perplexity

can you give me consolidated updated clear code and also a separate updated YAML file.

Here is a **consolidated, updated, and clear Python code framework** for your agentic AI app, along with a separate YAML semantic layer file. This design supports all your target queries—including aggregations, filtering, group-by, explanations (causal/why), and weather lookups—using a hierarchical multi-agent pattern.

1. semantic_layer.yaml (save this in your working directory)

```
tables:
  efficiency_data:
    table: "your_project.your_dataset.efficiency_data_sample"
    fields:
      Region: "Region"
      Country: "Country"
      City: "City"
      HubID: "Hub ID"
      ScopeAvailability: "Scope Available"
      Density: "Density"
      TargetKmsCollected: "Target Kms Collected"
      TotalKmsDriven: "Total Kms Driven"
      Efficiency: "Efficiency"
      Classification: "Classification"
      HubName: "Hub Name"
      Locality1: "Locality1"
      Locality2: "Locality2"
      C1Kms: "C1 kms"
      C2Kms: "C2 kms"
      C3Kms: "C3 kms"
      C1Count: "C1 Count"
      C2Count: "C2 Count"
      C3Count: "C3 Count"
      Deadends: "Deadends"
      Unpaved: "Unpaved"
      LessDriven: "Less Driven"
      CollectedC1Km: "Collected C1 km"
      CollectedC2Km: "Collected C2 km"
      CollectedC3Km: "Collected C3 km"
      CollectedC1Count: "Collected C1 Count"
      CollectedC2Count: "Collected C2 Count"
      CollectedC3Count: "Collected C3 Count"
      Date: "Date"
    metrics:
      C1Percent:
        expression: "(`C1 kms` / NULLIF(`C1 kms` + `C2 kms` + `C3 kms`, 0))"
        description: "Percentage of C1 kms"
      C2Percent:
        expression: "(`C2 kms` / NULLIF(`C1 kms` + `C2 kms` + `C3 kms`, 0))"
        description: "Percentage of C2 kms"
      C3Percent:
        expression: "(`C3 kms` / NULLIF(`C1 kms` + `C2 kms` + `C3 kms`, 0))"
        description: "Percentage of C3 kms"
  weather_data:
    table: "external_api"
    fields:
      City: "City"
      State: "State"
      Date: "Date"
      Temperature: "Temperature"
      Precipitation: "Precipitation"
      WeatherDescription: "WeatherDescription"
    metrics:
      AvgTemperature:
        expression: "AVG(Temperature)"
        description: "Average temperature"
      TotalPrecipitation:
        expression: "SUM(Precipitation)"
        description: "Total precipitation"
```

2. Python Code (Colab/Notebook Ready, Modular, and Extensible)

Install Required Packages

```
!pip install google-cloud-bigquery pandas langchain-google-genai pyyaml requests
```

Authenticate for BigQuery (Colab only)

```
from google.colab import auth
auth.authenticate_user()
```

Semantic Layer Loader

```
import yaml

class SemanticLayer:
    def __init__(self, yaml_path):
        with open(yaml_path, 'r') as f:
            self.data = yaml.safe_load(f)

    def get_table_info(self, table_name):
        return self.data["tables"].get(table_name)

    def generate_sql(self, table_name, metrics=[], dimensions=[], filters={}, aggregatior
        table_info = self.get_table_info(table_name)
        if not table_info:
            raise ValueError(f"Table {table_name} not found in semantic layer")
        select_parts = []
        for field, agg_func in aggregations.items():
            if field in table_info["fields"]:
                col_name = table_info["fields"][field]
                select_parts.append(f"{agg_func.upper()}(`{col_name}`) AS `{agg_func.lowe
        for metric in metrics:
            if metric in table_info["metrics"]:
                select_parts.append(f"{table_info['metrics'][metric]['expression']} AS `{
        for dim in dimensions:
            if dim in table_info["fields"]:
                col_name = table_info["fields"][dim]
                select_parts.append(f"`{col_name}` AS `{dim}`")
        if not select_parts:
            select_parts.append("*")
        sql = f"SELECT {', '.join(select_parts)} FROM `{table_info['table']}`"
        where_parts = []
        for field, value in filters.items():
            if field in table_info["fields"]:
                where_parts.append(f"`{table_info['fields'][field]}` = '{value}'")
        if where_parts:
            sql += f" WHERE {' AND '.join(where_parts)}"
        if group_by:
            group_cols = [f"`{table_info['fields'][field]}`" for field in group_by if fie
            if group_cols:
                sql += " GROUP BY " + ", ".join(group_cols)
        return sql
```

BigQuery Agent

```
from google.cloud import bigquery
import pandas as pd

class BigQueryAgent:
    def __init__(self, project_id):
        self.client = bigquery.Client(project=project_id)
    def run_query(self, sql):
        try:
            query_job = self.client.query(sql)
            return query_job.result().to_dataframe()
        except Exception as e:
            print(f"BigQuery error: {e}")
            return pd.DataFrame()
```

Weather Data Agent (Current Weather and Monthly Stats)

```
import requests
from datetime import datetime

class WeatherDataAgent:
    def __init__(self, api_key):
        self.api_key = api_key
        self.current_url = "https://api.openweathermap.org/data/2.5/weather"
        self.stats_url = "https://api.openweathermap.org/data/2.5/statistics"  # Example,

    def get_current_weather(self, city):
        params = {"q": city, "appid": self.api_key, "units": "metric"}
        response = requests.get(self.current_url, params=params)
        try:
            data = response.json()
            return {
                "api_request": response.url,
                "data": data
            }
        except Exception as e:
            return {"api_request": response.url, "error": str(e)}

    def get_monthly_weather_stats(self, city, year, month):
        params = {"q": city, "month": month, "year": year, "appid": self.api_key, "units'
        response = requests.get(self.stats_url, params=params)
        try:
            data = response.json()
            return data.get("result", data)
        except Exception as e:
            return {"error": str(e)}

    def compare_weather(self, city, year, month):
        prev_month = month - 1 if month > 1 else 12
        prev_year = year if month > 1 else year - 1
        this_month_stats = self.get_monthly_weather_stats(city, year, month)
        prev_month_stats = self.get_monthly_weather_stats(city, prev_year, prev_month)
        return {
            "this_month": this_month_stats,
            "previous_month": prev_month_stats
        }

    def process_query(self, query):
        if "now" in query.lower() or "current" in query.lower():
            city = self._extract_city(query)
            result = self.get_current_weather(city)
            return {
                "sql": result["api_request"],
                "result": result["data"] if "data" in result else result["error"],
                "summary": f"Current weather in {city}: {result['data'].get('weather', [
            }
        else:
            city = self._extract_city(query)
            today = datetime.today()
            stats = self.get_monthly_weather_stats(city, today.year, today.month)
            return {
                "sql": self.stats_url,
                "result": stats,
                "summary": f"Weather stats for {city}, {today.month}/{today.year}"
            }

    def _extract_city(self, query):
        # Simple extraction; for robust use LLM or regex
        words = query.split()
        for i, w in enumerate(words):
            if w.lower() == "in" and i+1 < len(words):
                return " ".join(words[i+1:]).replace("?", "")
        return "New York"
```

Manager Agent (with Explanation Capability)

```
from langchain_google_genai import ChatGoogleGenerativeAI
import ast
import os
from datetime import datetime

class ManagerAgent:
    def __init__(self, semantic_layer, data_agent, weather_agent, gemini_api_key, table_r
        os.environ["GOOGLE_API_KEY"] = gemini_api_key
        self.semantic_layer = semantic_layer
        self.data_agent = data_agent
        self.weather_agent = weather_agent
        self.llm = ChatGoogleGenerativeAI(model="gemini-1.5-flash-latest", temperature=0.
        self.table_name = table_name

    def handle_query(self, user_query):
        # Decide agent
        routing_prompt = (
            f"User question: {user_query}\n"
            f"Should this query be answered by the 'efficiency' agent (BigQuery) or the '
            f"Return 'efficiency', 'weather', or 'explanation' only."
        )
        route = self.llm.invoke(routing_prompt).content.strip().lower()

        if "weather" in route:
            return self.weather_agent.process_query(user_query)
        elif "explanation" in route or "why" in user_query.lower() or "compare" in user_c
            # Explanation: fetch both efficiency and weather data, ask LLM to correlate
            city, year, month = self._extract_city_year_month(user_query)
            eff_sql = self.semantic_layer.generate_sql(
                table_name=self.table_name,
                metrics=["Efficiency"],
                filters={"City": city},
                aggregations={"Efficiency": "AVG"},
                group_by=[]
            ) + f" AND EXTRACT(YEAR FROM `Date`) = {year} AND EXTRACT(MONTH FROM `Date`)
            prev_month = month - 1 if month > 1 else 12
            prev_year = year if month > 1 else year - 1
            eff_sql_prev = self.semantic_layer.generate_sql(
                table_name=self.table_name,
                metrics=["Efficiency"],
                filters={"City": city},
                aggregations={"Efficiency": "AVG"},
                group_by=[]
            ) + f" AND EXTRACT(YEAR FROM `Date`) = {prev_year} AND EXTRACT(MONTH FROM `Da
            eff_this = self.data_agent.run_query(eff_sql)
            eff_prev = self.data_agent.run_query(eff_sql_prev)
            weather_data = self.weather_agent.compare_weather(city, year, month)
            prompt = (
                f"Efficiency data for {city}: This month: {eff_this.to_dict() if hasattr(
                f"Weather data for {city}: This month: {weather_data['this_month']}, Prev
                "Analyze whether the change in efficiency can be attributed to weather co
                "If yes, explain how weather factors (temperature, rainfall, wind, etc.)
                "If not, say so."
            )
            explanation = self.llm.invoke(prompt)
            return {
                "efficiency_sql": {"this_month": eff_sql, "previous_month": eff_sql_prev}
                "weather_api_calls": weather_data,
                "explanation": explanation.content
            }
        else:
            # Data query
            prompt = (
                f"You are a data analyst. Given the user question, extract the following
                f"- 'metrics': list of metrics to select\n"
                f"- 'dimensions': list of fields to use as group by\n"
                f"- 'filters': dict of field-value filters\n"
                f"- 'aggregations': dict of field-aggregation_function (e.g., SUM, AVG, M
                f"- 'group_by': list of fields to group by\n"
                f"Available fields: {list(self.semantic_layer.get_table_info(self.table_n
                f"Available metrics: {list(self.semantic_layer.get_table_info(self.table_
                f"User question: {user_query}\n"
                f"Return ONLY a valid Python dictionary with these keys."
            )
            response = self.llm.invoke(prompt)
            try:
                parsed = ast.literal_eval(response.content.strip())
                metrics = parsed.get("metrics", [])
                dimensions = parsed.get("dimensions", [])
                filters = parsed.get("filters", {})
                aggregations = parsed.get("aggregations", {})
                group_by = parsed.get("group_by", [])
            except Exception as e:
                print("LLM parsing error:", e)
                metrics, dimensions, filters, aggregations, group_by = [], [], {}, {}, []
            sql = self.semantic_layer.generate_sql(
                table_name=self.table_name,
                metrics=metrics,
                dimensions=dimensions,
                filters=filters,
                aggregations=aggregations,
                group_by=group_by
            )
            result_df = self.data_agent.run_query(sql)
            if hasattr(result_df, 'empty') and result_df.empty:
                return {"sql": sql, "result": "No data found", "summary": "No results mat
            summary_prompt = (
                f"User question: {user_query}\n"
                f"SQL run: {sql}\n"
                f"Result (first 5 rows):\n{result_df.head(5).to_markdown()}\n"
                f"Summarize key findings in 2-3 sentences."
            )
            summary = self.llm.invoke(summary_prompt)
            return {"sql": sql, "result": result_df, "summary": summary.content}

    def _extract_city_year_month(self, query):
        # Naive implementation; for robust use LLM or regex
        city = "New York City"
        year = datetime.now().year
        month = datetime.now().month
        for token in query.split():
            if token.isdigit() and len(token) == 4:
                year = int(token)
            elif token.capitalize() in ["January", "February", "March", "April", "May", "
                                        "July", "August", "September", "October", "Novembe
                month = datetime.strptime(token, "%B").month
            elif token.capitalize() in ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
                                        "Aug", "Sep", "Oct", "Nov", "Dec"]:
                month = datetime.strptime(token, "%b").month
        # Extract city if "in" is present
        if "in" in query.lower():
            city = query.split("in")[-1].replace("?", "").strip()
        return city, year, month
```

Example Usage

```
# ==== CONFIGURATION ====
PROJECT_ID = "your_project_id"
GEMINI_API_KEY = "your_gemini_api_key"
SEMANTIC_LAYER_PATH = "semantic_layer.yaml"
TABLE_NAME = "efficiency_data"
WEATHER_API_KEY = "your_openweathermap_api_key"
# =======================

semantic_layer = SemanticLayer(SEMANTIC_LAYER_PATH)
data_agent = BigQueryAgent(project_id=PROJECT_ID)
weather_agent = WeatherDataAgent(api_key=WEATHER_API_KEY)
manager = ManagerAgent(semantic_layer, data_agent, weather_agent, GEMINI_API_KEY, TABLE_N

# Example queries
queries = [
    "What is the total Target km Driven?",
    "What is the efficiency in the city New York?",
    "Why is the efficiency low in Jan 2025?",
    "Compare C1 percent of high Density and Low density?",
    "How is the weather in Florida now?"
]

for user_query in queries:
    response = manager.handle_query(user_query)
    print("\nUser Query:", user_query)
    print("Actual SQL or API Call:\n", response.get("sql", response.get("efficiency_sql",
    print("Result Preview:\n", response.get("result", response.get("explanation", "")))
    print("Summary/Explanation:\n", response.get("summary", response.get("explanation", '
```

This framework will answer all your target questions, provide actual SQL/API calls, and support both data and explanation queries.
You can further refine entity extraction and prompt engineering for even more robust handling.