

UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA



*Corso di Laurea Magistrale in
Ingegneria Informatica e dell'Automazione*

Corso di Data Science

Implementazione di un chatbot tramite framework Rasa per la gestione finanziaria personale

Studenti:

Angelone Mattia Domenico
Romanelli Marco
Giannelli Edoardo
Ali Waqar Badar

Anno Accademico 2022-2023

Chatbot

Indice

1. Introduzione.....	3
2. Rasa	3
2.1 Intent.....	4
2.2 Storie.....	4
2.3 Pipeline	4
2.4 Action	5
3. Dataset	6
4. Implementazione	8
4.1 Pipeline	8
4.2 Intent.....	9
4.3 Storie.....	13
4.4 Action	16
5. Esempio di utilizzo	21

1. Introduzione

Nell'era digitale in cui viviamo, l'interazione umana con le macchine sta diventando sempre più comune e naturale. Uno dei modi in cui questa interazione sta evolvendo è attraverso i chatbot, che sono diventati una parte integrante delle esperienze utente su piattaforme web e applicazioni mobili. I chatbot offrono un'interfaccia conversazionale che permette agli utenti di interagire con sistemi automatizzati attraverso il linguaggio naturale, fornendo risposte immediate alle loro domande e richieste.

L'obiettivo di questo lavoro è la realizzazione di un chatbot personale per la gestione delle proprie finanze con il quale si può tenere traccia degli acquisti effettuati, si possono eliminare acquisti fatti in precedenza e si può chiedere informazioni riguardo le proprie spese.

2. Rasa

RASA è una delle piattaforme di sviluppo di chatbot open-source più popolari e potenti disponibili oggi. Fondata nel 2016, RASA utilizza tecniche di intelligenza artificiale (IA) e apprendimento automatico per creare chatbot altamente personalizzabili e scalabili. L'obiettivo principale di RASA è quello di fornire agli sviluppatori gli strumenti necessari per creare chatbot sofisticati, in grado di comprendere il contesto, gestire conversazioni complesse e fornire risposte accurate ed efficaci agli utenti.

Nella presente relazione, esploreremo in dettaglio il concetto di chatbot e approfondiremo le funzionalità e i vantaggi offerti dalla piattaforma RASA. Esamineremo i principali elementi costitutivi di un chatbot RASA, tra cui l'elaborazione del linguaggio naturale (NLP), il machine learning e la gestione dei dialoghi. Analizzeremo anche alcuni esempi concreti di applicazioni dei chatbot RASA in diversi settori, come il customer service, il marketing e l'assistenza sanitaria, per comprendere appieno il loro impatto e il loro potenziale.

RASA è composto da 2 moduli che lavorano insieme per creare e gestire un chatbot:

- **RASA NLU (Natural Language Understanding):** Questo modulo si occupa dell'elaborazione del linguaggio naturale. Utilizza algoritmi di machine learning per analizzare e comprendere il testo degli utenti, identificando gli intenti (le intenzioni dietro le frasi degli utenti) e le entità (informazioni specifiche come nomi, date, luoghi, ecc.). RASA NLU permette di addestrare il modello per riconoscere in modo accurato gli intenti e le entità specifiche del dominio del chatbot.
- **RASA Core:** Questo modulo gestisce il flusso di conversazione e la logica del chatbot. Utilizzando tecniche di machine learning basate su algoritmi di apprendimento di rinforzo, RASA Core permette di creare dialoghi dinamici e personalizzati. I modelli di RASA Core possono essere addestrati per comprendere il contesto delle conversazioni precedenti, prendere decisioni e generare risposte coerenti e appropriate.

I vari componenti con i quali si codifica un chatbot tramite Rasa sono specificati nelle seguenti sottosezioni.

2.1 Intent

Gli intenti sono fondamentali per comprendere il significato di una frase e la finalità della richiesta dell'utente. Quando il chatbot analizza un messaggio in ingresso, cerca di identificarne l'intento corrispondente tra quelli definiti nel file "nlu.yml". Affinché il modello possa essere addestrato correttamente, è necessario fornire diversi esempi per ciascun intento. Durante la classificazione del messaggio, viene assegnato un punteggio a ogni intento e viene selezionato quello con il punteggio più alto. È possibile visualizzare i punteggi associati agli intenti per ogni frase di input utilizzando il comando "\$rasa shell nlu".

2.2 Storie

Le storie rappresentano possibili scenari di conversazione tra l'utente e il chatbot. Per insegnare all'interprete come agire in risposta alle richieste dell'utente, è necessario definire diverse conversazioni che includano intenti e azioni. In base alla classificazione dell'intento, viene selezionata una delle storie presenti nel file "stories.yml".

2.3 Pipeline

La pipeline NLU (Natural Language Understanding) è composta da una serie di componenti che elaborano il testo in ingresso per consentire la classificazione degli intenti e l'estrazione di eventuali entità. Nel progetto descritto, la pipeline è composta dai seguenti componenti:

- **WhitespaceTokenizer:** Suddivide il testo in token separati da spazi bianchi.
- **RegexFeaturizer:** Estrae entità e classifica gli intenti. Durante l'addestramento, vengono definite espressioni regolari che indicano la presenza o l'assenza di determinati pattern nel messaggio dell'utente. Queste funzioni vengono utilizzate per classificare gli intenti e estrarre le entità.
- **LexicalSyntacticFeaturizer:** Estrae entità utilizzando caratteristiche lessicali e sintattiche. Analizza ogni token nel messaggio dell'utente e crea caratteristiche basate sulla configurazione specificata.
- **CountVectorsFeaturizer:** Crea una rappresentazione vettoriale del messaggio, dell'intento e della risposta dell'utente utilizzando CountVectorizer di sklearn. Questo componente considera anche i token composti solo da cifre come una singola caratteristica.
- **DIETClassifier:** Utilizza l'architettura DIET (Dual Intent and Entity Transformer) per la classificazione degli intenti e il riconoscimento delle entità. Viene addestrato per un numero specificato di epoche e può essere configurato per una migliore generalizzazione del modello ai set di test.

- EntitySynonymMapper: Mappa i sinonimi delle entità per garantire una corretta identificazione delle entità stesse.
- ResponseSelector: Seleziona la risposta del chatbot tra una serie di risposte candidate.
- FallbackClassifier: Classifica un messaggio utente come "nlu_fallback" se il classificatore precedente non riesce a classificare un'intento con una confidenza superiore alla soglia del FallbackClassifier.

2.4 Action

Dopo ogni messaggio dell'utente, il modello del chatbot prevede un'azione che l'assistente virtuale dovrebbe eseguire successivamente. Le azioni possono essere di due tipi:

- Utterances Action: Sono frasi predefinite utilizzate in determinate situazioni per inviare messaggi, immagini, pulsanti o altri tipi di risposte all'utente. Sono definite nel file "domain.py" sotto la voce "responses" e hanno la forma "utter_<nome>".
- Custom Action: Sono azioni personalizzate che consentono di eseguire del codice. Le azioni personalizzate sono definite nel file "actions.py" e possono essere utilizzate per gestire situazioni che richiedono interazioni complesse, come chiamate API, calcoli o interrogazioni di database.

Quando il chatbot prevede un'azione personalizzata, viene inviata una richiesta POST al server delle azioni con i dettagli dell'azione prevista, l'ID della conversazione, il contenuto del tracker e il contenuto del dominio. Dopo l'esecuzione dell'azione personalizzata, il server delle azioni restituisce una risposta al chatbot, che a sua volta invia la risposta all'utente e aggiorna il tracker della conversazione con gli eventi rilevanti.

Le azioni personalizzate sono implementate come classi Python e possono utilizzare metodi come "tracker.get_slot()" per ottenere il valore di uno slot, "tracker.latest_message" per accedere all'ultimo messaggio dell'utente e "dispatcher.utter_message()" per inviare risposte all'utente. Per consentire l'esecuzione delle azioni personalizzate, è necessario avviare il server delle azioni utilizzando il comando "\$rasa run action".

3. Dataset

Per poter portare a termine il lavoro è stato creato un file csv che simula un database dove sono presenti tutti gli acquisti effettuati dall'utente.

giorno	mese	anno	oggetto	tipologia	prezzo	data
1	5	2023	Smartphone	Elettronica	500	01/05/2023
2	5	2023	Tastiera	Elettronica	80	02/05/2023
3	5	2023	Cuffie	Elettronica	100	03/05/2023
4	5	2023	Libro	Cultura	20	04/05/2023
5	5	2023	Orologio	Accessori	200	05/05/2023
6	5	2023	Scarpe	Abbigliamento	70	06/05/2023
7	5	2023	Giacca	Abbigliamento	150	07/05/2023
8	5	2023	Laptop	Elettronica	1000	08/05/2023
9	5	2023	Borsa	Accessori	50	09/05/2023
10	5	2023	Televisore	Elettronica	800	10/05/2023
11	5	2023	Mouse	Elettronica	30	11/05/2023
12	5	2023	Tablet	Elettronica	300	12/05/2023
13	5	2023	Profumo	Bellezza	60	13/05/2023
14	5	2023	Gioielli	Accessori	250	14/05/2023
15	5	2023	Fotocamera	Elettronica	400	15/05/2023
16	5	2023	Maglietta	Abbigliamento	25	16/05/2023
17	5	2023	Pantaloni	Abbigliamento	60	17/05/2023
18	5	2023	Auricolari	Elettronica	50	18/05/2023
19	5	2023	Gioco da tavolo	Intrattenimento	30	19/05/2023
20	5	2023	Lampada	Casa	40	20/05/2023
21	5	2023	Giocattolo	Intrattenimento	15	21/05/2023

Il dataset rappresenta una tabella con informazioni su diversi oggetti, organizzate in colonne:

- "Giorno": rappresenta il giorno in cui è stato registrato l'oggetto.
- "Mese": indica il mese in cui è stato registrato l'oggetto.
- "Anno": rappresenta l'anno in cui è stato registrato l'oggetto.
- "Oggetto": specifica il nome dell'oggetto registrato.
- "Tipologia": indica la categoria o la tipologia di appartenenza dell'oggetto.
- "Prezzo": rappresenta il prezzo dell'oggetto registrato.
- "Data": indica la data completa di registrazione dell'oggetto.

Gli oggetti nel dataset sono vari e coprono diverse categorie:

- Elettronica: smartphone, tastiera, cuffie, laptop, tablet, fotocamera, mouse, auricolari, microfono, casse Bluetooth, stampante, macchina fotografica.
- Abbigliamento: scarpe, giacca, maglietta, pantaloni.
- Accessori: orologio, borsa, gioielli, portafoglio, occhiali da sole, zaino, braccialetto, binocolo.
- Cultura: libro.
- Intrattenimento: gioco da tavolo, puzzle, strumento musicale, videogiochi.
- Bellezza: profumo, olio essenziale.
- Casa: lampada, cassettera, vaso.
- Sport: palla da calcio.

Con questo file .csv il chatbot potrà aggiungere un nuovo acquisto effettuato da un utente, potrà eliminare un acquisto presente nel dataset e potrà estrapolare informazioni sulle spese fatte quando verrà richiesto dall'utente.

4. Implementazione

In questa sezione analizziamo l'implementazione del chatbot suddividendo la trattazione nelle sottosezioni: pipeline, intent, storie ed action.

4.1 Pipeline

Al fine di integrare i moduli specificati alla sezione 2.3, la pipeline presente nel file config.yml è stata implementata nel modo seguente:

```
# The config recipe.
# https://rasa.com/docs/rasa/model-configuration/
recipe: default.v1

# Configuration for Rasa NLU.
# https://rasa.com/docs/rasa/nlu/components/
language: it

pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: DIETClassifier
  epochs: 100
  constrain_similarities: true
- name: EntitySynonymMapper
- name: ResponseSelector
  epochs: 100
  constrain_similarities: true
- name: FallbackClassifier
  threshold: 0.6
  ambiguity_threshold: 0.1
```


Nelle policies è stata definita una regola che definisce un'action di default eseguita dal chatbot in caso in cui non riesce a interpretare la richiesta fatta dall'utente.

```
# Configuration for Rasa Core.
# https://rasa.com/docs/rasa/core/policies/
policies:
|
- name: MemoizationPolicy
- name: RulePolicy
  core_fallback_threshold: 0.6
  core_fallback_action_name: "action_my_fallback"
  enable_fallback_prediction: true
- name: UnexpectEDIntentPolicy
  max_history: 5
  epochs: 100
- name: TEDPolicy
  max_history: 5
  epochs: 100
  constrain_similarities: true

assistant_id: 20230610-182456-gentle-dingo
```

4.2 Intent

Gli intent implementati permettono all'utente di poter inserire, eliminare e chiedere informazioni statistiche relativamente alle proprie finanze personali.

Di seguito, un elenco degli intent sviluppati con le rispettive implementazioni:

- **add:** Richiesta di poter registrare un nuovo acquisto:

```
- intent: add
examples: |
- aggiungi un [tablet](oggetto) in [elettronica](tipologia) acquistato il [25-5-2023](data) a [500](prezzo)
- per favore, aggiungi il [orologio](oggetto) che ho comprato il [25-5-2023](data) in [accessori](tipologia). Il prezzo è di [500](prezzo)
- ho appena acquistato una [giacca](oggetto) in [abbigliamento](tipologia) al prezzo di [500](prezzo) il [25-5-2023](data)
- [gioielli](oggetto) [accessori](tipologia) [500](prezzo) [25-5-2023](data)
- ho effettuato un acquisto di [abbigliamento](tipologia): un [pantaloni](oggetto) a [500](prezzo). Aggiungi questo oggetto al [25-5-2023](data)
- potresti inserire il [gioco da tavolo](oggetto) che ho appena acquistato al prezzo di [500](prezzo) oggi [25-5-2023](data) nella sezione [intrat
- aggiungi il [vaso](oggetto) in [casa](tipologia) che ho comprato il [25-5-2023](data) al prezzo di [500](prezzo) alla tabella degli acquisti?
```

- **delete:** Richiesta di poter cancellare un acquisto:

```
- intent: delete
examples: |
  - elimina l'acquisto [libro](oggetto)
  - rimuovi il [libro](oggetto)
  - per favore, rimuovi il [binocolo](oggetto) dagli acquisti
  - Ho bisogno di eliminare il [olio essenziale](oggetto) dagli acquisti. Potresti farlo?
  - Rimuovi dagli acquisti il [cassettiera](oggetto) che ho acquistato
  - Il [strumento musicale](oggetto) deve essere rimosso dagli acquisti. Puoi farlo?
  - Potresti eliminare il [tè](oggetto)
```

- **info_day:** Richiesta di informazioni sugli acquisti effettuati in una specifica data:

```
- intent: info_day
examples: |
  - dimmi cosa ho acquistato il [25-5-2023](data)
  - per favore puoi dirmi cosa ho comprato il [25-5-2023](data)?
  - Vorrei sapere quali sono gli articoli che ho comprato il [25-5-2023](data)
  - Potresti dirmi quali sono gli oggetti che ho acquistato il [25-5-2023](data)?
  - Mi piacerebbe conoscere gli acquisti effettuati il [25-5-2023](data). Puoi elencarli?
  - acquisti del [25-5-2023](data)
  - oggetti comprati il [25-5-2023](data)
```

- **info_stats_month:** Richiesta di informazioni statistiche sugli acquisti effettuati in un mese specifico:

```
- intent: info_stats_month
examples: |
  - Dimmi quanto ho speso nel mese di [Maggio](mese)
  - mostrami la spesa totale di [Giugno](mese)
  - puoi dirmi la spesa totale nel mese di [Luglio](mese)
  - quanto ho speso nel mese di [Aprile](mese)
  - fammi il calcolo di quanto ho speso a [Gennaio](mese)
  - calcola la mia spesa nel mese di [Febbraio](mese)
  - spesa di [Marzo](mese)
  - dammi informazioni sulle mie spese nel mese di [Ottobre](mese)
  - elencami la statistica nel mese di [Novembre](mese) delle mie spesa
  - puoi mostrarmi una panoramica dei miei acquisti nel mese di [Dicembre](mese)
  - Mi piacerebbe avere un resoconto delle mie spese nel mese di [Gennaio](mese)
  - Puoi darmi un'analisi delle mie spese nel mese di [Febbraio](mese)?
  - Dimmi quali sono le statistiche delle mie spese per il mese di [Marzo](mese)
  - Puoi dirmi quali soldi ho speso nel mese di [Aprile](mese)? Voglio sapere tutti i dettagli
  - Sarebbe possibile ottenere un riepilogo delle mie spese nel mese di [Maggio](mese)?
```

- **info_tipology:** Richiesta di informazioni sugli acquisti, relativamente a una tipologia specifica:

```
- intent: info_stats_month
examples: |
  - Dimmi quanto ho speso nel mese di [Maggio](mese)
  - mostrami la spesa totale di [Giugno](mese)
  - puoi dirmi la spesa totale nel mese di [Luglio](mese)
  - quanto ho speso nel mese di [Aprile](mese)
  - fammi il calcolo di quanto ho speso a [Gennaio](mese)
  - calcola la mia spesa nel mese di [Febbraio](mese)
  - spesa di [Marzo](mese)
  - dammi informazioni sulle mie spese nel mese di [Ottobre](mese)
  - elencami la statistica nel mese di [Novembre](mese) delle mie spesa
  - puoi mostrarmi una panoramica dei miei acquisti nel mese di [Dicembre](mese)
  - Mi piacerebbe avere un resoconto delle mie spese nel mese di [Gennaio](mese)
  - Puoi darmi un'analisi delle mie spese nel mese di [Febbraio](mese)?
  - Dimmi quali sono le statistiche delle mie spese per il mese di [Marzo](mese)
  - Puoi dirmi quali soldi ho speso nel mese di [Aprile](mese)? Voglio sapere tutti i dettagli
  - Sarebbe possibile ottenere un riepilogo delle mie spese nel mese di [Maggio](mese)?
```

- **info_stats:** Richiesta di informazioni statistiche sugli acquisti in generale:

```
- intent: info_stats
examples: |
  - dammi informazioni sulle mie spese
  - elencami la statistica delle mie spesa
  - puoi mostrarmi una panoramica dei miei acquisti
  - Mi piacerebbe avere un resoconto delle mie spese
  - Puoi darmi un'analisi delle mie spese?
  - Dimmi quali sono le statistiche delle mie spese
  - Puoi dirmi quali soldi ho speso? Voglio sapere tutti i dettagli
  - Sarebbe possibile ottenere un riepilogo delle mie spese?
```

- **info_range:** Richiesta di informazioni sugli acquisti tra due date specifiche:

```
- intent: info_range
examples: |
  - dimmi quanto ho speso nel periodo [1-3-2023/28-4-2023](range)
  - mostrami la spesa totale del [2-2-2023/3-2-2023](range)
  - puoi dirmi la spesa totale dal [6-4-2023/15-9-2025](range)
  - quanto ho speso dal [8-5-2023/4-8-2023](range)
  - fammi il calcolo di quanto ho speso dal [11-7-2023/3-1-2023](range)
  - calcola la mia spesa dal [18-9-2023/9-11-2023](range)
  - spesa dal [22-7-2023/16-8-2023](range)
```

- **help**: Richiesta di aiuto sulle azioni che posso essere effettuate:

```
- intent: help
examples: |
  - aiuto
  - aiutami
  - quali sono i comandi?
  - cosa posso chiederti?
  - cosa posso chiedere a te?
  - cosa posso fare?
  - cosa puoi dirmi?
  - quali cose posso chiederti?
  - cosa puoi fare?
  - quali informazioni puoi darmi?
  - comandi
```

- **curiosity**: Richiesta di curiosità in merito alla finanza:

```
- intent: curiosity
examples: |
  - dimmi qualcosa di interessante
  - sai qualche curiosità?
  - qualche curiosità
  - fatti interessanti
  - qual è un fatto interessante?
  - conosci qualche evento interessante?
  - conosci qualche bella curiosità?
  - dimmi una curiosità
```

4.3 Storie

Per insegnare all'interprete come agire in base alle nostre richieste, è stato necessario definire una almeno una conversazione per ogni intent; tutte le conversazioni sono introdotte da un intent standard di saluto (greet).

Di seguito le storie implementate rispettivamente per gli intent della sottosezione precedente:

- add:

```
- story: nuova spesa
  steps:
    - intent: greet
    - action: utter_greet
    - intent: add
    - action: action_add
```

- delete:

```
- story: eliminazione spesa
  steps:
    - intent: greet
    - action: utter_greet
    - intent: delete
    - action: action_delete
```

- info_day:

```
- story: info giorno
  steps:
    - intent: greet
    - action: utter_greet
    - intent: info_day
    - action: action_info_day
```

- info_stats_month:

```
- story: info statistiche mese
  steps:
    - intent: greet
    - action: utter_greet
    - intent: info_stats_month
    - action: action_stats_month
```

- info_tipology:

```
- story: info tipologia
  steps:
    - intent: greet
    - action: utter_greet
    - intent: info_tipology
    - action: action_info_tipology
```

- info_stats:

```
- story: info statistiche
  steps:
    - intent: greet
    - action: utter_greet
    - intent: info_stats
    - action: action_info_stats
```


- info_range:

```
- story: info periodo
  steps:
    - intent: greet
    - action: utter_greet
    - intent: info_range
    - action: action_info_range
```

- help:

```
- story: aiuto
  steps:
    - intent: greet
    - action: utter_greet
    - intent: help
    - action: utter_help
```

- curiosity:

```
- story: curiosità
  steps:
    - intent: greet
    - action: utter_greet
    - intent: curiosity
    - action: utter_curiosity
```

4.4 Action

Per gli intent elencati alla sezione 4.2 abbiamo implementato le seguenti action:

- **action_add:**

```
#Classe per definire l'azione di inserimento elementi nel file
class ActionAdd(Action):

    def name(self) -> Text:
        return "action_add"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        data = (str(tracker.get_slot('data'))).replace('-', '/')
        oggetto = str(tracker.get_slot('oggetto'))
        tipologia = str(tracker.get_slot('tipologia'))
        prezzo = str(tracker.get_slot('prezzo'))
        data_oggetto = datetime.strptime(data, "%d/%m/%Y")
        giorno = data_oggetto.day
        mese = data_oggetto.month
        anno = data_oggetto.year

        def inserisci_elemento( giorno, mese, anno, oggetto, tipologia, prezzo, data):
            dataframe = pd.read_csv('./File_Spesa.csv', delimiter = ',')
            nuovo_elemento = pd.DataFrame({'giorno': [giorno], 'mese': [mese], 'anno': [anno], 'oggetto': [oggetto],
            |   |   |   'tipologia': [tipologia], 'prezzo': [prezzo], 'data': [data]})
            pd.concat([dataframe, nuovo_elemento]).to_csv('./File_Spesa.csv', lineterminator='\n', index=False)
            #print("Nuovo elemento inserito correttamente.")

        inserisci_elemento(giorno, mese, anno, oggetto, tipologia, prezzo, data)

        output="Ho appena salvato {}, in : {} , prezzo : {}, data : {}.".format(oggetto, tipologia, prezzo, data)

        dispatcher.utter_message(text=output)
        return []
```

Innanzitutto è stata necessaria l'implementazione della funzione `spli_date()`, la quale dà la possibilità di inserire le date sia nella notazione `gg-mm-aaaa`, sia `gg/mm/aaaa`.

La classe, infine, inserisce, attraverso la funzione `inserisci_elemento()`, una nuova riga del dataset con i campi passati per argomento.

- **action_delete:**

```
#Classe per eliminare un acquisto
class ActionDelete(Action):

    def name(self) -> Text:
        return "action_delete"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        oggetto = str(tracker.get_slot('oggetto'))
        output = ""

        #Funzione per modificare il dataframe
        def delete_oggetto(oggetto):
            dataframe = pd.read_csv('./File_Spesa.csv', delimiter = ',')
            try:
                dataframe_filtrato = dataframe[dataframe['oggetto'] != oggetto]
                dataframe_filtrato.to_csv('./File_Spesa.csv', lineterminator='\n', index=False)
                return "Ho appena eliminato questo oggetto comprato: {}".format(oggetto)
            except:
                return "Oggetto non trovato."

        output = delete_oggetto(oggetto)

        dispatcher.utter_message(text=output)
        return []
```

L'azione `action_delete` si compone di una funzione `delete_oggetto()` che prende in input l'oggetto desiderato e lo elimina, filtrando e aggiornando il dataset per tutte le righe che non lo contengono.

- **action_info_day:**

```
class ActionInfoDay(Action):

    def name(self) -> Text:
        return "action_info_day"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        data = str(tracker.get_slot('data'))

        data = data.replace('-', '/')

        #Funzione per 'richiesta info su spesa su un dato giorno'
        def info_spesa_day(data):
            try:
                dataframe = pd.read_csv('./File_Spesa.csv', delimiter = ',')
                filtro = dataframe['data'] == data
                righe_selezionate = dataframe[filtro]

                if righe_selezionate.empty:
                    return "Non ci sono stati acquisti il giorno : {}".format(data)
                else:
                    somma_prezzi = righe_selezionate['prezzo'].sum()
                    oggetti = righe_selezionate['oggetto'].tolist()
                    oggetti_stringa = ', '.join(oggetti)
                    return "Il giorno {}, hai speso {} euro (anticamente si usavano le rupie), comprando : {}".format(data, somma_prezzi, oggetti_stringa)
            except:
                return "Non ci sono stati acquisti il giorno : {}".format(data)

        #concatena_con_backslash(giorno, mese, anno) # Esempio giorno da cercare

        output = info_spesa_day(data)
        dispatcher.utter_message(text=output)
        return []
```

Preso una data in input, che sia scritta con / o – come separatori, l'azione restituisce la somma degli oggetti acquistati in quella data, insieme a una stringa contenente la lista degli oggetti stessi.

- **action_info_tipology:**

```
#Classe per calcolo statistica per tipologia
class ActionInfoTipology(Action):

    def name(self) -> Text:
        return "action_info_tipology"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        tipologia = str(tracker.get_slot('tipologia'))
        output = ""

        #Funzione per modificare il dataframe
        def stats_tipology(tipologia):
            dataframe = pd.read_csv('./File_Spesa.csv', delimiter = ',')
            try:
                prezzo_totale = dataframe['prezzo'].sum()
                dataframe_filtrato = dataframe[dataframe['tipologia'] == tipologia]
                prezzo_tipologia = dataframe_filtrato['prezzo'].sum()
                prezzo_percentuale = (prezzo_tipologia / prezzo_totale) * 100

                return "Per la tipologia: {}, hai speso : {} , in percentuale : {}.".format(tipologia, prezzo_tipologia, prezzo_percentuale)
            except:
                return "C'è qualcosa che non va, non ho trovato la tipologia : {}.".format(tipologia)

        output = stats_tipology(tipologia)

        dispatcher.utter_message(text=output)
        return []
```

Questa azione si compone di un'unica funzione che calcola il prezzo totale e percentuale di tutti gli oggetti acquistati per qula tipologia passatagli per argomento.

- **action_info_stats:**

```
#Classe per calcolo statistica totale
class ActionInfoStats(Action):

    def name(self) -> Text:
        return "action_info_stats"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        output = ""

        #Funzione per modificare il dataframe
        def stats():
            dataframe = pd.read_csv('./File_Spesa.csv', delimiter = ',')
            try:
                prezzo_totale = dataframe['prezzo'].sum()
                prezzo_medio = dataframe['prezzo'].mean()
                spesa_per_categoria = dataframe.groupby('tipologia')['prezzo'].sum()
                # Trova le due categorie con la spesa totale più alta e le salva in una stringa
                categorie_top = spesa_per_categoria.nlargest(2)
                categorie_top_stringa = categorie_top.to_string()

                return "Il prezzo totale che hai sostenuto è di : {} , la media per oggetto acquistato è : {}, le due categorie con la spesa più alta sono: {}".format(prezzo_totale, prezzo_medio, categorie_top_stringa)
            except:
                return "C'è qualcosa che non va."

        output = stats()

        dispatcher.utter_message(text=output)
        return []
```

Anche questa funzione si compone di un'unica funzione, la quale calcola prezzo totale, prezzo medio e le due categorie in cui si è speso di più; non prende in input nulla, dal momento che realizza un'analisi statistica su tutto il dataset.

- **action_stats_month:**

```
#Classe per calcolo statistica per mese
class ActionStatsMonth(Action):

    def name(self) -> Text:
        return "action_stats_month"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        mese_testuale = str(tracker.get_slot('mese'))
        mesi = {"gennaio": 1, "febbraio": 2, "marzo": 3, "aprile": 4, "maggio": 5, "giugno": 6, "luglio": 7, "agosto": 8, "settembre": 9, "ottobre": 10, "novembre": 11, "dicembre": 12}
        output = ""

        try:
            mese_lower = mese_testuale.lower()
            mese_numerico = mesi[mese_lower]
        except:
            output = "C'è un errore nel mese"

        #Funzione per modificare il dataframe
        def stats_month():
            dataframe = pd.read_csv('./File_Spesa.csv', delimiter = ',')
            try:
                df_month = dataframe[dataframe['mese'] == mese_numerico]
                spesa_totale = df_month['prezzo'].sum()
                spesa_media = df_month['prezzo'].mean()
                # Calcola la spesa totale per categoria
                spesa_per_categoria = df_month.groupby('tipologia')['prezzo'].sum()
                # Trova le due categorie con la spesa totale più alta e le salva in una stringa
                categorie_top = spesa_per_categoria.nlargest(2)
                categorie_top_stringa = categorie_top.to_string()
                return "Il prezzo totale che hai sostenuto nel mese di {} è di {}, la media del mese è di {}. Le due categorie su cui hai speso di più sono : {}".format(mese_testuale, spesa_totale, spesa_media, categorie_top_stringa)
            except:
                return "C'è qualcosa che non va, forse non hai fatto spese nel mese di : {}".format(mese_testuale)

        output = stats_month()

        dispatcher.utter_message(text=output)
        return []
```

Al fine di rendere più naturale la scelta del mese di cui visualizzare le statistiche, la porzione di codice che precede la funzione stats_month() è finalizzata a trasformare il nome del mese scritto come parola dall'utente, in un numero.

Successivamente, l'esecuzione passa alla funzione stats_month(), la quale calcola la spesa totale, la spesa media e le due tipologie in cui si è speso di più nel mese selezionato.

- **action_info_range:**

```
#Classe per range temporale
class ActionStatsRange(Action):

    def name(self) -> Text:
        return "action_info_range"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        range = str(tracker.get_slot('range'))

        data_inizio, data_fine = range.split('/')
        data_inizio = data_inizio.replace('-', '/')
        data_fine = data_fine.replace('-', '/')

        output = ""

        def is_date_between(date, date1, date2):

            date1_obj = datetime.datetime.strptime(date1, "%d/%m/%Y")
            date2_obj = datetime.datetime.strptime(date2, "%d/%m/%Y")
            date_obj = datetime.datetime.strptime(date, "%d/%m/%Y")

            return date1_obj <= date_obj <= date2_obj

        def filter_rows_by_date(date1, date2):
            dataframe = pd.read_csv('./File_Spesa.csv', delimiter= ',')
            filtered_rows = []
            #print(type(dataframe))
            for index, row in dataframe.iterrows():
                date_str = row['data'].strip()

                if is_date_between(date_str, date1, date2):
                    filtered_rows.append(row)

            return filtered_rows
```

```
def sum_prices_and_get_items():

    try:
        filtered_rows = filter_rows_by_date(data_inizio, data_fine)
        oggetti = []
        df_filtered = pd.DataFrame(filtered_rows, columns=['giorno', 'mese', 'anno', 'oggetto', 'tipologia', 'prezzo', 'data'])
        somma_prezzi = df_filtered['prezzo'].sum()
        oggetti = df_filtered['oggetto'].tolist()
        oggetti_stringa = ', '.join(oggetti)

        return "Dal giorno {} al giorno {}, hai speso in totale {} euro. Gli oggetti acquistati sono : {}".format(data_inizio, data_fine, somma_prezzi, oggetti_stringa)
    except:
        return "C'è stato un errore con le date. 1: {} 2: {}".format(data_inizio, data_fine)

output = sum_prices_and_get_items()
dispatcher.utter_message(text=output)
return []
```

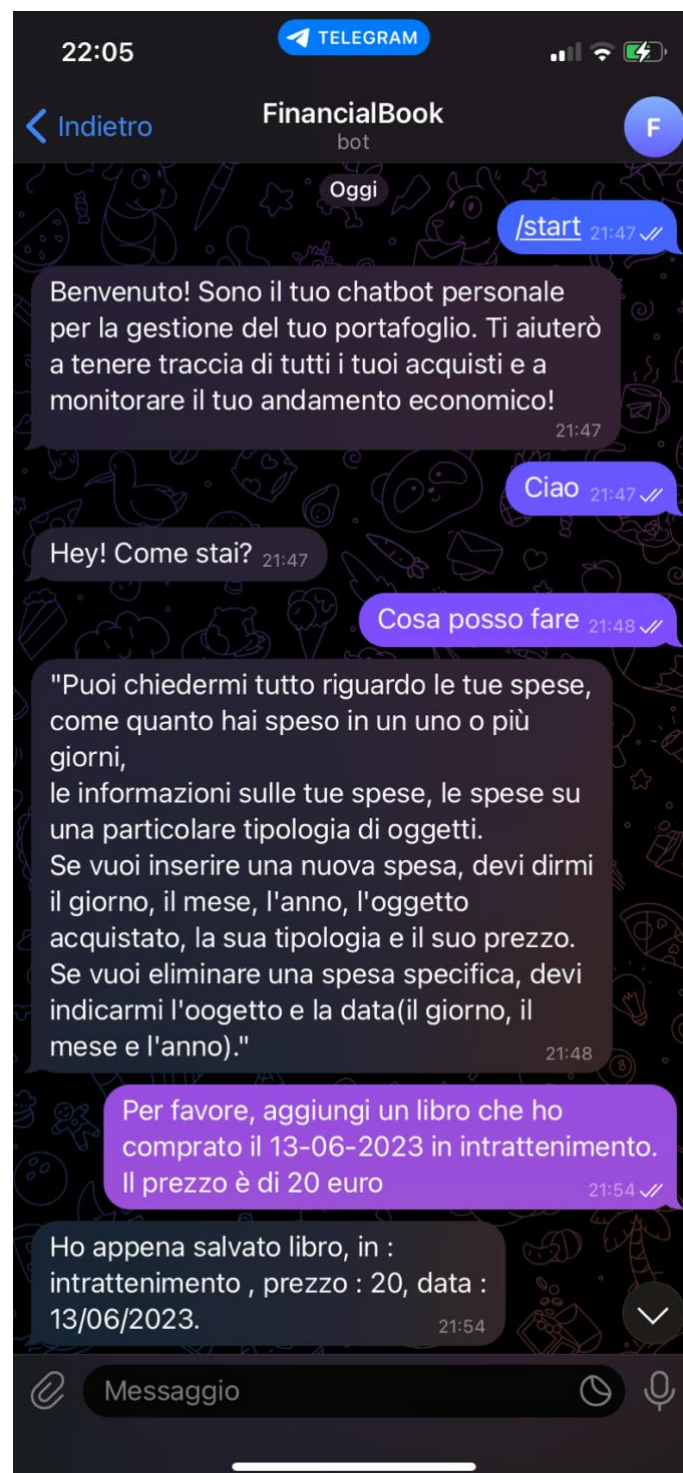
Dopo aver separato le date inserite in una stringa della forma gg-mm-aaaa/gg-mm-aaaa, questa azione si compone di tre funzioni:

- **is_date_between:** Dal momento che è necessaria una ricerca per date, abbiamo implementare una funzione che trasformi le date 1 e 2 da formato stringa e formato datetime.
- **filter_rows_by_date:** Questa è la funzione che prende tutte le righe del dataset il cui campo "data" è compreso tra le date specificate.

- `sum_prices_and_get_items`: Infine, vengono sommate le spese e viene creata una stringa degli elementi acquistati nel range di date risultante dalla funzione precedente.

5. Esempio di utilizzo

Si riportano di seguito degli esempi di utilizzo del chatbot usando l'app Telegram come interfaccia:



22:05

TELEGRAM



< Indietro

FinancialBook

bot

F

Dimmi quanto ho speso nel mese di Maggio

22:01 ✓✓

Il prezzo totale che hai sostenuto nel mese di Maggio è di 5160 , la media del mese è di 177.93103448275863. Le due categorie su cui hai speso di più sono : tipologia elettronica 3890 accessori 730

22:01

Mostrami la spesa totale di Giugno

22:02 ✓✓

Il prezzo totale che hai sostenuto nel mese di Giugno è di 490 , la media del mese è di 61.25. Le due categorie su cui hai speso di più sono : tipologia intrattenimento 220 casa 180

22:02

Elencami la statistica delle mie spese

22:02 ✓✓

Il prezzo totale che hai sostenuto è di : 5729 , la media per oggetto acquistato è : 150.76315789473685, le due categorie su cui hai speso di più sono : tipologia elettronica 3969 accessori 780

22:02

Fammi il calcolo di quanto ho speso dal 10-5-2023/23-5-2023

22:03 ✓✓

Dal giorno 10/5/2023 al giorno 23/5/2023, hai speso in totale 2400 euro. Gli oggetti acquistati sono : Televisore, Mouse,



Messaggio



22:06

TELEGRAM



< Indietro

FinancialBook

bot

F

intrattenimento 220

casa 180

22:02

Elencami la statistica delle mie spese 22:02 ✓✓

Il prezzo totale che hai sostenuto è di : 5729
, la media per oggetto acquistato è :
150.76315789473685, le due categorie su
cui hai speso di più sono : tipologia
elettronica 3969
accessori 780

22:02

Fammi il calcolo di quanto ho speso dal
10-5-2023/23-5-2023 22:03 ✓✓

Dal giorno 10/5/2023 al giorno 23/5/2023,
hai speso in totale 2400 euro. Gli oggetti
acquistati sono : Televisore, Mouse,
Profumo, Gioielli, Fotocamera, Maglietta,
Pantaloni, Auricolari, Gioco da tavolo,
Lampada, Giocattolo, Macchina fotografica,
Portafoglio

22:03

Dimmi cosa ho acquistato il 21-5-2023 22:03 ✓✓

Il giorno 21/5/2023, hai speso 15 euro
(anticamente si usavano le rupie),
comprando : Giocattolo.

22:03

Dimmi una curiosità 22:05 ✓✓

Nel 2010, qualcuno ha acquistato due pizze
per 10.000 bitcoin, rendendolo uno degli
acquisti più costosi della storia.

22:05



Messaggio



22:38

TELEGRAM

4G

< Indietro

FinancialBook

bot

F

comprato il 26-3-2023 in elettronica. il
prezzo è di 30 euro

22:19 ✓✓

Ho appena salvato vaso, in : elettronica ,
prezzo : 30, data : 26/3/2023.

22:19

Dimmi cosa ho acquistato il 21-5-2030

22:24 ✓✓

Non ci sono stati acquisti il giorno :
21/5/2030

22:25

Elimina il Vaso

22:28 ✓✓

Ho appena eliminato questo oggetto
comprato: vaso.

22:28

Ho effettuato un acquisto di abbigliamento:
gioielli a 700. Aggiungi questo oggetto al
27-2-2023

22:36 ✓✓

Ho appena salvato gioielli, in :
abbigliamento , prezzo : 700, data :
27/2/2023.

22:36

Dimmi cosa ho acquistato il 27-2-2023

22:37 ✓✓

Il giorno 27/2/2023, hai speso 700.0 euro
(anticamente si usavano le rupie),
comprando : gioielli.

22:37

Elimina l'acquisto gioielli

22:38 ✓✓

Ho appena eliminato questo oggetto
comprato: gioielli.

22:38



Messaggio

