1. Importazione delle librerie ¶

In [ ]:
```
!pip install -U langchain-openai langchain &>/dev/null
```

In [ ]:
```
import os
os.environ["OPENAI_API_KEY"] = "**********************"
```

In [ ]:
```
import sqlite3
import pandas as pd
from datetime import datetime, timedelta
import random
from langchain_openai import ChatOpenAI
from langchain.prompts import PromptTemplate
from IPython.display import display, Markdown
```

2. Connessione con SQLite Database

In [ ]:
```
# Connessione a SQLite database
conn = sqlite3.connect('attuari.db')
cursor = conn.cursor()
```

3. Creazione delle tabelle

In [ ]:
```
cursor.execute('''
DROP TABLE IF EXISTS polizze;
''')
# Creazione tabella polizze
cursor.execute('''
CREATE TABLE polizze (
    numero_polizza INTEGER PRIMARY KEY,
    prodotto TEXT,
    data_decorrenza DATE,
    data_scadenza DATE
);
''')

cursor.execute('''
DROP TABLE IF EXISTS riserve;
''')
# Creazione tabella riserve
cursor.execute('''
CREATE TABLE riserve (
    numero_polizza INTEGER PRIMARY KEY,
    importo_riserva NUMERIC,
    data_riserva DATE
);
''')

conn.commit()
```

In [ ]:
```
pd.read_sql_query("SELECT * FROM polizze;", conn)
```

Out[6]:

numero_polizza	prodotto	data_decorrenza	data_scadenza

In [ ]:
```
pd.read_sql_query("SELECT * FROM riserve;", conn)
```

Out[7]:

numero_polizza	importo_riserva	data_riserva

4. Generazione e inserimento dei dati

In [ ]:
```
random.seed(0)

# generazione di 100 polizze
numero_polizza = list(range(1000, 1100))

# Creazione del dataframe polizze
prodotti = ["CAPITALE","MISTA"]
data_iniziale = datetime(2020, 1, 1)
polizza = []
for i in numero_polizza:
    prodotto = random.choice(prodotti)
    data_decorrenza = data_iniziale + timedelta(days=random.randint(0, 90)
    data_scadenza = data_decorrenza + timedelta(days=365*8)
    polizza.append((i, prodotto, data_decorrenza.date(), data_scadenza.date

# Inserimento dei dati in tabella polizze
cursor.executemany('''
INSERT INTO polizze (numero_polizza, prodotto, data_decorrenza, data_scader
VALUES (?, ?, ?, ?)
''', polizza)
conn.commit()
```

In [ ]:
```
pd.read_sql_query("SELECT *  FROM polizze;", conn)
```

Out[9]:

	numero_polizza	prodotto	data_decorrenza	data_scadenza
0	1000	MISTA	2020-02-23	2028-02-21
1	1001	CAPITALE	2020-02-03	2028-02-01
2	1002	MISTA	2020-02-21	2028-02-19
3	1003	MISTA	2020-03-02	2028-02-29
4	1004	MISTA	2020-03-15	2028-03-13
...	...	...	...	...
95	1095	CAPITALE	2020-02-28	2028-02-26
96	1096	CAPITALE	2020-02-12	2028-02-10
97	1097	CAPITALE	2020-03-10	2028-03-08
98	1098	MISTA	2020-01-18	2028-01-16
99	1099	CAPITALE	2020-03-02	2028-02-29

100 rows × 4 columns

In [ ]:
```
# Creazione dataframe riserve
riserva = []
for i in numero_polizza:
  importo_riserva = round(random.uniform(1000.00, 100000.00), 2)
  data_riserva = data_iniziale + timedelta(days=365*5)
  riserva.append((i, importo_riserva, data_riserva.date()))

# Inserimento dei dati in tabella riserve
cursor.executemany('''
INSERT INTO riserve (numero_polizza, importo_riserva, data_riserva)
VALUES (?, ?, ?)
''', riserva)
conn.commit()
```

In [ ]:
```
pd.read_sql_query("SELECT * FROM riserve", conn)
```

Out[11]:

	numero_polizza	importo_riserva	data_riserva
0	1000	35870.33	2024-12-30
1	1001	29499.91	2024-12-30
2	1002	36560.92	2024-12-30
3	1003	94743.68	2024-12-30
4	1004	63741.04	2024-12-30
...	...	...	...
95	1095	34487.34	2024-12-30
96	1096	62433.27	2024-12-30
97	1097	5079.09	2024-12-30
98	1098	17222.19	2024-12-30
99	1099	98209.49	2024-12-30

100 rows × 3 columns

Prompt Engineering per la programmazione

Utilizzo di LangChain e GPT-4o-mini per creare una query di collegamento tra tabelle

In [ ]:
```
# Definizione del prompt
prompt = PromptTemplate(
    input_variables=["descrizione"],
    template="""
Sei un esperto analista in SQL. Scrivi una query in SQL per collegare le du

Tabelle:
1. polizze (numero_polizza INTEGER PRIMARY KEY, prodotto TEXT, data_decorre
2. riserve (numero_polizza INTEGER PRIMARY KEY, importo_riserva NUMERIC, da

Regole:
{descrizione}

SQL Query:
"""
)

chat = ChatOpenAI(temperature=0, model_name="gpt-4o-mini")
chain = prompt | chat
regole = "Recupera tutte le polizze con tutti i dati dalle tabelle."

sql_query = chain.invoke({"descrizione": regole})

display(Markdown(sql_query.content))
```

```
<IPython.core.display.Markdown object>
```

Esecuzione SQL query di collegamento tabelle

In [ ]:
```
pd.read_sql_query(
'''
SELECT
    p.numero_polizza,
    p.prodotto,
    p.data_decorrenza,
    p.data_scadenza,
    r.importo_riserva,
    r.data_riserva
FROM
    polizze p
LEFT JOIN
    riserve r ON p.numero_polizza = r.numero_polizza;
''', conn)
```

Out[13]:

	numero_polizza	prodotto	data_decorrenza	data_scadenza	importo_riserva	data_riserv
0	1000	MISTA	2020-02-23	2028-02-21	35870.33	2024-12-3
1	1001	CAPITALE	2020-02-03	2028-02-01	29499.91	2024-12-3
2	1002	MISTA	2020-02-21	2028-02-19	36560.92	2024-12-3
3	1003	MISTA	2020-03-02	2028-02-29	94743.68	2024-12-3
4	1004	MISTA	2020-03-15	2028-03-13	63741.04	2024-12-3
...	...	...	...	...	...	
95	1095	CAPITALE	2020-02-28	2028-02-26	34487.34	2024-12-3
96	1096	CAPITALE	2020-02-12	2028-02-10	62433.27	2024-12-3
97	1097	CAPITALE	2020-03-10	2028-03-08	5079.09	2024-12-3
98	1098	MISTA	2020-01-18	2028-01-16	17222.19	2024-12-3
99	1099	CAPITALE	2020-03-02	2028-02-29	98209.49	2024-12-3

100 rows × 6 columns

Prompt Engineering per il debugging

In [ ]:
```
pd.read_sql_query(
'''
SELECT p.prodotto, SUM(r.importo_riserva) as riserva, r.data_riserva
FROM polizze
INNER JOIN riserve ON p.numero_polizza = r.numero_polizza
GROUP BY p.prodotto,r.data_riserva
''', conn)
```

```
---------------------------------------------------------------------------
--
OperationalError                          Traceback (most recent call las
t)
/usr/local/lib/python3.11/dist-packages/pandas/io/sql.py in execute(self,
sql, params)
   2673         try:
-> 2674             cur.execute(sql, *args)
   2675             return cur

OperationalError: no such column: p.prodotto

The above exception was the direct cause of the following exception:

DatabaseError                             Traceback (most recent call las
t)
<ipython-input-14-75eddf6be234> in <cell line: 0>()
----> 1 pd.read_sql_query(
      2 '''
      3 SELECT p.prodotto, SUM(r.importo_riserva) as riserva, r.data_rise
rva
      4 FROM polizze
      5 INNER JOIN riserve ON p.numero_polizza = r.numero_polizza

/usr/local/lib/python3.11/dist-packages/pandas/io/sql.py in read_sql_quer
y(sql, con, index_col, coerce_float, params, parse_dates, chunksize, dtyp
e, dtype_backend)
    524
    525     with pandasSQL_builder(con) as pandas_sql:
--> 526         return pandas_sql.read_query(
    527             sql,
    528             index_col=index_col,

/usr/local/lib/python3.11/dist-packages/pandas/io/sql.py in read_query(se
lf, sql, index_col, coerce_float, parse_dates, params, chunksize, dtype,
dtype_backend)
   2736         dtype_backend: DtypeBackend | Literal["numpy"] = "numpy",
   2737     ) -> DataFrame | Iterator[DataFrame]:
-> 2738         cursor = self.execute(sql, params)
   2739         columns = [col_desc[0] for col_desc in cursor.descriptio
n]
   2740

/usr/local/lib/python3.11/dist-packages/pandas/io/sql.py in execute(self,
sql, params)
   2684
   2685             ex = DatabaseError(f"Execution failed on sql '{sql}':
{exc}")
-> 2686             raise ex from exc
   2687
   2688     @staticmethod

DatabaseError: Execution failed on sql '
SELECT p.prodotto, SUM(r.importo_riserva) as riserva, r.data_riserva
FROM polizze
INNER JOIN riserve ON p.numero_polizza = r.numero_polizza
GROUP BY p.prodotto,r.data_riserva
': no such column: p.prodotto
```

Utilizzo di LangChain e GPT-4o-mini per risolvere un problema nell'esecuzione di una query

In [ ]:
```
# Definizione del prompt
prompt = PromptTemplate(
    input_variables=["descrizione"],
    template="""
Sei un esperto analista in SQL.
Fornisci la query corretta in SQL per collegare le due tabelle con raggrupp
problema illustrato nella descrizione fornita.

Tabelle:
1. polizze (numero_polizza INTEGER PRIMARY KEY, prodotto TEXT, data_decorre
2. riserve (numero_polizza INTEGER PRIMARY KEY, importo_riserva NUMERIC, da

Regole:
{descrizione}

SQL Query:
"""
)

chat = ChatOpenAI(temperature=0, model_name="gpt-4o-mini")
chain = prompt | chat

regole = """
Ho eseguito la seguente query per effettuare un raggruppamento della riserv
""
pd.read_sql_query(
'''
SELECT p.prodotto, SUM(r.importo_riserva) as riserva, r.data_riserva
FROM polizze
INNER JOIN riserve ON p.numero_polizza = r.numero_polizza
GROUP BY p.prodotto,r.data_riserva
''', conn)
""
ho il seguente messaggio di errore:
""
DatabaseError: Execution failed on sql '
SELECT p.prodotto, SUM(r.importo_riserva) as riserva, r.data_riserva
FROM polizze
INNER JOIN riserve ON p.numero_polizza = r.numero_polizza
GROUP BY p.prodotto,r.data_riserva
': no such column: p.prodotto
""
L'obiettivo è recuperare i dati dalle tabelle e fare un raggruppamento per
I campi da vedere sono prodotto, riserva e data riserva"""

sql_query = chain.invoke({"descrizione": regole})

display(Markdown(sql_query.content))
```

```
<IPython.core.display.Markdown object>
```

Esecuzione SQL query di collegamento tabelle e raggruppamento

In [ ]:
```
pd.read_sql_query(
'''
SELECT polizze.prodotto, SUM(riserve.importo_riserva) AS riserva, MAX(riser
FROM polizze
INNER JOIN riserve ON polizze.numero_polizza = riserve.numero_polizza
GROUP BY polizze.prodotto
''', conn)
```

Out[16]:

	prodotto	riserva	data_riserva
0	CAPITALE	2636216.29	2024-12-30
1	MISTA	2448529.81	2024-12-30

Prompt Engineering per la documentazione

Utilizzo di LangChain e GPT-4o-mini per documentare il codice

In [ ]:
```
# Definizione del prompt
prompt = PromptTemplate(
    input_variables=["descrizione"],
    template="""
Sei un esperto analista in SQL. L'obiettivo è scrivere la documentazione di

Tabelle:
1. polizze (numero_polizza INTEGER PRIMARY KEY, prodotto TEXT, data_decorre
2. riserve (numero_polizza INTEGER PRIMARY KEY, importo_riserva NUMERIC, da

Regole:
{descrizione}

SQL Query:
"""
)

chat = ChatOpenAI(temperature=0, model_name="gpt-4o-mini")
chain = prompt | chat
regole = """
Ti chiedo di documentare linea per linea la seguente query scritta in SQL:
""
cursor.execute('''
DROP TABLE IF EXISTS polizze;
''')
# Creazione tabella polizze
cursor.execute('''
CREATE TABLE polizze (
    numero_polizza INTEGER PRIMARY KEY,
    prodotto TEXT,
    data_decorrenza DATE,
    data_scadenza DATE
);
''')
cursor.execute('''
DROP TABLE IF EXISTS riserve;
''')
# Creazione tabella riserve
cursor.execute('''
CREATE TABLE riserve (
    numero_polizza INTEGER PRIMARY KEY,
    importo_riserva NUMERIC,
    data_riserva DATE
);
''')
conn.commit()
""
"""
sql_query = chain.invoke({"descrizione": regole})
display(Markdown(sql_query.content))
```

```
<IPython.core.display.Markdown object>
```

Prompt Engineering per la traduzione

Utilizzo di LangChain e GPT-4o-mini per effetuare traduzione di codice da R a Python

In [ ]:
```
# Definizione del prompt
prompt = PromptTemplate(
    input_variables=["descrizione"],
    template="""
Sei un esperto actuarial data scientist. Converti il codice da R in Python,

Dataframe:
eta_assicurato da 18 a 79 anni
eta_veicolo da 0 a 9 anni
esposizione da 0 a 1 anno
premio segue una distribuzione gamma

Regole:
{descrizione}

SQL Query:
"""
)
chat = ChatOpenAI(temperature=0, model_name="gpt-4o-mini")
chain = prompt | chat
regole = """
Puoi tradurre il codice utilizzato per creare un dataframe con R in Python
Il codice impiegato è il seguente:
set.seed(123)
n_rows <- 3000
eta_assicurato <- sample(18:79, size = n_rows, replace = TRUE)
eta_veicolo <- sample(0:9, size = n_rows, replace = TRUE)
esposizione <- round(runif(n_rows, min = 0, max = 1), 2)
premio <- round(rgamma(n_rows, shape = 2, rate = 1/100), 2)

df <- data.frame(
  eta_assicurato = eta_assicurato,
  eta_veicolo = eta_veicolo,
  esposizione = esposizione,
  premio = premio
)

"""

sql_query = chain.invoke({"descrizione": regole})

display(Markdown(sql_query.content))
```

```
<IPython.core.display.Markdown object>
```

Esecuzione codice in Python per creazione dataset

In [ ]:
```
import pandas as pd
import numpy as np
from scipy.stats import gamma

# Imposta riproducibilità
np.random.seed(123)

# Numero di righe
n_rows = 3000

# Genera i dati
eta_assicurato = np.random.choice(range(18, 80), size=n_rows, replace=True)
eta_veicolo = np.random.choice(range(0, 10), size=n_rows, replace=True)
esposizione = np.round(np.random.uniform(0, 1, size=n_rows), 2)
premio = np.round(gamma.rvs(a=2, scale=100, size=n_rows), 2)

# Crea il DataFrame
df = pd.DataFrame({
    'eta_assicurato': eta_assicurato,
    'eta_veicolo': eta_veicolo,
    'esposizione': esposizione,
    'premio': premio
})

# Visualizza le prime righe del DataFrame
print(df.head())
```

```
   eta_assicurato  eta_veicolo  esposizione  premio
0              63            8         0.91   62.57
1              20            7         0.35   29.48
2              46            0         0.61  489.75
3              52            3         0.19   70.91
4              56            3         0.04   76.60
```

In [ ]:
```
cursor.close()
conn.close()
```

Riferimenti

In [ ]:
```
# https://platform.openai.com/docs/overview
# https://python.langchain.com/api_reference/openai/chat_models/langchain_
# https://python.langchain.com/docs/integrations/chat/openai/
# https://pypi.org/project/langchain-openai/
```