

Corso di Laurea in Ingegneria Gestionale  
Classe L-8



**Politecnico  
di Torino**

# Applicazione per la creazione di un itinerario gastronomico personalizzato

**Relatore**  
Carlo Masone

**Candidato**  
Lorenzo Grivet Talocia

Ottobre 2024

# Indice

<b>1</b>	<b>Proposta di progetto</b>	<b>4</b>
1.1	Titolo della proposta . . . . .	4
1.2	Descrizione del problema proposto . . . . .	4
1.3	Descrizione della rilevanza gestionale del problema . . . . .	4
1.4	Descrizione del data-set per la valutazione . . . . .	4
1.5	Descrizione preliminare degli algoritmi coinvolti . . . . .	5
1.6	Descrizione preliminare delle funzionalità previste per l'applicazione software . . . . .	5
<b>2</b>	<b>Descrizione del problema affrontato</b>	<b>6</b>
2.1	Contesto . . . . .	6
2.2	Input e output . . . . .	6
2.3	Potenzialità e criticità . . . . .	6
<b>3</b>	<b>Descrizione del dataset</b>	<b>7</b>
<b>4</b>	<b>Descrizione algoritmi utilizzati</b>	<b>8</b>
4.1	Pattern MVC e DAO . . . . .	8
4.2	Algoritmi principali utilizzati . . . . .	8
4.2.1	Compilazione automatica dei parametri . . . . .	8
4.2.2	Salvataggio delle scelte . . . . .	9
4.2.3	Ricerca dei migliori ristoranti . . . . .	9
4.2.4	Creazione del grafo . . . . .	10
4.2.5	Algoritmo di ricorsione . . . . .	11
<b>5</b>	<b>Diagramma delle classi</b>	<b>13</b>
<b>6</b>	<b>Videate dell'applicazione</b>	<b>14</b>
<b>7</b>	<b>Risultati sperimentali</b>	<b>17</b>
7.1	Utilizzo di SQL . . . . .	17
7.2	Algoritmo ricorsivo . . . . .	18

<b>8</b>	<b>Valutazioni dei risultati ottenuti</b>	<b>19</b>
8.1	Punti di forza . . . . .	19
8.2	Punti di debolezza . . . . .	19
8.3	Conclusioni . . . . .	19
<b>9</b>	<b>Licenza</b>	<b>20</b>

# Capitolo 1

## Proposta di progetto

### 1.1 Titolo della proposta

Applicazione per la creazione di un itinerario gastronomico personalizzato

### 1.2 Descrizione del problema proposto

Lo scopo dell'applicazione è quello di generare automaticamente un percorso gastronomico personalizzato all'interno di una città desiderata, elaborato rispettando i vincoli scelti dall'utente, in modo tale da massimizzare l'indicatore di qualità dei ristoranti. L'utente può infatti, per ogni città scelta, selezionare le proprie preferenze di tipologia di cucina, prezzo e le proprie eventuali intolleranze alimentari.

Il programma offre inoltre altre funzionalità come la ricerca dei migliori ristoranti che rispettano i vincoli inseriti che l'utente può inserire all'interno del proprio percorso.

### 1.3 Descrizione della rilevanza gestionale del problema

Il programma che si vuole implementare si sviluppa intorno alla creazione di una sequenza di ristoranti che sia la migliore possibile per l'utente che lo utilizza. Il servizio quindi fornito dall'applicazione, che utilizza come sorgente dati un estratto delle informazioni presenti sulla nota piattaforma di viaggi TripAdvisor, aggiunge una funzionalità che ora il sito non offre.

### 1.4 Descrizione del data-set per la valutazione

Il dataset utilizzato è stato ottenuto raccogliendo le informazioni sui locali presenti sulla piattaforma di viaggi TripAdvisor. I dati riguardano più di 125000 ristoranti oltre ad informazioni sulla città, tipo di cucina, rating, recensioni e fascia di prezzo.

Il dataset è disponibile all'indirizzo: [www.kaggle.com/damienbeneschi/krakow-ta-restaurants-data-raw](https://www.kaggle.com/damienbeneschi/krakow-ta-restaurants-data-raw). Sono state apportate alcune piccole modifiche rispetto alla fonte originale.

## 1.5 Descrizione preliminare degli algoritmi coinvolti

Il software è realizzato in linguaggio Python sfruttando il pacchetto Flet per l'interfaccia grafica. La prima parte dell'applicazione sfrutta interrogazioni del database mediante query SQL mentre la seconda utilizza algoritmi più complessi. Il programma costruisce un grafo, servendosi del pacchetto Python NetworkX, e si serve di un algoritmo ricorsivo per calcolare l'itinerario.

## 1.6 Descrizione preliminare delle funzionalità previste per l'applicazione software

All'apertura del programma l'interfaccia grafica presenta due schede. La prima permette, impostando dei filtri, di effettuare delle interrogazioni del database ricavando i migliori dieci ristoranti che rispettino i vincoli inseriti. E' successivamente possibile aggiungere uno o più ristoranti appena trovati alla propria lista personale, la quale può essere opportunamente modificata in qualunque momento.

La seconda parte si occupa invece, sfruttando un algoritmo ricorsivo, di costruire un itinerario tra i ristoranti (di lunghezza impostata dall'utente) in modo tale da scegliere i migliori possibili. E' inoltre possibile specificare eventuali intolleranze alimentari e scegliere alcune tra le tante disponibili tipologie di cucina. L'applicazione si assicurerà di calcolare il percorso in modo da evitare due ristoranti consecutivi che presentino le stesse tipologie di cucina.

## Capitolo 2

# Descrizione del problema affrontato

### 2.1 Contesto

L'applicazione realizzata propone un utile strumento che può essere di aiuto agli utenti semplificando e filtrando le ricerche e la navigazione tra il vasto numero di informazioni disponibili sulla rete.

Il meccanismo delle recensioni online, in particolare, è diventato uno strumento essenziale per la valutazione di un ristorante da parte del pubblico, andando a costituire la colonna portante di tante piattaforme come TripAdvisor, fonte del data-set su cui si basa l'applicazione. Tuttavia, il grande numero di recensioni e di informazioni disponibili rischia di generare un effetto controproducente, ovvero confondere gli utenti senza riuscire ad far risaltare le indicazioni importanti.

Questa è la criticità che la prima parte dell'applicazione si propone di risolvere, rendendo disponibile uno strumento che consenta all'utente di selezionare le proprie preferenze, o di non selezionarne nessuna, scartando un'enorme gamma di informazioni che non vengono presentate.

La seconda parte dell'applicazione risolve, invece, la difficoltà di pianificare autonomamente un itinerario di ristoranti in cui recarsi semplicemente esprimendo le proprie preferenze.

### 2.2 Input e output

Durante la propria esecuzione, l'applicazione riceve in input due tipologie di indicazioni. Il data-set contenente tutte le informazioni dei ristoranti di trentuno città europee e, da parte dell'utente, il programma recepisce una grande serie di informazioni: preferenze di tipologia di cucina, la durata del soggiorno nella città scelta, la fascia di prezzo cercata ed eventualmente le intolleranze o abitudini alimentari dei clienti (diete vegetariane, celiache o halal).

L'output che è restituito si concentra sempre sul ricavare la miglior soluzione rispettando i vincoli inseriti, così nel caso della ricerca dei migliori ristoranti come nella ricerca dell'Itinerario gastronomico.

### 2.3 Potenzialità e criticità

Le potenzialità dell'applicazione si concentrano sulla semplificazione ed efficienza della navigazione dell'utente. Le criticità possono, invece, essere legate alla veridicità delle informazioni derivate dalle recensioni che, per loro natura, non garantiscono di essere sicuramente affidabili.

## Capitolo 3

# Descrizione del dataset

Il data-set utilizzato si compone di una tabella "ta\_restaurants\_curated" contenente informazioni riguardo ai ristoranti registrati su TripAdvisor di trentuno città europee, in totale sono presenti più di 120 mila ristoranti in tutta Europa. Il data-set è stato scaricato all'indirizzo: <https://www.kaggle.com/damienbeneschi/krakow-ta-restaurants-data-raw>. La tabella "ta\_restaurants\_curated" contiene, per ogni ristorante:

- **Nome del ristorante:** Il nome del ristorante.
- **Città:** La città in cui si trova il ristorante.
- **Tipologie di cucina:** Un elenco di tipologie di cucina offerte dal ristorante.
- **Ranking:** La posizione del ristorante nella classifica della propria città.
- **Rating:** Un numero tra 1 e 5 che rappresenta il valore medio delle recensioni date dagli utenti.
- **Range di prezzo:** Suddiviso in tre classi che misurano in modo approssimato quanto è caro il ristorante.
- **Numero di recensioni degli utenti:** Il numero totale di recensioni degli utenti.
- **Esempio di due recensioni:** Due recensioni prese come esempio.
- **URL TripAdvisor:** Il link URL collegato alla pagina di TripAdvisor.com.
- **ID univoco:** Un ID univoco per ogni ristorante.

## Capitolo 4

# Descrizione algoritmi utilizzati

### 4.1 Pattern MVC e DAO

Il software è stato realizzato in Python seguendo il pattern MVC (*Model, View, Controller*):

- *View*: la view si occupa della rappresentazione grafica delle informazioni ricevute dal controller.
- *Controller*: il controller gestisce l'interazione tra gli input dell'utente ottenuti tramite la view e la logica del programma interrogando il model.
- *Model*: il model gestisce la logica applicativa del programma, l'accesso ai dati e l'elaborazione

Questo garantisce la separazione delle funzionalità di ciascuna classe aumentando la scalabilità del codice.

L'interazione con il database è invece affidata al pattern DAO (*Data Access Object*) che è utilizzato per fornire un'interfaccia unificata per l'accesso ai dati, in modo che l'applicazione non debba preoccuparsi dell'estrazione dei dati

### 4.2 Algoritmi principali utilizzati

#### 4.2.1 Compilazione automatica dei parametri

All'avvio dell'applicazione l'utente è tenuto ad effettuare tramite dei menu a tendina le prime due scelte: una città (all'interno delle trentuno che sono presenti nel data-set) e una fascia di prezzo (è anche possibile scegliere di non specificare una fascia di prezzo in particolare). Una volta selezionate le scelte, si compiranno automaticamente i campi di testo facenti riferimento al minimo e massimo Rating disponibile per la combinazione di città e fascia di prezzo scelta. Questi due valori sono estratti dal database tramite la seguente query SQL parametrica:

```
1 select min(t.Rating),max(t.Rating)
2 from ta_restaurants_curated t
3 where t.City =%s
4 and t.Price_Range = %s
5 and t.Rating >0
6 and not isnull(t.Number_of_Reviews)
```

Listing 4.1: Query SQL per estrarre il minimo e massimo Rating



All'utente è comunque lasciata la possibilità di compilare i campi manualmente e il programma, nel caso sia inserito un input non consentito, creerà un messaggio di errore.

Uguualmente, il menu a tendina che permette di selezionare una tipologia di cucina (oppure di sceglierne qualsiasi) sarà automaticamente popolato nel momento in cui l'utente modifica i parametri precedentemente citati; anche una modifica del Rating può comportare l'eliminazione o l'aggiunta di una o più tipologie di cucina che è possibile selezionare. La query SQL sfruttata è la seguente:

```

1 select Cuisine_Style
2 from ta_restaurants_curated t
3 where City =%s
4 and Price_Range = %s
5 and Rating >=%s
6 and Rating <=%s
7 and not isnull(t.Number_of_Reviews)
8 and not (Cuisine_Style like "%Vegan Options%" or Cuisine_Style like "%Vegetarian
    Friendly%" or Cuisine_Style like "%Gluten Free Options%")

```

Listing 4.2: Query SQL per estrarre le tipologie di cucina

## 4.2.2 Salvataggio delle scelte

Ogni volta che l'utente effettua una scelta da un menu a tendina o digita del testo negli appositi campi, l'applicazione, grazie al *Controller*, memorizza gli input in variabili di classe che sono utilizzate per interagire con il *Model*. Il codice Python è simile per ognuno dei parametri di input:

```

1 def getSelectedPrezzo(self,e):
2     if e.control.data is None:
3         pass
4     else:
5         self.selectedPrezzo=e.control.data
6
7         self.view.ddPrezzo_R.value=self.selectedPrezzo
8
9         if self.selectedCitta is not None:
10            self.fillRating(self.selectedCitta, self.selectedPrezzo)

```

Listing 4.3: Codice per memorizzare input dell'utente

La funzione *getSelectedPrezzo* viene chiamata dal codice quando l'utente seleziona dal menu una fascia di prezzo, andando così a modificare la variabile *selectedPrezzo* del *Controller*. Le ultime due righe, inoltre, verificano che se l'utente abbia già selezionato la città e, nel caso sia già stata scelta, esegue la funzione *fillRating* che, come precedentemente spiegato, compila automaticamente i campi relativi ai minimo e massimo *Rating*.

## 4.2.3 Ricerca dei migliori ristoranti

Una volta effettuate le scelte, l'utente può premere sul bottone *Trova Migliori* comandando all'applicazione di stampare a video i migliori dieci ristoranti che rispettano i vincoli inseriti. Anche quest'ultima ricerca è stata eseguita tramite una query SQL, il codice del *Model* risulta quindi molto semplice e rapido:

```

1 def getTopDieci(self,citta,prezzo,min,max,cucina):
2     res=DAO.getTopDieciDAO(citta,prezzo,min,max,cucina)
3     return res

```

Listing 4.4: Codice per ricavare migliori dieci ristoranti

#### 4.2.4 Creazione del grafo

Alla pressione del bottone *Calcola*, una volta che le scelte sono già state effettuate, l'applicazione effettua una query per estrarre dal database i ristoranti filtrati secondo i vincoli inseriti dall'utente, eliminando quelli che hanno un fattore, dato dal prodotto di *Rating* e numero di recensioni, inferiore alla media. Grazie a questa cernita il numero di operazioni che il programma deve svolgere si riduce notevolmente eliminando i ristoranti che, in ogni caso, non sarebbero rientrati nell'itinerario migliore.

Una volta trovati gli effettivi nodi del grafo, il programma corregge il *Rating* di ciascuno di questi tenendo in considerazione del numero di recensioni ricevute: la correzione è effettuata tramite il calcolo di un valore da sommare al *Rating*. Inizialmente ogni ristorante è assegnato all'interno di una classe in base al numero di recensioni ottenute:

```

1 def assegnaClasse(self, r):
2     if r.Number_of_Reviews < 5:
3         r.classe=0
4     elif r.Number_of_Reviews >=5 and r.Number_of_Reviews<30:
5         r.classe=1
6     elif r.Number_of_Reviews>=30 and r.Number_of_Reviews<100:
7         r.classe=2
8     elif r.Number_of_Reviews>=100 and r.Number_of_Reviews<500:
9         r.classe=3
10    elif r.Number_of_Reviews>=500:
11        r.classe=4

```

Listing 4.5: funzione che assegna classe ad un ristorante in base al numero di recensioni

Successivamente è calcolato il termine additivo e modificato il *Rating* del ristorante (*r*):

$$\text{deltaRating} = r.\text{Rating} - 3 \quad (4.1)$$

$$\text{correzione} = \text{deltaRating} \times r.\text{classe} \times 0.1 \quad (4.2)$$

$$r.\text{Rating} = r.\text{Rating} + \text{correzione} \quad (4.3)$$

La creazione del grafo è uno dei passaggi più delicati siccome da questa dipende la parte centrale dell'applicazione, ovvero la ricerca dell'itinerario. La funzione *creaGrafo* è responsabile, servendosi della libreria *NetworkX*, della creazione del grafo. Inizialmente viene creato il grafo (non orientato e non pesato) e, successivamente, sono aggiunti i nodi precedentemente ricavati tramite il metodo di *NetworkX* *add\_nodes\_from*.

Dal momento che l'itinerario cercato non può avere due ristoranti considerati simili in due giorni consecutivi, due nodi sono collegati da un arco se almeno uno dei due ristoranti ha più di metà delle sue tipologie di cucina diverse dall'altro. In questo modo è possibile riconoscere due ristoranti come differenti nonostante presentino un numero limitato di tipologie di cucina in comune.

Infine, con l'ultima riga del codice sottostante, gli archi sono aggiunti al grafo tramite il metodo di *NetworkX* *add\_edges\_from*.

```

1 def creaGrafo(self, nodi):
2     self.grafo=nx.Graph()
3     self.grafo.add_nodes_from(nodi)
4     archi=[]
5     for i in range(len(nodi)):
6         for j in range(i+1, len(nodi)):
7             a=nodi[i]
8             b=nodi[j]
9
10            d=len(a.setCucine-b.setCucine)
11            e=len(b.setCucine-a.setCucine)

```

```

12
13         if d>math.ceil(len(a.setCucine)/2) or e>math.ceil(len(b.setCucine)/2):
14             daAggiungere=True
15         else:
16             daAggiungere=False
17
18         if daAggiungere:
19             archi.append((a,b))
20
21     self.grafo.add_edges_from(archi)

```

Listing 4.6: Funzione che crea il grafo

#### 4.2.5 Algoritmo di ricorsione

L'algoritmo ricorsivo utilizzato dall'applicazione per la creazione dell'itinerario è il più complesso tra quelli descritti; nonostante averne già scartati una parte, come precedentemente descritto, il numero di ristoranti eleggibili nel percorso può essere molto alto quindi, inevitabilmente, il tempo di calcolo del programma può subire grossi aumenti.

```

1 def ricorsione(self, parziale, ultimo,giorni):
2
3     ammissibili = self.getAmmissibili(parziale,ultimo,giorni)
4
5     if self.isTerminale(parziale,ammissibili):
6         c=self.calcolaPuntaggio(parziale)
7
8         if c> self.maxPunteggio:
9             print(c)
10            self.solBest=copy.deepcopy(parziale)
11            self.maxPunteggio=c
12            return
13     else:
14         for a in ammissibili:
15             parziale.append(a)
16             self.ricorsione(parziale,a,giorni)
17             parziale.pop()

```

Listing 4.7: Ricorsione per ricavare migliore itinerario

Inizialmente sono trovati i nodi ammissibili tramite la funzione *getAmmissibili* che restituisce i successori del nodo di partenza che non siano già presenti nella soluzione parziale e che abbiano un *Rating* minore, in modo da ottenere il miglior ristorante all'inizio del percorso:

```

1 def getAmmissibili(self, parziale, ultimo,giorni):
2     amm=[]
3     if len(parziale)==giorni:
4         return amm
5     else:
6         vic= list(self.grafo.neighbors(ultimo))
7         for a in vic:
8             if a not in parziale:
9                 if ultimo.Ranking < a.Ranking:
10                     amm.append(a)
11     return amm

```

Listing 4.8: Funzione che restituisce i nodi ammissibili nella soluzione parziale

Successivamente se la soluzione parziale trovata è terminale, ovvero non ci sono nodi ammissibili, viene calcolato il punteggio, dato dalla somma dei *Rating*:

```
1 def calcolaPuntaggio(self, parziale):  
2     somma=0  
3     for i in parziale:  
4         somma+=i.Rating  
5     return somma
```

Listing 4.9: Funzione per il calcolo del punteggio

Se il punteggio è maggiore del massimo precedentemente trovato, la soluzione è memorizzata come la migliore.

Se la soluzione non è terminale, il programma continua ad aggiungere ristoranti alla lista tra gli ammissibili, eseguendo una ricorsione ad ogni aggiunta di un nodo fino al raggiungimento del numero massimo di ristoranti concesso, pari al numero di giorni che l'utente imposta.

Il comando *parziale.pop* presente all'ultima riga della ricorsione (4.2.5) garantisce il *backtracking* dell'algoritmo per poter esplorare sistematicamente tutte le possibilità di soluzione.

## Capitolo 5

# Diagramma delle classi

Il *Model* e il *Controller* sono le classi principali.

Model
+grafo: nx.Graph +aumentareCucine: bool +listaUtente: list +cucine_R: dict +solBest: list +maxPunteggio: int +maxDaCercare: int
+getAllCitta() +getAllPrezzi() +getMinMaxRating(citta, prezzo) +getCucine(citta, prezzo, min, max) +esisteRistorante(citta, prezzo, min, max) +getTopDieci(citta, prezzo, min, max, cucina) +aggiungiRistorante(ristorante) +rimuoviRistorante(ristorante) +svuotaLista() +checkCucina(nome, valore) +calcola(citta, prezzo, gorni, veg, cel, hal) +creaGrafo() +assegnaClasse(ristorante) +ricorsione(parziale, ultimo, giorni) +getAmmissibili(parziale, ultimo, giorni) +isTerminale(parziale, ammissibili) +calcolaPunteggio(parziale)

(a) Model

Controller
+halal: bool +vegetariano: bool +celiaco: bool +view: View +model: Model +selectedCitta: String +selectedPrezzo: String +selectedCucina: String +selectedRistorante: Ristorante +selectedDaLista: Ristorante +minRating: int +maxRating: int +minOk: bool +maxOk: bool +link: String
+fillDDIniziali() +fillRating(citta, prezzo) +fillDDCucine(min, max) +fillDDLista(risto, bool) +handleTopDieci(e) +handleAggiungi(e) +handleRimuovi(e) +handleSvuota(e) +handleCalcola(e) +getSelectedCitta(e) +getSelectedPrezzo(e) +getSelectedCucina(e) +getSelectedMinRating(e) +getSelectedMaxRating(e) +getSelectedRistorante(e) +getSelectedDaLista(e) +open_link(e) +handlecheckCucina(e) +getVegetariano(e) +getCeliaco(e) +getHalal(e) +stampa(e)

(b) Controller

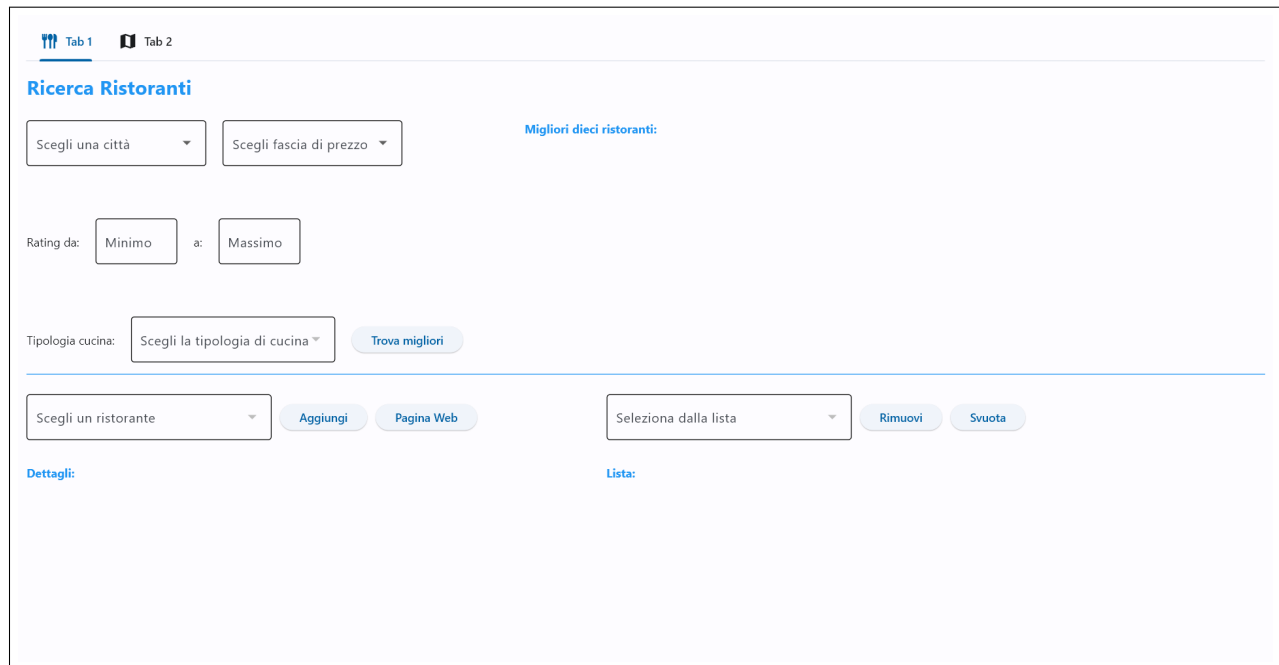
Figura 5.1: Diagramma delle classi

## Capitolo 6

# Videate dell'applicazione

The screenshot displays the initial interface of a restaurant search application. At the top, there are two tabs: 'Tab 1' (active) and 'Tab 2'. Below the tabs, the title 'Ricerca Ristoranti' is shown. The main area contains several search filters: a dropdown for 'Scegli una città', a dropdown for 'Scegli fascia di prezzo', a rating range selector with 'Minimo' and 'Massimo' inputs, and a dropdown for 'Scegli la tipologia di cucina' with a 'Trova migliori' button. Below these filters, there is a section for managing the search results, including a dropdown for 'Scegli un ristorante', buttons for 'Aggiungi' and 'Pagina Web', a dropdown for 'Seleziona dalla lista', and buttons for 'Rimuovi' and 'Svuota'. The bottom of the screen features two labels: 'Dettagli:' on the left and 'Lista:' on the right, indicating the sections for detailed information and the list of results.

Figura 6.1: Prima schermata all'avvio dell'applicazione



Tab 1 Tab 2

## Ricerca Ristoranti

Scegli una città ▼ Scegli fascia di prezzo ▼

Rating da: Minimo a: Massimo

Tipologia cucina: Scegli la tipologia di cucina ▼ Trova migliori

---

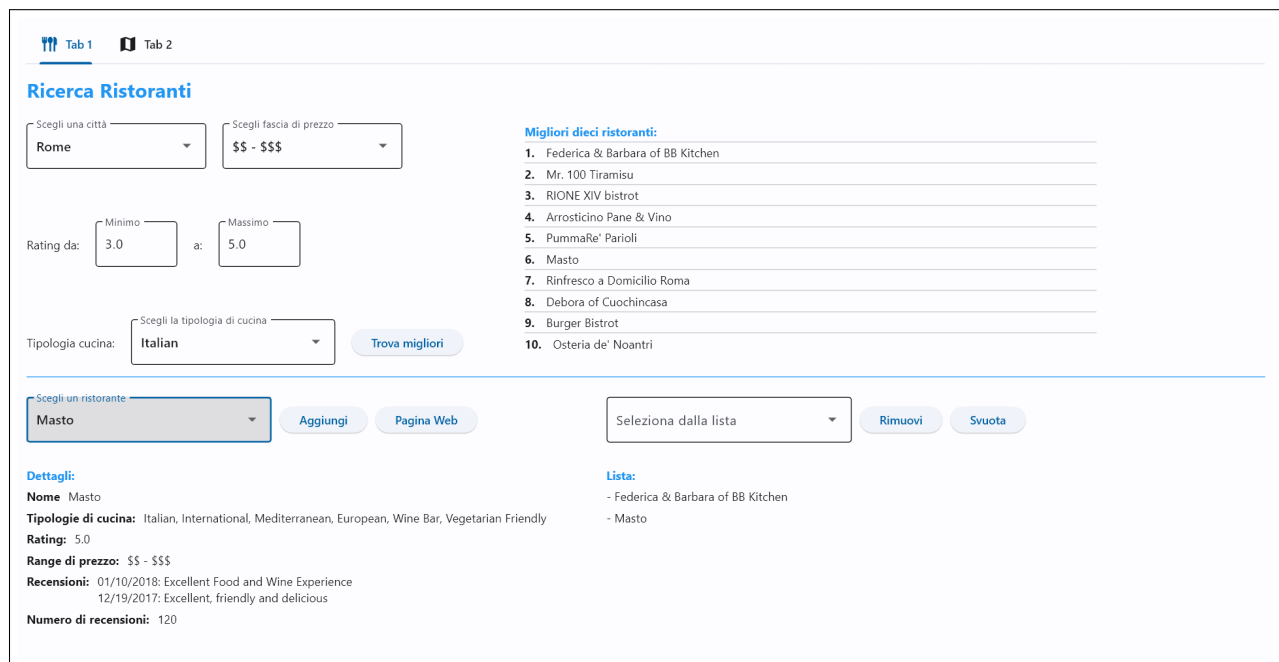
Scegli un ristorante ▼ Aggiungi Pagina Web Seleziona dalla lista ▼ Rimuovi Svuota

**Dettagli:**

**Migliori dieci ristoranti:**

**Lista:**

Figura 6.2: Seconda schermata all'avvio dell'applicazione



Tab 1 Tab 2

## Ricerca Ristoranti

Scegli una città ▼ Scegli fascia di prezzo ▼

Rome \$\$\$ - \$\$\$

Rating da: Minimo 3.0 a: Massimo 5.0

Tipologia cucina: Scegli la tipologia di cucina ▼ Italian Trova migliori

---

Scegli un ristorante ▼ Aggiungi Pagina Web Seleziona dalla lista ▼ Rimuovi Svuota

**Dettagli:**

**Nome:** Masto

**Tipologie di cucina:** Italian, International, Mediterranean, European, Wine Bar, Vegetarian Friendly

**Rating:** 5.0

**Range di prezzo:** \$\$ - \$\$\$

**Recensioni:** 01/10/2018: Excellent Food and Wine Experience  
12/19/2017: Excellent, friendly and delicious

**Numero di recensioni:** 120

**Migliori dieci ristoranti:**

1. Federica & Barbara of BB Kitchen
2. Mr. 100 Tiramisu
3. RIONE XIV bistrot
4. Arrostitino Pane & Vino
5. PummaRe' Parioli
6. Masto
7. Rinfresco a Domicilio Roma
8. Debora of Cuochincasa
9. Burger Bistrot
10. Osteria de' Noantri

**Lista:**

- Federica & Barbara of BB Kitchen
- Masto

Figura 6.3: Prima schermata dopo utilizzo dell'utente

Tab 1

Tab 2

### Itinerario

Città

Rome

Prezzo

\$\$ - \$\$\$

Numero giorni:

5

☒ Vegetariano

☐ Celiaco

☐ Halal

☐ European

☐ Fast Food

☐ French

☐ Fusion

☐ Gastropub

☐ German

☐ Greek

☒ Grill

☐ Healthy

Calcola

**Tipologie di cucina:** Italian, Pizza, Mediterranean, Barbecue, Grill, Vegetarian Friendly

**Rating:** 4.95

Figura 6.4: Seconda schermata dopo utilizzo dell'utente



## Capitolo 7

# Risultati sperimentali

### 7.1 Utilizzo di SQL

L'utilizzo di SQL è molto vantaggioso: è infatti evidente come si riduca, quasi dimezzandosi, il tempo impegnato dallo svolgimento della query rispetto al corrispondente codice in Python utile per raggiungere lo stesso obiettivo. Il grafico sottostante rappresenta la relazione tra tutte le possibili combinazioni di città e fascia di prezzo e il tempo impiegato rispettivamente da SQL (in viola) e da Python (in verde) per ricavare il risultato.

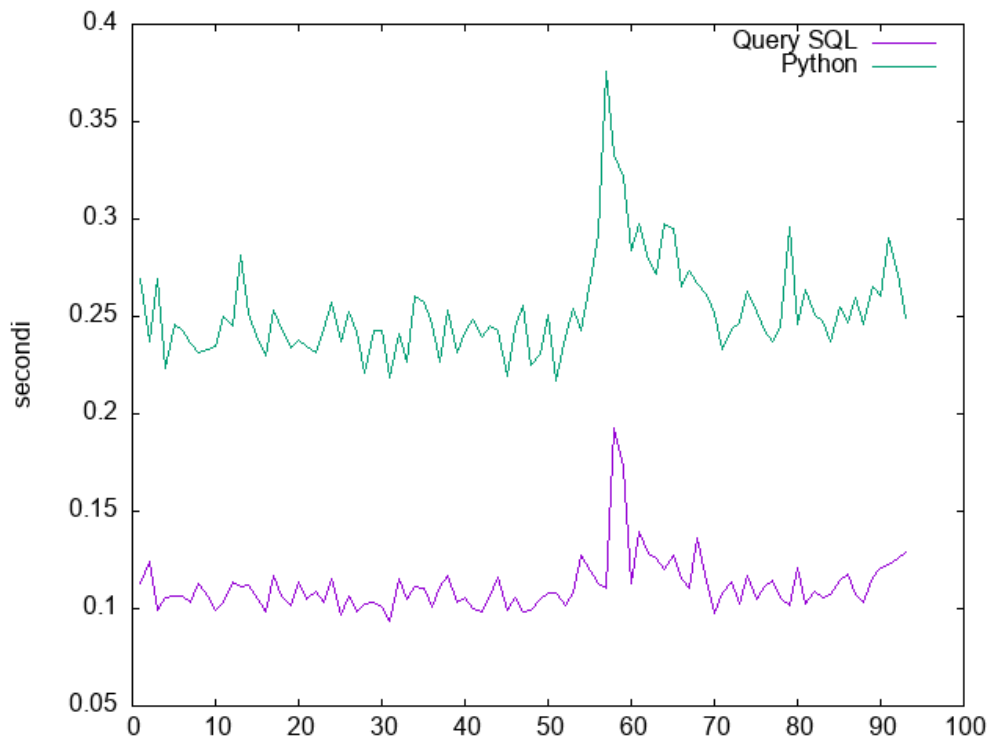


Figura 7.1: Grafico tempi SQL-Python

## 7.2 Algoritmo ricorsivo

La ricorsione è, a livello di complessità computazionale, sicuramente l'algoritmo che ha un impatto maggiore sull'esecuzione del programma. Se il numero di dati cresce, di conseguenza si verificherà un sensibile aumento dei tempi impiegati dall'algoritmo ricorsivo. Il grafico seguente evidenzia il tempo (in secondi) impiegato dall'applicazione per l'esecuzione del codice, per ognuna delle tre fasce di prezzo. Il test è stato effettuato richiedendo un itinerario di cinque giorni, scegliendo casualmente tre tipologie di cucina tra quelle disponibili.

Come è possibile osservare dal grafico, la seconda fascia di prezzo, quella che comprende la maggior parte dei ristoranti, può subire grandi aumenti (ad esempio per le città di Bruxelles e Madrid) del tempo di esecuzione a differenza delle altre due che rimangono circa costanti.

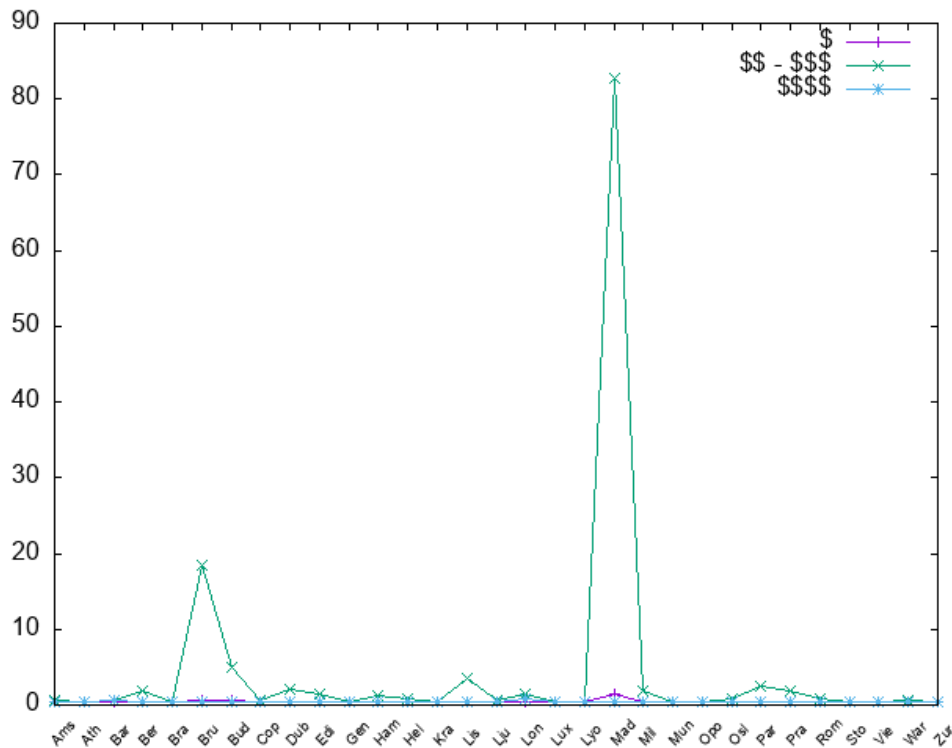


Figura 7.2: Grafico tempi di ricorsione

## Capitolo 8

# Valutazioni dei risultati ottenuti

### 8.1 Punti di forza

Nel complesso, l'applicazione, grazie ad un interfaccia grafica intuitiva, risulta semplice da utilizzare ed è in grado di soddisfare le richieste dell'utente in tempi ragionevolmente brevi.

Un altro aspetto importante per il corretto funzionamento è la robustezza che caratterizza l'interazione con il data-set e con gli input dell'utente. Il programma, infatti, effettua un elevato numero di controlli sugli input ricevuti, evitando così di generare degli errori comportando l'arresto dell'esecuzione.

### 8.2 Punti di debolezza

Le criticità possono sorgere nel momento in cui l'utente selezioni particolari combinazioni di input che obbligano il programma ad aumentare esponenzialmente le operazioni da svolgere, rischiando di completare l'esecuzione in tempi elevati.

### 8.3 Conclusioni

Gli obiettivi del progetto sono, in conclusione, stati raggiunti: l'applicazione fornisce un utile strumento che semplifica la navigazione offrendo delle soluzioni personalizzate per ciascun utente.

# Capitolo 9

## Licenza



Questo documento è condiviso con licenza Creative Commons BY-NC-SA 4.0.

Tu sei libero di:

- Condividere — riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare questo materiale con qualsiasi mezzo e formato
- Modificare — remixare, trasformare il materiale e basarti su di esso per le tue opere

Il licenziante non può revocare questi diritti fintanto che tu rispetti i termini della licenza.

Alle seguenti condizioni:

- Attribuzione — Devi riconoscere una menzione di paternità adeguata , fornire un link alla licenza e indicare se sono state effettuate delle modifiche . Puoi fare ciò in qualsiasi maniera ragionevole possibile, ma non con modalità tali da suggerire che il licenziante avalli te o il tuo utilizzo del materiale.
- NonCommerciale — Non puoi utilizzare il materiale per scopi commerciali.
- StessaLicenza — Se remixi, trasformi il materiale o ti basi su di esso, devi distribuire i tuoi contributi con la stessa licenza del materiale originario.
- Divieto di restrizioni aggiuntive — Non puoi applicare termini legali o misure tecnologiche che impongano ad altri soggetti dei vincoli giuridici su quanto la licenza consente loro di fare.

Note:

Non sei tenuto a rispettare i termini della licenza per quelle componenti del materiale che siano in pubblico dominio o nei casi in cui il tuo utilizzo sia consentito da una eccezione o limitazione prevista dalla legge.

Non sono fornite garanzie. La licenza può non conferirti tutte le autorizzazioni necessarie per l'utilizzo che ti prefiggi. Ad esempio, diritti di terzi come i diritti all'immagine, alla riservatezza e i diritti morali potrebbero restringere gli usi che ti prefiggi sul materiale.

Per una copia della licenza completa, visita <https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode>