# Relazione

CORSO DI BIG DATA

Primo Progetto di

Oleksandr Pooddubnyy (533951)

Davide Fadale (547045) Gruppo: ByteBuilders

Git: https://github.com/OlekasndrPoddubnyy/BigDataFirstProject.git

# DataWhiz Enterprise

## Obiettivi del Progetto

Il progetto *DataWhiz Enterprise* nasce all’interno del corso di Big Data con l’obiettivo di realizzare un assistente intelligente in grado di comprendere richieste espresse in linguaggio naturale e tradurle automaticamente in analisi eseguibili su dataset strutturati. Il fine ultimo è semplificare l’accesso a informazioni complesse da parte di utenti non tecnici, offrendo risposte precise e leggibili, supportate da una solida infrastruttura di elaborazione dati.

Il sistema è pensato per operare in un contesto realistico, in cui l’utente può porre domande come *"Mostrami il reparto con più turnover"* oppure *"Quali sono i prodotti più acquistati nell’e-commerce?"*. Tali richieste vengono interpretate semanticamente, collegate ai dataset pertinenti e trasformate in job Spark che restituiscono i risultati richiesti.

Un obiettivo centrale è garantire la massima modularità del progetto. Ogni componente ha una responsabilità ben definita: dal parser semantico che comprende l’intento dell’utente, al generatore di job Spark, fino al modulo di esecuzione e alla presentazione dei risultati. Il sistema è progettato per supportare più domini applicativi, tra cui risorse umane, vendite e attività e-commerce. È inoltre prevista la possibilità di combinare più dataset in un’unica analisi, rendendo l’assistente utile anche in scenari più complessi che richiedono incroci tra fonti differenti.

Il progetto supporta due modalità operative. In modalità *test*, tutte le analisi sono predefinite e vengono eseguite offline, senza il coinvolgimento dell’intelligenza artificiale. Questo consente di verificare la correttezza delle pipeline e l'integrità dei dati in un contesto controllato. In modalità *produzione*, invece, l’utente interagisce direttamente con l’assistente attraverso una shell: la richiesta viene analizzata da un modello GPT-4o che genera automaticamente il codice Spark necessario per produrre l’analisi desiderata.

Particolare attenzione è stata riservata alla chiarezza dei risultati. Oltre alla classica visualizzazione tabellare tramite Spark, l’utente può richiedere una spiegazione testuale del risultato, elaborata da un modello linguistico avanzato. Questo rende il sistema accessibile anche a chi non ha competenze tecniche specifiche in ambito Big Data.

Infine, tra i requisiti fondamentali vi era la possibilità di eseguire tutto il sistema in locale, senza dipendenze da servizi esterni se non per la componente AI. Il progetto doveva inoltre evitare di diventare un semplice traduttore da linguaggio naturale a SQL: l’obiettivo era creare una piattaforma più articolata, capace di selezionare ed eseguire job Spark costruiti appositamente per ogni dominio, con possibilità di estensione e riuso futuro.

## Architettura e tecnologie utilizzate

L’architettura del progetto *DataWhiz Enterprise* è pensata per garantire modularità, estendibilità e una chiara separazione delle responsabilità tra i componenti. Il sistema è organizzato in più livelli funzionali, ognuno con un ruolo specifico: dall’interazione con l’utente alla generazione del codice Spark, fino all’esecuzione e restituzione dei risultati. La progettazione è stata guidata dall’esigenza di supportare l’esecuzione sia in ambienti locali che su infrastrutture distribuite.

### Architettura generale

Il sistema si compone dei seguenti macro-componenti:

**Interfaccia utente (CLI)**: l’utente interagisce con il sistema tramite terminale. In modalità *test*, vengono eseguite domande predefinite; in modalità *produzione*, l’utente può porre domande libere in linguaggio naturale.

**Parser semantico (NLP)**: la domanda dell’utente viene interpretata e tradotta in un’intent JSON strutturato, contenente *dominio*, *intento* e *filtri*. In modalità offline, questo processo è realizzato tramite regole statiche; in produzione, ci si affida a modelli GPT-4o tramite le API OpenAI.

**Query builder**: a partire dall’intent JSON, viene generato il codice Spark necessario a eseguire la richiesta. Il builder sceglie se utilizzare job statici (per richieste note) oppure generare dinamicamente codice tramite AI.

**Esecutore Spark**: esegue il codice prodotto all’interno di una sessione Spark configurata localmente. Il codice restituisce un DataFrame, che può essere mostrato a schermo oppure descritto in linguaggio naturale.

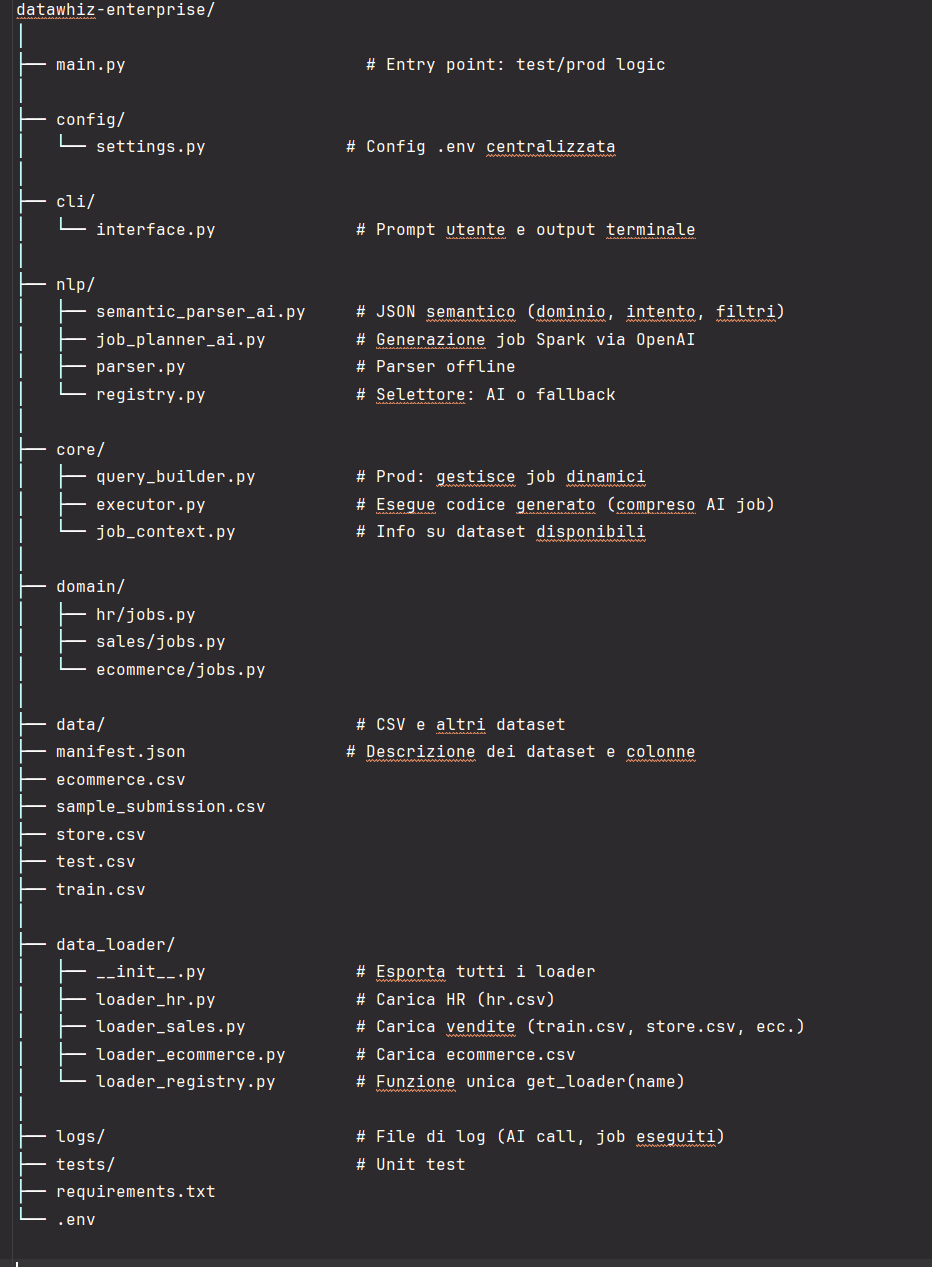
**Risposta AI (output arricchito)**: quando richiesto, il sistema può fornire una spiegazione testuale del risultato, sempre tramite GPT-4o. In questo modo, anche un utente non esperto può comprendere il significato dell’output.

**Caricamento dati**: ogni dominio dispone di un loader dedicato, che carica i dataset dal disco e li pre-elabora. La presenza di un manifest centralizzato (manifest.json) consente al sistema di conoscere struttura e scopo di ogni dataset disponibile.

### Struttura del progetto

Il progetto è suddiviso nei seguenti pacchetti principali:

* + - main.py: punto di ingresso, gestisce modalità test/prod.
    - cli/: interazione da terminale.
    - nlp/: parsing semantico e generazione job via AI.
    - core/: query builder e motore di esecuzione.
    - domain/: job Spark per ciascun dominio (es. HR, vendite).
    - data\_loader/: moduli per il caricamento dei dataset.
    - config/: impostazioni centralizzate (.env, Spark, API key).
    - data/: cartella contenente i dataset CSV.
    - logs/ e tests/: directory dedicate a logging e testing.



### Tecnologie utilizzate

Le principali tecnologie utilizzate sono:

* **Python 3.10**: linguaggio principale, scelto per la sua flessibilità e vasta disponibilità di librerie per il data processing.
* **Apache Spark (PySpark)**: motore di elaborazione distribuito, utilizzato per processare dataset di grandi dimensioni.
* **OpenAI GPT-4o**: modello linguistico usato per interpretare richieste in linguaggio naturale e generare codice Spark personalizzato.
* **Pandas**: utilizzato in modo accessorio per convertire e sintetizzare risultati Spark da mostrare in formato JSON.
* **dotenv**: per la gestione delle variabili di ambiente in modo sicuro.
* **Windows e Linux (compatibilità)**: il progetto è stato sviluppato per funzionare sia su ambienti Windows (con winutils.exe) che su Linux/WSL, rendendo il sistema più portabile.

### Caricamento e gestione dei dati

Ogni dominio (es. *hr*, *sales*, *ecommerce*) è supportato da un modulo dedicato nel package data\_loader/. I loader leggono i file CSV da data/, li pre-elaborano in DataFrame e li rendono disponibili agli script Spark in modo trasparente. La configurazione di dataset e metadati (colonne, descrizione, percorso) è centralizzata in manifest.json.

## Risultati

L’applicazione *DataWhiz Enterprise* è stata testata in modalità sia offline (test) che interattiva (produzione), dimostrando la sua capacità di interpretare domande espresse in linguaggio naturale, trasformarle in job Spark validi e restituire risposte precise, sia in forma tabellare sia in forma testuale semplificata. I risultati ottenuti evidenziano la solidità dell’approccio modulare e l’efficacia dell’integrazione tra Spark e modelli di intelligenza artificiale.

### Modalità test (offline)

In modalità test, il sistema esegue una serie di domande predefinite per verificare la correttezza dell’intero flusso di parsing, generazione ed esecuzione. Esempi di domande e risultati ottenuti includono:

#### "Mostrami il reparto con più turnover"

Il sistema ha correttamente identificato l’intento come una richiesta HR e ha generato un job Spark per analizzare la colonna Attrition per reparto. Il risultato mostrava il reparto con il maggior numero di abbandoni registrati.

#### "Mostrami le vendite del 2015 per ogni negozio"

In questo caso, il sistema ha incrociato i dati tra train.csv e store.csv per restituire una tabella aggregata delle vendite per negozio nel solo anno 2015.

#### "Quali sono i prodotti più acquistati"

Utilizzando ecommerce.csv, è stato effettuato il conteggio degli eventi di tipo

“acquisto” per prodotto, fornendo una classifica ordinata dei prodotti più popolari.

In tutti i casi, il sistema ha restituito risultati coerenti, stampati in formato tabellare Spark, a conferma della correttezza dell’architettura e della logica di job planning statica.

### Modalità interattiva (produzione)

La modalità interattiva rappresenta l’aspetto più innovativo e dinamico del progetto

*DataWhiz Enterprise*. In questa configurazione, l’applicazione non lavora più su un insieme di domande predefinite, ma attiva un’interfaccia testuale (shell) che consente all’utente di porre **domande libere in linguaggio naturale**, come se interagisse con un assistente intelligente.

#### Flusso operativo

Il processo si articola in più fasi, orchestrate automaticamente:

#### Input dell’utente

L’utente digita una domanda aperta, ad esempio:

* Qual è il reparto con il turnover più alto?

#### Parsing semantico della domanda

Il modulo NLP analizza la domanda utilizzando il modello GPT-4o, che restituisce un intent strutturato in formato JSON, contenente:

* + il **dominio** (es. hr, sales, ecommerce),
  + l’**intento** (es. turnover\_analysis, top\_sales\_2015),
  + eventuali **filtri** (es. year: 2015).

#### Generazione del job Spark

In base all’intent ricevuto, un secondo prompt viene costruito per chiedere nuovamente a GPT-4o di generare un **codice Spark valido**, capace di ottenere il risultato desiderato a partire dai dataset disponibili. Il modello è aiutato da una descrizione dettagliata dei dataset (estratta da manifest.json), così da creare job pertinenti e ottimizzati.

#### Esecuzione del job Spark

Il codice Spark generato viene eseguito localmente tramite exec, in un contesto controllato in cui i DataFrame sono già precaricati (grazie ai loader definiti per ciascun dominio). Il

risultato della computazione viene assegnato alla variabile result.

#### Scelta della forma di risposta

Una volta ottenuto il risultato, l’utente può scegliere:

**Risposta semplice (1)**: viene mostrata la tabella risultante direttamente tramite result.show() (formato Spark).

**Risposta elaborata (2)**: il contenuto della tabella viene convertito in dizionario Python

(toPandas().to\_dict(...)) e passato di nuovo a GPT-4o, che genera una spiegazione **testuale e semplificata**, adatta anche a utenti non tecnici.

#### Interazione successiva

Dopo la risposta, l’utente può decidere se:

rivedere lo stesso risultato in formato diverso (passare da semplice a elaborato o viceversa) o porre una nuova domanda.

## Conclusioni

Il progetto *DataWhiz Enterprise* ha rappresentato un’iniziativa ambiziosa ma estremamente formativa, in cui sono stati integrati in modo armonico diversi aspetti dell’ingegneria dei dati, dell’analisi su larga scala e dell’intelligenza artificiale.

#### Versatilità e scalabilità

Uno degli aspetti più rilevanti emersi è la **versatilità** della piattaforma: grazie alla

distinzione tra modalità **test** e modalità **produzione**, l'applicazione è risultata facilmente testabile in ambienti controllati, ma al tempo stesso estensibile e pronta per contesti più realistici, dove l'interazione è dinamica e imprevedibile. La scelta di utilizzare **Apache Spark** come motore di elaborazione ha garantito la possibilità di gestire anche dataset voluminosi e distribuiti, mantenendo performance solide anche su macchine locali.

#### Ruolo dell’intelligenza artificiale

L’integrazione di **modelli linguistici avanzati** (in particolare GPT-4o) ha aggiunto un livello completamente nuovo all’esperienza utente. Non solo l’utente può porre domande

complesse senza conoscere linguaggi di interrogazione, ma ottiene anche **risposte testuali elaborate** che spiegano i risultati in modo comprensibile. Questo approccio trasforma l’analisi dati da un’attività puramente tecnica a una **forma di conversazione intelligente con il dato**.

#### Manutenibilità e modularità

Dal punto di vista tecnico, la struttura del progetto è stata pensata in modo **modulare** e

facilmente estendibile: ciascun dominio (hr, sales, ecommerce) ha un proprio modulo con i job specifici, mentre i loader sono separati per garantire coerenza nel caricamento dei dataset. Le componenti sono isolate in cartelle logiche (e.g. core, nlp, client), e i parametri di configurazione sono centralizzati nel file .env.

#### Limitazioni e sviluppi futuri

Nonostante i risultati positivi, si segnalano alcune **possibili migliorie**:

* L’attuale modalità AI è basata su prompt generativi: in scenari produttivi, potrebbe essere utile **raffinare il prompting** o integrare validazioni automatiche del codice generato.
* Manca un'interfaccia grafica: un'estensione Web permetterebbe di rendere lo strumento più accessibile a profili non tecnici.
* Alcuni job complessi potrebbero beneficiare di **ottimizzazioni Spark avanzate**, come l'uso di cache, broadcast join o salvataggio intermedio dei risultati.
* Infine, l’inclusione di **analytics avanzate o machine learning** aprirebbe scenari predittivi interessanti (es. previsione del turnover o delle vendite).

#### Conclusione

In sintesi, *DataWhiz Enterprise* dimostra come sia possibile coniugare l’efficienza del

calcolo distribuito con l’intelligenza conversazionale, creando uno strumento accessibile, potente e potenzialmente pronto per essere utilizzato anche in contesti reali. È un esempio concreto di come **Big Data e AI** possano collaborare per semplificare l’accesso alla

conoscenza contenuta nei dati.

## Evoluzioni Future

Il progetto *DataWhiz Enterprise*, nella sua forma attuale, fornisce una base solida per l’interazione tra utenti e sistemi di analisi dati distribuiti, ma presenta ampi margini di evoluzione. Un primo asse di sviluppo riguarda le **performance computazionali**: attualmente i job Spark generati sono funzionali ma non ottimizzati. Un'evoluzione naturale prevede l’integrazione di un **ottimizzatore fisico** a valle della generazione del codice, capace di introdurre automaticamente strategie come il caching di DataFrame intermedi, il broadcast di tabelle piccole e l’uso di partizionamenti intelligenti per ridurre lo shuffle.

Un secondo asse fondamentale è lo sviluppo di una **interfaccia web interattiva**, che sostituisca l'attuale terminale testuale. Questo frontend permetterebbe non solo di scrivere domande, ma anche di **navigare tra i dataset**, **visualizzare grafici interattivi**, **esportare risultati** e **salvare le sessioni utente**. L’aggiunta di una dashboard richiederebbe un backend asincrono (es. con FastAPI) e un frontend moderno (React, Vue, o Svelte) collegato ai componenti Spark in locale. In questo contesto, il sistema potrebbe evolversi in una piattaforma "low-code" per l’analisi aziendale intelligente, accessibile anche a utenti non tecnici.

Infine, per scenari più ambiziosi, il progetto potrebbe supportare l’integrazione di **fonti dati esterne** (es. PostgreSQL, API REST, data lake) e **moduli di pianificazione automatica** per la creazione di report periodici, rendendo *DataWhiz* uno strumento versatile per la business intelligence assistita da AI.