Code Summary: Floor Plan Room Area Computation using Windmill & Gemini API

I have used **windmill dataflow** to automate the process and **Gemini 1.5 Flash** for vision llm

The workflow consists of two main steps:

1. Extracting Room Information from Floor Plan

Code :

```
#requirements:
#google-generativeai
#wmill
#pandas
#pillow

import json
import logging
import re
import requests
from wmill import task
import google.generativeai as genai
from PIL import Image
import io

# Configure Gemini API
api_key = " "  # Replace with your API key
genai.configure(api_key=api_key)

# Load Gemini Vision Model
model = genai.GenerativeModel("gemini-1.5-flash")

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def download_image_from_drive(url: str):
    """
    Downloads an image from a Google Drive URL and returns it as a PIL Image
object.
    """
```

```
    try:
        # Extract the file ID from the Google Drive URL
        file_id = url.split("/d/")[1].split("/")[0]
        direct_download_url =
f"https://drive.google.com/uc?export=download&id={file_id}"

        # Download the image
        response = requests.get(direct_download_url)
        response.raise_for_status()  # Raise an error for bad status codes

        # Open the image using Pillow
        image = Image.open(io.BytesIO(response.content))
        return image
    except Exception as e:
        logger.error(f"Failed to download image from Google Drive: {str(e)}")
        raise

@task()
def extract_room_data(image_url: str):
    """
    Extracts room names, dimensions, and generates structured output from an
image hosted on Google Drive.
    """
    try:
        # Download the image from Google Drive
        image = download_image_from_drive(image_url)

        # Convert the image to bytes
        img_byte_arr = io.BytesIO()
        image.save(img_byte_arr, format='JPEG')  # Save as JPEG format
        img_byte_arr = img_byte_arr.getvalue()

        # Generate content using the Gemini API
        response = model.generate_content(
            contents=[
                "Analyze this floor plan image and extract **all** room names
and their dimensions (length and width in meters). "
                "Return the response in valid JSON format with the following
structure:",
                {
                    "mime_type": "image/jpeg",  # Specify JPEG MIME type
                    "data": img_byte_arr
```

```
                }
            ]
        )

        # Log the raw response for debugging
        logger.info(f"Raw response from Gemini API: {response.text}")

        # Attempt to parse the response as JSON
        response_text = response.text.strip()
        try:
            room_data = json.loads(response_text).get("rooms", [])
        except json.JSONDecodeError:
            # Fallback: Try to extract JSON if parsing fails
            room_data = extract_json_from_text(response_text)
            if room_data:
                room_data = room_data.get("rooms", [])

        if not room_data:
            return {"error": "No valid room data found in the response"}

        return room_data
    except Exception as e:
        logger.error(f"Failed to extract room details: {str(e)}")
        return {"error": f"Failed to extract room details: {str(e)}"}

def extract_json_from_text(text: str):
    """
    Attempts to extract JSON from a text response using regex.
    """
    try:
        # Look for JSON-like content in the text
        json_match = re.search(r"\{.*\}", text, re.DOTALL)
        if json_match:
            return json.loads(json_match.group(0))
        else:
            return None
    except Exception as e:
        logger.error(f"Failed to extract JSON from text: {str(e)}")
        return None

@task()
def main(image_url: str):
```

```
    """
    Orchestrates the workflow.
    """
    logger.info("Starting room data extraction...")
    room_data = extract_room_data(image_url)
    logger.info(f"Extracted room data: {room_data}")

    # Return the extracted room data
    return room_data
```

- **Image Retrieval**: The image is downloaded from a **Google Drive URL** using requests. It is opened and processed using **Pillow (PIL)**
- **Conversion to Bytes :** The image is converted to JPEG format and stored in a byte array (io.BytesI0), which is required for Gemini API input.
- **Vision Model Processing**: Uses the **Gemini 1.5 Flash** model to analyze the image and extract room details (names, length, and width).
- **JSON Extraction**: Parses the model's response into structured room data.
- **Error Handling**: Ensures valid room information is extracted, logging errors when necessary.

2. Computing Room Areas & Generating "Thinking" Explanations

Code:

```
#requirements:
#google-generativeai
#wmill
#pandas
#pillow

import json
import io
import os
import logging
import re
import base64
import pandas as pd
import google.generativeai as genai
from PIL import Image
from wmill import task

# Configure Gemini API
api_key = " "  # Replace with your API key
```

```
genai.configure(api_key=api_key)

# Load Gemini Vision Model
model = genai.GenerativeModel("gemini-1.5-flash")

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

@task()
def compute_room_areas(room_data):
    """
    Computes area for each room and generates a "thinking" explanation.
    """
    if isinstance(room_data, dict) and "error" in room_data:
        return room_data  # Skip computation if there's an error

    for room in room_data:
        try:
            # Ensure the keys exist and values are valid
            if "length" not in room or "width" not in room:
                room["area"] = None
                room["thinking"] = "Missing dimensions (length or width)."
                continue

            # Convert length and width to floats
            length = float(room["length"])
            width = float(room["width"])

            # Compute the area
            room["area"] = round(length * width, 2)

            # Use "name" instead of "room_name" if "room_name" is not present
            room_name = room.get("room_name", room.get("name", "Unknown
Room"))

            # Generate a "thinking" explanation
            thinking_prompt = (
                f"Explain how the area for {room_name} was calculated using "
                f"length={length}m and width={width}m."
            )
            thinking_response = model.generate_content(thinking_prompt)
```

```
            room["thinking"] = thinking_response.text
        except (ValueError, KeyError) as e:
            room["area"] = None
            room["thinking"] = f"Error calculating area: {str(e)}"

    return room_data

@task()
def main(room_data):
    """
    Orchestrates the workflow by computing room areas and generating
explanations.
    """
    logger.info("Starting room area computation...")

    # Validate the input
    if room_data is None:
        logger.error("room_data is None. Please provide valid room data.")
        return {"error": "room_data is None. Please provide valid room data."}

    if not isinstance(room_data, list):
        logger.error("room_data must be a list of room dictionaries.")
        return {"error": "room_data must be a list of room dictionaries."}

    # Compute room areas and generate thinking explanations
    processed_data = compute_room_areas(room_data)
    logger.info(f"Processed room data: {processed_data}")

    # Convert the processed data to a DataFrame for better visualization
    df = pd.DataFrame(processed_data)
    logger.info(f"Processed room data as DataFrame:\n{df}")

    # Return the processed room data
    return processed_data
```

- **Area Calculation**: Computes the area of each room using `length × width` from the previous node (extract room information). Used that previous node output as input in this current node
- **Model Thinking** : Uses Gemini to generate a textual explanation of how the area was calculated for each room.
- **Data Output**: Stores results in a structured format (JSON and DataFrame) with columns:

    - **Room Name**
    - **Area (m²)**
    - **Thinking (AI explanation of the calculation process)**
- **Validation & Logging**: Handles missing or incorrect data, ensuring robustness.

Final Output:

A structured table (CSV/JSON) containing **room names, areas, and AI-generated explanations**.