

# PROGETTO TRIPADVISOR

By Analisti Anonimi

# OBIETTIVO DEL PROGETTO

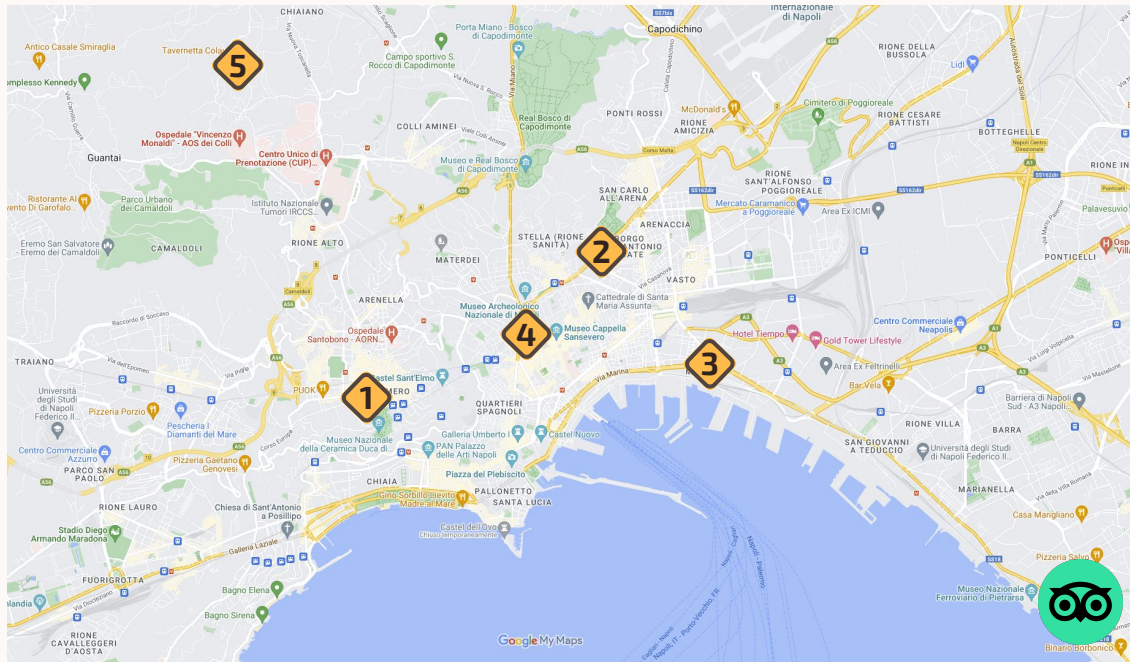
Scaricare 400 recensioni dei primi 5 ristoranti di Napoli, utilizzando questo link di TripAdvisor:

[I MIGLIORI 10 ristoranti a Napoli - Aggiornamento di maggio 2023](#)

- Scaricare 400 recensione dei primi 5 ristoranti di Napoli
- Eliminare le stopwords
- Creare una TAG Cloud delle 50 parole più usate per ogni ristorante con rating 4 e 5;
- Creare una TAG Cloud delle 50 parole più usate per ogni ristorante con rating 1, 2, 3;
- Eliminare le stopwords dalle TAG Cloud;
- Creare un Database contenente due tabelle: ristoranti e recensioni.

## POSIZIONI RISTORANTI SU MAPPA

- 1° Ristorante
- 2° Ristorante
- 3° Ristorante
- 4° Ristorante
- 5° Ristorante



# INTRODUZIONE

L'organizzazione del progetto è partita da un *brainstorming* iniziale di tutti i componenti degli "Analisti Anonimi", andando a definire come sarebbe stato svolto il progetto.



# COMPOSIZIONE DEL PROGETTO

01

## DRIVER E DOWNLOAD HTML

Importazione delle librerie, setup del driver selenium e download delle pagine HTML

02

## ESTRAZIONE DATI RECENSIONI

Estrazione dei dati delle recensioni dall'HTML

03

## ESTRAZIONE DATI RISTORANTI

Estrazione dei dati dei ristoranti dall'HTML

04

## GENERAZIONE WORDCLOUD

Generazione delle wordcloud ed eliminazione delle stopwords

05

## CREAZIONE DATABASE

Definizione tabelle "ristoranti" e "recensioni" e importazione dei dati in quest'ultime

06

## GENERAZIONE GRAFICO RECENSIONI

Generazione grafico interattivo tramite libreria "plotly"



01

# DRIVER E DOWNLOAD HTML

```
from bs4 import BeautifulSoup
import time
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
import re
from cleantext import clean
from wordcloud import WordCloud
import pandas as pd
import matplotlib.pyplot as plt
import sqlite3
import plotly.express as px
```

1. **Selenium** serve per l'automazione del browser. Permette di controllare, tramite un driver, un browser web in maniera automatizzata.
2. **ChromeDriverManager** facilita l'installazione e l'aggiornamento automatico dei driver Chrome necessari per controllare il browser tramite Selenium.
3. **By** è una classe in Selenium che fornisce metodi per identificare gli elementi della pagina web in base a diversi criteri come l'ID, il nome o la classe.
4. **re** è il modulo delle espressioni regolari in Python, che fornisce uno strumento per la manipolazione e il controllo di stringhe di testo.
5. **cleantext** è una libreria python per la pulizia di testo. Fornisce funzioni per rimuovere caratteri indesiderati. Nel nostro caso le emoji dalle recensioni.
6. **WordCloud** serve per generare "nuvole di parole" in cui la dimensione delle parole rappresenta la loro frequenza.
7. **Matplotlib** è stata utilizzata in concomitanza della libreria WordCloud, dato che ci ha permesso di visualizzare la "nuvola di parole", che è in tutti i sensi un grafico.
8. **Time** è una libreria Python per gestire il tempo, in questo caso è stata utilizzata per creare dei piccoli intervalli nei processi di web-scraping in modo da non dare sospetti allo "snort" di TripAdvisor.
9. **Pandas** è una libreria multi-funzione, in questo progetto è stata utilizzata per creare dei DataFrame contenenti i dati dei ristoranti e delle recensioni per essere poi trasferiti nel database. Inoltre è stata utilizzata per creare il DataFrame delle parole con il relativo conteggio, che poi è stato utilizzato per la creazione delle WordCloud.
10. **Plotly.express** è una sotto-componente della libreria Plotly che permette la creazione di grafici interattivi che danno la possibilità di visualizzare meglio dei dati. In particolare è stata utilizzata per mostrare la percentuale di ogni rating (1,2,3,4,5 stelle) delle recensioni per ogni ristorante.
11. **SQLite3** è stata utilizzata per la creazione del Database. Essa permette di creare, gestire e interrogare database SQLite utilizzando il linguaggio SQL.

```
def start_driver():  
    driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))  
    return driver
```

La funzione **start\_driver()** utilizza una componente della libreria Selenium e si occupa dell'attivazione del driver web.

Nel nostro caso utilizziamo come browser web Chrome e con la libreria driver manager ci scarichiamo il driver aggiornato alla corrispettiva versione del nostro browser



```

def get_html(driver):
    num_rev = [400, 290, 400, 90, 400]
    ID_rist = ["d3741499", "d23807140", "d4318643", "d11785758", "d2062029"]

    for i in range(len(ID_rist)):
        listaHTML = []

        driver.get(f"https://www.tripadvisor.it/Restaurant_Review-g187785-{ID_rist[i]}-Reviews")
        time.sleep(5)
        try:
            driver.find_element(By.ID, "onetrust-accept-btn-handler").click()
        except:
            pass

        listaHTML.append(driver.page_source)
        time.sleep(5)

        for k in range(10, (num_rev[i]-9), 10):
            try:
                driver.get(f"https://www.tripadvisor.it/Restaurant_Review-g187785-{ID_rist[i]}-Reviews-or{k}")
                listaHTML.append(driver.page_source)
                time.sleep(5)
                print(i+1,k+10)
            except:
                print(f"fine{i}")
                break

        html_uniti = '\n'.join(listaHTML)

        f = open(f"ristorante_{i+1}.html", 'w', encoding="utf-8")
        f.write(html_uniti)
        f.close()

    driver.quit()

```

La funzione **get\_html(driver)** prende come parametro il driver web inizializzato in precedenza. Definiamo poi due liste, la prima contiene il numero massimo di recensioni estraibili per ogni ristorante (comunque limitato superiormente a 400), parametro che andrà in una stringa formattata successivamente.

la seconda contiene i codici univoci di ogni ristorante, sempre da inserire nei link per aprire le pagine dei vari ristoranti.

A questo punto cicliamo su tutti e 5 i ristoranti. Per ognuno, prima apriamo la pagina iniziale in cui troviamo già 10 recensioni e che quindi ci scarichiamo con il metodo `page_source` di Selenium.. Nel frattempo, se si presenta il pulsante per accettare i cookies cerchiamo di accettarlo. Dopodichè cicliamo da 10 al numero massimo di recensioni per ogni ristorante a intervalli di 10. Inseriamo questo valore nella stringa formattata sottostante che tramite il metodo `get` raggiunge i link con le varie pagine delle recensioni e le scarica una ad una. Infine le varie pagine sono unite con la funzione `join` e il file html unito di ogni ristorante viene salvato nella directory sempre in formato html. Concludiamo chiudendo il driver.

The background features decorative wavy lines in the corners. On the left, a light beige circle is partially visible, with a dark brown wavy line extending from it towards the bottom. On the right, a dark brown wavy line extends from the top towards the right edge, and a light beige shape is partially visible at the bottom right.

**02**

# **ESTRAZIONE DATI RECENSIONI**

```
# get all reviews info

def get_all_reviews(n):

    ID_rist = ["d3741499", "d23807140", "d4318643", "d11785758", "d2062029"]

    html = open(f"ristorante_{n}.html", 'r', encoding = "utf-8")

    all_info_rev = []

    soup = BeautifulSoup(html, "html.parser")
    reviews_boxes = soup.find_all("div", {"class": "review-container"})

    for r in reviews_boxes:

        dct = {
            "ID": ID_rist[n-1],
            "rating": get_rate(r),
            "reviewdate": get_date(r),
            "title": get_title(r),
            "text": get_review(r)}

        all_info_rev.append(dct)

    html.close()

    return all_info_rev
```

La funzione **get\_all\_reviews** si occupa di raccogliere i dati delle recensioni di un ristorante a nostra scelta, indicato con n (n che va da 1 a 5).

Procediamo aprendo il corrispettivo file e analizzandolo con beautiful soup. Con quest'ultimo, tramite il metodo find\_all troviamo tutti i contenitori con il tag div che contengono le informazioni di ogni recensione. Useremo questi contenitori per cercarci all'interno gli altri elementi tramite le funzioni di acquisizioni successive.

A questo punto, per ogni contenitore delle recensioni, creiamo un dizionario che conterrà come chiavi, Id del ristorante, autore della recensione, rating, data della recensione, titolo e testo della recensione, e come valori gli output delle corrispettive funzioni di acquisizione. Appendiamo il dizionario di ogni recensione alla lista dei dizionari e quando il ciclo è finito e quindi le recensioni sono state analizzate tutte, chiudiamo anche l'html del ristorante 'n' e ritorniamo in output proprio la lista di dizionari. questa ci servirà in seguito sia per la realizzazione delle wordcloud sia per la creazione dei database.

```
# Funzione Get Rate
def get_rate(r): # get rating

    if (r.find("span", {"class": "ui_bubble_rating bubble_10"})) is not None:
        stars = 1
    elif (r.find("span", {"class": "ui_bubble_rating bubble_20"})) is not None:
        stars = 2
    elif (r.find("span", {"class": "ui_bubble_rating bubble_30"})) is not None:
        stars = 3
    elif (r.find("span", {"class": "ui_bubble_rating bubble_40"})) is not None:
        stars = 4
    elif (r.find("span", {"class": "ui_bubble_rating bubble_50"})) is not None:
        stars = 5
    else:
        stars = ""

    return stars
```

→ **get\_rate(r)**: Questa funzione restituisce il rating (numero di stelle) associato a una recensione r. La funzione cerca un elemento <span> con una classe specifica all'interno della recensione. A seconda della classe trovata, viene assegnato un valore numerico corrispondente al numero di stelle (da 1 a 5). Se la classe non viene trovata, viene restituita una stringa vuota.

```
# Funzione Get Title Review
def get_title(r):
    value = r.find("span", {"class": "noQuotes"})
    if value is None:
        title = ""
    else:
        title = value.text.strip()
        title = clean(title, no_emoji= True)
    return title
```

→ **get\_title(r)**: Questa funzione restituisce il titolo di una recensione r. La funzione cerca un elemento <span> con una classe specifica che contiene il titolo della recensione. Se l'elemento non viene trovato, viene restituita una stringa vuota. Altrimenti, viene estratto il testo contenuto nell'elemento, pulito dagli spazi iniziali e finali e dalle emoji utilizzando la libreria cleantext. Il titolo pulito viene restituito come output della funzione.

```
# Funzione Get Date
def get_date(r): # get date
    value = r.find("div", {"class": "prw_rup prw_reviews_stay_date_hsx"})
    if value is None:
        date = ""
    else:
        date = value.text.strip()
        date = date[19:]
    return date
```

→ **get\_date(r)**: Questa funzione restituisce la data associata a una recensione r. La funzione cerca un elemento <div> con una classe specifica che contiene la data della recensione. Se l'elemento non viene trovato, viene restituita una stringa vuota. Altrimenti, viene estratto il testo contenuto nell'elemento e viene tenuta solo la parte relativa alla data (presumendo che ci sia una formattazione fissa). La data viene restituita come output della funzione.

```
# Funzione Get Reviews
```

```
def get_review(r):  
    review = r.find("div", {"class": "entry"}).text  
    review_piu = review.replace("Più", "")  
    cleaned_review = clean(review_piu, no_emoji=True)  
    return cleaned_review
```

→ **get\_review(r)**: Questa funzione estrae il testo della recensione r. La funzione cerca un elemento <div> con una classe specifica che contiene il testo della recensione. Il testo viene estratto come semplice testo all'interno dell'elemento, inclusa l'eventuale parte nascosta delle recensioni troppo lunghe. Viene rimossa la stringa "Più" che potrebbe essere presente per espandere le recensioni lunghe. Infine, il testo viene pulito dalle emoji utilizzando la libreria cleantext e viene restituito come output della funzione.



# ESTRAZIONE DATI RISTORANTI



## # ESTRAZIONE INFO GENERALI RISTORANTI

```
def get_info_rist():
```

```
    all_info_rist = []
```

creiamo una lista vuota

```
    ID_rist = ["d3741499", "d23807140", "d4318643", "d11785758", "d2062029"]
```

Forniamo l'ID dei ristoranti dal link della pagina web

```
    for i in range(1,6):
```

Creiamo un ciclo for per automatizzare l'estrazione delle variabili

```
        html = open(f"ristorante_{i}.html", 'r', encoding = "utf-8")
        soup = BeautifulSoup(html, "html.parser")
```

Apriamo i vari file html come stringa formattata (f), in modalità lettura (r)

```
        dct = {
            "ID": ID_rist[i-1],
            "name": get_name_rest(soup),
            "price": get_price(soup),
            "typecook": get_typecook(soup),
            "address": get_address(soup)}
```

Utilizziamo la libreria BeautifulSoup, mentre con "html.parser" definiamo la modalità per l'estrazione dei Tag dei ristoranti

Il dizionario conterrà: id del ristorante, nome del ristorante, fascia di prezzo, tipo di cucina ed indirizzo

```
        all_info_rist.append(dct)
```

```
        html.close()
```

Appendiamo i dizionari alla lista dei dizionari, che sarà l'output della funzione

```
    return all_info_rist
```

```
# get restaurant name
```

```
def get_name_rest(r):
```

```
    name_rest = r.find("h1", {"class": "HjBfq"}).text
```

```
    return name_rest
```

La funzione `get_name_rest` riceve un oggetto `r` come parametro e restituisce il nome del ristorante estratto da tale oggetto.

All'interno della funzione, viene utilizzato il metodo `find` sull'oggetto `r` per trovare un elemento `<h1>` con il valore dell'attributo `class` impostato su `"HjBfq"`. Il testo di questo elemento viene estratto utilizzando il metodo `text` e assegnato alla variabile `name_rest`.

Infine, il nome del ristorante viene restituito come risultato della funzione.

*# Funzione Get Price*

```
def get_price(r):
```

```
    box = r.find("span", {"class": "DsyBj DxyfE"})
```

```
    price = box.find("a", {"class": "dlMOJ"}).text
```

```
    return price
```

La funzione `get_price` riceve un oggetto `r` come parametro e restituisce il prezzo estratto da tale oggetto.

All'interno della funzione, viene utilizzato il metodo `find` sull'oggetto `r` per trovare un elemento `<span>` con il valore dell'attributo `class`.

Il risultato di questa operazione viene assegnato alla variabile `box`.

Successivamente, viene utilizzato il metodo `find` sull'oggetto `box` per trovare un elemento `<a>` con il valore dell'attributo `class`, ed il testo di questo elemento viene estratto utilizzando il metodo `text` e assegnato alla variabile `price`.

Infine, il prezzo viene restituito come risultato della funzione.

# Funzione Get Type cook

```
def get_typecook(r):  
  
    box = r.find("span", {"class": "DsyBj DxyfE"})  
    typecook_list = box.find_all("a", {"class": "dlMOJ"})  
  
    if typecook_list is None:  
        typecook = ""  
    else:  
        typecook = []  
  
        for i in typecook_list:  
            i = i.text  
            typecook.append(i)  
  
    typecook = ", ".join(typecook[1:])  
  
    return typecook
```

All'interno della funzione, viene utilizzato il metodo find sull'oggetto r per trovare un elemento <span> con il valore dell'attributo. Il risultato di questa operazione viene assegnato alla variabile box.

Successivamente, viene utilizzato il metodo find\_all sull'oggetto box per trovare tutti gli elementi <a> con il valore dell'attributo class. I risultati vengono assegnati alla lista typecook\_list.

Successivamente, viene eseguito un controllo per verificare se typecook\_list è vuota.

Se typecook\_list è None, viene assegnata una stringa vuota alla variabile typecook. In caso contrario, viene inizializzata una lista vuota typecook e viene eseguito un loop su ogni elemento i in typecook\_list.

All'interno del loop, viene estratto il testo di ogni elemento i utilizzando il metodo text e viene aggiunto alla lista type cook utilizzando il metodo append.

Dopo il loop, viene utilizzato il metodo join per concatenare tutti gli elementi della lista type cook, separati da una virgola. La concatenazione inizia dall'elemento con indice 1 (typecook[1:]) per escludere il primo elemento.





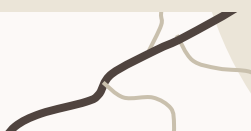
```
# Funzione Get Address
```

```
def get_address(r):
```

```
    address = r.find_all("a", {"class": "AYHFM"})  
    address = address[1].text
```

```
    return address
```

La funzione `get_address` riceve un oggetto `r` come parametro e restituisce l'indirizzo estratto da tale oggetto. All'interno della funzione, viene utilizzato il metodo `find_all` sull'oggetto `r` per trovare tutti gli elementi `<a>` con il valore dell'attributo `class`. Il risultato di questa operazione viene assegnato alla variabile `address`. Successivamente, si accede al secondo elemento nella lista `address` (indicizzato come `address[1]`) e si estrae il testo dell'elemento utilizzando il metodo `text`. Questo testo rappresenta l'indirizzo desiderato. Infine, l'indirizzo viene restituito come risultato della funzione.





# GENERAZIONE WORDCLOUD

```
def generate_wordcloud(n):

    reviews = get_all_reviews(n)
    Dataframe = pd.DataFrame.from_dict(reviews)

    DF_45 = Dataframe[Dataframe["rating"] >= 4]
    DF_13 = Dataframe[Dataframe["rating"] < 4]

    reviews_45 = " ".join(DF_45["text"].tolist())
    reviews_13 = " ".join(DF_13["text"].tolist())

    REG = r"\w+"
    review_words_45 = re.findall(REG, reviews_45)
    review_words_13 = re.findall(REG, reviews_13)

    f = open("stopwords-it.txt", encoding="utf8")
    stopwords = []
    for i in f:
        stopwords.append(i.strip())
    f.close()

    cleaned_review45_words = []
    for k in review_words_45:
        if k not in stopwords:
            cleaned_review45_words.append(k)
    unique_review45_words = set(cleaned_review45_words)

    cleaned_review13_words = []
    for k in review_words_13:
        if k not in stopwords:
            cleaned_review13_words.append(k)
    unique_review13_words = set(cleaned_review13_words)

    dict_review45 = {}
    for j in unique_review45_words:
        dict_review45[j] = cleaned_review45_words.count(j)

    dict_review13 = {}
    for j in unique_review13_words:
        dict_review13[j] = cleaned_review13_words.count(j)
```

Abbiamo creato la funzione ***generate\_wordcloud*** che prende come input il parametro *n*, ovvero il numero che identifica il ristorante. Successivamente richiamiamo la funzione ***get\_all\_reviews*** che ci serve per prenderci tutte le recensioni del ristorante. Convertiamo tutte le recensioni in un ***DataFrame Pandas***, andando poi a dividerlo secondo le recensioni di rating  $\geq 4$  e  $< 4$ . Abbiamo poi concatenato in una lista le recensioni, separandole con lo spazio. Per rimuovere caratteri indesiderati (come ad esempio le ***emoji***) e prenderci solo le parole escludendo anche le punteggiature e gli spazi è stata utilizzata la libreria ***re***, che ci ha permesso di farlo utilizzando le espressioni regolari.

In seguito è stato utilizzato il file delle ***stopwords*** italiane, che contiene un insieme di parole che non aggiungono particolare significato al testo, che di conseguenza possono essere rimosse dalle recensioni.

Tramite dei cicli *for* andiamo ad “appendere” la lista di ***“parole pulite”*** che andremo poi ad utilizzare per generare le word clouds. Queste liste verranno poi convertite in *set* (perchè sappiamo che il *set* è una sequenza di elementi unici non ordinati).

Successivamente vengono creati due dizionari, in cui ***la chiave è una parola unica e il valore è il conteggio di quante volte appare quella parola*** nelle recensioni corrispondenti. Poi abbiamo convertito i dizionari in *DataFrame*, per poi rinominarli con la colonna “count”.



```

wordcloud_df45 = pd.DataFrame.from_dict(dict_review45, orient="index")
wordcloud_df45.columns = ["count"]

wordcloud_df13 = pd.DataFrame.from_dict(dict_review13, orient="index")
wordcloud_df13.columns = ["count"]

wordcloud45 = WordCloud(background_color = "white",
                        max_words = 50, width = 600, height = 600)

wordcloud45.generate_from_frequencies(wordcloud_df45.to_dict()["count"])

plt.title(f"Wordcloud delle recensioni con rating 4 o 5 del ristorante {n}")
plt.imshow(wordcloud45, interpolation="bilinear")
plt.axis("off")
plt.savefig(f"wordcloud_45_ristorante_{n}.png")
plt.show()

wordcloud13 = WordCloud(background_color = "white",
                        max_words = 50, width = 600, height = 600)

wordcloud13.generate_from_frequencies(wordcloud_df13.to_dict()["count"])

plt.title(f"Wordcloud delle recensioni con rating 1, 2 o 3 del ristorante {n}")
plt.imshow(wordcloud13, interpolation="bilinear")
plt.axis("off")
plt.savefig(f"wordcloud_13_ristorante_{n}.png")
plt.show()

count45 = wordcloud_df45.sort_values(['count'], ascending=False)[0:10].rename_axis('parole per rating 4 e 5', axis=1) #
count13 = wordcloud_df13.sort_values(['count'], ascending=False)[0:10].rename_axis('parole per rating da 1 a 3', axis=1)

return count45, count13

```

In seguito andiamo a generare le word clouds per le recensioni con rating da 1 a 3 e da 4 a 5, e lo generiamo a partire dalle frequenze delle parole nel DataFrame.

Infine, i due DataFrame contenenti i 10 termini più frequenti per ciascun rating vengono restituiti come output della funzione, oltre i grafici che abbiamo generato in precedenza.





# DATABASE

# DATABASE

Tramite la libreria Sqlite3 abbiamo creato un database contenente due tabelle:

- **Ristoranti**: contiene le informazioni su ogni ristorante:
  - ID Ristorante
  - Nome
  - Prezzo
  - Genere Cucina
  - Indirizzo
- **Recensioni**: contiene le informazioni riguardanti ogni recensione:
  - ID Ristorante
  - Autore
  - Titolo
  - Data
  - Rating
  - Testo

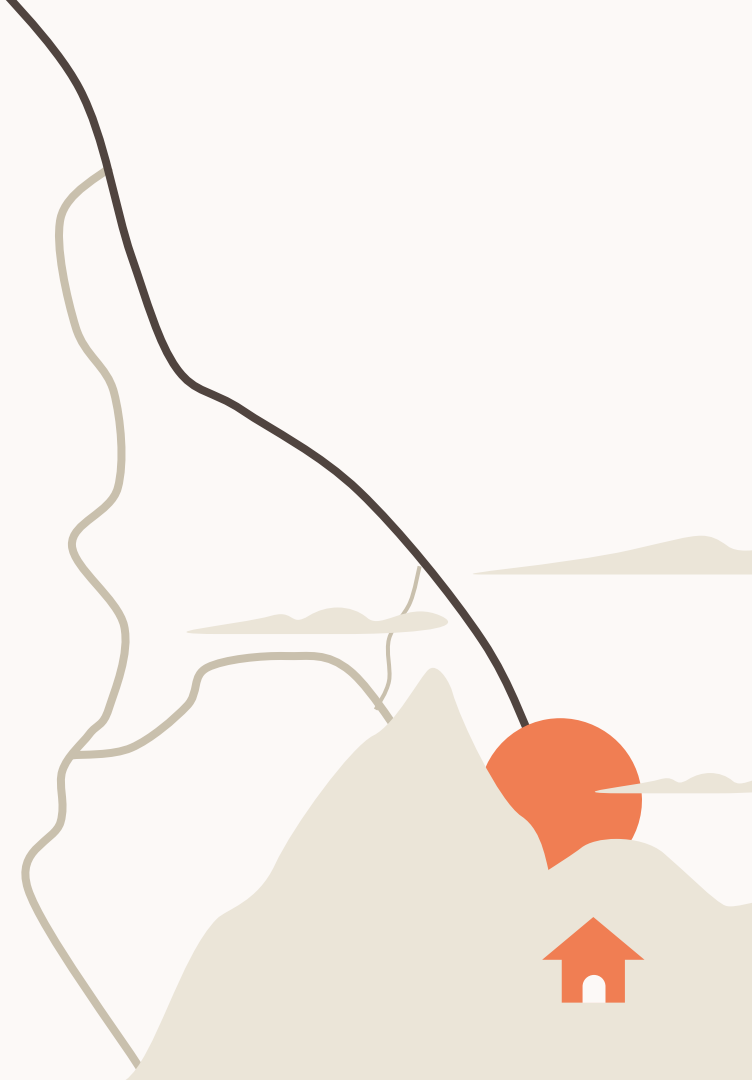
# DATABASE

Dopo aver creato la base dati abbiamo generato tramite la libreria **Pandas** un DataFrame ed inserito i dati all'interno delle tabelle del DB tramite il metodo "to\_sql".

```
dfRistoranti.to_sql("ristoranti", db, if_exists='replace', index = False)  
dfRecensioni.to_sql("recensioni", db, if_exists='replace', index = False)
```



# GRAFICI RECENSIONI

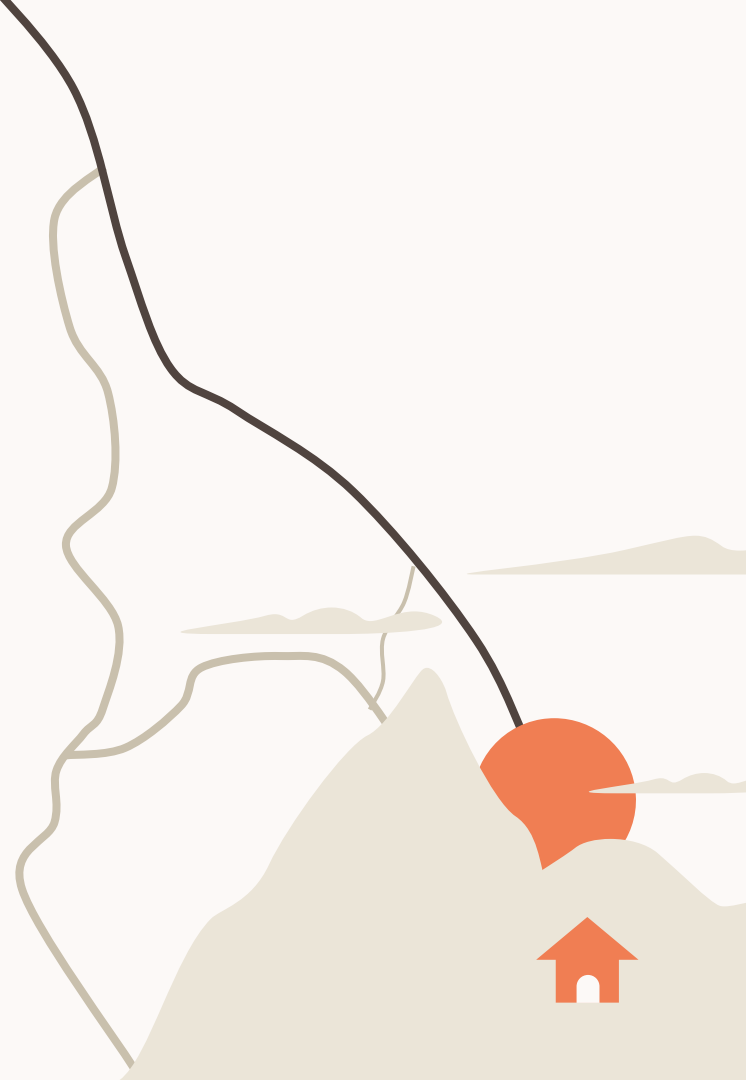


# GRAFICI RECENSIONI

Per presentare meglio la distribuzione delle stelle nelle recensioni per ogni ristorante abbiamo generato 5 grafici tramite la libreria "plotly", in particolare con una sua sotto-componente, ovvero "plotly.express".

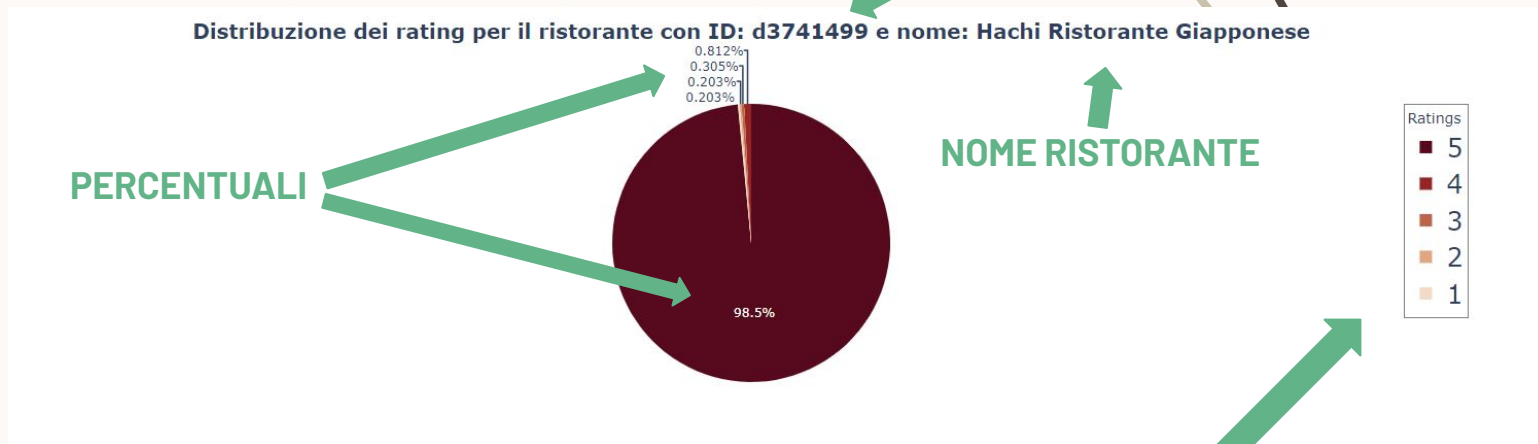
La funzione creata genera anche dei file HTML (che mantengono l'interattività del grafico) con all'interno i relativi grafici.

Una proprietà di questa libreria è infatti la generazione di grafici interattivi.



# GRAFICI RECENSIONI

Esempio di grafico Plotly:

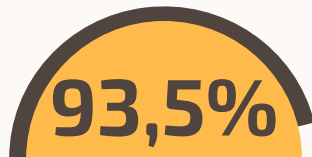


# PERCENTUALI DI RECENSIONI A 5 STELLE



## 1° Ristorante

Il primo ristorante su Napoli totalizza il 98,5% di recensioni a 5 stelle



## 2° Ristorante

Il secondo si attesta sul 93,5% di recensioni a 5 stelle



## 3° Ristorante

Il terzo ottiene il 90,1% di recensioni a 5 stelle



## 4° Ristorante

Contro le aspettative il quarto ristorante si attesta al 95,6% di recensioni a 5 stelle



## 5° Ristorante

Infine il quinto ristorante che totalizza il 94,2% di recensioni a 5 stelle

# THANKS

PER QUALSIASI INFORMAZIONE  
RIGUARDANTE IL PROGETTO:

[a.caccese@lumsastud.it](mailto:a.caccese@lumsastud.it)

[d.mariani6@lumsastud.it](mailto:d.mariani6@lumsastud.it)

[s.desaraca@lumsastud.it](mailto:s.desaraca@lumsastud.it)

[v.valentini4@lumsastud.it](mailto:v.valentini4@lumsastud.it)

[e.corradi1@lumsastud.it](mailto:e.corradi1@lumsastud.it)

**ANALISTI ANONIMI**



# Font e colori usati

Questa presentazione è stata fatta usando i seguenti font:

**Exo 2**

<https://fonts.google.com/specimen/Exo+2>

**Barlow**

<https://fonts.google.com/specimen/Barlow>

#ebe5d8

#c9c0ad

#ffba4c

#f07e53

#63b389

#878e91