DOCUMENTAZIONE CASO DI STUDIO
INGEGNERIA DELLA CONOSCENZA 2021-2022

Progetto realizzato da: Bartolomeo Roberto Siciliano

Matricola: 724558

E-mail: b.siciliano1@studenti.uniba.it

Link al repository del progetto: https://github.com/SicilianoBartolomeo/Ingegneria-della-Conoscenza

INTRODUZIONE:

Il progetto consiste nella realizzazione di un sistema esperto, basato su una base di conoscenza, utilizzato al fine di riconoscere e diagnosticare guasti o malfunzionamenti automobilistici. Il processo appena descritto è complesso e richiede l'intervento di esperti di dominio e ingegneri meccanici, proprio per questo il sistema non vuole sostituire l'aiuto di un esperto, ma si propone come un'assistenza temporanea. L'idea del progetto si basa su un sistema che una volta riconosciuto un guasto ha la possibilità di suggerire una soluzione al problema, o altrimenti, in caso non riesca a individuare un guasto, ha la possibilità di fornire all'utente indicazioni stradali per rivolgersi ad un meccanico o eventualmente suggerire un prezzo di vendita dell'auto attraverso l'uso di modelli di apprendimento supervisionato. Quest'ultima componente non è interattiva, ovvero non permette all'utente di effettuare query al sistema poiché necessita della realizzazione di un'interfaccia utente, argomento considerato marginale ai fini del corso; dunque, consiste in un'analisi delle prestazioni di vari modelli di apprendimento supervisionato (regressori), utilizzati per predire il prezzo di un auto usata a partire da una serie di caratteristiche dette feature.

LIBRERIE E STRUMENTI UTILIZZATI NELLA REALIZZAZIONE:

Il progetto è stato realizzato in **Python** versione 3.10.2 . La base di conoscenza e la ricerca sul grafo per calcolare un percorso sono state realizzate attraverso l'IDE *Visual Studio Code* mentre la parte di apprendimento supervisionato è stata realizzata come *Notebook* su *Google Colab* così da sfruttare gli strumenti messi a disposizione da *Google*, come l'uso di una GPU per addestrare i modelli più velocemente.

Librerie utilizzate:

- Experta, utilizzata per realizzare il sistema di diagnostica basato su base di conoscenza;
- OSMnx, utilizzata per rappresentare Bari sottoforma di grafo tramite le API di OpenStreetMap;

- Json, utilizzata per gestire file json
- Requests, utilizzata per fare request alle API di OpenStreetMap
- Networkx, utilizzata per gli algoritmi di ricerca su grafo;
- Prettytable, utilizzata per la visualizzazione di elenchi tramite tabella;
- Pandas, utilizzata per caricare i dataset,
- Numpy, utilizzata per contenere e gestire dataset;
- Sklearn, utilizzata per la realizzazione dei modelli di apprendimento supervisionato;
- Matplotlib, utilizzata per stampare grafici;

MANUALE UTENTE:

COME AVVIARE SISTEMA DI DIAGNOSTICA E NAVIGATORE

Per avviare i sistemi è necessario installare le librerie come spiegato nel file *Read.me* .

Per avviare il sistema di diagnostica è sufficiente avviare il file "***knowladgeBase.py***". Una volta avviato viene proposto un questionario attraverso il quale l'utente inserisce i sintomi da lui notati durante la prova di accensione del proprio veicolo. L'utente può rispondere inserendo l'intero che corrisponde alla risposta che vuole dare. Per tutte le risposte di questo tipo, anche nel programma del navigatore, il sistema gestisce eventuali errori di input, di valore o di tipo, segnalandoli all'utente e riproponendo le domande. Una volta risposto alle domande il sistema risponderà con il problema identificato e una soluzione suggerita.

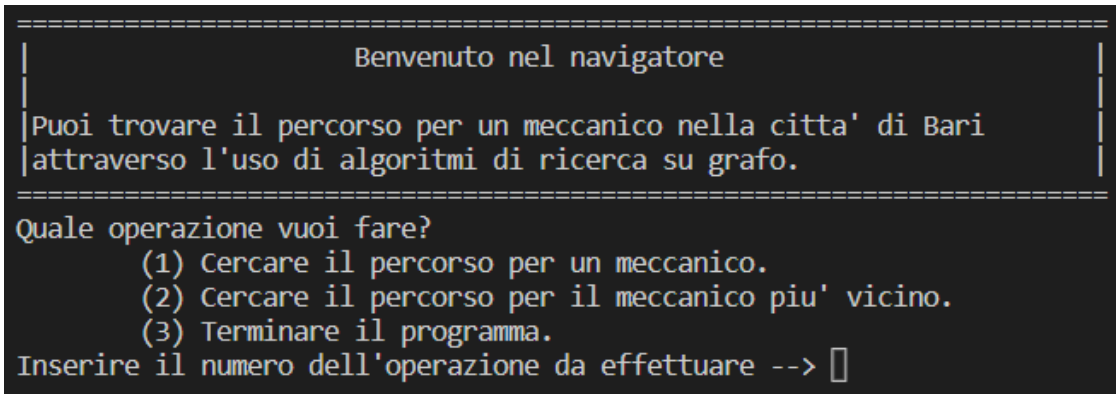
```
Quando provi a mettere in moto la macchina noti che le luci:
        (1) diventano piu' fioche
        (2) non diventano piu' fioche
        (3) a volte si' a volte no
 --> 1

Quando provi a mettere in moto la macchina senti odore di carburante:
        (1) si'
        (2) no
        (3) a volte si' a volte no
 --> 3
La batteria e' scarica
Si consiglia di ricaricare la batteria.
```

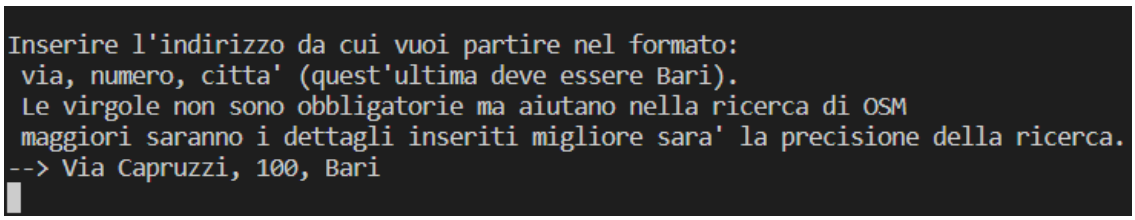
Se il sistema non identifica alcun problema l'utente può scegliere, attraverso un menu, se ripetere il test, trovare il percorso per un meccanico o terminare il programma.

```
Nessun problema identificato
Quale operazione vuoi effettuare?
        (1) Cercare un meccanico
        (2) Riprovare il test
        (3) Terminare il programma
 -->
```

Se l'utente decide di trovare il percorso per un meccanico viene eseguito il file "***navigator.py***". Quest'ultimo può essere avviato come appena descritto, o se si vuole provarlo individualmente, può essere avviato senza bisogno di eseguire prima "*knowladgeBase.py*". Una volta avviato l'utente può scegliere se cercare il percorso per un meccanico, cercare il percorso per il meccanico più vicino alla propria posizione o terminare il programma.



Se si sceglie una delle prime due opzioni, il sistema chiede all'utente di inserire la propria posizione di partenza. L'indirizzo può essere specificato inserendo via, numero e la città di Bari. Il sistema funziona solo se si inserisce la città di Bari poiché solo essa è stata rappresentata mediante grafo per effettuare la ricerca del percorso; dunque, se non si inserisce correttamente, non è garantito il corretto funzionamento. Inoltre, non è obbligatorio inserire virgole o maiuscole nell'indirizzo, tuttavia maggiori saranno i dettagli inseriti, migliore sarà la ricerca di *OpenStreetMap* nel trovare la posizione corretta.



Una volta inserito l'indirizzo di partenza, se è stato scelto di trovare il percorso per il meccanico più vicino verrà stampato direttamente il percorso sulla mappa.

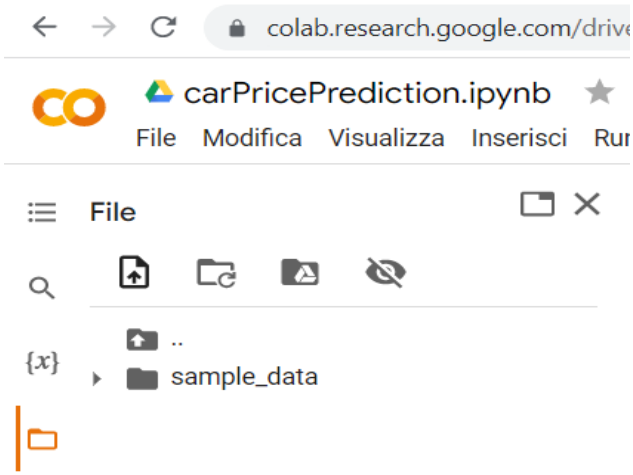
Mentre se è stato scelto di trovare il percorso per un meccanico, l'utente sceglie il meccanico da una lista proposta dal sistema. Successivamente gli verrà chiesto il tipo di algoritmo da utilizzare e dopo di ciò verrà stampato il percorso. Il percorso stampato può essere zoommato e ci si può navigare tramite le opzioni della finestra.

```
Ecco una lista delle officine di Bari:

+----+-----------------------------------------------+-----------------------------------------+
| Id |                     Nome                      |                Indirizzo                |
+----+-----------------------------------------------+-----------------------------------------+
| 1  | Autofficina Meccanica ElettrautoColaianni Lorenzo | Viale Ottorino Respighi, 55, 70132 Bari BA |
| 2  |                 Officina Calabria             |    Via Bruno Buozzi, 60, 70132 Bari BA  |
| 3  |                 Attolico Officina             | Viale Caduti di Nassiriya, 6, 70124 Bari BA |
| 4  |          Officine Meccaniche Longo Snc        |    2 trav. S. Caterina, Bari, BA 70100  |
| 5  |           Autofficina Raffaele Pansini        | Corso della Carboneria, 26, 70123 Bari BA |
| 6  |              Autofficina MANNARINI            |   Via Giovanni Bovio, 24, 70123 Bari BA |
| 7  |                Bosch Car Service              | Via Scipione Crisanzio, 216, 70123 Bari BA |
| 8  |       Autofficina Autorizzata Sapone Vito     | Viale Domenico Cotugno, 19, 70124 Bari BA |
| 9  |                  Officina C&C                 |       Via Timavo, 29, 70124 Bari BA     |
| 10 |                Autofficina Oreste             | Via Giuseppe Zanardelli, 89, 70125 Bari BA |
| 11 |           Autofficina Riparazioni Auto        | Via Giovanni Amendola, 109, 70126 Bari BA |
| 12 | Autofficina meccanico Bari - F.lli De Marzo S.a.s. |    Via Caldarola, 11/A, 70126 Bari BA   |
| 13 |            Autofficina Meccanica Auto         |       Via Padre Pio, 70126 Bari BA      |
+----+-----------------------------------------------+-----------------------------------------+
Inserire l'id dell'officina desiderata --> 7
Quale algoritmo vuoi che venga usato per la ricerca del percorso?
       (1) algoritmo di ricerca che minimizza la distanza euclidea
       (2) algoritmo di Dijkstra
       (3) algoritmo A*
       (4) tutti e tre i precedenti elencati
 -->
```

COME AVVIARE IL SISTEMA DI APPRENDIMENTO SUPERVISIONATO

Lo sviluppo e l'analisi degli algoritmi di apprendimento supervisionato sono avvenuti tramite lo sviluppo di un *notebook* su *Google Colab*. Non è necessario installare in locale dipendenze o librerie poiché basta avviare le finestre del *notebook* seguendo l'ordine, cliccando sul link del *notebook* contenuto su *Github*. Tuttavia, per provare il sistema, è necessario caricare i dataset, poiché i file caricati su *Colab* non godono della persistenza dopo la chiusura della finestra. Il dataset "**CarPrice.csv**" si trova nella cartella "**Resources**" e può essere facilmente caricato cliccando a sinistra su "file", e successivamente su "carica file" nel menu che si apre.



Il dataset "**veihcle.csv**" date le sue grandi dimensioni, verrà caricato da *Kaggle*. Per fare ciò è sufficiente generare un token su *Kaggle* e caricare su *Colab* il file che lo contiene chiamato "***kaggle.json***". Se si hanno difficoltà su quest'ultimo passaggio si possono consultare lo Step 2 e lo Step 3 su https://www.analyticsvidhya.com/blog/2021/06/how-to-load-kaggle-datasets-directly-into-google-colab/ .

SCELTE PROGETTUALI:

SISTEMA DI DIAGNOSTICA

Il sistema mira a diagnosticare il guasto di un'automobile basandosi sui dati forniti dall'utente, il quale viene coinvolto nella risposta a una serie di domande, mirate a individuare i sintomi dell'auto che non si accende e le possibili relazioni che questi hanno col guasto, al fine di suggerire una soluzione ad esso. Se il sistema non riconosce nessun guasto, l'utente ha la possibilità di ripetere il test, cercare il percorso per un meccanico a Bari o terminare il programma.

La procedura adottata per un sistema di diagnostica è logicamente un ***backward chaining***. In un linguaggio logico (come può essere il *Prolog*) ci limiteremmo a chiedere il guasto dell'auto, e il sistema, notando che ha degli atomi dal valore che non conosce, andrebbe in automatico a porre le domande a run-time sugli atomi che abbiamo definito '*askable*'; tuttavia per le limitazioni della

libreria utilizzata (**Experta**) siamo costretti ad operare con il ***forward chaining*** andando a porre prima le domande all'utente, le cui risposte sono considerate come fatti osservati, per poi determinare se è presente o meno un guasto a seconda della regola considerata vera.

L'idea generale si basa su una regola di derivazione, una forma generalizzata della regola di inferenza chiamata ***modus ponens***:

Se "$h \leftarrow a1 \wedge \ldots \wedge am$" è una clausola definita proposizionale nella base di conoscenza e ogni ai è stato derivato, allora h può essere derivato.

Dove:

- h è la **testa** dell'atomo
- $a1 \wedge \ldots \wedge am$ è il **corpo** della clausola, formato da ai **atomi**

Se $m > 0$ la clausola è detta regola, se $m = 0$ il corpo è vuoto e la clausola è detta clausola atomica o **fatto**. Tutte le clausole atomiche nella base di conoscenza sono sempre derivate in maniera diretta cioè sono considerate sempre vere e rappresentano la conoscenza di fondo della *knowladge base*.

BASE DI CONOSCENZA

La base di conoscenza è piuttosto semplice ma non per questo meno efficace ai fini dell'applicazione, potrebbe essere migliorata per poterla applicare a casi reali tramite l'intervento di esperti di dominio. Le regole sono state formulate prendendo spunto da articoli scientifici (vedi cartella '*References*'). La knowladge base è strutturata nel seguente modo:

Regole:

- battery_dead ← light=1 $\wedge$ engine $= 4$
- battery_dead ← light $= 2$
- replace_battery ← battery_dead

- out_of_gas ← light = 1 $\wedge$ engine $= 2 \wedge$ gas $= 2$
- out_of_gas ← light = 1 $\wedge$ engine $= 1 \wedge$ gas $= 2$
- out_of_gas ← light = 1 $\wedge$ engine $= 3 \wedge$ gas $= 2$
- out_of_gas ← light = 2 $\wedge$ engine $= 2 \wedge$ gas $= 2$
- out_of_gas ← light = 2 $\wedge$ engine $= 1 \wedge$ gas $= 2$
- out_of_gas ← light = 2 $\wedge$ engine $= 3 \wedge$ gas $= 2$
- out_of_gas ← light = 3 $\wedge$ engine $= 2 \wedge$ gas $= 2$
- out_of_gas ← light = 3 $\wedge$ engine $= 1 \wedge$ gas $= 2$
- out_of_gas ← light = 3 $\wedge$ engine $= 3 \wedge$ gas $= 2$
- refuel_gas ← out_of_gas

- battery_weak ← light = 1 $\wedge$ engine $= 2 \wedge$ gas $= 1 \wedge$ light_dim $= 1$
- battery_weak ← light = 1 $\wedge$ engine $= 2 \wedge$ gas $= 1 \wedge$ light_dim $= 3$

- battery_weak ← light = 3
- recharge_battery ← battery_weak

- not_identified ← light = 1 ∧ engine = 2 ∧ gas = 1 ∧ light_dim = 2
- not_identified ← light = 1 ∧ engine = 2 ∧ gas = 3
- not_identified ← light = 1 ∧ engine = 1 ∧ gas = 3
- not_identified ← light = 1 ∧ engine = 3 ∧ gas = 1
- not_identified ← light = 1 ∧ engine = 3 ∧ gas = 3
- not_identified ← light = 1 ∧ engine = 1 ∧ gas = 1 ∧ smell = 2
- select_operation ← not_identified

- car_is_flooded ← light = 1 ∧ engine = 1 ∧ gas = 1 ∧ smell = 1
- car_is_flooded ← light = 1 ∧ engine = 1 ∧ gas = 1 ∧ smell = 3
- wait_then_restart ← car_is_flooded

Fatti:

Fatti dichiarati dall'utente rispondendo alle domande:

- light:
  - 1 = light up
  - 2 = nothing happens
  - 3 = sometimes good sometimes bad
- gas (gas tantk):
  - 1 = not empty
  - 2 = empty
  - 3 = not sure
- light_dim (when try to start):
  - 1 = dim
  - 2 = not dim
  - 3 = sometimes do sometimes don't
- engine (when try to start):
  - 1 = cranks normally
  - 2 = cranks slowly
  - 3 = sometimes good sometimes bad
  - 4 = nothing happens
- smell (smell of gasoline when try to start):
  - 1 = SMELL
  - 2 = NOT SMELL
  - 3 = SOMETIMES

Fatti derivati da regole (rappresentano i problemi emersi, hanno dominio binario):

- battery_dead
- battery_weak
- out_of_gas
- engine_is_flooded
- not_identified

Le **regole** permettono di individuare quattro categorie di problemi ovvero, problemi relativi a un **guasto alla batteria**, alla **batteria scarica**, al **serbatoio vuoto** o a un **motore ingolfato** dal carburante che bagna la candela, e per ognuna di queste regole, derivate a partire dai fatti, verrà derivata una possibile soluzione suggerita all'utente. Per le regole che identificano che nessun guasto è stato individuato verrà chiesto all'utente di scegliere se ripetere il test, cercare il percorso per un meccanico o terminare il programma.

Le regole sono derivate prendendo in considerazione fatti. Questi si dividono in fatti dichiarati prendendo in considerazione le risposte dell'utente (**osservazioni**) e fatti derivati da regole. Le osservazioni sono: *light* che identifica se si accendono le luci quando si gira la chiave, gas che identifica se è presente carburante nel serbatoio, *light_dim* che identifica se le luci diventano più fioche quando si prova a mettere in moto il veicolo, *engine* che identifica come si sente girare il motore quando si prova a mettere in moto il veicolo e *smell* che identifica se si sente odore di carburante quando si prova a mettere in moto il veicolo. I fatti derivati hanno valore binario (True, False), quest'ultimi nel momento in cui la regola corrispondente risulta vera vengono dichiarati veri per identificare il problema e suggerire una soluzione e sono*: battery_dead* che indica che la batteria è rotta e dunque da sostituire, *battery_weak* che la batteria è scarica e bisogna caricarla, *out_of_gas* che indica che il serbatoio è vuoto e bisogna fare rifornimento*, engine_is_flooded* che indica che il motore è ingolfato e bisogna attendere prima di riprovare ad accendere il veicolo, e *not_identified* che indica che non è stato identificato nessun problema dal sistema. Infine, ci sono fatti "di supporto" come *question* e *order_question* utilizzati per gestire il flusso delle domande.

SISTEMA ESPERTO

Un **sistema esperto** è un'applicazione dell'intelligenza artificiale che vede un programma cercare di risolvere dei problemi, provando a riprodurre i comportamenti di persone esperte in un determinato campo di attività, per fare inferenza.

È principalmente composto da una "***knowledge base***" (sopra descritta), che rappresenta e memorizza fatti e regole riguardanti il mondo, un "***inference engine***" che si occupa di mettere in pratica le nozioni apprese dalla base di conoscenza, e da una "*user interface*" che permette una facile interazione tra il sistema e l'utente.

IMPLEMENTAZIONE DEL SISTEMA ESPERTO

L' implementazione del sistema si basa su un sistema esperto realizzato tramite il linguaggio **Python** utilizzando la libreria **Experta**, la quale permette di associare un insieme di fatti ad un insieme di regole relativi agli stessi, ed eseguire azioni in base alle regole di abbinamento. Le regole sono formate da due componenti chiamati **LHS** (Left-Hand-Side) e **RHS** (Right-Hand-Side). Il

primo descrive le condizioni in base alle quali la regola viene applicata, mentre il secondo è l'insieme di azioni da eseguire quando viene applicata la regola.

Nel momento in cui si avvia il sistema, un fatto denominato "*question*" viene impostato a *True* per iniziare a porre all'utente le domande per comprendere i sintomi riscontrati durante il tentativo di accensione del veicolo. A seconda della domanda l'utente può rispondere inserendo il numero della risposta che vuole dare. Il sistema è tollerante agli errori gestendo i casi in cui l'utente immette input errati di valore o di tipo, sollevando e gestendo le opportune eccezioni.
Per ogni sintomo è stata associata una regola che, quando viene applicata, si occupa di domandare all'utente se riscontra il sintomo e, in base alla sua risposta, il fatto relativo al sintomo viene avvalorato. Nel momento in cui tutti i fatti relativi ai sintomi sono avvalorati, il sistema applica le regole opportune per derivare i problemi emersi. Quando un problema è stato derivato viene impostato a *True* il fatto relativo e di conseguenza verrà derivata la regola per suggerire una soluzione all'utente a seconda del problema emerso. Ad esempio, se l'utente risponde dicendo che girando la chiave le luci si accendono, che è presente carburante nel serbatoio e che quando prova a mettere in moto il motore gira lentamente e le luci diventano più fioche, verrà derivata la regola che indica che la batteria è scarica, avvalorando il fatto corrispondente al problema e, di conseguenza, suggerendo all'utente di ricaricare la batteria.

RICERCA SU GRAFO

Il sistema permette all'utente di trovare il percorso per un meccanico nella città di Bari a partire dalla propria posizione. L'idea si basa sul fatto che nel momento in cui il sistema di diagnostica non riesce a segnalare alcun guasto, l'utente ha l'opportunità di trovare il percorso stradale per un meccanico che risolva i problemi al veicolo. In particolare, l'utente può scegliere se trovare il percorso per un meccanico da lui scelto fra un elenco di meccanici proposti dal sistema, o trovare il percorso per il meccanico più vicino alla propria posizione.

IMPLEMENTAZIONE

Per realizzare ciò la città di Bari è stata rappresentata sottoforma di grafo al fine di poter applicare gli algoritmi di ricerca per trovare il cammino tra due nodi. È stata utilizzata la libreria *OSMnx* che consente di scaricare dati geospaziali da ***OpenStreetMap*** e modellare, progettare, visualizzare e analizzare reti stradali del mondo reale. È stato creato un grafo da *OSM* con centro in Via Capruzzi e che si estende nel raggio di 3,5 km i cui nodi sono gli incroci tra le strade e gli archi le strade che li collegano. L'elenco dei meccanici è stato creato manualmente sul file "*meccanici.json*" indicando id, nome, indirizzo e le coordinate ovvero latitudine e longitudine.

Per cercare il percorso per un meccanico, l'utente immette il proprio indirizzo specificando via, eventuale numero civico e la città di Bari. Da questo indirizzo vengono recuperate le coordinate tramite una *request* all'*API* di *OpenStreetMap* chiamata *Nominatim* e, una volta ottenute, verrà impostata come partenza il nodo del grafo più vicino a tali coordinate. Chiaramente maggiori sono i dettagli dell'indirizzo e migliore sarà la precisione del sistema nell'individuare l'origine. Dopo di ciò l'utente immette l'id del meccanico scelto dall'elenco proposto e verrà impostata come destinazione il nodo più vicino alle coordinate del meccanico selezionato. Infine, l'utente sceglie

quale algoritmo verrà usato dal sistema, potendo scegliere tra l'algoritmo che minimizza la distanza euclidea, l'algoritmo di Dijkstra, l'algoritmo A* o tutti e tre per fare un confronto, e il sistema lo applica per trovare il cammino e stampare il percorso sulla mappa di Bari.

Per cercare il percorso per il meccanico più vicino l'utente immette il proprio indirizzo e il sistema si occupa di trovare il meccanico più vicino e stamparne il percorso. Per fare ciò il sistema fa una ricerca del cammino minimo, ovvero cerca i percorsi dal nodo di origine verso tutti i meccanici, tramite l'algoritmo che minimizza la distanza euclidea, e ne restituisce il più breve.

ALGORITMI UTILIZZATI

- **Algoritmo di ricerca che minimizza la distanza euclidea**: è stato utilizzato un algoritmo messo a disposizione da *OSMnx* ovvero:
  osmnx.distance.shortest_path(*G,orig,dest,weight='length')*
  Tale algoritmo prende in input un grafo, il nodo di origine e di destinazione per cui calcolare il percorso e un peso da minimizzare ovvero *lenght* che rappresenta la distanza euclidea tra due nodi. Restituisce una lista di nodi che rappresentano il cammino minimo tra origine e destinazione.
- **Algoritmo di Dijkstra**: è un algoritmo utilizzato per cercare i cammini minimi in un grafo ciclico e con pesi non negativi sugli archi. L'algoritmo inizia dal nodo scelto, noto anche come nodo di origine, tenendo traccia del percorso più breve attualmente noto, da ciascun nodo al nodo di origine. Aggiorna i valori del percorso se trova un percorso più breve. Quando l'algoritmo trova il percorso più breve tra il nodo di origine e un altro nodo, quel nodo viene contrassegnato come "visitato" e aggiunto al percorso. Questo processo continua finché tutti i nodi non sono stati aggiunti al percorso. Il risultato dell'algoritmo è un percorso che collega il nodo sorgente a tutti gli altri nodi nel grafico seguendo il percorso più breve per ciascun nodo.
- **Algoritmo A*** : è un algoritmo che garantisce di trovare il cammino dal costo minimo.
  astar_path(G, source, target, heuristic=None, weight='weight')
  L'algoritmo prende in input il grafo, il nodo di origine e di destinazione, una funzione euristica che stima la distanza tra un nodo e l'obiettivo, e un peso ovvero il costo derivato dal peso degli archi percorsi tra due nodi. Nella scelta del cammino da espandere si considerano il costo del percorso e l'euristica. Quindi per ogni percorso parziale della frontiera, l'algoritmo usa una stima del costo di un percorso completo che lo estenda. Per essere ammissibile, l'euristica h, deve sottostimare il costo effettivo minimo per raggiungere il nodo obiettivo. La frontiera, contenente i nodi dei percorsi parziali, è implementata solitamente per mezzo di una coda con priorità, dove l'ordine è dato dalla somma del costo più quella dell'euristica.

APPRENDIMENTO SUPERVISIONATO

La compravendita dei veicoli usati è un'attività in via di sviluppo con una stima del valore di mercato che si è moltiplicata negli ultimi anni. L'ascesa di siti online e altri strumenti simili hanno reso più semplice, per acquirenti e venditori, migliorare la comprensione delle variabili che

decidono il valore di un veicolo usato. Proprio per questo, i modelli di apprendimento potrebbero essere utilizzati per stimare il costo di un qualsiasi veicolo. L'idea di implementare questi modelli nel caso di studio nasce dal fatto che, nel momento in cui il sistema di diagnostica o un meccanico segnalano problemi al veicolo dell'utente, esso potrebbe pensare di vendere il veicolo e/o comprarne uno nuovo. Dunque, l'utente, dovrebbe avere l'opportunità di interrogare un modello, che predica il valore di vendita del proprio veicolo o il valore di acquisto di un veicolo, in base a varie caratteristiche chiamate feature, come i chilometri percorsi, il tipo di motore ecc… . Il sistema, mancando di interfaccia interattiva, non si può interrogare ma è limitato ad un'analisi di modelli di apprendimento supervisionato, per valutare le prestazioni di predizione di valori considerando anche dataset diversi. Il sistema è stato implementato come *notebook* di *Google Colab* per sfruttare gli strumenti messi a disposizione come l'esecuzione su GPU.

ISPEZIONE E PREPROCESSING DEI DATI

I dataset utilizzati per addestrare e valutare i modelli di regressione sono contenuti nella cartella *Resources* e sono:

- **CarPrice.csv**: contenente più di 5000 veicoli venduti da siti web in India come *cars24, cardekho, AutoPortal* e *olx*, quest'ultimo conta in India più di 200 milioni di utenti attivi e 8,5 milioni di transizioni mensili.
- **Vehicles.csv**: contenente più di 400 mila veicoli venduti negli Stati Uniti su *Craigslist*, noto sito web di vendita di veicoli e non solo. Il dataset è disponibile su *Kaggle* al seguente indirizzo, https://www.kaggle.com/datasets/austinreese/craigslist-carstrucks-data.

Per entrambi i dataset sono stati analizzati e preprocessati per poi essere divisi casualmente in training test e test set, dove il primo contiene il 70% dei dati e il secondo, il restante 30%.

Il **primo dataset** contiene feature quali brand e modello del veicolo; varient che rappresenta la variante del modello (ad esempio modello Hyunadi i10 varient Sportz1.1); tipo di carburante; chilometri percorsi; trasmissione; numero di proprietari che ha avuto il veicolo; luogo di vendita; data dell'annuncio e prezzo in rupie indiane. Quest'ultimo è rappresentato come un intero ed è il target ovvero il valore da predire. Tutte le altre feature sono rappresentate come oggetti. Si è notato che nella colonna varient ci sono 49 valori nulli che sono stati sostituiti dal valore più frequente ovvero dalla moda. Infine, sono stati trasformati i valori delle feature da tipo oggetto a tipo ordinale e sono stati normalizzati i valori. Non sono stati gestiti i valori outliers, cioè troppo distanti dalla media, poiché il dataset è di piccole dimensioni (5000 tuple circa).

Il **secondo dataset**, di dimensioni molto maggiori del precedente, contiene 26 feature differenti alcune delle quali non sono state tenute in considerazione poiché molto comuni, poco informative o ripetitive come link, descrizioni o id. Dunque, le feature utilizzate sono: prezzo; anno del veicolo; casa automobilistica; modello; condizioni del veicolo; numero di cilindri; tipo di carburante utilizzato; odometro ovvero miglia percorsi dal veicolo; stato del veicolo; tipo di cambio (automatico, manuale o altro); numero di ruote motrici; dimensioni; tipo del veicolo come suv, berlina ecc.; colore; stato di quotazione. Il prezzo in dollari è il target ed è rappresentato come

intero. Per migliorare le prestazioni dei modelli sono stati gestiti i valori outliers. Per il prezzo si è notata una grande differenza tra il 75% dei valori ed il valore massimo. Si è deciso di eliminare il 10% dei valori agli estremi della distribuzione. Nella colonna che rappresenta le miglia percorse dal sono stati gestiti i valori nulli e sono stati mantenuti solo i valori minori di 3000000 poiché si è notato graficamente che quelli superiori fossero outliers. Sono state eliminate le tuple contenenti valori nulli nella colonna riguardante l'anno del veicolo poiché quest'ultimo non poteva essere rimpiazzato. Infine, sono stati trasformati i valori di tipo oggetto in valori di tipo ordinale.

ANALISI DEI MODELLI DI APPRENDIMENTO SUPERVISIONATO

Un problema di regressione è una generalizzazione del problema di classificazione, in cui il modello restituisce come output un valore continuo, piuttosto che un insieme finito. In altre parole, un modello di regressione stima una funzione multivariata a valori continui.

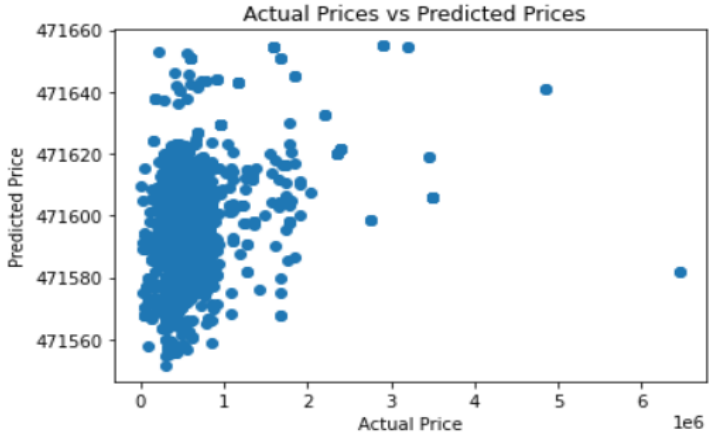
SUPPORT VECTOR REGRESSOR

Una SVM risolve problemi di classificazione binaria trovando la retta di separazione delle classi ovvero l'**iperpiano** che massimizza il **margine** tra le classi stesse, dove con margine si intende la distanza minima dall'iperpiano ai punti delle due classi. Questo obiettivo viene raggiunto utilizzando una parte del training set, i cosiddetti vettori di supporto. I **vettori di supporto** sono i valori di una classe più vicini alla retta di separazione, quelli che più si avvicinano all'altra classe, ovvero, i valori classificabili con più difficoltà. Una volta trovati questi esempi, tutti gli altri saranno ininfluenti ai fini della classificazione, perché sono loro a definire la retta di separazione e il margine. Quindi rispetto ad altri modelli che cercano di minimizzare l'errore tra valori predetti e reali, una SVM cerca gli esempi più rappresentativi e difficili da classificare. Inoltre, possono essere utilizzate funzioni matematiche dette **kernel** che prendono dati in input e li trasformano nella forma richiesta, qualora non sia possibile determinare un iperpiano linearmente separabile.

Quando una Support Vector Machine effettua operazioni di regressione prende il nome di **Support Vector Regressor**. La SVR utilizza gli stessi principi della SVM con alcune differenze. Viene impostato un margine di tolleranza per minimizzare l'errore per individuare l'iperpiano che massimizza il margine. Quest'ultima, ha una buona capacità di generalizzazione ed è robusta ai valori outliers. L'addestramento ha una complessità più che quadratica al numero di esempi, dunque, non scala bene dataset che contengono più di 10000 esempi. Per dataset di grandi dimensioni solitamente si usano altri modelli come LinearSVR o SGD Regressor.

- SVR addestrato sul primo dataset:

```
SVR Regressor Score: -0.06325370730174851
SVR Regressor r2_score: -0.06820672247987791
Mean absolute error of SVR Regressor: 271198.4321420298
Mean squared error of SVR Regressor: 297284446193.9865
Root Mean Square error of SVR Regressor: 545237.9720764012
```



Come possiamo notare il coefficiente di determinazione R^2 è negativo sia sul training set (molto grave) che sul test set e gli errori di predizione calcolati sono molto grandi; dunque, non è il modello corretto da applicare per questo task o per questo dataset. Non è stato possibile addestrare l'SVR sul secondo dataset dato che la complessità è quadratica rispetto al numero di esempi che nel secondo dataset sono più di 400 mila.

REGRESSIONE LINEARE

La **regressione lineare** rappresenta un metodo di stima del valore atteso di una variabile dipendente Y, dati i valori di altre variabili indipendenti, anche dette feature *X1, X2, …, Xk* che rappresentano le caratteristiche degli esempi a disposizione. La regressione lineare prende questo nome dal fatto che la Y da predire è rappresentata come una combinazione lineare delle variabili indipendenti con dei pesi *w*.

Ovvero,

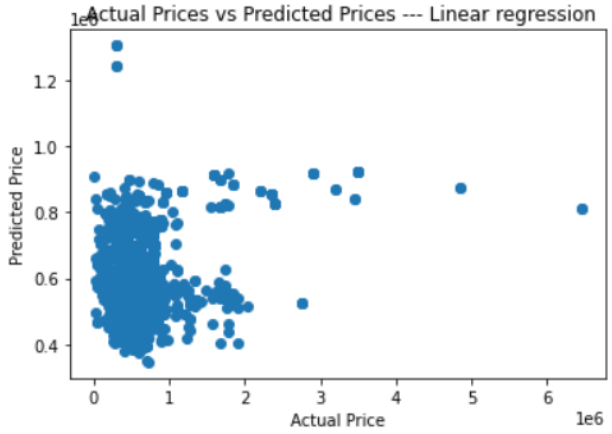
$$Y^(e) = w0 + w1\ X1(e) + \ldots + wn\ Xn(e) + \varepsilon$$

dove *Y^(e)* è il valore da predire; *w0, w1, …, wn* rappresentano i pesi che durante l'addestramento sono impostati per minimizzare la funzione d'errore; *X1(e), X2(e), …, Xn(e)* rappresentano i valori delle feature sull'esempio e; ε rappresenta l'errore o bias.

Dunque, l'obiettivo è far passare la curva vicino ai valori degli esempi a disposizione e lo fa gestendo i valori del vettore W dei parametri in modo da minimizzare l'errore di predizione della funzione di errore che di solito è la funzione dell'errore quadratico.

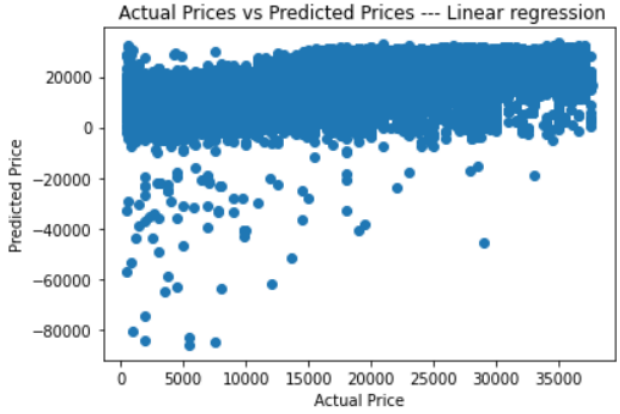
- Linear regression addestrato sul primo dataset:

```
Linear Regression Score: 0.055020586349048384
Linear Regression r2_score: 0.061364713139148486
Mean absolute error of Linear Regressor: 297255.7327343899
Mean squared error of Linear Regression: 261224410556.7481
Root Mean Square error of Linear Regression: 511101.1744818712
```



- Linear regression addestrato sul secondo dataset:

```
Linear Regression Score: 0.5254535170139525
Linear Regression r2_score: 0.5234859854486539
Mean absolute error of Linear Regressor: 5388.143721885424
Mean squared error of Linear Regression: 49342469.12361189
Root Mean Square error of Linear Regression: 7024.419486591892
```



- Differenze di prestazioni del modello in base al dataset utilizzato:

Model	R2 score on test set	Mean absolute error	Mean squared error	Root mean squared error
Linear regression trained on 1st dataset	0.061	297255.733	261224410556.748	511101.174
Linear regression trained on 2nd dataset	0.523	5388.144	49342469.124	7024.419

Come possiamo notare nemmeno questo modello ha ottenuto buoni risultati. Tuttavia, con il secondo dataset possiamo vedere alcuni miglioramenti. L' R^2 score è passato da essere nemmeno 0,1 ad essere 0,52, questo è un buon miglioramento anche se non lo fa arrivare alla sufficienza, poiché 0,5 sta a significare che ha una probabilità del 50% di fare una predizione corretta, più o meno come lanciare una moneta sperando che esca testa. Le funzioni d'errore ci segnalano anche una diminuzione degli errori di predizione diminuiti di un fattore di più o meno 10^2. Quindi cambiando dataset le prestazioni sono migliorate pur non rendendo il modello adatto al compito.

SGD REGRESSOR

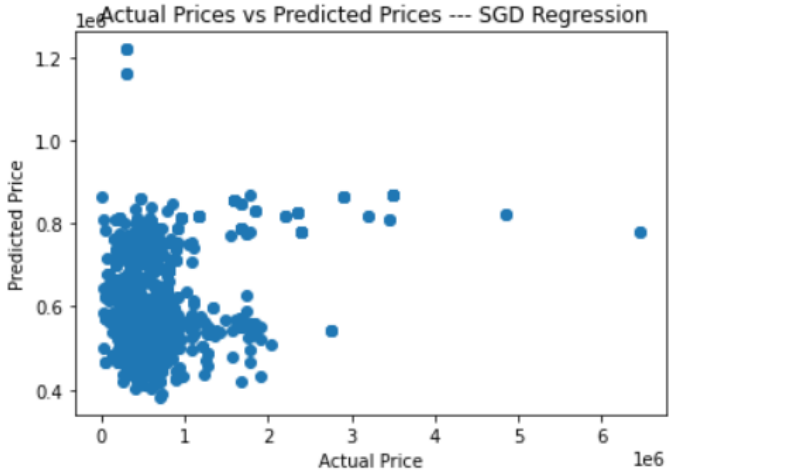
La **discesa di gradiente** è l'algoritmo di ottimizzazione per eccellenza; è utilizzato per apprendere i coefficienti di un modello di ML. Come nel caso della regressione linerare, l'obiettivo del Gradient Descent è quello di trovare i coefficienti del modello che minimizzano la funzione di costo, quelli dunque che ci offrono il modello migliore. Il primo step è quello di inizializzare i coefficienti (pesi) ad un valore casuale al quale sarà associato un certo valore della funzione di costo. Ad ogni passo decrementa ogni peso in base al valore della sua derivata parziale.
Ovvero,

$$wi = wi - \eta * [\, \partial / \partial\, wi\, (\, error(E,w\hat{}) \,) \,]$$

Il valore della derivata ci dirà se e quanto velocemente la funzione di costo sta crescendo o decrescendo in quel preciso punto. Ad esempio, se la pendenza è negativa, il valore della derivata sarà negativo e sottraendola al valore corrente del coefficiente si registra inevitabilmente un aumento del valore di *wi*. Se al contrario la derivata fosse stata positiva allora si sarebbe ottenuta una riduzione del coefficiente. Prima di effettuare la sottrazione si deve moltiplicare la derivata per un fattore costante η detto learning rate che permette di controllare la velocità di ogni step del *Gradient Descent*; a valori di η infinitesimali (η =0,000001) corrispondono step troppo piccoli e, conseguentemente, un tempo eccessivamente lungo per raggiungere la convergenza nel punto di minimo. Allo stesso modo, ponendo η grande (η =100) si rischia che il coefficiente wi rimbalzi da una parte all'altra della curva (apprendimento instabile). Quindi, il learning rate è un iperparametro che si deve ottimizzare; Il calcolo progressivo della derivata della funzione di costo e la sottrazione sono operazioni da ripetere per un numero predefinito di cicli chiamato epoche, le quali con il passare del tempo permetteranno al coefficiente di convergere verso il valore ottimale; man mano che il coefficiente si avvicina a tale punto la pendenza della funzione si riduce, il valore della derivata diminuisce e gli step successivi saranno sempre più brevi. Il tutto fino ad arrivare al punto di minimo in cui la derivata è nulla oppure al raggiungimento di un intorno di questo sufficientemente piccolo da ritornare una buona approssimazione del coefficiente. Ad una funzione lineare utilizza l'errore quadratico che essendo una funzione convessa avrà un unico punto di minimo locale che corrisponde a quello globale.

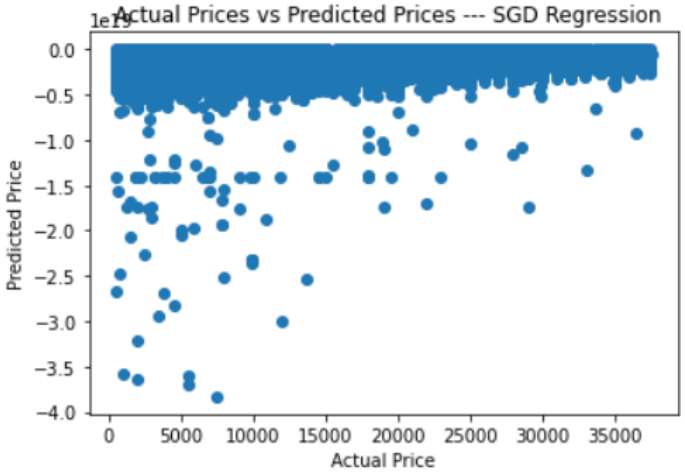
- SGD Regressor addestrato sul primo dataset:

```
SGD Regressor Score: 0.0534783866049946
SGD Regressor r2_score: 0.05977384892331561
Mean absolute error of SGD Regressor: 294452.1040109455
Mean squared error of SGD Regressor: 261667151814.05432
Root Mean Square error of SGD Regressor: 511534.11598255526
```



- SGD Regressor addestrato sul secondo dataset:

```
SGD Regressor Score: -3.0782512383176433e+28
SGD Regressor r2_score: -3.0878647190222692e+28
Mean absolute error of SGD Regressor: 1.3797399913531395e+18
Mean squared error of SGD Regressor: 3.197447816927306e+36
Root Mean Square error of SGD Regressor: 1.788140882852161e+18
```



- Differenze di prestazioni del modello in base al dataset utilizzato:

Model	R2 score on test set	Mean absolute error	Mean squared error	Root mean squared error
SGD regressor trained on 1st dataset	0.062	298369.361	260937694706.706	510820.609
SGD regressor trained on 2nd dataset	-4112692837753551762397069312.000	550886496531365952.000	42586453528091961785388325744410624 0.000	652582971951398400.000

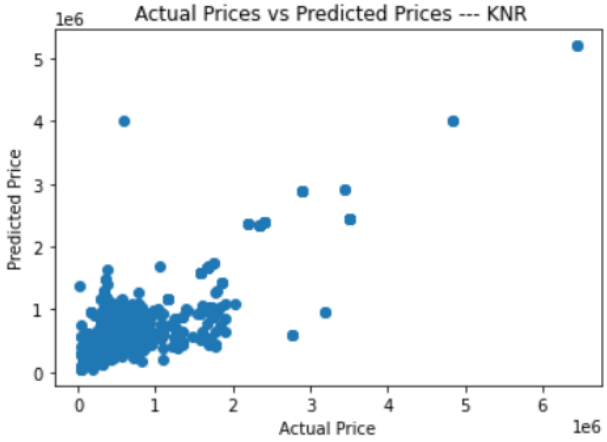
L'SGD Regressor ha avuto valori pessimi che sono andati addirittura a peggiorare quando è stato addestrato sul secondo dataset. L'R^2 score è diventato negativamente grande ed anche le funzioni di errore ci segnalano errori molto grandi. Dunque, il modello fa predizioni di prezzo completamente errate su entrambi i dataset ma a maggior modo nel secondo, più numeroso e con più caratteristiche. Possiamo concludere che nemmeno questo si è rivelato un modello adatto al task o ai dataset.

KNR

Il **k nearest neighbors** è un tipo di apprendimento di tipo *lazy learning* o anche detto basato su istanze poiché durante la fase di training non costruisce alcun modello interno generale, ma memorizza semplicemente istanze di dati di addestramento. Per problemi di classificazione, un'etichetta di classe viene assegnata sulla base di un voto a maggioranza, ad es. se si vuole predire *Y^(e)* sulla base di *X1(e),X2(e), …, Xn(e),* calcola la similarità tra "e" e tutti gli esempi del training set, assegnando ad "e" la categoria di maggioranza dei k esempi più simili. I problemi di regressione utilizzano un concetto simile al problema di classificazione, ma in questo caso viene presa la media dei k vicini più vicini per fare una previsione su una classificazione.

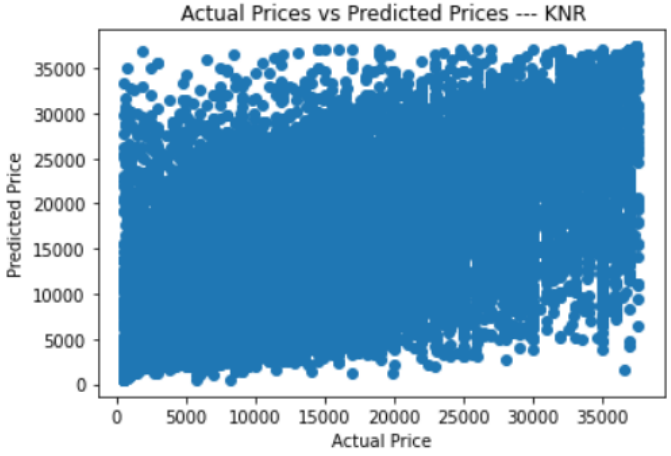
- KNR addestrato sul primo dataset:

```
K Neighbors Regressor Score: 0.748169225500505
K Neighbors Regressor r2_score: 0.6045261241639399
Mean absolute error of K Neighbors Regressorr: 198276.38798679868
Mean squared error of K Neighbors Regressor: 110061310875.45856
Root Mean Square error of K Neighbors Regressor: 331754.89578220027
```



- KNR addestrato sul secondo dataset:

```
K Neighbors Regressor Score: 0.8098253059053128
K Neighbors Regressor r2_score: 0.6997127822601594
Mean absolute error of K Neighbors Regressorr: 3499.6630375358363
Mean squared error of K Neighbors Regressor: 31094390.337069143
Root Mean Square error of K Neighbors Regressor: 5576.234422714771
```



- Differenze di prestazioni del modello in base al dataset utilizzato:

Model	R2 score on test set	Mean absolute error	Mean squared error	Root mean squared error
KNR trained on 1st dataset	0.605	198276.388	110061310875.459	331754.896
KNR trained on 2nd dataset	0.700	3499.663	31094390.337	5576.234

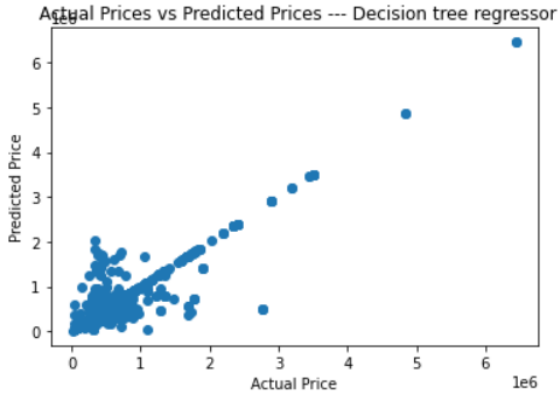
Il knr sul primo dataset ha avuto un punteggio dato dal coefficiente di determinazione appena sufficiente, mentre le funzioni di errore ci dicono che ha fatto errori di predizioni molto grandi. Con il secondo dataset si è comportato meglio con una forte diminuzione di MAE, MSE e RMSE. Ad esempio, il RMSE è passato da più di 330 mila a 5000 così come il MAE passato da quasi 200 mila a nemmeno 3500. Quindi continua a sbagliare ma molto meno che nel primo dataset. Possiamo concludere che pur essendo una tecnica che non addestra alcun modello si è comportato meglio rispetto ai modelli precedentemente descritti.

DECISION TREE REGRESSOR

Gli **alberi di decisione** sono un modello di apprendimento supervisionato usato per task di classificazione o regressione. L'obiettivo è creare un modello che predica il valore di una variabile target apprendendo semplici regole decisionali dedotte dalle feature dei dati. Il modello crea una struttura ad albero simile ad un diagramma di flusso i cui nodi interni rappresentano le condizioni prese in base alle feature, i nodi foglia rappresentano l'output e gli archi (max 2 uscenti per nodo) rappresentano il risultato della condizione di nodo ovvero se l'esempio preso in considerazione soddisfa o meno la condizione espressa dal nodo sottoforma di funzione booleana. Può essere usato come regressore per produrre output di tipo continuo. È un modello semplice da interpretare poiché può essere visualizzato e la sua complessità è logaritmica in base al numero di esempi.

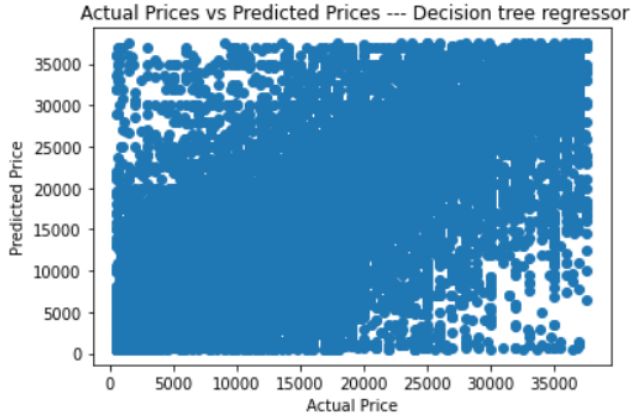
- Albero di decisione addestrato sul primo dataset:

```
Decision Tree Regressor Score: 0.9979397374843745
Decision Tree Regressor r2_score: 0.8131568119761867
Mean absolute error of Decision Tree Regressor: 76913.98415841584
Mean squared error of Decision Tree Regressor: 51998899190.436966
Root Mean Square error of Decision Tree Regressor: 228032.6713224159
```



- Albero di decisione addestrato sul secondo dataset:

```
Decision Tree Regressor Score: 0.9999443872449817
Decision Tree Regressor r2_score: 0.8056592311933846
Mean absolute error of Decision Tree Regressor: 2282.881400127252
Mean squared error of Decision Tree Regressor: 20123759.409947302
Root Mean Square error of Decision Tree Regressor: 4485.951338339205
```



- Differenze di prestazioni del modello in base al dataset utilizzato:

Model	R2 score on test set	Mean absolute error	Mean squared error	Root mean squared error
Decision tree trained on 1st dataset	0.813	76124.259	52008234996.477	228053.141
Decision tree trained on 2nd dataset	0.806	2282.881	20123759.410	4485.951

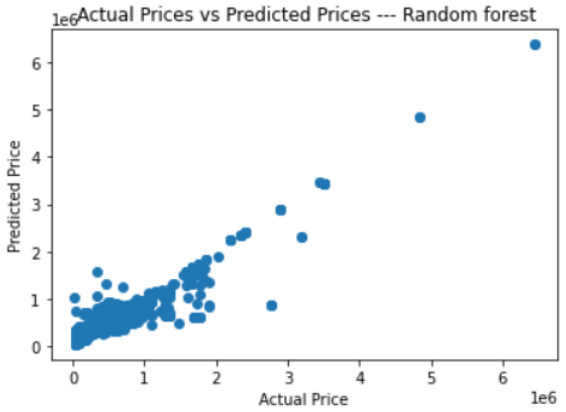
L'r^2 è buono con un valore di 0,8 , sul secondo dataset è diminuito di 0,01 dunque è peggiorato ma in maniera ininfluente. Le funzioni di errore hanno dato risultati migliori sul secondo dataset. Come possiamo notare il MAE è passato da più di 75000 a poco più di 2200 così come il RMSE. Possiamo concludere che tra i modelli utilizzati gli alberi di decisione si sono rivelati uno mei migliori per la predizione del prezzo.

RANDOM FOREST REGRESSOR

Il **random forest** è un modello di apprendimento supervisionato per regressione e classificazione basato sull'*ensemble learning*, ovvero la combinazione di più modelli per migliorare le prestazioni. Uno degli svantaggi principali degli alberi di decisione è che possono andare facilmente in overfitting, cioè non riescono a generalizzare dai dati appresi con la conseguenza di avere buone prestazioni sui dati di training ma non riuscendo a fare corrette predizioni per valori mai visti. Una soluzione potrebbe essere di limitare il partizionamento o di potare gli alberi riducendo però il loro potere predittivo. Proprio per questo il random forest combina la semplicità degli alberi di decisione con la flessibilità e la potenza di un modello di insieme. In particolare, si addestrano un certo numero di alberi di decisione su una parte del dataset e durante la predizione, ogni albero produce come output un proprio valore di predizione, e questi valori verranno aggregati per formare la predizione finale data dal "voto" di ogni albero. La predizione finale può essere basata su una media delle singole predizioni. In questo modo, pur non offrendo la stessa capacità di interpretazione di un singolo albero, le loro prestazioni sono migliori in generale.

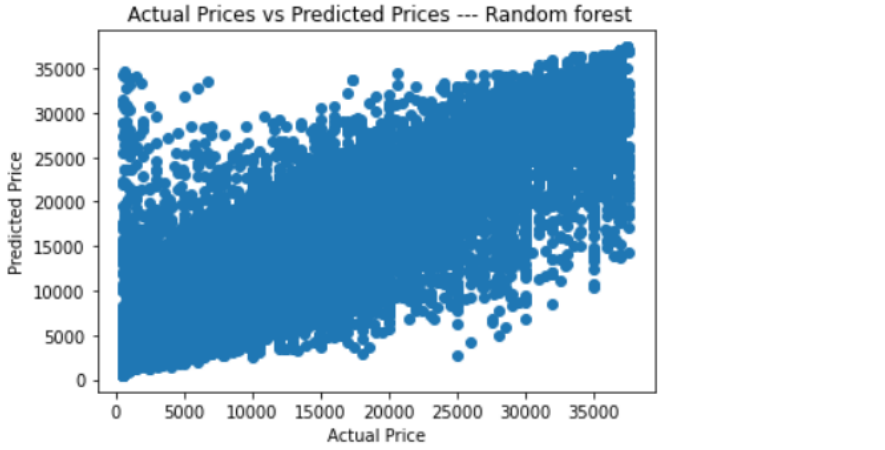
- Random forest regressor addestrato sul primo dataset:

```
Random Forest Regressor Score: 0.9807585294357091
Random Forest Regressor r2_score: 0.8773383627970348
Mean absolute error of Random Forest Regressor: 89255.54549274928
Mean squared error of Random Forest Regressor: 34137022467.407406
Root Mean Square error of Random Forest Regressor: 184762.06988288317
```



- Random forest regressor addestrato sul secondo dataset:

```
Random Forest Regressor Score: 0.9861673996991084
Random Forest Regressor r2_score: 0.8991629553340378
Mean absolute error of Random Forest Regressor: 1793.0012500432777
Mean squared error of Random Forest Regressor: 10441558.088550996
Root Mean Square error of Random Forest Regressor: 3231.3399834358183
```



- Differenze di prestazioni del modello in base al dataset utilizzato:

Model	R2 score on test set	Mean absolute error	Mean squared error	Root mean squared error
Random forest trained on 1st dataset	0.881	88773.815	33229526543.130	182289.678
Random forest trained on 2nd dataset	0.899	1793.001	10441558.089	3231.340

Il random forest ha avuto risultati molto buoni. L'R quadro su entrambi i dataset è di quasi 0,9 e considerando che il massimo è 1 questi sono ottimi valori. Gli errori, come negli altri modelli sono più alti sul primo dataset ma che vanno ad abbassarsi sul secondo. Possiamo concludere che tra tutti i modelli provati il random forest è quello che ha avuto risultati migliori e quindi si rivela il più adatto al compito preso in considerazione.

FUNZIONI DI ERRORE UTILIZZATE

Poiché è stato considerato un task di regressione, come output di una predizione abbiamo numeri reali confrontabili aritmeticamente, dunque, per valutare le predizioni, sono state utilizzate le seguenti metriche:

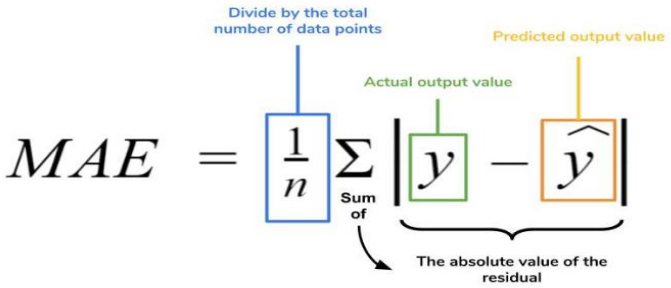
- **R^2** : il coefficiente di determinazione rappresenta la proporzione della varianza di y che è stata esposta dalle variabili indipendenti nel modello. Da un indicazione della bontà dell'addestramento e da una misura di quanto sia probabile che campioni non visti siano

predetti correttamente, attraverso la proporzione della varianza. Il miglior punteggio possibile è 1 e può raggiungere valori negativi;

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \epsilon_i^2$.

- **MAE**: il *mean absolute error* rappresenta la media dell'errore assoluto. Quest'ultimo è la somma delle differenze assolute tra valori effettivi e predetti.



- **MSE**: il *mean squared error*, o errore quadratico medio indica la discrepanza quadratica media fra i valori dei dati osservati e i valori predetti o stimati. È una metrica che penalizza i grandi errori poiché la differenza tra il valore reale e predetto è elevata al quadrato, dunque bisogna cercare di minimizzarlo.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

- **RMSE**: il root *mean squared error* rappresenta la radice dell'errore quadratico medio. Minimizzare il RMSE equivale a minimizzare il MSE poiché le funzioni hanno gli stessi punti di minimo.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

CONFRONTO FRA I MODELLI

Si consiglia di vedere i risultati sul notebook di Colab.

Modelli addestrati sul primo dataset:

Model	R2 score on training set	R2 score on test set	Mean absolute error	Mean squared error	Root mean squared error
SVR	-0.063	-0.068	271198.432	297284446193.987	545237.972
Linear Regression	0.055	0.061	297255.733	261224410556.748	511101.174
SGD Regressor	0.054	0.061	295957.340	261359027758.152	511232.851
K Neighbors Regressor	0.748	0.605	198276.388	110061310875.459	331754.896
Decision Tree Regressor	0.998	0.809	77560.428	53243212855.356	230744.909
Random Forest Regressor	0.982	0.883	87240.374	32437119054.722	180103.079

Modelli addestrati sul secondo dataset:

Model	R2 score on training set	R2 score on test set	Mean absolute error	Mean squared error	Root mean sq
Linear Regression	0.525	0.523	5388.144	49342469.124	7024.
SGD Regressor	-2944584675384672258209559347 2.000	-2953978951749416574100753612 8.000	1331913386973544960.000	3058810670148489406322274226180980736.000	174894558810
K Neighbors Regressor	0.810	0.700	3499.663	31094390.337	5576.
Decision Tree Regressor	1.000	0.806	2281.483	20066165.210	4479.
Random Forest Regressor	0.986	0.899	1792.329	10425343.874	3228.

Come possiamo notare, sul primo dataset, i modelli sono in ordine crescente di prestazione. L'SVR ha avuto risultati molto negativi, perfino sul training set e ciò significa che non ha fatto predizioni corrette nemmeno per esempi già visti; la regressione lineare e l'SGD regressor hanno avuto risultati simili insufficienti; il knr ha avuto risultati appena sufficienti mentre gli alberi di decisione hanno avuto buoni risultati che sono migliorati ancora di più con il random forest. Successivamente, provando a cambiare dataset, i risultati, in generale sono migliorati tranne che per l'SGD Regressor che ha avuto pessimi risultati con errori di predizione molto grandi. La regressione lineare è il modello che ha avuto il miglioramento maggiore con il secondo dataset mentre il knr, gli alberi di decisione e il random forest hanno avuto prestazioni simili a quelle registrate sul primo dataset. Possiamo concludere dicendo che il modello che ha avuto prestazioni maggiori su entrambi i dataset è stato il random regressor che si rivela il modello migliore per predire il prezzo di un veicolo.

CROSS VALIDATION

Addestrare i parametri di una funzione di predizione e testarli sugli stessi dati su cui sono stati addestrato è una metodologia sbagliata: un modello che viene testato sugli stessi dati che ha visto in precedenza avrà ottime prestazioni ma sbaglierà a predire per esempi che non ha mai visto prima. Questa situazione è chiamata overfitting. Quindi training e test set devono essere indipendenti ad ogni prova e ciò richiede una grande quantità di dati. Una soluzione a questo problema è chiamata cross validation. La più utilizzata è la **k-fold cross validation**. L'idea è quella di dividere il dataset in un determinato numero k di parti chiamate folds e di costruire,

successivamente, k modelli con gli stessi iperparametri utilizzando ogni volta una delle k fold come test set e le restanti k-1 come training. Ogni modello, quindi, è identico all'altro in termini sia di tipologia (ad esempio, regressione lineare, SVR, albero decisionale) sia di settaggi degli iperparametri, differenziando esclusivamente per il train e test set che viene utilizzato. Per ognuno dei modelli se ne calcola l'errore ottenendo, attraverso la loro media, l'errore complessivo del modello.

I risultati della k-fold cross validation sono stati ottenuti con k = 5, considerando il primo dataset, in base alla funzione R^2 e moltiplicati per 100.

Model	Cross Validation Score
SVR	-6.938
Linear Regression	-3.980
SGD Regressor	-3.713
K Neighbors Regressor	41.907
Decision Tree Regressor	60.540
Random Forest Regressor	78.856

Il valore ottimo sarebbe stato 100 poiché R^2 è stato moltiplicato per 100. Come possiamo notare l'SVR, la Linear Regression e l'SGD Regressor hanno ottenuto risultati negativi, anche peggiori delle valutazioni precedenti senza l'uso della cross validation, possiamo concludere che sono modelli non applicabili al task o al dataset considerato. Il knr ha avuto risultati positivi ma comunque non sufficienti poiché inferiori a 50. Per gli alberi di decisione c'è stato un punteggio sufficiente che è migliorato con l'uso del random forest, il quale ha raggiunto un punteggio buono pari quasi a 80.

HYPERPARAMETER TUNING

L'**hyperparameter tuning** o optimization è il problema della scelta del set di iperparametri ottimale all'algoritmo di apprendimento utilizzato. Gli iperparametri sono parametri utilizzati per controllare l'addestramento di un modello, mentre i parametri (tipicamente pesi) sono quelli addestrati durante la fase di learning per minimizzare le funzioni d'errore. Per trovare gli iperparametri migliori è stata utilizzata la **Grid search** che consiste in una ricerca esaustiva nello spazio degli iperparametri, specificati manualmente, dell'algoritmo di apprendimento. Vengono scelti alcuni valori per diversi iperparametri da ottimizzare e la Grid Search proverà tutte le combinazioni possibili di questi, ovvero addestrerà un modello per ogni possibile combinazione e, infine, ritornerà quello migliore misurando le prestazioni mediante la tecnica della Cross Validation. Poiché la ricerca esaustiva è un tipo di ricerca "a forza bruta", ha una grande complessità computazionale e quindi è difficilmente applicabile a dataset di grandi dimensioni. Proprio per questo è stata applicata al primo dataset. I modelli presi in considerazione sono stati

l'SVR e il Random Forest Regressor poiché c'era una scelta più ampia di iperparametri e ci sono stati i risultati migliori. Il valore di k preso della k-fold cross validation è pari a 3.

GRID SEARCH APPLICATA A SVR

Iperparametri presi in considerazione dalla Grid Search:

```
def hpTunintSvm(xtrain,ytrain):
 param = {'kernel' : ('linear', 'poly', 'rbf', 'sigmoid'),
 'C' : [1,5,10],
 'degree' : [3,8],
 'coef0' : [0.01,10,0.5],
 'gamma' : ('auto','scale')}

 grids = GridSearchCV(SVR(),param,cv = 3,n_jobs = -1, verbose = 2)
 grids.fit(xtrain,ytrain)
 print("Best parameter for Support Vector Regressor:\n")
 print(grids.best_params_)
```

I risultati della Grid search sono stati trovati esegendola su Visual Studio Code poiché su Colab il processo finiva in uno stato perenne di "wait" poiché necessita di molte risorse.

I migliori iperparametri trovati dalla Grid Search sono:

kernel = 'poly', gamma = 'auto', degree = 8, coef0 = 10, C = 1

Le prestazioni del SVR dopo l'hyperparameter tuning sono:

```
SVR after hyperparameter tuning
Score of Hyper Parameter Tuned SVR is: 0.44861251350779885
Accuracy for predicting price of car is 45.18069831268197 %
Mean absolute error of Hyper Parameter Tuned SVR: 186698.92007008506
Mean squared error of Hyper Parameter Tuned SVR: 152563407436.9196
Root Mean Square error of Hyper Parameter Tuned SVR: 390593.66026206774

Compare svr's hyperparameter tuning on 1st dataset:
+--------+----------------------+---------------------+--------------------+--------------------------+
| Model  | R2 score on test set | Mean absolute error | Mean squared error | Root mean squared error |
+--------+----------------------+---------------------+--------------------+--------------------------+
| Before |        -0.068        |      271198.432     |  297284446193.987  |        545237.972        |
| After  |        0.452         |      186698.920     |  152563407436.920  |        390593.660        |
+--------+----------------------+---------------------+--------------------+--------------------------+
```

Le prestazioni dell'SVR sono migliorate di molto seppur il modello resta inadeguato al task. Il coefficiente di determinazione è passato da essere negativo ad avere un valore di quasi 0.5, un buon miglioramento suppur insufficiente. Anche i valori dati dalle altre funzioni di errore sono migliorati, ad esempio il RMSE è passato da quasi 550mila a nemmeno 400mila.

GRID SEARCH APPLICATA A RANDOM FOREST REGRESSION

Iperparamenti presi in considerazione dalla Grid Search:

```
def hpTuningRf(xtrain, ytrain):
 print("\nHyper parameter tuning:")
 parameter = { 'bootstrap': [True, False],
  'max_features': ['sqrt', 'log2'],
  'min_samples_leaf': [1, 2, 4],
  'min_samples_split': [2, 5, 10],}

 gvc = GridSearchCV(RandomForestRegressor(),parameter,cv=5)
 gvc.fit(xtrain,ytrain)
 print("Best parameter for Random Forest Regressor:\n")
 print(gvc.best_params_)
```

I migliori iperparametri trovati dalla Grid Search sono:

```
Hyper parameter tuning:
Best parameter for Random Forest Regressor:

{'bootstrap': False, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2}
```

Le prestazioni del random forest regressor dopo l'hyperparameter tuning sono:

```
Score of Hyper Parameter Tuned Random Forest Regressor is: 0.9979397374843745
Accuracy for predicting price of car is 88.43380679578046 %
Mean absolute error of Hyper Parameter Tuned Random Forest Regressor: 62006.76436523652
Mean squared error of Hyper Parameter Tuned Random Forest Regressor: 32188988038.81877
Root Mean Square error of Hyper Parameter Tuned Random Forest Regressor: 179412.89819524897

Compare random forest regressor's hyperparameter tuning on 1st dataset:
+--------+----------------------+---------------------+--------------------+-------------------------+
| Model  | R2 score on test set | Mean absolute error | Mean squared error | Root mean squared error |
+--------+----------------------+---------------------+--------------------+-------------------------+
| Before |        0.883         |      87240.374      |  32437119054.722   |        180103.079       |
| After  |        0.884         |      62006.764      |  32188988038.819   |        179412.898       |
+--------+----------------------+---------------------+--------------------+-------------------------+
```

Ci sono stati miglioramenti seppur non di molto, gli errori di predizione sono diminuiti e si rivela ancora come il modello migliore applicato al task.

SVILUPPI FUTURI

Il sistema potrebbe essere migliorato ed ampliato. La base di conoscenza potrebbe essere migliorata aggiungendo più regole con l'aiuto di un ingegnere meccanico o un esperto di dominio, per applicarla a casi reali, come strumento di diagnostica. Il sistema del navigatore potrebbe essere ampliato non considerando solo la città di Bari ma una mappa più grande in cui si potrebbe navigare anche tramite un interfaccia grafica o cambiare il tipo di percorso in base a come si vuole percorrerlo, ad esempio in bici o a piedi. Il sistema di predizione di un prezzo di un veicolo

potrebbe essere migliorato aggiungendo un interfaccia in cui l’utente inserisce le caratteristiche di un veicolo che vuole acquistare o vendere e il sistema potrebbe suggerire un prezzo di partenza.