  -0.0003

Difference between Train and Test metrics:
RMSE: 0.5031
MAE: 0.0643
R2: 0.0004
```

Abbiamo i seguenti parametri:
alpha: Questo è il parametro di regolarizzazione. I valori elencati (0.1, 0.5, 1.0, 2.0, 5.0) rappresentano diversi livelli di regolarizzazione.

max_iter: Questo è il numero massimo di iterazioni per il solutore. Se il modello non converge (cioè, non trova una soluzione) entro questo numero di iterazioni, si fermerà e restituirà ciò che ha finora.



Anche in questo caso i risultati non sono buoni

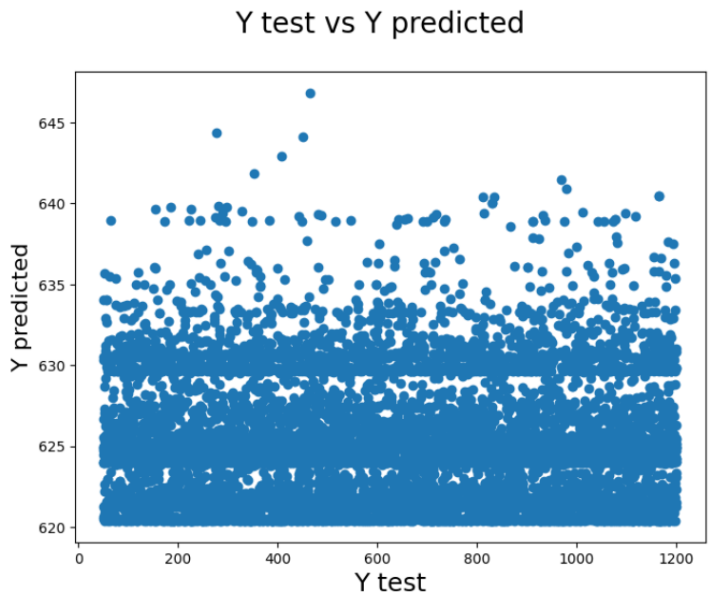
2.5.4 Testo il modello Ridge

```
from sklearn.linear_model import Ridge
parameters = {
    'alpha': [0.1, 0.5, 1.0, 2.0, 5.0],
    'fit_intercept': [True, False],
    'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga']
}

tryModello(Ridge(), parameters, X_train, Y_train, X_test, Y_test, cv=kfold)
```

```
Time elapsed: 0.004 seconds

Train metrics:
Mean Squared log error train:  0.5817
Root Mean Squared Error train:  331.7137
Mean Absolute Error train:  286.8572
R2 score pred train:  0.0001

Test Metrics:
Mean Squared log Error test:  0.5618
Root Mean Squared Error test:  331.2319
Mean Absolute Error test:  286.8033
R2 score pred test:  -0.0004

Difference between Train and Test metrics:
RMSE: 0.4818
MAE: 0.054
R2: 0.0005
```

Abbiamo i seguenti parametri:
alpha: Questo è un parametro di regolarizzazione che aiuta a prevenire l'overfitting. Valori più alti di alpha aumentano la regolarizzazione e rendono il modello più semplice. I valori nell'array [0.1, 0.5, 1.0, 2.0, 5.0] rappresentano diverse intensità di regolarizzazione.

solver: Questo parametro specifica il metodo da utilizzare per calcolare i pesi del modello Ridge Regression. 'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag' e 'saga' sono diversi algoritmi che possono essere utilizzati per risolvere problemi di regressione.



Anche in questo caso i risultati non sono ottimali

2.5.5 Testo la regressione con Random Forest v1

La **Random Forest** è un algoritmo di machine learning che combina più alberi decisionali per fare previsioni. Questo processo di aggregazione rende il modello più robusto e preciso, riducendo l'overfitting.
Migliora le prestazioni combinando le previsioni di molti alberi decisionali, riducendo sia la varianza (overfitting) che il bias (underfitting). Inoltre, la sua capacità di gestire un gran numero di variabili e la sua robustezza agli outliers la rendono un modello potente e versatile.

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV

# Definizione della griglia dei parametri per la ricerca casuale
param_dist = {
    'n_estimators': [int(x) for x in np.linspace(start=10, stop=200, num=10)],
    'max_features': [1, 2, 4, 6, 8, 10],  # Imposta solo valori validi per max_features
    'max_depth': [int(x) for x in np.linspace(1, 20, num=10)],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True]  # Imposta bootstrap su True
}

# Esecuzione della ricerca casuale
random_search = RandomizedSearchCV(
    RandomForestRegressor(random_state=0, oob_score=True), param_distributions=param_dist, n_iter=10, cv=kfold, scoring='neg_mean_squared_error', random_state=0, n_jobs=-1
)
random_search.fit(X_train, Y_train)

# Uso della tua funzione tryModello con il modello ottimizzato
tryModello(random_search.best_estimator_, {}, X_train, Y_train, X_test, Y_test, cv=kfold)
```

```
Time elapsed: 1.024 seconds

Train metrics:
Mean Squared log error train:  0.5721
Root Mean Squared Error train:  327.8976
Mean Absolute Error train:  282.9871
R2 score pred train:  0.023

Test Metrics:
Mean Squared log Error test:  0.5595
Root Mean Squared Error test:  330.2699
Mean Absolute Error test:  285.385
R2 score pred test:  0.0054

Difference between Train and Test metrics:
RMSE: -2.3723
MAE: -2.398
R2: 0.0176
```

Abbiamo i seguenti parametri:
n_estimators: Questo parametro rappresenta il numero di alberi nella foresta. Nel codice, abbiamo indicato tramite un array di numeri, interi da 10 a 200, divisi in modo uniforme in 10 valori.

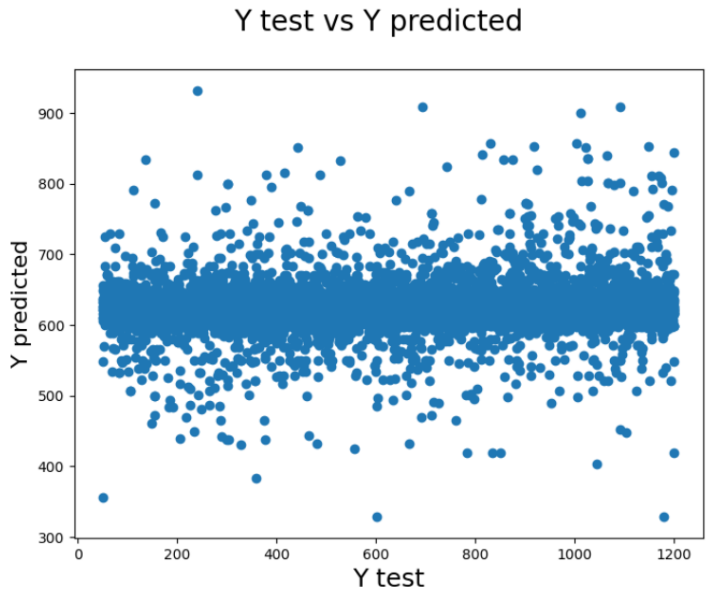
max_features: Indica il numero massimo di caratteristiche che ogni albero può utilizzare per la divisione. I valori possibili che abbiamo inseriti sono: 1, 2, 4, 6, 8 e 10.

max_depth: Rappresenta la profondità massima degli alberi nella foresta. Abbiamo creato un array di numeri interi da 1 a 20, divisi in modo uniforme in 10 valori.

min_samples_split: È il numero minimo di campioni necessari per dividere un nodo interno. I valori possibili sono: [2,5,10].

min_samples_leaf: Indica il numero minimo di campioni richiesto per essere un nodo foglia. I valori possibili sono [1,2,4].

bootstrap: Un valore booleano che indica se i campioni vengono utilizzati con o senza sostituzione per la costruzione degli alberi.



I risultati ottenuti dalla Random Forest sembrano per ora essere i migliori.

2.5.6 Testo la regressione con Random Forest v2

Si è tentato di migliorare ulteriormente la predizione della random forest testando un set di parametri diverso.

```
param_dist = {
    'n_estimators': [int(x) for x in np.linspace(start=50, stop=200, num=10)],
    'max_features': [1, 2, 4, 6, 8, 10, 'sqrt', 'log2'],
    'max_depth': [int(x) for x in np.linspace(5, 30, num=10)],
    'min_samples_split': [5, 10, 15],
    'min_samples_leaf': [2, 4, 6],
    'bootstrap': [True]
}

random_search = RandomizedSearchCV(
    RandomForestRegressor(random_state=0, oob_score=True, bootstrap=True), param_distributions=param_dist, n_iter=10, cv=kfold, random_state=0, n_jobs=-1
)
random_search.fit(X_train, Y_train)

tryModello(random_search.best_estimator_, {}, X_train, Y_train, X_test, Y_test, cv=kfold)
```

```
Time elapsed: 1.927 seconds

Train metrics:
Mean Squared log error train:  0.5711
Root Mean Squared Error train:  327.4778
Mean Absolute Error train:  282.5946
R2 score pred train:  0.0255

Test Metrics:
Mean Squared log Error test:  0.5596
Root Mean Squared Error test:  330.2774
Mean Absolute Error test:  285.242
R2 score pred test:  0.0054

Difference between Train and Test metrics:
RMSE: -2.7996
MAE: -2.6473
R2: 0.0201
```

Abbiamo i seguenti parametri:

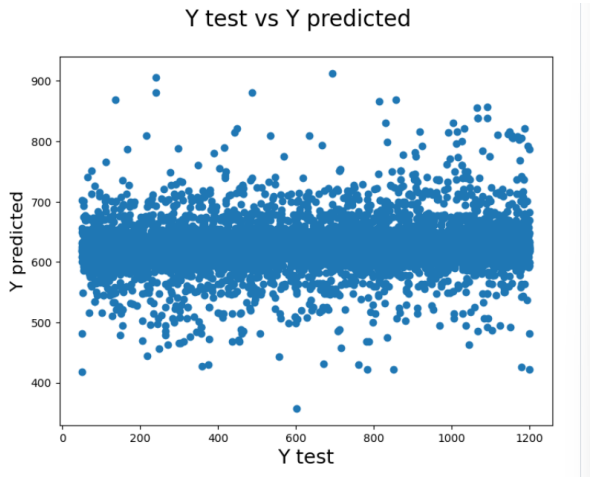
n_estimators: Questo parametro rappresenta il numero di alberi nella foresta. Nel codice, viene creato un array di numeri interi da 50 a 200, divisi in modo uniforme in 10 valori.

max_features: Indica il numero massimo di caratteristiche che ogni albero può utilizzare per la divisione. I valori possibili qui sono 1, 2, 4, 6, 8, 10, 'sqrt', 'log2'. 'sqrt' e 'log2' indicano rispettivamente la radice quadrata e il logaritmo in base 2 del numero totale di caratteristiche.

max_depth: Rappresenta la profondità massima degli alberi nella foresta. Abbiamo creato un array di numeri interi da 5 a 30, divisi in modo uniforme in 10 valori.

min_samples_split: È il numero minimo di campioni necessari per dividere un nodo interno. I valori possibili sono: [5,10,15].

min_samples_leaf: Indica il numero minimo di campioni richiesto per essere un nodo foglia. I valori possibili sono [2,4,6].
bootstrap: Un valore booleano che indica se i campioni vengono utilizzati con o senza sostituzione per la costruzione degli alberi.



2.5.6 Testo la regressione con Xgboost v1

XGBoost è un algoritmo di apprendimento automatico basato su gradient boosting. Utilizza il parallelismo e l'ottimizzazione dell'hardware, la regolarizzazione (L1 e L2) e gestisce i dati mancanti. Queste caratteristiche lo rendono potente e flessibile, spesso fornendo risultati migliori rispetto ad altri algoritmi.

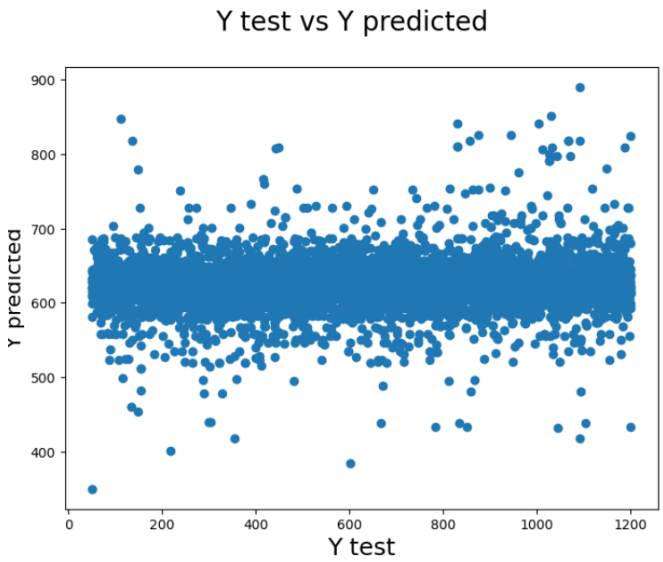
```
import xgboost as xgb

xgb_model = xgb.XGBRegressor()
parameters = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5]
}
tryModello(xgb_model, parameters, X_train, Y_train, X_test, Y_test, cv=kfold)
```

```
Time elapsed: 0.023 seconds

Train metrics:
Mean Squared log error train:  0.5763
Root Mean Squared Error train:  329.584
Mean Absolute Error train:  284.6489
R2 score pred train:  0.0129

Test Metrics:
Mean Squared log Error test:  0.5605
Root Mean Squared Error test:  330.7336
Mean Absolute Error test:  285.9077
R2 score pred test:  0.0026

Difference between Train and Test metrics:
RMSE: -1.1496
MAE: -1.2588
R2: 0.0103
```

Abbiamo i seguenti parametri:
n_estimators: Questo parametro rappresenta il numero di alberi che vengono costruiti nel modello. Nel nostro caso abbiamo individuato i possibili valori i quali sono [100, 200, 300].

learning_rate: Questo parametro, noto anche come tasso di apprendimento, riduce la contribuzione di ciascun albero aggiungendo un fattore di scala. Questo può aiutare a prevenire l'overfitting. Nel nostro codice, i valori possibili indicati sono [0.01, 0.1, 0.2].

max_depth: Questo parametro controlla la profondità massima di ciascun albero. Può essere utilizzato per controllare il sovradattamento poiché alberi più profondi possono catturare più dettagli dei dati. I valori possibili che abbiamo indicato sono [3, 4, 5].



2.5. 7 Testo la regressione con Xgboost v2

Si è tentato nuovamente l'xgboost con un set diverso di parametri per tentare di ottenere risultati migliori

```python
import xgboost as xgb

xgb_model = xgb.XGBRegressor()
parameters = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 1],
    'max_depth': [3, 4, 5]
}
tryModello(xgb_model, parameters, X_train, Y_train, X_test, Y_test, cv=kfold)
```

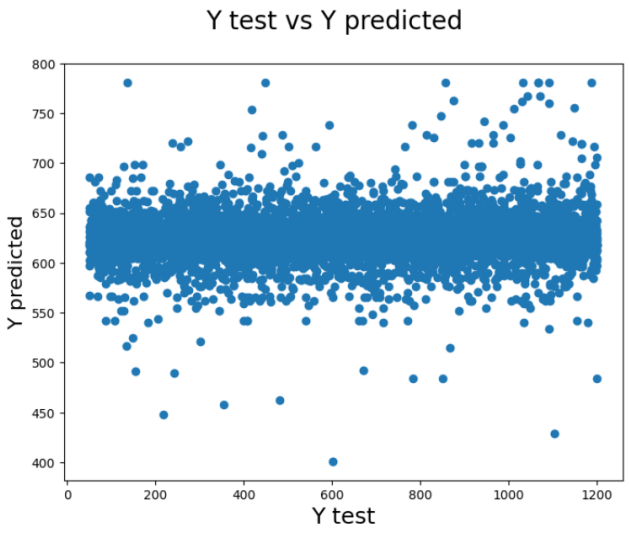
```
Time elapsed: 0.023 seconds

Train metrics:
Mean Squared log error train:  0.5781
Root Mean Squared Error train:  330.2525
Mean Absolute Error train:  285.4213
R2 score pred train:  0.0089

Test Metrics:
Mean Squared log Error test:  0.561
Root Mean Squared Error test:  330.8662
Mean Absolute Error test:  286.2581
R2 score pred test:  0.0018

Difference between Train and Test metrics:
RMSE: -0.6137
MAE: -0.8368
R2: 0.0071
```

Abbiamo i seguenti parametri:

n_estimators: Questo parametro determina il numero di alberi da costruire nel modello. Più alberi ci sono, più complesso è il modello, e più alta è la capacità di apprendimento. Tuttavia, un numero eccessivo di alberi può portare a un sovradattamento. Nel tuo caso, stai considerando 100, 200 o 300 alberi.

learning_rate (o tasso di apprendimento): Questo parametro controlla la velocità con cui il modello si adatta al problema. Un tasso di apprendimento più basso richiede più alberi per ottenere risultati simili. Nel tuo caso, stai considerando un tasso di apprendimento di 0.01, 0.1 o 1.

max_depth: Questo parametro controlla la profondità massima di ciascun albero. Puoi pensare a questo parametro come un controllo del grado di specializzazione di ciascun albero. Un valore più alto porterà a un albero più specializzato (più profondo), mentre un valore più basso porterà a un albero più generalizzato (meno profondo). Nel tuo caso, stai considerando una profondità massima di 3, 4 o 5.



2.5.7 Elasticnet

```
from sklearn.linear_model import ElasticNet
elasticnet_model = ElasticNet()
parameters = {
    'alpha': [0.1, 0.5, 1.0],
    'l1_ratio': [0.1, 0.5, 0.9],
    'max_iter': [1000, 2000, 3000]
}
tryModello(elasticnet_model, parameters, X_train, Y_train, X_test, Y_test, cv=kfold)
```

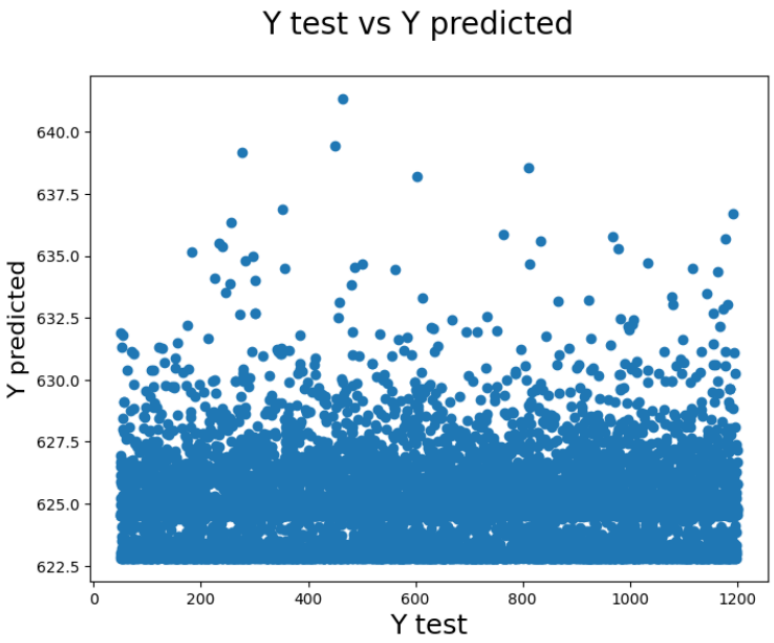
```
Time elapsed: 0.002 seconds

Train metrics:
Mean Squared log error train:  0.5817
Root Mean Squared Error train:  331.7224
Mean Absolute Error train:  286.8689
R2 score pred train:  0.0001

Test Metrics:
Mean Squared log Error test:  0.5618
Root Mean Squared Error test:  331.2139
Mean Absolute Error test:  286.7997
R2 score pred test:  -0.0003

Difference between Train and Test metrics:
RMSE: 0.5085
MAE: 0.0693
R2: 0.0003
```

Abbiamo i seguenti parametri:

alpha: Questo è un parametro di regolarizzazione che bilancia la quantità di riduzione che si desidera applicare durante l'apprendimento del modello. Un valore di alpha più alto aumenta la regolarizzazione e rende il modello più robusto ai rumori, ma può portare a un sottodimensionamento se è troppo alto. Un alpha di 0 equivale a una regressione lineare ordinaria1**.**

l1_ratio: Questo parametro determina il mix tra la penalità L1 e L2 nel modello Elastic Net. Un l1_ratio di 0 corrisponde a una penalità L2 (cioè Ridge), mentre un l1_ratio di 1 corrisponde a una penalità L1 (cioè Lasso). Per valori compresi tra 0 e 1, la penalità è una combinazione di L1 e L21.

max_iter: Questo parametro controlla il numero massimo di iterazioni per cui l'algoritmo deve eseguire la ricerca del modello ottimale. Un numero più alto di iterazioni permette una ricerca più approfondita, ma richiede più tempo



Anche in questo caso i risultati non sono buoni

2.5.8 Testo con regressione polinomiale di grado 2

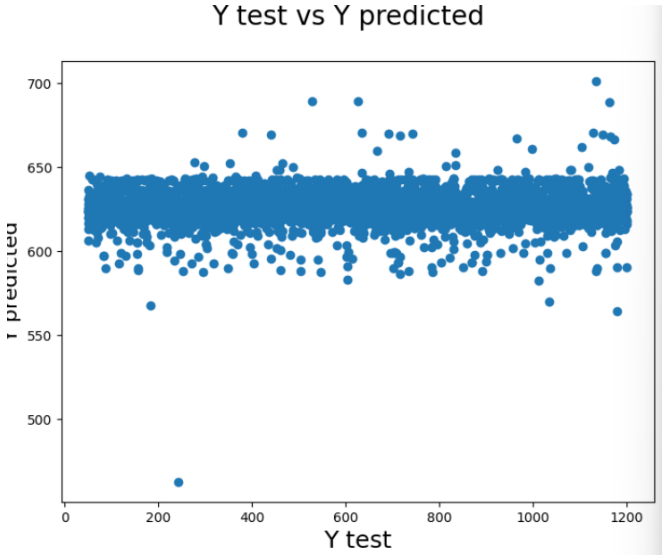
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
parameters = {'fit_intercept': [True, False], 'copy_X': [True, False]}
polyfeats = PolynomialFeatures(degree=2)
X_train_poly = polyfeats.fit_transform(X_train)
X_test_poly = polyfeats.transform(X_test)

tryModello(LinearRegression(), parameters, X_train_poly, Y_train, X_test_poly, Y_test, cv=kfold)
```

```
Time elapsed: 0.012 seconds

Train metrics:
Mean Squared log error train:  0.5816
Root Mean Squared Error train:  331.6899
Mean Absolute Error train:  286.8197
R2 score pred train:  0.0003

Test Metrics:
Mean Squared log Error test:  0.5616
Root Mean Squared Error test:  331.1722
Mean Absolute Error test:  286.765
R2 score pred test:  0.0

Difference between Train and Test metrics:
RMSE: 0.5177
MAE: 0.0547
R2: 0.0003
```

Abbiamo i seguenti parametri:

fit_intercept: Questo parametro determina se calcolare l'intercetta per questo modello. Se impostato su False, non verrà utilizzata alcuna intercetta nei calcoli (cioè, si presume che i dati siano centrati)2.

copy_X: Se True, X sarà copiato; altrimenti, potrebbe essere sovrascritto



Conclusione:

Il modello che ha mostrato le migliori prestazioni è il Random Forest v2. Questo modello ha la differenza più piccola tra le metriche di addestramento e di test, indicando che è il modello più stabile e meno soggetto a sovradattamento. Inoltre, ha il valore più alto di R2 sul set di test.

Tuttavia, nessuno dei modelli ha mostrato un punteggio R2 elevato e quindi è molto improbabile ottenere predizioni accurate.

3 Knowledge Base

3.1 Introduzione

Una knowledge base, o KB, è un deposito di informazioni che fornisce risposte all'utente attraverso la conoscenza che contiene. Essa rappresenta fatti del mondo, ragiona su tali fatti e applica regole e altre forme di logica per dedurre nuovi fatti o mettere in luce contraddizioni. Pertanto, la KB può essere definita come un insieme di assiomi, ovvero proposizioni che si presume siano vere, creando così un ambiente che facilita la raccolta, l'organizzazione e la diffusione della conoscenza. Nel contesto del nostro caso di studio, la KB viene impiegata per permettere all'utente di formulare domande relative all'ambito di interesse approfondito.

3.2 Gestione della KB

Per la gestione della knowledge base, abbiamo impiegato l'estensione Pyswip del linguaggio Python, che si basa sul linguaggio di programmazione logica Prolog. Sfruttando le funzionalità di Prolog, abbiamo iniziato a popolare la knowledge base con fatti, che vengono generati automaticamente prelevando i dati pertinenti direttamente dal dataset. In questo modo, se il dataset viene aggiornato con nuovi dati, la KB si aggiorna di conseguenza. Successivamente, abbiamo implementato delle regole, basandoci su come poter gestire i fatti presenti nella KB e su come potrebbe avvenire l'interazione da parte dell'utente.

3.3 Fatti

I fatti rappresentano gli assiomi sempre veri della knowledge base, e sono la base per il funzionamento delle regole.

1) `"neighbourhood(id, neighbourhood)"`
   Rappresenta la relazione tra un airbnb e il suo quartiere.

2) `"price(id, price)"`
   Rappresenta la relazione tra un airbnb e il suo prezzo.

3) `"minimum_nights(id, minimum nights)"`
   Rappresenta la relazione tra un airbnb e il numero minimo di notti richieste.

4) `"number_of_reviews(id, number of reviews)"`
   Rappresenta la relazione tra un airbnb e il numero di recensioni ricevute.

5) `"review_rate_number(id, review rate number)"`
   Rappresenta la relazione tra un airbnb e il numero di recensioni ricevute.

6) `"cancellation_policy(id, cancellation policy)"`
   Rappresenta la relazione tra un airbnb e la sua politica di cancellazione.

7) `"room_type(id, room type)"`
   Rappresenta la relazione tra un airbnb e il tipo di stanza offerta.

8) `"host_identity_verified(id, host identity verified)"`
   Rappresenta la relazione tra un airbnb e la verifica dell'identità dell'host.

Sono poi stati creati i due insiemi corrispondenti a tutti i quartieri considerati centrali di New York e tutti i quartieri considerati periferici.

```
quartieri_centrali = ["Kensington", "Midtown", "Harlem", "Murray Hill", "Hells Kitchen",
                      "Upper West Side", "Chinatown", "West Village", "Chelsea", "SoHo",
                      "East Village", "Lower East Side", "Kips Bay", "Greenwich Village",
                      "Little Italy", "Tribeca", "Financial District", "Flatiron District"]
```

```
quartieri_periferici = ["Clinton Hill", "East Harlem", "Bedford-Stuyvesant", "Upper East Side",
                        "South Slope", "Bushwick", "Flatbush", "Fort Greene",
                        "Prospect-Lefferts Gardens", "Long Island City", "Williamsburg",
                        "Greenpoint", "Park Slope", "Brooklyn Heights", "Gowanus",
                        "Washington Heights", "Flatlands", "Cobble Hill", "Flushing", "DUMBO"]
```

E sono stati utilizzati per rappresentare sotto forma di fatti i quartieri centrali e periferici.

`"posizione_centrale(quartiere)"`
Indica che il quartiere è in posizione centrale

`"quartieri_periferici(quartiere)"`
Indica che il quartiere è in posizione periferica

Si sono poi volute rappresentare le politiche di cancellazione esistenti.

`"cancellazione_stretta('strict')"`
Rappresenta il fatto che esiste una politica di cancellazione stretta, con l'argomento 'strict'.

`"cancellazione_media('moderate')"`
Rappresenta il fatto che esiste una politica di cancellazione moderata, con l'argomento 'moderate'.

`"cancellazione_flessibile('flexible')"`
Rappresenta il fatto che esiste una politica di cancellazione flessibile, con l'argomento 'flexible'.

3.4 Regole

Le regole costituiscono l'elemento centrale dell'interazione con l'utente. Esse fungono da domande che l'utente può porre alla knowledge base, senza necessità di conoscere la sintassi specifica. Le regole sfruttano i fatti presenti nella KB per fornire all'utente le informazioni da lui richieste.

Sono stati previsti due task principali che l’utente potrebbe richiedere alla base di conoscenza.

```
Scegliere quale funzione eseguire:
1) Trova gli airbnb date determinate caratteristiche
2) Dati i gusti dell'utente, trova l'airbnb col punteggio migliore
3) Esci
```

3.4.1 Trova gli airbnb date determinate caratteristiche

Il **primo task** consiste nell’ottenere tutti gli airbnb date alcune scelte dell’utente.
Per far questo sono stare create una serie di regole corrispondenti a tutte le combinazioni possibili.
Se ne presenta una per ogni tipo come esempio.

```
"prezzo_economico(ID) :- price(ID, Prezzo), Prezzo < 100."
```

La regola afferma che un Airbnb con ID ha un prezzo considerato economico se il suo prezzo è inferiore a 100.

```
"ottima_recensione(ID) :- review_rate_number(ID, Voto), Voto >=
4."
```

La regola afferma che un Airbnb con ID ha un'ottima recensione se il suo numero di recensioni è maggiore o uguale a 4.

```
"economico_e_ben_recensito(ID) :- prezzo_economico(ID),
ottima_recensione(ID)."
```

La regola afferma che un Airbnb con ID è economico e ben recensito se soddisfa contemporaneamente le condizioni di prezzo economico e ottima recensione.

```
"economico_ben_recensito_centrale(ID) :-
economico_e_ben_recensito(ID), neighbourhood(ID, Quartiere),
posizione_centrale(Quartiere)."
```

La regola afferma che un Airbnb con ID è economico, ben recensito e situato in una posizione centrale se soddisfa le condizioni di economicità e buona recensione, e si trova in un quartiere centrale.

```
"economico_ben_recensito_centrale_stretto(ID) :-
economico_ben_recensito_centrale(ID), cancellation_policy(ID,
Politica), cancellazione_stretta(Politica)."
```

La regola afferma che un Airbnb con ID è economico, ben recensito, situato in una posizione centrale e ha una politica di cancellazione stretta se soddisfa le condizioni di economicità, buona recensione, posizione centrale e politica di cancellazione stretta.

```
"economico_ben_recensito_centrale_stretto_notti(ID, Notti) :-
economico_ben_recensito_centrale_stretto(ID), minimum_nights(ID,
MinNotti), MinNotti =< Notti."
```

La regola afferma che un Airbnb con ID è economico, ben recensito, situato in una posizione centrale, ha una politica di cancellazione stretta e richiede un numero minimo di notti che non supera Notti se soddisfa le condizioni di economicità, buona recensione, posizione centrale, politica di cancellazione stretta e numero di notti.

Questa regola rappresenta la regola finale che a sua volta chiama le precedenti in base alle occorrenze, per poter rappresentare tutte le combinazioni vi sono 36 regole di questo tipo.

Esempio di esecuzione:

```
Inserisci la fascia di prezzo che preferisci per l'airbnb
1) Economico
2) Medio
3) Costoso
2
Vuoi che gli airbnb mostrati abbiano buone o cattive recensioni ?
1) Ben recensito
2) Mal recensito
1
Vuoi che l'airbnb sia in una posizione centrale o periferica ?
1) centrale
2) periferico
2
Quale tipo di politica di cancellazione deve avere l'airbnb ?
1) stretta
2) media
3) flessibile
1
Inserisci il numero di notti che vuoi soggiornare:
5
Gli id degli airbnb che rispettano i filtri sono:
1049386
1071478
1119528
```

3.4.1 Dati i gusti dell'utente, trova l'airbnb col punteggio migliore

Il **secondo task** consiste nel calcolare lo score di ogni airbnb in base a delle convenzioni decise a livello progettuale (ad esempio un prezzo economico fornisce più punti di un prezzo costoso) e in base alle preferenze dell'utente che avranno un impatto sullo score (se l'utente preferisce una stanza condivisa, gli id corrispondenti a stanze condivise avranno uno score maggiore).
Dopodichè viene effettuato un confronto tra tutti in modo da ottenere l'airbnb con lo score maggiore rispetto ai gusti dell'utente e le convenzioni.

1)

```
has_reviews(AirbnbId, ReviewScore) :-
    number_of_reviews(AirbnbId, NumberOfReviews),
    (
        (NumberOfReviews >= 0, NumberOfReviews =< 256)  -> ReviewScore = 5;
        (NumberOfReviews > 256, NumberOfReviews =< 512)  -> ReviewScore = 10;
        (NumberOfReviews > 512, NumberOfReviews =< 768)  -> ReviewScore = 15;
        (NumberOfReviews > 768, NumberOfReviews =< 1024) -> ReviewScore = 20
    ).
```

Rappresenta la regola che calcola uno 'Score' sulla base del numero di recensioni per un determinato Airbnb id.

2)

```
user_pref_score(AirbnbId, PrefRoomType, PrefCancellationPolicy, PrefPriceMax, BonusScore) :-
    room_type(AirbnbId, RoomType),
    cancellation_policy(AirbnbId, CancellationPolicy),
    price(AirbnbId, Price),
    (RoomType == PrefRoomType -> ScoreRoomType = 15 ; ScoreRoomType = 0),
    (CancellationPolicy == PrefCancellationPolicy -> ScoreCancellationPolicy = 8 ; ScoreCancellationPolicy = 0),
    (Price =< PrefPriceMax -> ScorePrice = 5 ; ScorePrice = 0),
    BonusScore is ScoreRoomType + ScoreCancellationPolicy + ScorePrice.
```

Rappresenta la regola che calcola uno 'Score Bonus' sulla base delle preferenze dell'utente per un determinato AirbnbId.

3)

```
price_index(AirbnbId, PriceIndex) :-
    price(AirbnbId, Price),
    (
        (Price >= 50, Price =< 100)  -> PriceIndex is 15;
        (Price > 100, Price =< 300)  -> PriceIndex is 10;
        (Price > 300, Price =< 500)  -> PriceIndex is 5;
        (Price > 500, Price =< 700)  -> PriceIndex is 3;
        (Price > 700, Price =< 900)  -> PriceIndex is 2;
        (Price > 900, Price =< 1200) -> PriceIndex is 1
    ).
```

Rappresenta la regola che assegna uno 'Score' sulla base del prezzo per un determinato AirbnbId.

4)

```
room_type_score(AirbnbId, RoomTypeScore) :-
    room_type(AirbnbId, RoomType),
    (RoomType = 'Entire home/apt' -> RoomTypeScore = 15;
     RoomType = 'Private room'    -> RoomTypeScore = 5;
     RoomType = 'Shared room'     -> RoomTypeScore = 0;
     RoomType = 'Hotel room'      -> RoomTypeScore = 10
    ).
```

Rappresenta la regola che assegna uno 'Score' sulla base del tipo di stanza per un determinato airbnb id.

5)

```
cancellation_policy_score(AirbnbId, CancellationPolicyScore) :-
    cancellation_policy(AirbnbId, CancellationPolicy),
    (CancellationPolicy = 'flexible' ->
        CancellationPolicyScore = 10;
        CancellationPolicyScore = 5
    ).
```

Rappresenta la regola che assegna uno 'Score' sulla base della politica di cancellazione per un determinato AirbnbId.

6)

```
host_identity_verified_score(AirbnbId, HostIdentityVerifiedScore) :-
    host_identity_verified(AirbnbId, HostIdentityVerified),
    (HostIdentityVerified = 'true' ->
        HostIdentityVerifiedScore = 5;
        HostIdentityVerifiedScore = 0
    ).
```

Rappresenta la regola che assegna uno 'Score' sulla base della verifica dell'identità dell'host per un determinato AirbnbId.

7)

```
overall_score(AirbnbId, OverallScore) :-
    price_index(AirbnbId, PriceIndex),
    has_reviews(AirbnbId, ReviewScore),
    room_type_score(AirbnbId, RoomTypeScore),
    cancellation_policy_score(AirbnbId, CancellationPolicyScore),
    host_identity_verified_score(AirbnbId, HostIdentityVerifiedScore),
    OverallScore is PriceIndex + ReviewScore + RoomTypeScore + CancellationPolicyScore + HostIdentityVerifiedScore.
```

Rappresenta la regola che calcola uno 'Score complessivo' per un determinato AirbnbId.

8)

```
final_score(AirbnbId, PrefRoomType, PrefCancellationPolicy, PrefPriceMax, FinalScore) :-
    overall_score(AirbnbId, OverallScore),
    user_pref_score(AirbnbId, PrefRoomType, PrefCancellationPolicy, PrefPriceMax, BonusScore),
    FinalScore is OverallScore + BonusScore.
```

Rappresenta la regola che calcola lo 'Score finale' considerando sia lo 'Score complessivo' che lo 'Score Bonus' per un determinato AirbnbId.

9)

```
highest_review_score([AirbnbId], PrefRoomType, PrefCancellationPolicy, PrefPriceMax, AirbnbId) :-
    final_score(AirbnbId, PrefRoomType, PrefCancellationPolicy, PrefPriceMax, _).
```

Questa regola è progettata per gestire il caso in cui hai una lista contenente un solo Airbnb Id. Essa afferma che il miglior punteggio di recensione tra gli Airbnb nella lista è l'Airbnb Id stesso, ottenuto calcolando lo 'Score finale' considerando le preferenze dell'utente.

10)

```
highest_review_score([AirbnbId1, AirbnbId2 | Rest], PrefRoomType, PrefCancellationPolicy, PrefPriceMax, HighestReviewScoreAirbnbId) :-
    final_score(AirbnbId1, PrefRoomType, PrefCancellationPolicy, PrefPriceMax, FinalScore1),
    final_score(AirbnbId2, PrefRoomType, PrefCancellationPolicy, PrefPriceMax, FinalScore2),
    (FinalScore1 >= FinalScore2 ->
        highest_review_score([AirbnbId1 | Rest], PrefRoomType, PrefCancellationPolicy, PrefPriceMax, HighestReviewScoreAirbnbId);
        highest_review_score([AirbnbId2 | Rest], PrefRoomType, PrefCancellationPolicy, PrefPriceMax, HighestReviewScoreAirbnbId)
    ).
```

Questa regola è progettata per gestire il caso in cui hai una lista contenente almeno due AirbnbId. Essa compara i punteggi finali di recensione tra i primi due AirbnbId nella lista (AirbnbId1 e AirbnbId2). Se il punteggio finale di AirbnbId1 è maggiore o uguale a quello di AirbnbId2, allora l'AirbnbId con il punteggio più alto è AirbnbId1; altrimenti, è AirbnbId2. Quindi, la regola continua a confrontare il vincitore tra AirbnbId1 e AirbnbId2 con gli altri AirbnbId nella lista (Rest), fino a quando non rimane solo un Airbnb Id con il punteggio più alto, che sarà assegnato a HighestReviewScoreAirbnbId.

Esempio di esecuzione:

```
Scegliere quale funzione eseguire:
1) Trova gli airbnb date determinate caratteristiche
2) Dati i gusti dell'utente, trova l'airbnb col punteggio migliore
3) Esci
2
Quale tipo di stanza preferisci?
1) Stanza privata
2) Intero appartamento/casa
3) Stanza condivisa
4) Stanza di hotel
2
Quale tipo di politica di cancellazione deve avere l'airbnb ?
1) stretta
2) moderata
3) flessibile
3
Inserisci il prezzo massimo che vorresti pagare (Tra 50 e 1200): 100
L'airbnb con lo score migliore è quello con id: 15901285
Price: 61.0
Neighbourhood: Jamaica
Minimum Nights: 1
Number of Reviews: 259.0
Review Rate: 2.0
Cancellation Policy: flexible
Room Type: Entire home/apt
Host Identity Verified: verified
```

4 Ontologia

4.1 Introduzione

L'ontologia, nel campo dell'informatica, è un sistema di rappresentazione formale che descrive la realtà e la conoscenza. È una struttura dati che permette di definire le entità (come oggetti, idee, ecc.) e le loro interazioni in uno specifico ambito di conoscenza.

Abbiamo scelto di sviluppare e implementare un'ontologia che potesse rappresentare il settore degli alloggi in affitto. Questo permette agli utenti, di visualizzare tutti gli alloggi disponibili sui vari provider (AirBnB, Booking…) e di visualizzare informazioni come Città, Prezzi, Pagamenti effettuati, Prenotazioni …

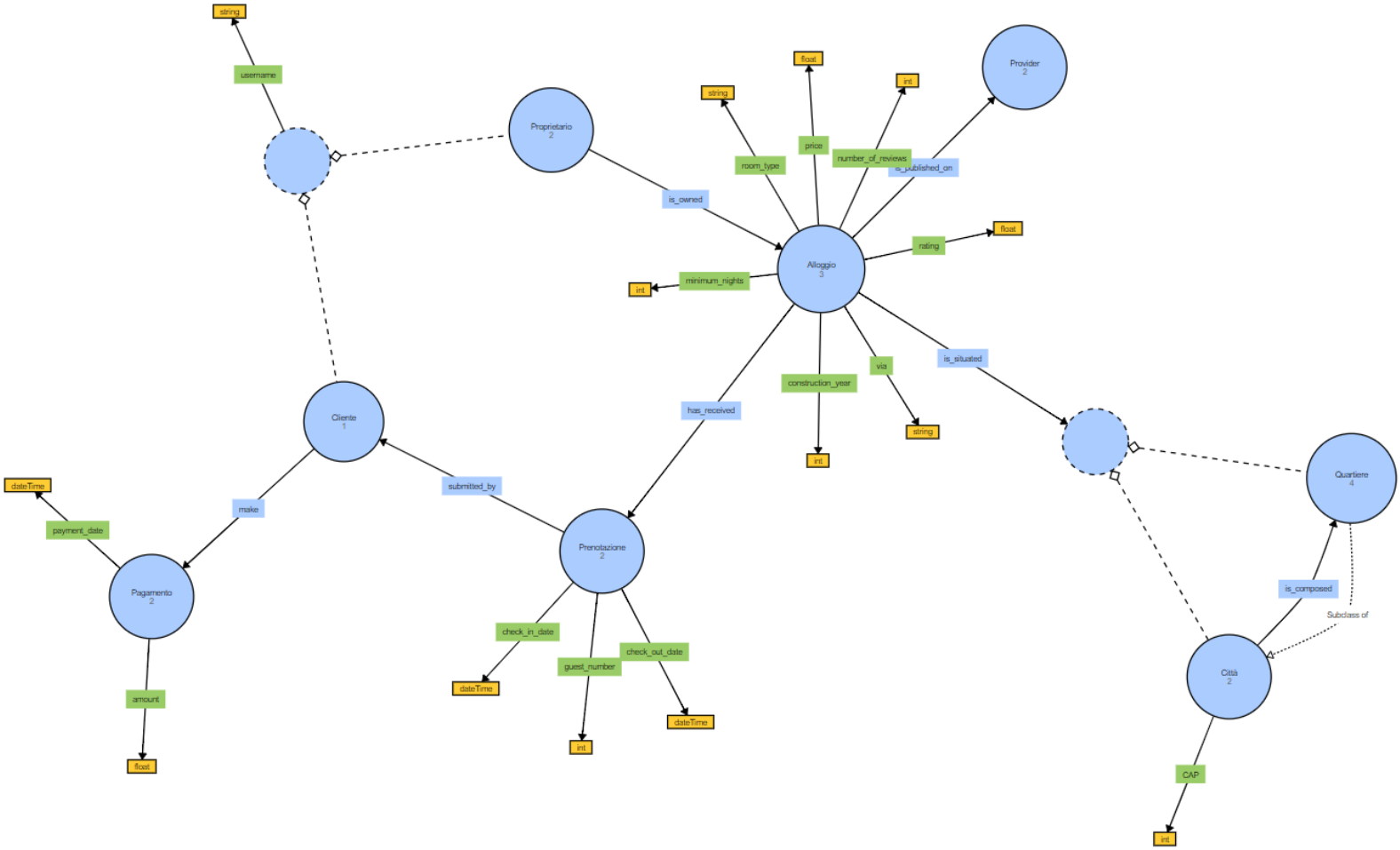


Figura 4.1 – Struttura della nostra ontologia

4.1 Protégé

Nell'elaborazione dell'Ontologia, abbiamo optato per l'utilizzo di Protégé, un editor di ontologie open source. L'ontologia si articola in varie classi, la maggior parte delle quali rappresenta una categoria correlata all'alloggio e ne fornisce una descrizione dettagliata (città, quartiere, prezzo, prenotazione). Inoltre, abbiamo classi che rappresentano i "proprietari" coinvolti nella prenotazione (alloggio, cliente) e la prenotazione stessa.

Riferendosi al grafo, la classe "Thing" è la radice principale da cui si diramano le altre classi. Le classi "Provider", "Alloggio" e "Città" sono collegate direttamente a "Thing". Sotto la classe "Città" troviamo la sottoclasse "Quartiere". Ogni classe ha attributi specifici associati, rappresentati dai rettangoli verdi nel grafo. Infine, le linee tratteggiate indicano le relazioni tra diverse classi, come tra "Alloggio" e "Prenotazione".

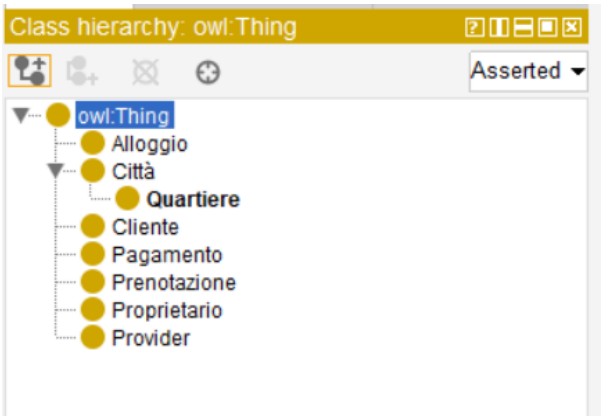


Figura 4.2 – Classi presenti nell'ontologia

Nel processo di costruzione dell'Ontologia, non solo abbiamo definito diverse classi, ma abbiamo anche introdotto una serie di proprietà, specificamente object-properties e data-properties. Le object-properties facilitano la creazione di relazioni tra due individui, che possono appartenere alla stessa classe o a classi differenti. D'altra parte, le data-properties consentono di associare un individuo a un tipo di dato primitivo.

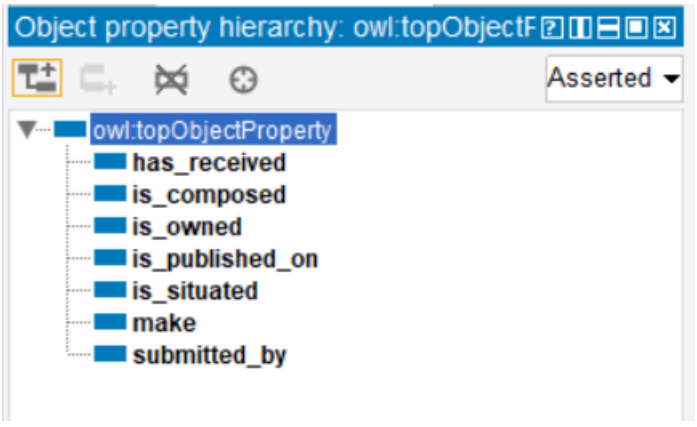


Figura 4.3 – Object property

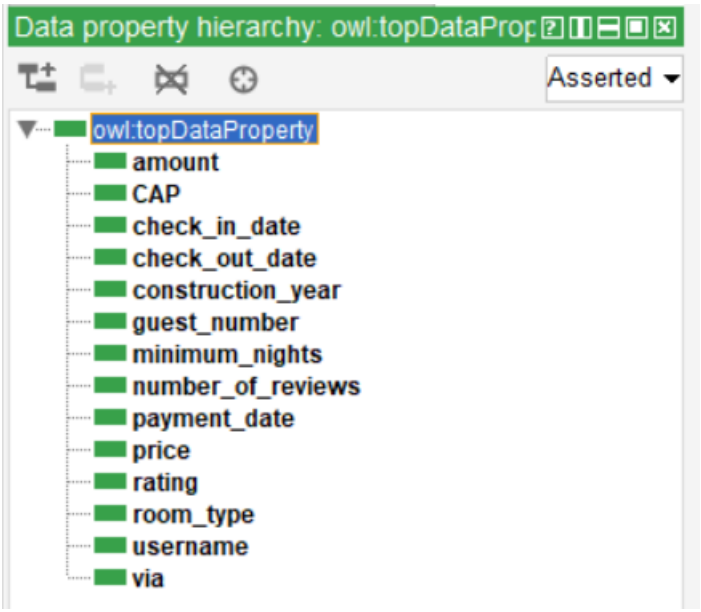


Figura 4.4 – Data property

Successivamente si è passati alla creazione degli individui stessi che rappresentano alcuni esempi.

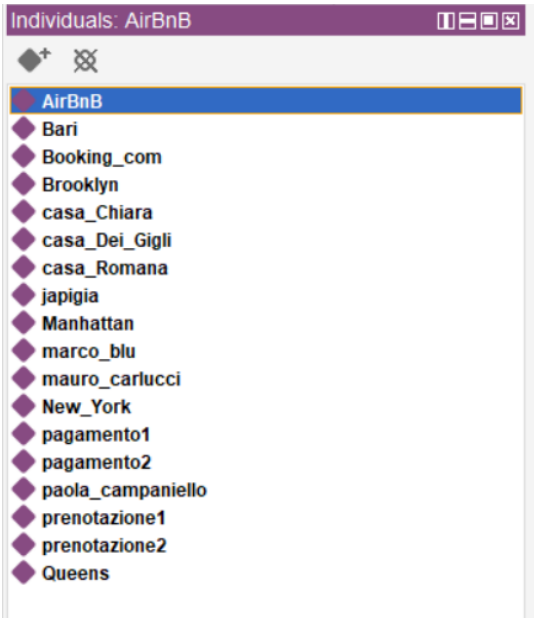


Figura 4.5 – Individui presenti nell'ontologia

Figura 4.6 – Individui della Classe Alloggio

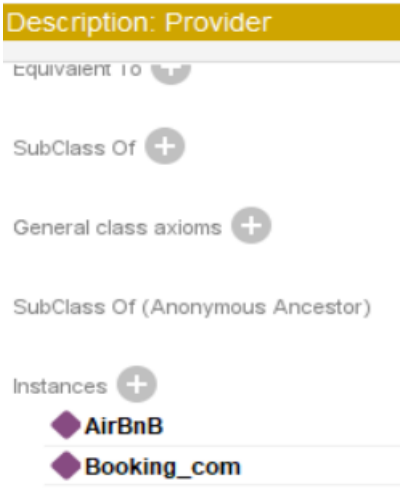


Figura 4.7 – Individui della Classe Provider

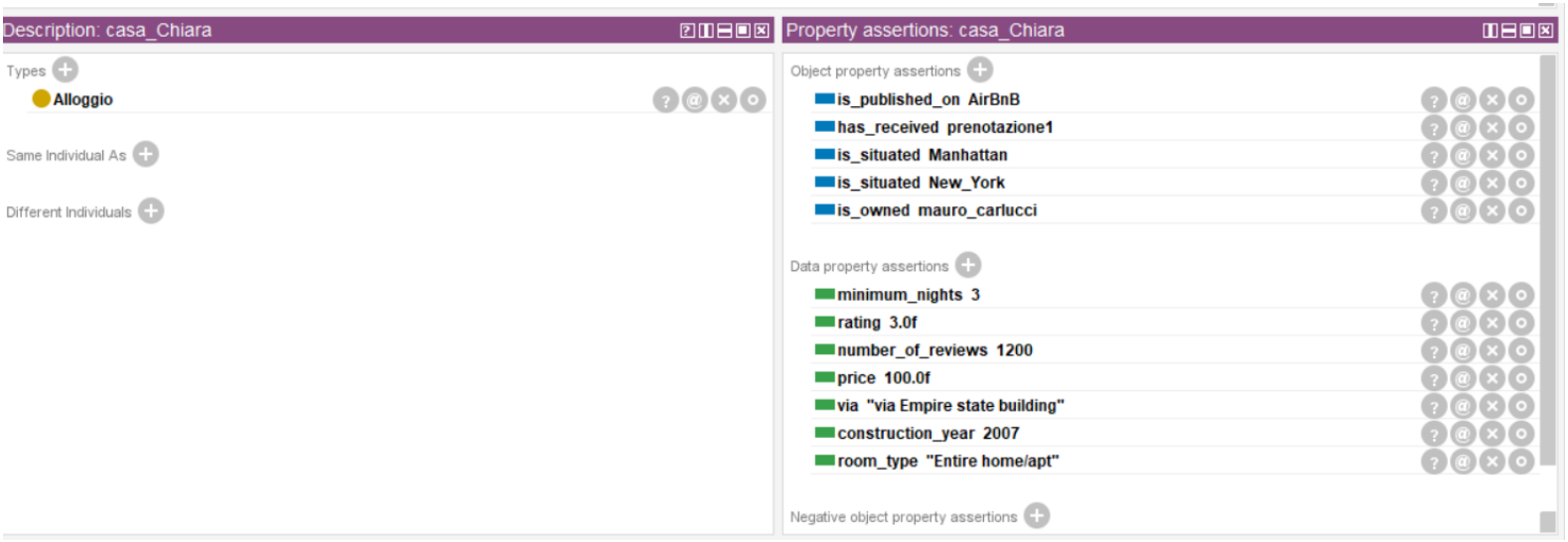


Figura 4.8- Individuo con le proprietà annesse

4.2 Owlready2

Per l'interrogazione dell'ontologia in Python, abbiamo fatto ricorso alla libreria Owlready2. Questa libreria ci ha permesso di eseguire query sull'ontologia che avevamo precedentemente sviluppato con l'editor Protégé, in modo semplice e intuitivo.
Durante l'esecuzione del programma, l'utente avrà la possibilità di scegliere tra la visualizzazione delle classi, delle object properties, delle data properties o di alcune query di esempio.

Nella figura successiva vedremo com'è possibile visualizzare il menu per l'esplorazione all'interno della nostra ontologia, dov'è possibile effettuare le varie scelte:

```
Benvenuto nell'ontologia

Seleziona cosa vorresti esplorare:
1) Visualizzazione Classi
2) Visualizzazione proprietà d'oggetto
3) Visualizzazione proprietà dei dati
4) Esegui query
5) Esci dall' Ontologia

Inserisci qui la tua scelta:
```

Selezionando la prima scelta, ci verranno mostrate le varie classi presenti nell'ontologia:

```
Inserisci qui la tua scelta: 1

Classi presenti nell'ontologia:

[ontology.Alloggio, ontology.Prenotazione, ontology.Città, ontology.Quartiere, ontology.Proprietario, ontology.Provider, ontology.Cliente, ontology.Pagamento]

Vorresti esplorare meglio qualche classe in particolare?

1) Alloggio
2) Città
3) Proprietario
4) Provider
5) Quartiere
6) Prenotazione
7) Cliente
8) Pagamento

Inserisci qui la tua scelta:
```

Volendo visionare in maniera più dettagliata le classi, abbiamo la possibilità di selezionarla tramite questa funzionalità, in questo caso abbiamo selezionato la seconda scelta, per visionare più nello specifico la classe Città, noteremo quindi la lista che ci verrà fornita delle città presenti:

```
Vorresti esplorare meglio qualche classe in particolare?

1) Alloggio
2) Città
3) Proprietario
4) Provider
5) Quartiere
6) Prenotazione
7) Cliente
8) Pagamento

Inserisci qui la tua scelta: 2

Lista delle città presenti:

[ontology.Città, ontology.Bari, ontology.New_York, ontology.Quartiere, ontology.japigia, ontology.Brooklyn, ontology.Manhattan, ontology.Queens]

Vuoi esaminare un'altra classe? (y/n):
```

Abbiamo dunque anche la possibilità di visionare le varie **data properties** contenute nella nostra ontologia, scegliendo la terza scelta proposta:

```
Seleziona cosa vorresti esplorare:
1) Visualizzazione Classi
2) Visualizzazione proprietà d'oggetto
3) Visualizzazione proprietà dei dati
4) Esegui query
5) Esci dall' Ontologia

Inserisci qui la tua scelta: 3

Proprietà dei dati presenti nell'ontologia:

[ontology.CAP, ontology.amount, ontology.check_in_date, ontology.check_out_date, ontology.construction_year, ontology.guest_number, ontology.minimum_nights, ontology.number_o
```

Ora vediamo la possibilità anche di visualizzare le diverse **object properties**, selezionando in questo caso la seconda scelta:

```
Seleziona cosa vorresti esplorare:
1) Visualizzazione Classi
2) Visualizzazione proprietà d'oggetto
3) Visualizzazione proprietà dei dati
4) Esegui query
5) Esci dall' Ontologia

Inserisci qui la tua scelta: 2

Proprietà d'oggetto presenti nell'ontologia:

[ontology.has_received, ontology.is_composed, ontology.is_owned, ontology.is_published_on, ontology.is_situated, ontology.make, ontology.submitted_by]
```

Ora mostriamo la possibilità di visualizzare alcune Query d'esempio, selezionando la quarta scelta. Successivamente, ci verrà inoltre richiesto di selezionare quale delle query disponibili si vuole eseguire:

```
Seleziona cosa vorresti esplorare:
1) Visualizzazione Classi
2) Visualizzazione proprietà d'oggetto
3) Visualizzazione proprietà dei dati
4) Esegui query
5) Esci dall' Ontologia

Inserisci qui la tua scelta: 4
1) Lista di alloggi che presentano il proprietario 'mauro_carlucci
2) Lista di alloggi pubblicati su 'AirBnB'
3) Prenotazioni effettuate da Marco Blu
4) Torna indietro
```

```
inserisci la tua scelta: 1
[ontology.casa_Chiara]

1) Lista di alloggi che presentano il proprietario 'mauro_carlucci
2) Lista di alloggi pubblicati su 'AirBnB'
3) Prenotazioni effettuate da Marco Blu
4) Torna indietro

inserisci la tua scelta: 2
[ontology.casa_Chiara, ontology.casa_Dei_Gigli, ontology.casa_Romana]

1) Lista di alloggi che presentano il proprietario 'mauro_carlucci
2) Lista di alloggi pubblicati su 'AirBnB'
3) Prenotazioni effettuate da Marco Blu
4) Torna indietro

inserisci la tua scelta: 3
[ontology.prenotazione1, ontology.prenotazione2]
```

CONCLUSIONE

Questo progetto ha un ampio margine di espansione in quanto potrebbe essere espanso in futuro in tanti modi.
Si può infatti espandere il dataset per prendere in considerazione non solo New York ma anche altre città in America e non, in modo da poter studiare a fondo eventuali correlazioni.
Si può inoltre espandere il dataset in modo da prendere in considerazione anche altri provider per alloggi in affitto, allargando dunque la ricerca anche ad altre piattaforme come Booking, Trivago, Agoda…