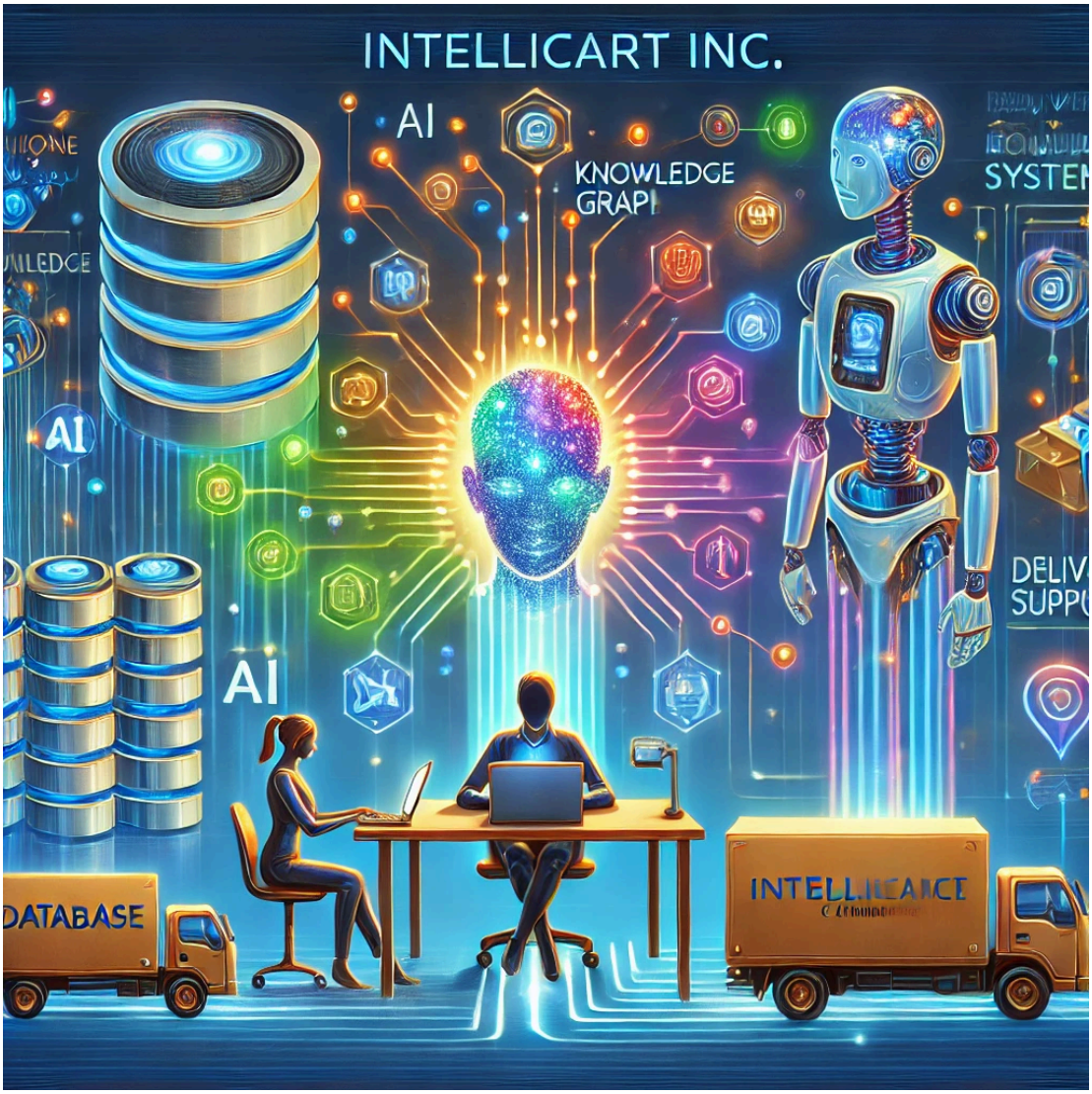

Introduction

Imagine an e-commerce company, `IntelliCart Inc.`, running a successful online shop where customers order products, triggering a chain of deliveries. While some deliveries go off smoothly, others might encounter issues, leading to customer inquiries. On top of that, users may raise tickets for claims that need prompt and efficient resolution. This scenario mirrors the **real-world** challenges of managing customer service in a thriving e-commerce business, touching on multiple facets of operations, from order handling to post-delivery support.

Let's consider a typical customer message that demands a response:

Hello,

Thanks for the quick delivery. Could you please send me my last order total cost?

Regards,
John

Addressing this request isn't as straightforward as it seems. To respond effectively, we need to:

- Analyze and accurately interpret the intent behind the customer's message.
- Query the database to fetch the relevant details (e.g., the total cost of the last order).
- Craft a clear, friendly, and satisfactory reply.

This use case is quite realistic and includes different aspects of an e-commerce company's activities. Addressing all this seamlessly requires a robust AI-powered customer care system, that we'll hopefully be building in this tutorial.

> Customer Message
>
> *Hello,*
>
> *Thanks for the quick delivery. Could you please send me my last order total cost ?*
>
> *Regards,*
> *John*

Requirements

This tutorial will be using the following packages: ***pandas, pydantic, openai, python-dotenv, jinja2***. You can install any of them using `pip`.

For the LLM, any OpenAI API compatible LLM Provider will be just fine.

All the datasets used here will be provided to you afterward. These are toy datasets but designed to be as realistic as possible.

Datasets Presentation

For this use case, we will use the following datasets:

---

1. **Customers**

- **Id**: Unique identifier for the customer
- **Name**: Full name of the customer
- **eMail**: Email address of the customer
- **Country**: Country of the customer

---

2. **Products**

- **Id**: Unique identifier for the product

- **Name**: Name of the product
- **ShortDescription**: Short description of the product
- **Color**: Color of the product
- **Size**: Size of the product
- **OriginCountry**: Country of origin of the product
- **DelivrableCountries**: List of countries where the product can be delivered
- **Universe**: Universe of the product (Cloth, Electronics, etc.)

---

3. **Orders**

- **Id**: Unique identifier for the order
- **CustomerId**: Foreign key referencing the customer who made the order
- **Date**: Date of the order
- **CustomerName**: Name of the customer
- **CustomerCountry**: Country of the customer

---

4. **OrderItems**

- **Id**: Unique identifier for the order item
- **OrderId**: Foreign key referencing the order
- **ProductId**: Foreign key referencing the product
- **ProductName**: Name of the product
- **Quantity**: Quantity of the product in the order
- **UnitPrice**: Unit price of the product
- **Currency**: Currency of the order

---

5. **Deliveries**

- **Id**: Unique identifier for the delivery
- **OrderId**: Foreign key referencing the order
- **SentDate**: Date when the order was sent
- **PlannedDeliveryDate**: Planned delivery date
- **EffectiveDeliveryDate**: Effective delivery date
- **AnyDeliveryIssue**: Whether there was any issue with the delivery
- **DeliveryCountry**: Country of the delivery

Data Loading

In [1]:
```
import pandas as pd
from pathlib import Path
from IPython.display import Markdown as md
```

In [2]:
```
excel_crm_data_path = Path("../../data/crm_data.xlsx")
```

In [3]:
```
def load_sheet(sheet_name: str, verbose: bool = True) -> pd.DataFrame:
    df = pd.read_excel(excel_crm_data_path, sheet_name=sheet_name)
```

```
    if verbose:
        print(f"{sheet_name} Data Shape: {df.shape}")
        display(df.head())

    return df
```

In [ ]:

In [4]:
```
customer_df = load_sheet("Customers", verbose=True)
```

```
Customers Data Shape: (5, 4)
```

	Id	Name	eMail	Country
0	CLT001	John Doe	john.doe@example.com	France
1	CLT002	Jane Smith	jane.smith@example.com	USA
2	CLT003	Alice Johnson	alice.j@example.com	USA
3	CLT004	Bob Brown	bob.brown@example.com	Spain
4	CLT005	Charlie Davis	charlie.d@example.com	France

In [5]:
```
product_df = load_sheet("Products", verbose=True)
```

```
Products Data Shape: (40, 9)
```

	Id	Name	ShortDescription	Color	Size	OriginCountry	DelivrableCountries
0	PRD001	Classic T-Shirt	Cotton T-shirt with crew neck	White	S, M, L	USA	USA, Canada, Mexico
1	PRD002	Skinny Jeans	Denim jeans with slim fit	Blue	28-36	Italy	Europe, USA
2	PRD003	Wool Cardigan	Cozy cardigan with button front	Grey	M, L	UK	UK, Ireland, France
3	PRD004	Floral Dress	Summer dress with floral prints	Yellow	XS-L	India	Asia, Australia
4	PRD005	Leather Jacket	Premium genuine leather jacket	Black	M, L, XL	Pakistan	Worldwide

In [6]:
```
order_df = load_sheet("Orders", verbose=True)
```

```
Orders Data Shape: (6, 5)
```

	Id	CustomerId	Date	CustomerName	CustomerCountry
0	ORD001	CLT001	2024-10-20	John Doe	France
1	ORD002	CLT003	2024-11-21	Alice Johnson	USA
2	ORD003	CLT001	2024-11-25	John Doe	France
3	ORD004	CLT002	2024-11-27	Jane Smith	USA
4	ORD005	CLT003	2024-12-01	Alice Johnson	USA

In [7]:
```
order_item_df = load_sheet("OrderItems", verbose=True)
```

```
OrderItems Data Shape: (22, 7)
```

	Id	OrderId	ProductId	ProductName	Quantity	UnitPrice	Currency
0	ITM001	ORD001	PRD002	Skinny Jeans	3	2	USD
1	ITM002	ORD001	PRD011	Galaxy S23 Ultra	1	500	USD
2	ITM003	ORD001	PRD013	Pixel 8 Pro	4	500	USD
3	ITM004	ORD001	PRD025	Amber Essence	4	80	USD
4	ITM005	ORD001	PRD037	Torres Sangre de Toro	10	15	USD

In [8]:
```
delivery_df = load_sheet("Deliveries", verbose=False)
delivery_df.loc[
    delivery_df["EffectiveDeliveryDate"].isnull(), "EffectiveDeliveryDate"
] = None
print("Deliveries Data Shape:", delivery_df.shape)
delivery_df.head()
```

```
Deliveries Data Shape: (8, 9)
```

Out[8]:

	Id	OrderId	SentDate	PlannedDeliveryDate	EffectiveDeliveryDate	AnyDelivery
0	DLV001	ORD001	2024-10-20	2024-10-24	2024-10-23	
1	DLV002	ORD002	2024-11-21	2024-11-22	None	
2	DLV003	ORD002	2024-11-24	2024-11-28	2024-11-25	
3	DLV004	ORD003	2024-11-25	2024-11-25	2024-11-26	
4	DLV005	ORD004	2024-11-27	2024-12-01	None	

In [ ]:
```

```

Data Modeling

The use case can be addressed in several ways:

- In a one shot prompt to a LLM
- Using multi-steps LLM workflows
- Orchestrating autonomous LLM Agents

While all of these approaches will be studied sooner or later, we'll only be considering the first one in this introductory notebook.

Too much talk, let's deep dive now !

In the above data presentation section, we can see that our datasets are designed in a top-down way. While this is optimal for storage in DBMS, it's far from ideal for LLM querying. In fact, in order to correctly handle the request, the LLM should be able to see all the customer's context at a glance, meaning the data should be presented in a bottom-up way where all the customer's information is packed into a single and compact object.

Bellow, we'll be using Pydantic to model our data in such way.

Base Pydantic Model

In [9]:
```
import re
from datetime import date
from typing import Any, Literal, Optional

import pandas as pd
from pydantic import BaseModel as pBaseModel
```

In [10]:
```
DELIVERY_STATUS = Literal[
    "DELIVERED", "AT_CHECKPOINT", "IN_TRANSIT", "CANCELLED", "RETURNED"
]
CURRENCY = Literal["USD", "EUR", "GBP", "CHF", "JPY", "CNY"]
```

In [11]:
```
def clean_field_name(field_name: str) -> str:
    return re.sub(r"[^a-z]", "", field_name.lower())
```

In [12]:
```
class BaseModel(pBaseModel):
    """
    Base data model, with some helper methods for object instantiation from dict
    """

    @classmethod
    def retrieve_raw_field_values(
        cls, data: dict[str, Any], strict: bool = True
    ) -> dict[str, Any]:

        data_keys = {clean_field_name(key): key for key in data}
        field_values: dict[str, Any] = {}

        for field_name in cls.model_fields:
            field_name_ = clean_field_name(field_name)
            try:
                field_values[field_name] = data[data_keys[field_name_]]
```

```
            except KeyError as e:
                if strict:
                    raise ValueError(f"Field {field_name} not found in data") fr

        return field_values

    @classmethod
    def preprocess_raw_field_values(
        cls, raw_field_values: dict[str, Any]
    ) -> dict[str, Any]:
        return raw_field_values

    @classmethod
    def from_dict(
        cls, data: dict[str, Any], *, strict: bool | None = None, **kwargs: Any
    ):
        strict = not bool(kwargs) if strict is None else strict
        raw_field_values = cls.retrieve_raw_field_values(data, strict=strict)
        raw_field_values.update(kwargs)
        return cls(**cls.preprocess_raw_field_values(raw_field_values))
```

Product Data Model

In [13]:
```
class Product(BaseModel):
    id: str
    name: str
    short_description: str
    color: str
    size: str
    origin_country: str
    delivrable_countries: list[str]
    universe: str

    @classmethod
    def preprocess_raw_field_values(cls, raw_field_values: dict[str, Any]):
        """Ensure delivrable_countries is a list of strings"""

        raw_field_values["delivrable_countries"] = re.split(
            r"\s*[,;]\s*", raw_field_values["delivrable_countries"]
        )
        return raw_field_values
```

In [14]:
```
product = Product.from_dict(product_df.iloc[0].to_dict())
product
```

Out[14]:
```
Product(id='PRD001', name='Classic T-Shirt', short_description='Cotton T-shirt
with crew neck', color='White', size='S, M, L', origin_country='USA', delivrabl
e_countries=['USA', 'Canada', 'Mexico'], universe='Cloth')
```

Delivery Data Model

In [15]:
```
class Delivery(BaseModel):
    id: str
    status: DELIVERY_STATUS
    sent_date: date
    planned_delivery_date: date
    effective_delivery_date: Optional[date]
```

```
    any_delivery_issue: bool
    delivery_country: str
```

In [16]:
```
delivery = Delivery.from_dict(delivery_df.iloc[0].to_dict())
delivery
```

Out[16]:
```
Delivery(id='DLV001', status='DELIVERED', sent_date=datetime.date(2024, 10, 2
0), planned_delivery_date=datetime.date(2024, 10, 24), effective_delivery_date=
datetime.date(2024, 10, 23), any_delivery_issue=False, delivery_country='Franc
e')
```

Order Item Data Model

In [17]:
```
class OrderItem(BaseModel):
    id: str
    product: Product
    quantity: int
    unit_price: float
    currency: CURRENCY
```

In [18]:
```
order_item = OrderItem.from_dict(order_item_df.iloc[0].to_dict(), product=produc
order_item
```

Out[18]:
```
OrderItem(id='ITM001', product=Product(id='PRD001', name='Classic T-Shirt', sho
rt_description='Cotton T-shirt with crew neck', color='White', size='S, M, L',
origin_country='USA', delivrable_countries=['USA', 'Canada', 'Mexico'], univers
e='Cloth'), quantity=3, unit_price=2.0, currency='USD')
```

Order Data Model

In [19]:
```
class Order(BaseModel):
    id: str
    date: date
    items: list[OrderItem]
    deliveries: list[Delivery]
```

In [20]:
```
order = Order.from_dict(
    order_df.iloc[0].to_dict(), items=[order_item], deliveries=[delivery]
)


order
```

Out[20]:
```
Order(id='ORD001', date=datetime.date(2024, 10, 20), items=[OrderItem(id='ITM00
1', product=Product(id='PRD001', name='Classic T-Shirt', short_description='Cot
ton T-shirt with crew neck', color='White', size='S, M, L', origin_country='US
A', delivrable_countries=['USA', 'Canada', 'Mexico'], universe='Cloth'), quanti
ty=3, unit_price=2.0, currency='USD')], deliveries=[Delivery(id='DLV001', statu
s='DELIVERED', sent_date=datetime.date(2024, 10, 20), planned_delivery_date=dat
etime.date(2024, 10, 24), effective_delivery_date=datetime.date(2024, 10, 23),
any_delivery_issue=False, delivery_country='France')])
```

Customer Data Model

In [21]:
```
class Customer(BaseModel):
    id: str
```

```
    name: str
    email: str
    country: str
    orders: list[Order]
```

In [22]:
```
customer = Customer.from_dict(customer_df.iloc[0].to_dict(), orders=[order])
customer
```

Out[22]:
```
Customer(id='CLT001', name='John Doe', email='john.doe@example.com', country='F
rance', orders=[Order(id='ORD001', date=datetime.date(2024, 10, 20), items=[Ord
erItem(id='ITM001', product=Product(id='PRD001', name='Classic T-Shirt', short_
description='Cotton T-shirt with crew neck', color='White', size='S, M, L', ori
gin_country='USA', delivrable_countries=['USA', 'Canada', 'Mexico'], universe
='Cloth'), quantity=3, unit_price=2.0, currency='USD')], deliveries=[Delivery(i
d='DLV001', status='DELIVERED', sent_date=datetime.date(2024, 10, 20), planned_
delivery_date=datetime.date(2024, 10, 24), effective_delivery_date=datetime.dat
e(2024, 10, 23), any_delivery_issue=False, delivery_country='France')])])
```

> Well done ! Now we have our data models defined and the `Customer` data model does pack all the customer information into a single object.

From Pandas DataFrames to a Compcat CRM Data

Here, we'll pack the dataframes into a dictionary of `Customer` objects. Let's define a function to do this.

In [23]:
```
def build_crm_data(
    customer_df: pd.DataFrame,
    product_df: pd.DataFrame,
    order_df: pd.DataFrame,
    order_item_df: pd.DataFrame,
    delivery_df: pd.DataFrame,
) -> dict[str, Customer]:

    products_dict = {
        product_dict["Id"]: Product.from_dict(product_dict)
        for product_dict in product_df.to_dict(orient="records")
    }

    crm_data: dict[str, Customer] = {}

    for customer_dict in customer_df.to_dict(orient="records"):

        orders: list[Order] = []

        for order_dict in order_df.query("CustomerId == @customer_dict['Id']").t
            orient="records"
        ):
            items = [
                OrderItem.from_dict(
                    order_item_dict, product=products_dict[order_item_dict["Prod
                )
                for order_item_dict in order_item_df.query(
                    "OrderId == @order_dict['Id']"
                ).to_dict(orient="records")
            ]
```

```
            deliveries = [
                Delivery.from_dict(delivery_dict)
                for delivery_dict in delivery_df.query(
                    "OrderId == @order_dict['Id']"
                ).to_dict(orient="records")
            ]
            order = Order.from_dict(order_dict, items=items, deliveries=deliveri

            orders.append(order)

        customer = Customer.from_dict(customer_dict, orders=orders)
        crm_data[customer.id] = customer

    return crm_data
```

In [24]:
```
crm_data = build_crm_data(
    customer_df=customer_df,
    product_df=product_df,
    order_df=order_df,
    order_item_df=order_item_df,
    delivery_df=delivery_df,
)
display(md(f"**number of customers:** {len(crm_data)}"))
# print(crm_data["CLT001"].model_dump_json(indent=2))
```

number of customers: 5

LLM for Customer Care

Getting Ready: Settings

The code is designed to work with any OpenAI API compatible LLM Provider. Just make sure to set the required environment variables:

- OPENAI_API_KEY
- OPENAI_BASE_URL (optional)

You could put these values in a `.env` file in the root of the project and then load them with `load_dotenv()`.

The prompts are parametrized using Jinja2 template engine.

Analyzing the Customer Requests

In [25]:
```
msg_df = load_sheet("Messages", verbose=True)
```

```
Messages Data Shape: (3, 6)
```

	Id	CustomerId	CustomereMail	Title	Body	Date
0	MSG001	CLT001	john.doe@example.com	Total Order Cost	Hello,\n\nThanks for the quick delivery. Could...	2024-11-24
1	MSG002	CLT001	john.doe@example.com	Delivery Issue	Hello,\n\nHow is it possible that after so man...	2024-12-20
2	MSG003	CLT001	john.doe@example.com	Delivery Issue	Hello,\n\nHow is it possible that my order con...	2024-12-20

In [26]:
```
num_eqs = 20  # number of equal signs to display each side (left and right)
```

In [27]:
```
def display_msg(row: pd.Series) -> None:
    display(
        md(
            f"""
<font color="teal">{"="*num_eqs} {row['Title']} {"="*num_eqs}</font>
**Request Date:** <font color="lightblue">{row["Date"]}</font>
**Request Title:** <font color="red">{row["Title"]}</font>

{row["Body"]}
___
<br><br>
"""
        )
    )
```

In [28]:
```
for row_idx, row in msg_df.iloc[:3].iterrows():
    display_msg(row)
```

========== Total Order Cost ==========

Request Date: 2024-11-24

Request Title: Total Order Cost

Hello,

Thanks for the quick delivery. Could you please send me my last Nov order total cost ?

Regards, John

========== Delivery Issue ==========
Request Date: 2024-12-20
Request Title: Delivery Issue

Hello,

How is it possible that after so many days after my 12 Dec order, I still not having received my due items. Could you please confirm that the items are not delivered to me yet and tell me when that would be the case ?

Regards, John

========== Delivery Issue ==========
Request Date: 2024-12-20
Request Title: Delivery Issue

Hello,

How is it possible that my order containing my beloved PREMIUM AUSTRLIAN SHIRAZ still not delivered after so many days ? Could you please confirm that the items are not delivered to me yet and tell me when that would be the case ?

Regards, John

Based on the above requests, a good prompt should:

- **Identify the Core Inquiry:** Distinguish between order details, delivery issues, or other concerns.
- **Extract Relevant Details:** Include order dates, product names, or any identifiable information provided.
- **Provide Empathy and Reassurance:** Acknowledge the concern with professionalism.
- **Guide Action:** Ask the system to retrieve necessary information and provide a clear response or next steps.

These key points should help the LLM to overcomme challenges such as :

- **Understanding Requests**: Ambiguity, language variability, and multi-concern queries.
- **Reference Disambiguition**: Identifying relevant information with insufficient and partial details

- **Tone Management**: Accurately detecting sentiment and responding empathetically.
- **Edge Cases**: wrong claims, non identifiable products, etc.

In [ ]:

Building the Prompt

Let's recall that here we're using a one-shot approach, where all the customer data is provided to the LLM at once. Then the LLM will analyse the customer request along with the context and return a, hopefully, relevant response. **May God Save us from Hallucinations** 😂 !

In [29]:
```
import jinja2
import json


jinja_env = jinja2.Environment()
```

In [30]:
```
def render_jinja_template(template_str: str, **kwargs: Any) -> str:
    return jinja_env.from_string(template_str).render(**kwargs)
```

In [31]:
```
prompt_as_jinja_template = Path(
    "../../prompt_templates/pure_llm/rich_pure_llm.md"
).read_text(encoding="utf-8")

md(
    f"""<br><font color="teal">{"="*num_eqs} Tailored Prompt {"="*num_eqs}</font
    + render_jinja_template(
        prompt_as_jinja_template,
        customer_request_date="2024-11-20",
        customer_request_title="Not Delivered Order",
        customer_request_body="I have a problem with my order which was not deli
        customer_data=json.dumps(
            {
                "id": "CLT000",
                "name": "Test Customer",
                "email": "test_customer@intellicart.com",
                "country": "France",
                "orders": [],
            }
        ),
    )
    + "\n___"
)
```

Out[31]:

========== Tailored Prompt ==========

As a Customer Relation Manager for IntelliCart Inc., you're tasked with addressing customer claims and requests effectively, using personalized information from their historical data. Ensure that each response is thorough and aligned with IntelliCart's customer service standards.

Leverage the customer's data, such as orders, deliveries, and purchased products, to provide a detailed response. Consider the context, past interactions, and any specific details or nuances that could enhance the customer's experience.

Steps

1. **Analyze the Customer Request:**
    - Review the request date, title, and body to understand the customer's issue or need.
    - Identify the primary concern and any secondary points mentioned.
2. **Examine Customer Data:**
    - Inspect the JSON customer data for relevant historical details like past orders, delivery issues, or frequent product purchases.
    - Cross-check any claims made by the customer with their order history and interactions.
3. **Formulate a Response:**
    - Acknowledge the customer's request and express a willingness to assist.
    - Provide information or a solution based on their historical data. Highlight relevant past orders or interactions.
    - Offer additional support or follow-up actions if needed.
4. **Conclude and Sign Off:**
    - Apologize for any inconvenience if appropriate, thank the customer for their patience, and assure them that their issue is being addressed.
    - Include IntelliCart's contact details.

Output Format

- Responses should be structured into a well-organized, clear, and polite paragraph (or paragraphs if necessary).
- Include a sign-off with IntelliCart's customer support contact number and email address at the end.

Examples

Example 1:

Customer Request / Claim

- Request Date: 2023-11-05
- Request Title: Delayed Delivery of Order 1538

"Hello, I ordered a blender on November 5, 2023, and it hasn't been delivered yet. Could you help me track my order?"

Data


```
{
  "customer_name": "John Doe",
  "orders": [
    {
      "order_id": 1538,
      "product": "Blender",
      "order_date": "November 5, 2023",
      "estimated_delivery": "November 10, 2023",
      "status": "In Transit"
    }
  ]
}
```


Response:

"Dear John Doe,

Thank you for reaching out regarding your order. I understand your concern about the delay. Upon checking, your blender (Order ID: 1538) is currently in transit and was estimated to be delivered by November 10, 2023. We apologize for the delay and any inconvenience this may have caused. I have contacted our delivery partner to expedite your order, and they anticipate it will arrive shortly.

For further assistance or if your order doesn’t arrive soon, please feel free to reach us at +3366666666 or support@intellicart.com. Thank you for your patience!

Best regards, IntelliCart Customer Support Team Phone: +3366666666 Email: support@intellicart.com"

Notes

- Always personalize the response based on the customer's name and specific details of their request.
- Ensure that any offered resolution lies within company policy.
- Maintain a warm and understanding tone throughout the communication.

Input

Customer Request / Claim

- Request Date: 2024-11-20
- Request Title: Not Delivered Order

I have a problem with my order which was not delivered.

Customer Data

{"id": "CLT000", "name": "Test Customer", "email": "test_customer@intellicart.com", "country": "France", "orders": []}

In [ ]:

Finally Calling the LLM

In [32]:
```
import os
from dotenv import load_dotenv
from openai import OpenAI
from openai.types.chat.chat_completion import ChatCompletion
```

In [33]:
```
load_dotenv()
DEFAULT_MODEL_NAME = "gpt-4o-mini"
client = OpenAI(
    api_key=os.getenv("OPENAI_API_KEY"), base_url=os.getenv("OPENAI_BASE_URL") o
)
```

In [34]:
```
def llm(
    content: str, model_name: str | None = DEFAULT_MODEL_NAME, max_tokens: int =
) -> ChatCompletion:
    model_name = model_name or DEFAULT_MODEL_NAME
    return client.chat.completions.create(
        model=model_name,
        messages=[{"role": "user", "content": content}],
        max_tokens=max_tokens,
    )
```

In [35]:
```
def get_and_display_llm_res(
    customer_msg: pd.Series,
    customer_data: Customer,
    llm_res: ChatCompletion | None = None,
) -> ChatCompletion:
    if llm_res is None:
        rendered_prompt = render_jinja_template(  # Fill in the prompt parameter
            prompt_as_jinja_template,
            customer_request_date=customer_msg["Date"],
            customer_request_title=customer_msg["Title"],
            customer_request_body=customer_msg["Body"],
            customer_data=customer_data.model_dump_json(indent=0),
        )
        llm_res = llm(rendered_prompt)
```

```
    display_msg(customer_msg)
    display(
        md(
            f"""<font color="green">{"="*num_eqs} LLM Response {"="*num_eqs}</fo
        {llm_res.choices[0].message.content}
___
<br><br>"""
        )
    )

    return llm_res
```

In [36]:
```
customer_id = "CLT001"
customer_data = crm_data[customer_id]


customer_msg = msg_df.query("CustomerId == @customer_id").iloc[0]


llm_res = get_and_display_llm_res(customer_msg, customer_data=customer_data)
```

========== Total Order Cost ==========

Request Date: 2024-11-24

Request Title: Total Order Cost

Hello,

Thanks for the quick delivery. Could you please send me my last Nov order total cost ?

Regards, John

========== LLM Response ==========

Dear John Doe,

Thank you for your message and for acknowledging the quick delivery of your recent order. I'm pleased to provide you with the total cost for your last order placed in October 2024.

Your order (Order ID: ORD001) included the following items:

1. **Skinny Jeans** - 3 units at $2.00 each = 6.00
2. **Galaxy S23 Ultra** - 1 unit at $500.00 = 500.00
3. **Pixel 8 Pro** - 4 units at $500.00 each = 2,000.00
4. **Amber Essence** - 4 units at $80.00 each = 320.00
5. **Torres Sangre de Toro** - 10 units at $15.00 each = 150.00

Adding these amounts together gives you a total order cost of **$2,976.00** for your order in October. If you need more details or any further assistance, please do not hesitate to reach out.

Thank you for choosing IntelliCart, and I'm here to assist you with any other inquiries you may have.

Best regards,
IntelliCart Customer Support Team
Phone: +3366666666
Email: support@intellicart.com

> Bingo ! We got a response, and It seems to be quite convincing ! Not only the answer looks very natural and the tone is very friendly but the LLM manages to identify the relevant `order` despite the **partial** details provided by the customer. **Impressive** !
>
> Let's try it one time again !

In [37]:
```
customer_id = "CLT001"
customer_data = crm_data[customer_id]


customer_msg = msg_df.query("CustomerId == @customer_id").iloc[1]


llm_res = get_and_display_llm_res(customer_msg, customer_data=customer_data)
```

========== Delivery Issue ==========
Request Date: 2024-12-20
Request Title: Delivery Issue

Hello,

How is it possible that after so many days after my 12 Dec order, I still not having received my due items. Could you please confirm that the items are not delivered to me yet and tell me when that would be the case ?

Regards, John

========== LLM Response ==========
Dear John Doe,

Thank you for reaching out to us regarding the delivery of your recent order dated December 12, 2024. I understand your frustration about not having received your items yet. Upon reviewing your account, I found that the order (Order ID: ORD006) includes several items, such as the Classic T-Shirt, Polo Shirt, Galaxy S23 Ultra, Lavender Bliss, and Penfolds Grange.

Unfortunately, the delivery for this order has been cancelled due to unforeseen circumstances. I sincerely apologize for the inconvenience this has caused you. We are currently working with our delivery partners to resolve the issue, and I will ensure that you are updated on the status of your order as soon as possible.

In the meantime, if you would like to discuss alternative solutions or require further assistance, please do not hesitate to contact us. You can reach our customer support at +3366666666 or via email at support@intellicart.com. Thank you for your understanding and patience during this time.

Best regards,
IntelliCart Customer Support Team
Phone: +3366666666
Email: support@intellicart.com

> Again, we got a very interesting response ! The LLM seems to be able to understand the context of the request and the data provided by the customer.
>
> Let's try it one time more !

In [38]:
```
customer_id = "CLT001"
customer_data = crm_data[customer_id]


customer_msg = msg_df.query("CustomerId == @customer_id").iloc[2]


llm_res = get_and_display_llm_res(customer_msg, customer_data=customer_data)
```

========== Delivery Issue ==========

Request Date: 2024-12-20

Request Title: Delivery Issue

Hello,

How is it possible that my order containing my beloved PREMIUM AUSTRLIAN SHIRAZ still not delivered after so many days ? Could you please confirm that the items are not delivered to me yet and tell me when that would be the case ?

Regards, John

========== LLM Response ==========

Dear John Doe,

Thank you for your patience and for reaching out regarding your order containing your PREMIUM AUSTRALIAN SHIRAZ. I understand your concerns about the delivery delay, and I'm here to assist you.

Your recent order (Order ID: ORD006) placed on December 12, 2024, included five bottles of Penfolds Grange, a beloved choice! Unfortunately, the delivery for this order was cancelled due to an issue that arose on December 12, 2024, which was the planned delivery date. I apologize for any frustration this may have caused.

We are currently reviewing the situation to understand what went wrong and will work on resolving it promptly. I suggest we can either process a new delivery attempt or explore other options depending on your preference. Please let me know how you would like to proceed, and I will ensure your request is prioritized.

If you have any further questions or need additional assistance, feel free to reach out. Our team is here to help!

Best regards,
IntelliCart Customer Support Team
Phone: +3366666666
Email: support@intellicart.com

Again, an interesting response ! Let me recall that this one is a little tricky since the only information provided by the customer is the name of a product inside the order, and to make it worse, the product name is mispelled (very realistic, **isn't it?**). The LLM manages to overcome all these glitches and provide a very interesting response.

In [ ]:

Conclusion and Next Steps

In this tutorial, we've explored how to leverage the power of a single Large Language Model (LLM) to (partially) automate customer care processes for an e-commerce company. By analyzing customer requests, we crafted a tailored prompt that combines both the request and relevant customer data. Thanks to the **Pydantic** library, we were able to structure this customer data in a streamlined, bottom-up approach—packing everything into a single, compact object. This setup is perfect for querying LLMs.

What's truly exciting is that the LLM was able to generate responses that felt natural and relevant. This ability holds immense potential to not only automate but also support

customer care teams in handling inquiries with greater efficiency and accuracy.

Now, there are at least two key enhancements to take our solution to the next level:

1. **Enhancing the One-Shot Approach**
   By refining our current setup, we can make it even more powerful. Here's how:

   - **Advanced Prompt Engineering**: Experiment with techniques like:
     - **Chain of Thoughts** to improve reasoning and accuracy.
     - **Few-Shot Learning** to provide better context and examples to the model.
   - **Context Refinement**: Provide richer context to the LLM by integrating detailed data schemas, which help the model understand the nuances of customer information.
   - **Exploring Different LLMs**: Test various LLMs to see which best fits the needs of the business and improves the quality of responses.
   - **Fine-Tuning**: Fine-tune the LLM using specific customer data to further personalize the responses (though this should be reserved for scenarios where it's necessary).

2. **Shifting to a Multi-Step Approach**
   For a more sophisticated and scalable solution, consider adopting a **multi-step workflow** or **LLM Agents**. This method breaks down tasks into distinct steps, making the process more organized and adaptable. By leveraging workflows or agents, we can create dynamic systems that handle more complex customer interactions, evolving as they learn from ongoing inputs.

With these improvements, we're looking at an exciting future where AI not only supports customer care teams but enhances the entire experience for both customers and businesses alike.

Keep in mind though that while the one-shot approach offers simplicity and has demonstrated impressive results for our use case, it comes with **significant limitations**—particularly when dealing with extensive customer histories. In fact, this method relies on feeding all customer information into the LLM's context, which **can quickly exhaust token limits**, making it **inefficient for long interactions**. Furthermore, this approach **relinquishes control over the LLM**, which becomes solely responsible for processing and generating responses.

This lack of modularity complicates evaluation, as the output is a single, subjective result that's difficult to analyze or refine (though evaluation isn't the primary focus here). A more effective alternative would be to adopt a system with streamlined components—such as dedicated data extractors—that simplifies both the process and its assessment. Breaking the task into smaller, manageable pieces not only improves efficiency but also ensures greater control over the system's performance.

In the next tutorial, we will see how to use a LLM to automate customer care in a more advanced way, using a multi-steps LLM workflow. This will allow the LLM to handle more complex customer inquiries with precision. This approach, not only would hopefully enhance response quality but also gives us greater control over critical elements such as data extraction, validation, and even tone management.

The last part of this tutorial series will focus on **LLM Agents** that can be used to create more complex workflows.

In the final chapter of this series, we'll unlock the power of **LLM Agents**, which will empower you to design and implement even more advanced and dynamic workflows for better customer service.

Stay tuned!

Kossi NEROMA Sr Data Scientist