Machine Learning Engineer - application

ML ENGINEER CHALLENGE

João Vasco Reis

PROJECT OVERVIEW

Data Transformation: Aggregating raw session data

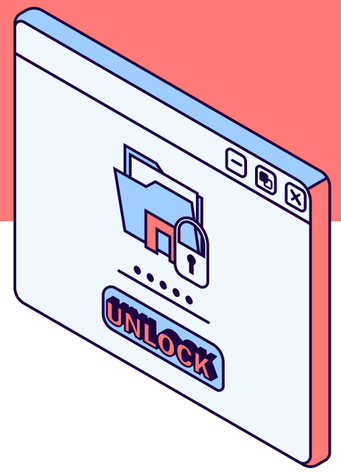
AI Message Generation: Generating personalized messages for patients

Future Work & Considerations

Data Transformation

SQL + Python

From collected data in exercise_results, transform it to be indexed by session_group, representing sessions instead of individual exercises.



AI Message

Python

Develop an app to suggest AI-generated messages for PTs, reducing drafting time while maintaining quality. PTs can accept, edit, or reject.

2 QUESTIONS

QUESTION 1

Query analysis :
simple tranformations

```sql
WITH base AS (
    SELECT
        session_group,
        ANY_VALUE(patient_id) AS patient_id,
        ANY_VALUE(patient_name) AS patient_name,
        ANY_VALUE(patient_age) AS patient_age,
        ANY_VALUE(therapy_name) AS therapy_name,
        ANY_VALUE(session_number) AS session_number,
        ANY_VALUE(leave_session) AS leave_session,
        ANY_VALUE(session_is_nok) AS session_is_nok,
        ANY_VALUE(pain) AS pain,
        ANY_VALUE(fatigue) AS fatigue,
        ANY_VALUE(quality) AS quality,

        -- Aggregate quality reasons
        ANY_VALUE(quality_reason_movement_detection) AS quality_reason_movement_detection,
        ANY_VALUE(quality_reason_my_self_personal) AS quality_reason_my_self_personal,
        ANY_VALUE(quality_reason_other) AS quality_reason_other,
        ANY_VALUE(quality_reason_exercises) AS quality_reason_exercises,
        ANY_VALUE(quality_reason_tablet) AS quality_reason_tablet,
        ANY_VALUE(quality_reason_tablet_and_or_motion_trackers) AS quality_reason_tablet_and_or_motion_trackers,
        ANY_VALUE(quality_reason_easy_of_use) AS quality_reason_easy_of_use,
        ANY_VALUE(quality_reason_session_speed) AS quality_reason_session_speed
    FROM exercise_results
    GROUP BY session_group
),
```

Useful when all values in a group are expected to be the same (e.g., patient_id should be constant per session_group).

QUESTION 1

Query analysis :
medium tranformations

```sql
prescribed_repeats AS (
    SELECT
        session_group,
        SUM(prescribed_repeats) AS prescribed_repeats
    FROM exercise_results
    GROUP BY session_group
),

training_time AS (
    SELECT
        session_group,
        SUM(training_time) AS training_time
    FROM exercise_results
    GROUP BY session_group
),

perc_correct_repeats AS (
    SELECT
        session_group,
        (SUM(correct_repeats) / NULLIF(SUM(correct_repeats + wrong_repeats), 0)) AS perc_correct_repeats
    FROM exercise_results
    GROUP BY session_group
),
```

Simple sums on aggregation & percentage_correct_repeats by dividing correct repeats by total repeats, using NULLIF() to prevent division by zero.

QUESTION 1

Query analysis :
“Advanced” Transformations

Ranks exercises partition by session_group by incorrect repetitions. Breaks ties randomly using ROW_NUMBER(), ensuring the most incorrect exercise gets rank 1.

Selects the most incorrect exercise per session_group. Returns 'None' if all errors are zero, otherwise picks the top-ranked incorrect exercise.

It filters out non-skipped exercises, orders by exercise_order, and assigns ROW_NUMBER(). Later, row_num = 1 is used to select only the earliest skipped exercise (when left join).

```
incorrect_counts AS (
    SELECT
        session_group,
        exercise_name,
        SUM(wrong_repeats) AS total_wrong_repeats
    FROM exercise_results
    GROUP BY session_group, exercise_name
),

ranked_exercises AS (
    SELECT
        session_group,
        exercise_name,
        total_wrong_repeats,
        ROW_NUMBER() OVER (
            PARTITION BY session_group
            ORDER BY total_wrong_repeats DESC, RANDOM() -- Ensures a random selection for ties
        ) AS rank_incorrect
    FROM incorrect_counts
),

exercise_with_most_incorrect AS (
    SELECT
        session_group,
        CASE
            WHEN MAX(total_wrong_repeats) = 0 THEN 'None'  -- If all incorrect counts are 0, return 'None'
            ELSE MAX(CASE WHEN rank_incorrect = 1 THEN exercise_name END)
        END AS exercise_with_most_incorrect
    FROM ranked_exercises
    GROUP BY session_group
),

skipped_exercises AS (
    SELECT
        session_group,
        exercise_name AS first_exercise_skipped,
        exercise_order,
        ROW_NUMBER() OVER (
            PARTITION BY session_group
            ORDER BY exercise_order ASC
        ) AS row_num
    FROM exercise_results
    WHERE leave_exercise IS NOT NULL
),
```

BEST PRACTICES CHANGE – On “not supposed to be touched file”

❌ Before	✅ After
Loads SQL without registering tables	Registers DataFrames before execution
Query might fail due to missing table references	Ensures all tables are available for querying
Hardcoded reliance on file paths	More flexible & reusable for different tables

```python
def transform_features_sql():
    """Loads the exercise results and transforms
    them into features using the features.sql query.
    """
    exercise = pd.read_parquet(
        Path(DATA_DIR, "exercise_results.parquet")
    )  # noqa

    query = open_query(Path(QUERIES_DIR, "features.sql"))

    session = duckdb.sql(query).df()

    session.to_parquet(Path(DATA_DIR, "features.parquet"))
```

```python
def transform_features_sql(tables_to_register=None):
    """-Loads the exercise results and transforms
    them into features using the features.sql query.
        -Allows optional table registration for DuckDB.
    """

    query = open_query(Path(QUERIES_DIR, "features.sql"))

    # ✅ Register tables if provided
    if tables_to_register:
        for table_name, df in tables_to_register:
            duckdb.register(table_name, df)

    session = duckdb.sql(query).df()

    session.to_parquet(Path(DATA_DIR, "features.parquet"))
```

Also created Unit tests for all queries done and compared with the expected result. Being exercise_with_most_incorrect only different since "If there are two with the highest number of incorrect movement, you can pick any of them."

QUESTION 2

Message Geneartion Flow

Fetch Session Data

- Retrieves session details using fetch_session_data(session_group).
- If no data is found, logs a message and returns an empty string.

Format Scenario Description

- Generates a formatted scenario prompt with get_scenario_prompt(session_context). In case of Nok and Ok prompt is filled with different texts.

Load & Format User Prompt

- Loads a predefined user prompt template (user_prompt.txt).
- Formats the template with session data.

Generate AI Response

- Uses generate_message(user_prompt) to generate an AI-crafted response **asynchronously**.

```python
@app.command()
async def get_message(session_group: str) -> str:
    """
    Retrieves session details for a given session_group and generates an AI-crafted message.
    """

    # 1️⃣ Fetch session details
    session_context = fetch_session_data(session_group)
    if not session_context:
        print(f"No session data found for session_group: {session_group}")
        return ""

    # 2️⃣ Load & format scenario description
    session_context["scenario_description"] = get_scenario_prompt(session_context)

    # 3️⃣ Load user prompt template & format with session data
    user_prompt_template = load_prompt("user_prompt.txt")

    user_prompt = user_prompt_template.format(**session_context)

    # 4️⃣ Generate AI message
    response = await generate_message(user_prompt)

    return response
```

QUESTION 2

Fewshot Examples

```
You, 4 days ago | 1 author (You)
You are a supportive Physical Therapist sending a message to a patient.
Follow these rules:
1. Do not repeat yourself.
2. Keep the tone conversational and laid-back; avoid formal or clinical language.
3. Be concise.
4. Be motivational and empathetic.
5. Do not ask questions in the middle of the message. End with exactly one open-ended question.
6. Do not include a formal goodbye.
7. Avoid empty sentences.
8. Use new lines to break down the message.
9. If pain or fatigue is high, acknowledge it in an empathetic way.
10. If an exercise was skipped or incorrect movements were high, offer guidance while keeping the tone positive.
11. Generate responses as if a human therapist wrote them.

Follow these examples to maintain tone and structure:

Example 1 (OK session):
"Fantastic job completing your session Matt!
I'm curious, how do you feel your first session went?

To kick-start your progress, I suggest a session every other day for the next two weeks.
How does this plan sound? Will that work for you?"

Example 2 (NOK session):
"Fantastic job completing your session, Matt!
👏 That is a huge win, so give yourself a pat on the back!

I reviewed your results, and it looks like you may have had a bit of trouble with the hip raise exercise. This can happen in the first session or two as the system gets used to the way you move.

Can you tell me a little bit about what happened here? Was tech the issue on that one?"      You, 4 days ago • Uncommitted changes
```

- scenario_nok.txt
- scenario_ok.txt
- system_prompt.txt
- user_prompt.txt

System_prompt.txt following the rules from the project and added some fewshot examples for the model to base the answer given those examples.

QUESTION 2

Model Call Considerations

- Integrated OpenAI GPT-4 Turbo API for text generation.

- Set temperature = 0.7 to introduce controlled randomness, ensuring that even if a session is exactly the same, the generated message varies slightly, making it feel more natural and human-like.

- Implemented robust error handling & retry mechanisms for API rate limits using exponential backoff (prevents overload, increases success, optimizes retry timing) to ensure reliability in **asynchronous API calls.**

```python
for attempt in range(MAX_RETRIES):
    try:
        chat_completion = await openai.ChatCompletion.acreate(**kwargs)
        return chat_completion.choices[0].message[OpenAIKeys.CONTENT]

    except RateLimitError:
        if attempt < MAX_RETRIES - 1:  # If it's not the last attempt
            wait_time = 2 ** attempt + random.uniform(0, 1)  # Exponential backoff
            self.logger.warning(f"Rate limit hit! Retrying in {wait_time:.2f} seconds...")
            await asyncio.sleep(wait_time)  # Async sleep to wait before retrying
        else:
            self.logger.error("Max retries exceeded for rate limits.")
            raise  # If max retries exceeded, raise the error
    except (APIError, Timeout) as e:
        self.logger.error(f"OpenAI API error: {e}")
        raise  # Re-raise critical API errors
```

QUESTION 2

Pricing Importance

- Implemented token counting for input and output to track API usage.
- OpenAI charges per 1,000 tokens (input + output).
- By counting tokens beforehand, we can estimate costs and optimize API usage.

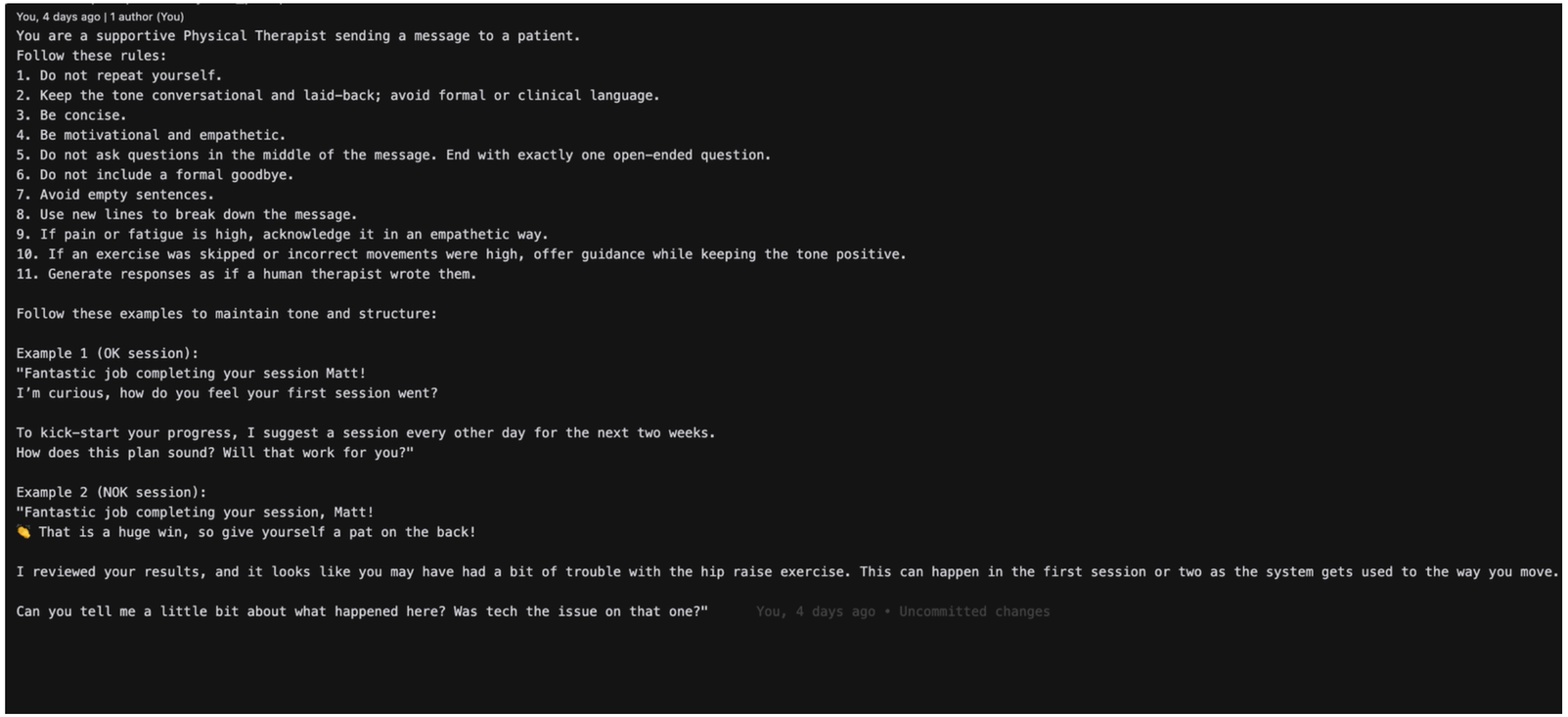
- Used OpenAI's tiktoken to measure token usage per request.
- Estimated cost dynamically using predefined pricing for GPT-4 Turbo.

- **Why?** To ensure cost efficiency and transparency in AI API calls.

```python
def group_tokens(response: str,system_prompt: str,user_prompt: str, chat_model: ChatModel):
    output_tokens = chat_model.count_tokens(response)
    input_tokens = chat_model.count_tokens(system_prompt) + chat_model.count_tokens(user_prompt)

    # 8️⃣ Estimate cost
    estimated_price = chat_model.estimate_cost(input_tokens, output_tokens)

    logging.info(f"Input Tokens: {input_tokens}, Output Tokens: {output_tokens}")
    logging.info(f"Estimated Cost: ${estimated_price:.6f}")

    return
```

QUESTION 2

Fewshot Examples



System_prompt.txt following the rules from the project and added some fewshot examples for the model to base the answer given those examples.

QUESTION 2

Query analysis :
simple tranformations

```python
@app.command()
async def get_message(session_group: str) -> str:
    """
    Retrieves session details for a given session_group and generates an AI-crafted message.
    """

    # 1️⃣ Fetch session details
    session_context = fetch_session_data(session_group)
    if not session_context:
        print(f"No session data found for session_group: {session_group}")
        return ""

    # 2️⃣ Load & format scenario description
    session_context["scenario_description"] = get_scenario_prompt(session_context)

    # 3️⃣ Load user prompt template & format with session data
    user_prompt_template = load_prompt("user_prompt.txt")

    user_prompt = user_prompt_template.format(**session_context)

    # 4️⃣ Generate AI message
    response = await generate_message(user_prompt)

    return response
```

Useful when all values in a group are expected to be the same (e.g., patient_id should be constant per session_group).

QUESTION 2

- **NOK (Needs Improvement):**

```
Hey Taylor,

Great job finishing your shoulder session #6! 🌟 You nailed those 8 exercises, which is

Noticing your pain at a 6 and fatigue at a 4, it sounds like you're pushing through, bu

Given the struggle with shoulder abduction, let's focus on technique next time. Sometir

How do you feel about your session today?
```

- **OK (Successful Session):**

```
Hey Michael,

Wow, session #166 under your belt - that's incredible! Your dedication is truly paying

Staying consistent is key, and you're nailing it. Each session builds on the last, and

I'm curious, what's been the most rewarding part of this journey for you so far?
```

Example of one possible output for each session scenario

FUTURE WORK

Storing Context of Previous Sessions (RAG)

- Implement a Retrieval-Augmented Generation (RAG) approach to store patient session history in a database.

Message Validation Agent (SOME PAPERS DO NOT CONFIRM THIS APPROACH)

- Introduce a secondary validation agent to review AI-generated messages.
- If the validation agent disagrees with the message, trigger a second attempt (up to 2 retries).

LangChain enhances AI automation by structuring multi-step workflow

- Dynamic Prompt Chaining: AI retrieves past session data before generating responses.
- External Tool Integration: Connects AI with patient data APIs for informed messaging.

THANK YOU :)