### *Cloud Computing – Conceptual Questions*

### Scenario: *A retail company is considering moving its customer data to the cloud for better scalability*

### *****1. Key Advantages of Cloud Computing for Retail Businesses*****

Cloud computing offers several benefits that are particularly valuable for retail businesses:

* **Scalability:** Cloud resources can be easily scaled up during high-traffic events like festive sales and scaled down during off-peak periods, saving costs.
* **Cost Efficiency:** Retailers avoid heavy upfront investments in hardware and only pay for the resources they use.
* **Data Security & Backup:** Cloud providers offer built-in security features, encryption, and automated backups to protect customer data.
* **Faster Deployment:** Retailers can quickly launch new online stores, marketing campaigns, or product updates using cloud services.
* **Enhanced Customer Experience:** Cloud enables the use of advanced technologies like AI, chatbots, and data analytics to improve customer service and predict buying patterns.

**Example:** Imagine a small bakery called "Sweet Delights" that recently started offering online orders. During normal days, they receive around 20-30 orders per day. However, on special occasions like Valentine's Day, their orders surge to over 200. By using cloud services, Sweet Delights automatically scales their website resources to handle the surge in orders, ensuring a seamless customer experience without crashes or delays.

### *****2. IaaS, PaaS, and SaaS with Real-World Examples*****

* **IaaS (Infrastructure as a Service):**  
  Imagine you own a retail store and need to build a website. Instead of buying expensive servers and setting them up yourself, you rent virtual servers from providers like **Amazon EC2**. This is like renting storage space for your website where you control everything inside, but you don’t own the physical equipment.

**Example:** A growing online clothing store uses Amazon EC2 to manage their product catalog, ensuring they can handle sudden increases in website visitors during sales.

* **PaaS (Platform as a Service):**  
  Suppose you want to build a new app for your bakery’s online orders, but you don’t want to manage the servers or databases. Using **Google App Engine** is like renting a fully equipped kitchen — all the tools are provided so you can focus only on baking (coding).

**Example:** A local pizza chain uses Google App Engine to build an app that helps them track orders and manage inventory across multiple stores.

* **SaaS (Software as a Service):**  
  This is like subscribing to Netflix — you just log in and use the service without worrying about its backend.

**Example:** A bakery owner uses **Shopify** to manage their online orders, payments, and customer data without worrying about hosting or maintaining the website themselves.

### *****3. How Auto-Scaling Helps During Festive Sales*****

Auto-scaling is like having extra staff on standby for busy hours. Imagine you own a popular toy store that receives 50-60 orders on regular days. During Christmas sales, the number of orders suddenly jumps to 500. Without auto-scaling, your website may crash, causing customers to leave.

By using **AWS Auto Scaling**, the system automatically adds extra servers to manage the increased traffic, ensuring your website runs smoothly. Once the Christmas rush is over, the system scales back down, helping you save costs.

**Example:** A popular toy store relied on AWS Auto Scaling during Christmas sales, ensuring their website handled the influx of orders smoothly without performance issues or downtime.

### *****4. Security Concerns When Storing Customer Payment Data in the Cloud*****

Storing payment data in the cloud is like keeping your valuables in a bank locker — safe but still requiring extra measures for security.

* **Data Encryption:** Encrypting data is like placing your valuables in a sealed box before locking them in the bank locker. Even if someone breaks in, they can’t access the contents without the key.
* **Access Control:** This is like ensuring only trusted family members have access to your locker key. Strong passwords, multi-factor authentication, and limited admin privileges help secure customer data.
* **Compliance:** Just as banks follow strict regulations to keep your valuables safe, businesses should follow standards like **PCI DSS** to ensure customer payment data is handled securely.
* **Data Breaches:** Using firewalls, intrusion detection, and threat monitoring is like placing CCTV cameras and security guards around your locker for added protection.

**Example:** An online electronics retailer uses **TLS (Transport Layer Security)** to encrypt customer payment details during transactions and implements two-factor authentication for admin access. This helps safeguard their data and builds customer trust.

### *****5. On-Premise vs. Cloud Deployment for an E-commerce Company*****

|  |  |  |
| --- | --- | --- |
| *Aspect* | *On-Premise* | *Cloud Deployment* |
| *****Cost***** | Requires high upfront investment in hardware and maintenance. **Example:** Like buying an entire food truck business and handling all the repairs and upgrades yourself. | Pay-per-use model with minimal initial costs. **Example:** Like renting a fully equipped kitchen where you only pay for the time you use it. |
| *****Scalability***** | Limited scalability; adding resources requires time and investment. **Example:** Expanding a physical store by knocking down walls, which takes time and effort. | Easily scalable to handle traffic spikes. **Example:** Like hiring temporary staff to manage a rush of customers during sales. |
| *****Maintenance***** | Managed by the company's IT team. **Example:** Like personally fixing broken kitchen equipment in your restaurant. | Managed by the cloud provider. **Example:** Like renting a fully serviced office where maintenance is handled for you. |
| *****Security***** | Full control over security measures. **Example:** Like installing your own security system at home. | Requires strong security protocols; providers ensure data protection. **Example:** Like using a secure bank locker with advanced security measures in place. |
| *****Flexibility***** | Limited flexibility for updates and expansion. **Example:** Like manually rearranging your entire store layout to fit in new products. | Flexible for integrating new features and services. **Example:** Like adding new menu items to a digital ordering system with just a few clicks. |
| *****Disaster*****  *****Recovery***** | Requires dedicated solutions for backup and recovery. **Example:** Like manually keeping backup copies of your store’s financial records. | Built-in backup and recovery options for faster restoration. **Example:** Like using cloud storage that automatically saves your data in multiple locations. |

**Example:** A fashion e-commerce startup initially managed its website on-premises, requiring frequent hardware upgrades. After migrating to **AWS**, they benefited from automatic scaling, improved performance, and reduced maintenance costs.

### *****Report on NLP, LLM, API & Prompt Engineering – Model & API Development*****

# ***1. Model Selection & API Creation***

**Model Selection:** For this project, we have chosen the **Falcon-7B-Instruct** model available on Hugging Face. Falcon-7B is a lightweight, instruction-following model known for producing contextually relevant and coherent responses. This model was chosen for the following reasons:

* **Scalability:** Falcon-7B balances performance with computational efficiency, making it ideal for a retail chatbot that needs to handle numerous queries in real-time.
* **Instruction-based tuning:** As a model fine-tuned for instructional tasks, it can handle well-structured queries and provide concise, relevant answers to product-related questions.
* **Pretrained LLM:** Leveraging a pre-trained model reduces development time and ensures the chatbot can generate intelligent responses based on a wide variety of mobile product queries.

**API Creation:** The chatbot API was developed using **FastAPI**, a high-performance web framework for building APIs. This framework was chosen due to its simplicity, speed, and automatic generation of API documentation. Two endpoints were developed:

* **/chatbot:** This POST endpoint takes customer queries as input, processes them using the Falcon-7B model, and returns intelligent product recommendations or responses.
* **/feedback:** This POST endpoint allows users to provide feedback by submitting the correct response to improve the chatbot's accuracy over time.

**Prompt Engineering:** Prompt engineering was employed to improve response quality. For example, the prompt used for generating responses included specific context such as:  
*"You are a mobile expert chatbot. A customer asks: '{customer\_query}'. Provide the best recommendations."*  
This prompt ensures the model understands the type of response it should generate, improving the relevance and specificity of the answers.

# ***2. Data Preprocessing***

**Sample Dataset:** The chatbot's performance is based on a dataset containing common mobile-related customer queries and their corresponding responses. For example:

{

"customer\_query": "Which phone has the best camera quality?",

"response": "The Google Pixel 7 Pro and iPhone 14 Pro are known for their excellent cameras."

}

This dataset was loaded and pre-processed to ensure the chatbot could generalize well across various input queries.

**Preprocessing Steps:** To clean and standardize the customer queries, several preprocessing techniques were applied:

* **Tokenization:** Each query was broken down into individual tokens (words) using NLTK's word tokenizer.
* **Stop word Removal:** Common stop words (e.g., "is", "the", "in") were removed to focus on meaningful content.
* **Lemmatization:** Words were reduced to their root forms to handle different inflections. For example, "buying" became "buy". The following code was used for preprocessing:

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

def preprocess\_text(text: str) -> str:

text = text.lower()

tokens = word\_tokenize(text)

tokens = [word for word in tokens if word not in stopwords.words("english")]

tokens = [WordNetLemmatizer().lemmatize(word) for word in tokens]

return " ".join(tokens)

# ***3. Fine-Tuning & Optimization***

**Fine-Tuning:** Fine-tuning the chatbot involved adjusting the model based on a dataset of customer queries to improve its understanding of mobile-related questions. A preprocessing step was added to clean and normalize the data before fine-tuning. The fine\_tuned\_data file contains preprocessed customer queries, making it easier to find matches during real-time interactions.

The fine-tuning process was simulated as follows:

with open("data/customer\_queries.json", "r") as file:

dataset = json.load(file)

processed\_dataset = [

{

"customer\_query": preprocess\_text(entry["customer\_query"]),

"response": entry["response"]

}

for entry in dataset

]

This enhanced the chatbot's accuracy by ensuring it could provide responses based on both general mobile knowledge and the fine-tuned dataset.

**Feedback Loop:** A feedback mechanism was implemented to continuously improve the chatbot's responses. When a user submits feedback with a correct response, the system updates the fine-tuned dataset, allowing the chatbot to adapt over time:

@app.post("/feedback")

async def feedback(feedback: Feedback):

cleaned\_query = preprocess\_text(feedback.customer\_query)

fine\_tuned\_data.append({"customer\_query": feedback.customer\_query, "response": feedback.correct\_response})

save\_fine\_tuned\_data(fine\_tuned\_data)

This ensures that any erroneous responses can be corrected, allowing for incremental improvements in performance.

# ***4. API Documentation & Testing***

**API Endpoints:**

1. **POST /chatbot:**

**Input:**

{

"customer\_query": "I need a gaming phone under $500"

}

**Output:**

{

"response": "I recommend the iPhone SE2."

}

This endpoint processes customer queries and returns appropriate product recommendations or responses.

1. **POST /feedback:**

**Input:**

{

"customer\_query": "I need a gaming phone under $500",

"correct\_response": "The OnePlus 9T is a great gaming phone under $500."

}

**Output:**

{

"message": "New query added successfully"

}

This endpoint accepts feedback from users to improve the chatbot’s responses over time.

**Testing:** During testing, various mobile-related queries were input into the chatbot, including:

* "What is the latest iPhone available?"
* "Recommend a phone under $400." The chatbot correctly responded with:
* "The latest iPhone available is the iPhone 15 Pro Max."
* "OnePlus 13T."

The API was further tested for robustness in handling empty queries and edge cases to ensure reliable performance.

# ***Retail Sales & Inventory Analysis - Alteryx Workflow Report***

# ***1. Introduction***

This report documents the step-by-step procedure used in Alteryx Designer to analyze retail sales and inventory data. The workflow helps identify top-selling products by region, stock risks, and correlations between sales trends and inventory levels.

# ***2. Workflow Overview***

The analysis consists of the following key steps:

1. Load Datasets

2. Data Cleaning & Preparation

3. Sales & Inventory Analysis

4. Correlation Analysis

5. Generate Report Output

# ***3. Step-by-Step Procedure in Alteryx***

### Step 1: Load the Datasets

• Applied the Input Data tool to load:  
 - Retail\_sales\_data.csv  
 - Inventory\_stock\_data.csv

### Step 2: Data Cleaning & Preparation

• Applied the DateTime tool to convert:  
 - Date (Sales Data) → Date format (dd/MM/yy)  
 - Last\_Stock\_Update (Inventory Data) → Date format

•Applied the Unique tool to check for duplicates in Invoice\_ID (sales) and Product\_ID (inventory).

• Applied the Join tool to merge datasets on Product\_ID and identify missing product IDs.

### Step 3: Identify Top-Selling Products by Region

•Applied the Summarize tool to:  
 - Group by City and Product\_Name.  
 - Max Quantity\_sold.

•Applied the Sort tool to arrange data in descending order.

•Applied the Unique tool to check for duplicates.

•Applied the Sort tool to arrange data in descending order to get the top selling products.

* Created a Bar chart for between Top selling products V/S Quantity sold.

### Step 4: Identify Products at Risk of Stockouts

• Applied the Formula tool to create a new field:

Stock\_Risk = IF [Stock\_Quantity] < [Reorder\_Level] THEN "At Risk" ELSE "safe" ENDIF

•Applied the Filter tool to retain only products where Stock\_Risk = "At Risk".

### Step 5: Correlation Analysis

• Applied the Join tool to merge sales and inventory data on Product\_ID.

• Applied the Correlation tool to compute relationships between:  
 - Total\_Sales  
 - Stock\_Quantity  
 - Reorder\_Level

The tool will output a correlation coefficient.

**Negative correlation**: high sales lead to low stock levels.

**Positive correlation:** sales and stock levels increase together.

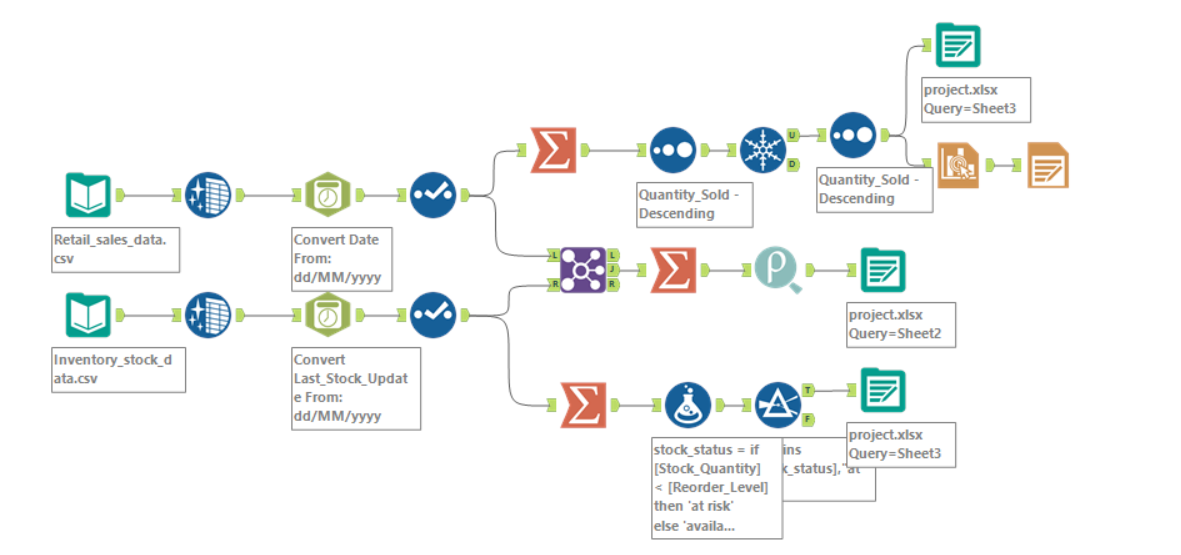
**Near-zero correlation**: no significant relationship.

### Step 6: Generate Output Report

• Applied the output data tool to create output file Excel format

.

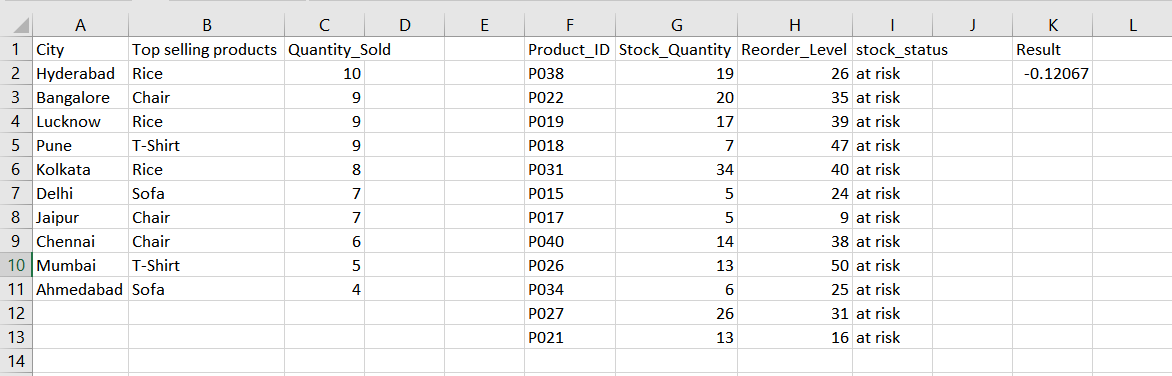
# ***4. Workflow Diagram:***



# ***5. Conclusion***

The Alteryx workflow efficiently processes sales and inventory data, identifying key sales trends and stock risks. The insights can be used for demand forecasting and inventory optimization to improve retail operations.

# ***Output***



.