Building A Dialogue Feature Extraction Pipeline Using Function Calling!

```
sample_data = \
"""Agent: Thank you for calling BrownBox Customer Support. My name is Tom. How may I assist you today?\nCustomer: Hi Tom, I'm tryin
```

```
print (sample_data)
```

```
Agent: Thank you for calling BrownBox Customer Support. My name is Tom. How may I assist you today?
Customer: Hi Tom, I'm trying to log in to my account to purchase an Oven Toaster Grill (OTG), but I'm unable to proceed as it's ask
Agent: Sure, I can assist you with that. May I know your registered mobile number or email address, please?
Customer: My registered mobile number is +1 123-456-7890.
Agent: Thank you. Let me check that for you. I'm sorry to inform you that we don't have this number on our records. Can you please
Customer: Oh, I'm sorry. I might have registered with a different number. Can you please check with my email address instead? It's
Agent: Sure, let me check that for you. (After a few moments) I see that we have your email address on our records. We'll be sendin
Agent: Please enter the verification code in the field provided and click on 'Verify'. Once your email address is verified, you'll
Customer: Okay, I entered the code, and it's verified now. Thank you for your help.
Agent: You're welcome. Is there anything else I can assist you with?
Customer: No, that's all. Thank you.
Agent: You're welcome. Have a great day!
```

Defining What's Important

```
from utils import query_raven
from typing import List
from dataclasses import dataclass
# Warning control
import warnings
warnings.filterwarnings('ignore')
```

```
from dataclasses import dataclass
schema_id = ("agent_name", "customer_email", \
             "customer_order", "customer_phone", "customer_sentiment")

dataclass_schema_representation = '''
@dataclass
class Record:
    agent_name : str # The agent name
    customer_email : str # customer email if provided, else ''
    customer_order : str # The customer order number if provided, else ''
    customer_phone : str # the customer phone number if provided, else ''
    customer_sentiment : str # Overall customer sentiment, either 'frustrated', or 'happy'. Always MUST have a value.
'''

# Let's call exec to insert the dataclass into our python interpreter so it understands this.
exec(dataclass_schema_representation)
```

Building The Database

```
def initialize_db():
    import sqlite3

    # Connect to SQLite database (or create it if it doesn't exist)
    conn = sqlite3.connect('extracted.db')
    cursor = conn.cursor()

    # Fixed table name
    table_name = "customer_information"

    # Fixed schema
    columns = """
    id INTEGER PRIMARY KEY,
    agent_name TEXT,
    customer_email TEXT,
    customer_order TEXT,
    customer_phone TEXT,
    customer_sentiment TEXT
    """
```

```
        # Ensure the table name is enclosed in quotes if it contains special characters
        quoted_table_name = f'"{table_name}"'

        # Check if a table with the exact name already exists
        cursor.execute(f"SELECT name FROM sqlite_master WHERE type='table' AND name={quoted_table_name}")
        if cursor.fetchone():
            print(f"Table {table_name} already exists.")
        else:
            # Create the new table with the fixed schema
            cursor.execute(f'''CREATE TABLE {quoted_table_name} ({columns})''')
            print(f"Table {table_name} created successfully.")

        # Commit the transaction and close the connection
        conn.commit()
        conn.close()
```

```
    !rm extracted.db
    initialize_db()
```

```
    Table customer_information created successfully.
```

Adding in Tools To Populate The Database

```
    from dataclasses import dataclass, fields
    def update_knowledge(results_list : List[Record]):
        """
        Registers the information necessary
        """
        import sqlite3
        from sqlite3 import ProgrammingError

        # Reconnect to the existing SQLite database
        conn = sqlite3.connect('extracted.db')
        cursor = conn.cursor()

        # Fixed table name
        table_name = "customer_information"

        # Prepare SQL for inserting data with fixed column names
        column_names = "agent_name, customer_email, customer_order, customer_phone, customer_sentiment"
        placeholders = ", ".join(["?"] * 5)
        sql = f"INSERT INTO {table_name} ({column_names}) VALUES ({placeholders})"

        # Insert each record
        for record in results_list:
            try:
                record_values = tuple(getattr(record, f.name) for f in fields(record))
                cursor.execute(sql, record_values)
            except ProgrammingError as e:
                print(f"Error with record. {e}")
                continue

        # Commit the changes and close the connection
        conn.commit()
        conn.close()
        print("Records inserted successfully.")
```

```
    my_record = Record(agent_name = "Agent Smith", \
                       customer_email = "", customer_order = "12346", \
                       customer_phone = "", customer_sentiment = "happy")
```

```
    update_knowledge([my_record])
```

```
    Records inserted successfully.
```

Building Tools To Pull Information Out

```
    import sqlite3
    def execute_sql(sql: str):
        """ Runs SQL code for the given schema. Make sure to properly leverage the schema to answer the user's question in the best wa
    # Fixed table name, assuming it's not dynamically generated anymore
    table_name = "customer_information"

    # Establish a connection to the database
    conn = sqlite3.connect('extracted.db')
    cursor = conn.cursor()

    # Execute the SQL statement
    cursor.execute(sql)

    # Initialize an empty list to hold query results
    results = []

    results = cursor.fetchall()
    print("Query operation executed successfully. Number of rows returned:", len(results))

    # Close the connection to the database
    conn.close()

    # Return the results for SELECT operations; otherwise, return an empty list
    return results
```

```
sql = '''
    SELECT agent_name
        FROM customer_information
        WHERE customer_sentiment = "happy"
    '''
# Print the final SQL command for debugging
print("Executing SQL:", sql)

execute_sql(sql)
```

```
Executing SQL:
    SELECT agent_name
        FROM customer_information
        WHERE customer_sentiment = "happy"

Query operation executed successfully. Number of rows returned: 1
[('Agent Smith',)]
```

Building The Pipeline

```
!rm extracted.db
initialize_db()
```

```
Table customer_information created successfully.
```

Attribution:

We will be using a handful of samples (~10-15 samples) in this lesson from a publically-available customer_service_chatbot on HuggingFace. The link to the public dataset is here: https://huggingface.co/datasets/SantiagoPG/customer_service_chatbot

```
from datasets import load_dataset
import os

cwd = os.getcwd()
dialogue_data = load_dataset(cwd + "/data/customer_service_chatbot", cache_dir="./cache")["train"]
```

```
Generating train split: 0 examples [00:00, ? examples/s]
```

```
sample_zero = dialogue_data[6]
dialogue_string = sample_zero["conversation"].replace("\n\n", "\n")
print (dialogue_string)
```

```
Agent: Hello, thank you for contacting BrownBox customer support. My name is Alex, how can I assist you today?
Customer: Hi, I'm calling about my order for a water purifier. I received it yesterday, but it's not working correctly. I want to r
Agent: I'm sorry to hear that. I'll be happy to help you with that. Can you please provide me with your order number?
Customer: Sure, it's 12345.
Agent: Thank you for the information. May I know the reason for the return?
```

```
Customer: As I mentioned earlier, the product is not working correctly. I want to return it and get a refund.
Agent: I'm sorry for the inconvenience. We would be happy to process your return and refund. However, since you have opted for Cash
Customer: What? That's too long. Why does it take so much time?
Agent: I understand your frustration, but the refund process takes time as we have to verify the product's condition and ensure tha
Customer: This is unacceptable. I need the refund immediately. Can't you do anything about it?
Agent: I'm sorry, but we cannot expedite the refund process. However, I can assure you that we will process your refund as soon as
Customer: Can you at least tell me the status of my refund?
Agent: Sure, I can check the status of your refund. Please allow me a moment to check that for you.
(Customer is put on hold for a few minutes)
Agent: Thank you for waiting. I have checked your refund status, and I see that your return has been received by our team. The refu
Customer: Alright, I understand. Is there anything else I need to do?
Agent: No, you don't have to do anything else. Our team will process your refund, and you will receive an email confirmation once i
Customer: Okay, thank you for your help.
Agent: You're welcome. I apologize for the inconvenience caused. Is there anything else I can assist you with?
Customer: No, that's all.
Agent: Alright, please feel free to contact us if you have any further questions or concerns. Have a great day!
Customer: You too. Bye.
Agent: Goodbye!
```

```
import inspect

prompt = "\n" + dialogue_string

signature = inspect.signature(update_knowledge)
signature = str(signature).replace("__main__.Record", "Record")
docstring = update_knowledge.__doc__

raven_prompt = f'''{dataclass_schema_representation}\nFunction:\n{update_knowledge.__name__}{signature}\n    """{docstring}"""\n\r
print (raven_prompt)
```

```

@dataclass
class Record:
    agent_name : str # The agent name
    customer_email : str # customer email if provided, else ''
    customer_order : str # The customer order number if provided, else ''
    customer_phone : str # the customer phone number if provided, else ''
    customer_sentiment : str # Overall customer sentiment, either 'frustrated', or 'happy'. Always MUST have a value.

Function:
update_knowledge(results_list: List[Record])
    """
    Registers the information necessary
    """


User Query:
Agent: Hello, thank you for contacting BrownBox customer support. My name is Alex, how can I assist you today?
Customer: Hi, I'm calling about my order for a water purifier. I received it yesterday, but it's not working correctly. I want to r
Agent: I'm sorry to hear that. I'll be happy to help you with that. Can you please provide me with your order number?
Customer: Sure, it's 12345.
Agent: Thank you for the information. May I know the reason for the return?
Customer: As I mentioned earlier, the product is not working correctly. I want to return it and get a refund.
Agent: I'm sorry for the inconvenience. We would be happy to process your return and refund. However, since you have opted for Cash
Customer: What? That's too long. Why does it take so much time?
Agent: I understand your frustration, but the refund process takes time as we have to verify the product's condition and ensure tha
Customer: This is unacceptable. I need the refund immediately. Can't you do anything about it?
Agent: I'm sorry, but we cannot expedite the refund process. However, I can assure you that we will process your refund as soon as
Customer: Can you at least tell me the status of my refund?
Agent: Sure, I can check the status of your refund. Please allow me a moment to check that for you.
(Customer is put on hold for a few minutes)
Agent: Thank you for waiting. I have checked your refund status, and I see that your return has been received by our team. The refu
Customer: Alright, I understand. Is there anything else I need to do?
Agent: No, you don't have to do anything else. Our team will process your refund, and you will receive an email confirmation once i
Customer: Okay, thank you for your help.
Agent: You're welcome. I apologize for the inconvenience caused. Is there anything else I can assist you with?
Customer: No, that's all.
Agent: Alright, please feel free to contact us if you have any further questions or concerns. Have a great day!
Customer: You too. Bye.
Agent: Goodbye!<human_end>
```

```
raven_call = query_raven(raven_prompt)
print (raven_call)
```

```
update_knowledge(results_list=[Record(agent_name='Alex', customer_email='', customer_order='12345', customer_phone='', customer_sen
```

```
exec(raven_call)
```

```
Records inserted successfully.
```

```
import inspect

sample_zero = dialogue_data[10]
dialogue_string = sample_zero["conversation"].replace("\n\n", "\n")

prompt = "\n" + dialogue_string
signature = inspect.signature(update_knowledge)
docstring = update_knowledge.__doc__
raven_prompt = f'''{dataclass_schema_representation}\nFunction:\n{update_knowledge.__name__}{signature}\n    """{docstring}"""\n\r

raven_call = query_raven(raven_prompt)
print (raven_call)
exec(raven_call)
```

```
update_knowledge(results_list=[Record(agent_name='John', customer_email='', customer_order='#BB789012', customer_phone='', customer
Records inserted successfully.
```

```
execute_sql(
    '''
    SELECT COUNT(customer_sentiment)
    FROM customer_information
    WHERE agent_name = "John" AND customer_sentiment = "happy"
    ''')
```

```
Query operation executed successfully. Number of rows returned: 1
[(1,)]
```

```
prompt = "how many customers John has made happy."

signature = inspect.signature(execute_sql)

docstring = execute_sql.__doc__

sql_schema_representation = \
"""
CREATE TABLE customer_information (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    agent_name TEXT,
    customer_email TEXT,
    customer_order TEXT,
    customer_phone TEXT,
    customer_sentiment TEXT
);
"""

raven_prompt = f'''{sql_schema_representation}\nFunction:\n{execute_sql.__name__}{signature}\n    """{docstring}"""\n\n\nUser Quer

print (raven_prompt)
```

```

CREATE TABLE customer_information (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    agent_name TEXT,
    customer_email TEXT,
    customer_order TEXT,
    customer_phone TEXT,
    customer_sentiment TEXT
);

Function:
execute_sql(sql: str)
    """ Runs SQL code for the given schema. Make sure to properly leverage the schema to answer the user's question in the best way


User Query:how many customers John has made happy.<human_end>
```

```
raven_call = query_raven(raven_prompt)

print (raven_call)
```

```
execute_sql(sql='SELECT COUNT(customer_sentiment) FROM customer_information WHERE agent_name = "John" AND customer_sentiment = "hap
eval(raven_call)
```

```
Query operation executed successfully. Number of rows returned: 1
[(1,)]
```

```
!rm extracted.db
initialize_db()
```

```
Table customer_information created successfully.
```

```
from tqdm import tqdm

for i in tqdm(range(0, 10)):
    data = dialogue_data[i]
    dialogue_string = data["conversation"].replace("\n\n", "\n")

    # Ask Raven to extract the information we want out of this dialogue.
    prompt = "\n" + dialogue_string
    signature = inspect.signature(update_knowledge)
    docstring = update_knowledge.__doc__
    raven_prompt = f'''{dataclass_schema_representation}\nFunction:\n{update_knowledge.__name__}{signature}\n    """{docstring}""'
    raven_call = query_raven(raven_prompt)
    print (raven_call)
    exec(raven_call)
```

```
 10%|█         | 1/10 [00:01<00:15,  1.68s/it]update_knowledge(results_list=[Record(agent_name='Tom', customer_email='johndoe@email
Records inserted successfully.
 20%|██        | 2/10 [00:02<00:11,  1.44s/it]update_knowledge(results_list=[Record(agent_name='Alex', customer_email='', customer_
Records inserted successfully.
 30%|███       | 3/10 [00:04<00:10,  1.48s/it]update_knowledge(results_list=[Record(agent_name='Sarah', customer_email='jane.doe@em
Records inserted successfully.
 40%|████      | 4/10 [00:06<00:10,  1.73s/it]update_knowledge(results_list=[Record(agent_name='BrownBox', customer_email='john.doe
Records inserted successfully.
 50%|█████     | 5/10 [00:08<00:08,  1.67s/it]update_knowledge(results_list=[Record(agent_name='Sarah', customer_email='', customer
Records inserted successfully.
 60%|██████    | 6/10 [00:09<00:06,  1.65s/it]update_knowledge(results_list=[Record(agent_name='Alex', customer_email='johnsmith@em
Records inserted successfully.
 70%|███████   | 7/10 [00:11<00:04,  1.54s/it]update_knowledge(results_list=[Record(agent_name='Alex', customer_email='', customer_
Records inserted successfully.
 80%|████████  | 8/10 [00:12<00:02,  1.43s/it]update_knowledge(results_list=[Record(agent_name='Rachel', customer_email='', custome
Records inserted successfully.
 90%|█████████ | 9/10 [00:13<00:01,  1.45s/it]update_knowledge(results_list=[Record(agent_name='Sarah', customer_email='', customer
Records inserted successfully.
100%|██████████| 10/10 [00:15<00:00,  1.56s/it]update_knowledge(results_list=[Record(agent_name='Sarah', customer_email='jane@email
Records inserted successfully.
```

```
signature = inspect.signature(execute_sql)

docstring = execute_sql.__doc__

schema_representation = \
"""
CREATE TABLE customer_information (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    agent_name TEXT,
    customer_email TEXT,
    customer_order TEXT,
    customer_phone TEXT,
    customer_sentiment TEXT
);
"""

raven_prompt = f'''{schema_representation}\nFunction:\n{execute_sql.__name__}{signature}\n    """{docstring}"""\n\n\n'''
raven_prompt = raven_prompt + "User Query: How many happy customers?<human_end>"
print (raven_prompt)
raven_call = query_raven(raven_prompt)

print (raven_call)
eval(raven_call)
```

```

CREATE TABLE customer_information (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    agent_name TEXT,
        customer_email TEXT,
        customer_order TEXT,
        customer_phone TEXT,
        customer_sentiment TEXT
    );

    Function:
    execute_sql(sql: str)
        """ Runs SQL code for the given schema. Make sure to properly leverage the schema to answer the user's question in the best way


    User Query: How many happy customers?<human_end>
    execute_sql(sql='SELECT COUNT(*) FROM customer_information WHERE customer_sentiment = "happy";')
    Query operation executed successfully. Number of rows returned: 1
    [(7,)]
```

```
raven_prompt = f'''{schema_representation}\nFunction:\n{execute_sql.__name__}{signature}\n    """{docstring}"""\n\n\n'''
raven_prompt = raven_prompt + \
"User Query: Give me the names and phone numbers of the ones"\
"who are frustrated and the order numbers?<human_end>"

print (raven_prompt)
raven_call = query_raven(raven_prompt)

print (raven_call)
eval(raven_call)
```

```
    CREATE TABLE customer_information (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        agent_name TEXT,
        customer_email TEXT,
        customer_order TEXT,
        customer_phone TEXT,
        customer_sentiment TEXT
    );

    Function:
    execute_sql(sql: str)
        """ Runs SQL code for the given schema. Make sure to properly leverage the schema to answer the user's question in the best way


    User Query: Give me the names and phone numbers of the oneswho are frustrated and the order numbers?<human_end>
    execute_sql(sql='SELECT agent_name, customer_phone, customer_order FROM customer_information WHERE customer_sentiment = "frustrated
    Query operation executed successfully. Number of rows returned: 3
    [('Sarah', '', 'BB123456'), ('Alex', '', '12345'), ('Sarah', '9876543210', '')]
```

```
Start coding or generate with AI.
```