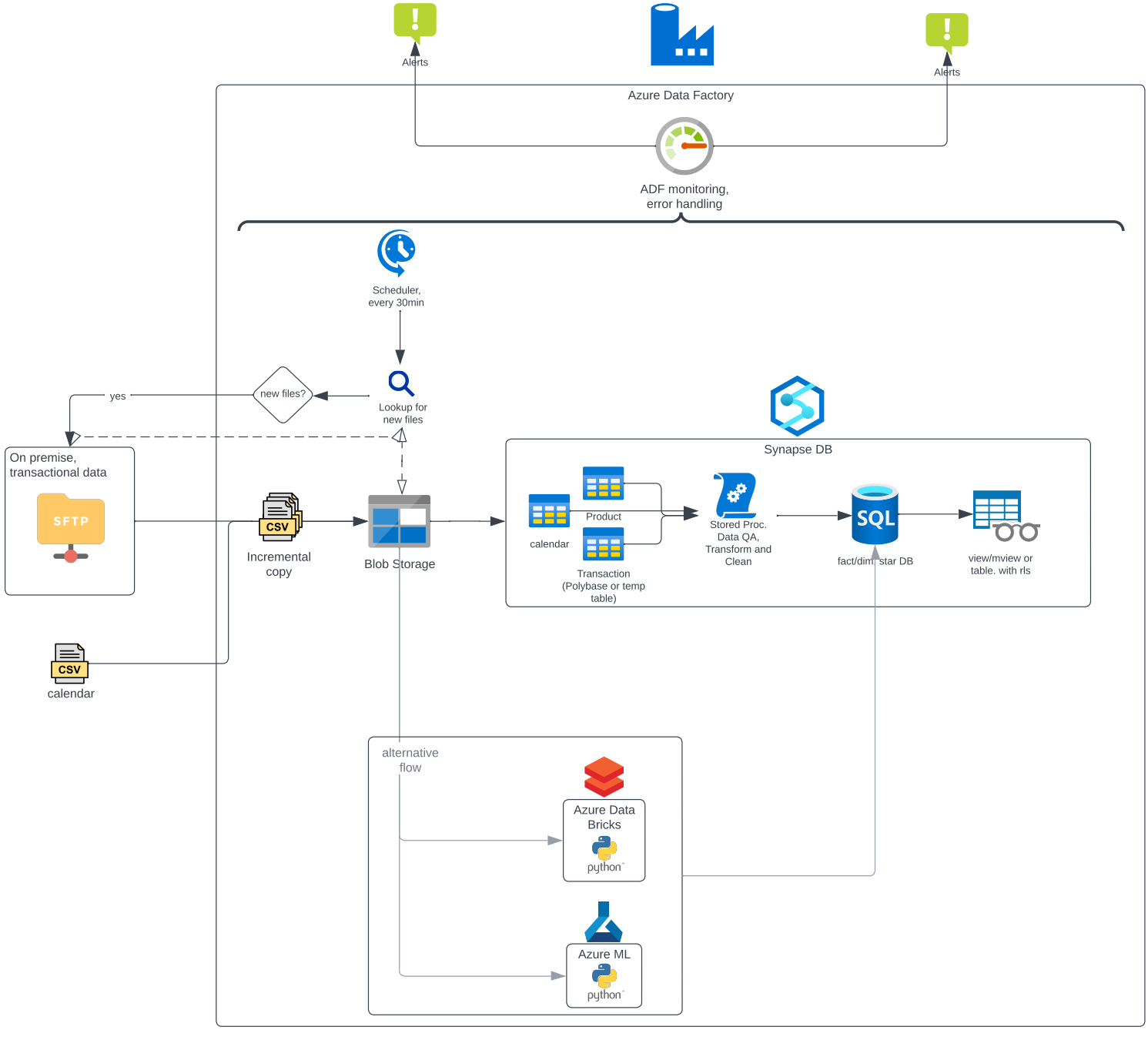
Alerts
Alerts
Azure Data Factory
ADF monitoring,
error handling
Scheduler,
every 30min
Lookup for
new files
new files?
yes
On premise,
transactional data
SFTP
CSV
Incremental
copy
Blob Storage
CSV
calendar
Synapse DB
calendar
Product
Transaction
(Polybase or temp
table)
Stored Proc.
Data QA,
Transform and
Clean
SQL
fact/dim star DB
view/mview or
table. with rls
alternative
flow
Azure Data
Bricks
python
Azure ML
python

2. How would you clean the data and make it ready for analysis?

1. Detecting and Handling Missing Values
Based on the nature of the data and its availability, options include:
· Replacing with Substitutes from a Reliable Source: Using data from a trusted external source to fill in the missing values.
· Impute missing values based on the conditions or rules derived from other data attributes. For instance, if a product's price is missing, one might use the average price of same/similar products as a substitute
· Using Indicator Variables or categorizing reasons of missing data
Instead of imputing missing values, add a binary indicator variable to denote whether the value was missing for a particular record or coding reasons if available. This approach can be useful in machine learning models to capture the pattern of missingness itself as a feature
· Model base imputation (mostly for ml tasks)
· Last Observation Carried Forward (LOCF) & Next Observation Carried Backward (NOCB)
· Keeping as Null: Maintain missing values as null where it makes sense for the analysis or downstream processes.

2. Detecting and Correcting Inaccuracies
Obvious Inconsistencies and Data Validation: Apply domain-specific rules to identify and correct errors (e.g., sales cannot be negative, dates must fall within a certain range, or address mismatched records). SQL CHECK constraints or Python assertions can be used for this purpose.

Identifying Outliers: Use statistical summaries and visualizations to pinpoint outliers that might indicate inaccuracies, such as unusually high or low values in sales data that could signify errors.
Anomaly Detection: Implement anomaly detection frameworks, such as the Isolation Forest model, to systematically identify data quality issues. Examples can be found in my GitHub repository: https://github.com/falko13/DataQualityChecks.

3. Standardizing Formats
Date and Time: Standardize date and time formats across the dataset to ensure consistency.
Numerical Data: Ensure numerical data uses a consistent number of decimal places and standardize units of measure where applicable.
Text Data: Standardize the casing of text data (e.g., converting all text to lower case), trim whitespace, and remove irrelevant special characters.
4. Deduplication
Identify Duplicates: Use methods to detect duplicate records within your data.
Resolve Duplicates: Determine the best approach for handling duplicates, which may involve keeping the first occurrence, aggregating data from duplicates, or deleting all but one record to ensure data uniqueness.
5. Data Type Conversions
Ensure Correct Data Types: Convert data to the appropriate type for analysis, such as transforming strings representing numbers to numeric data types, or ensuring all categorical data is treated as such.
6. Handling Erroneous or Inconsistent Entries
Regular Expressions: Employ regular expressions to identify and correct patterns of erroneous data, especially in text fields.
Normalization: Normalize text data to ensure consistency, such as replacing synonyms with a standard term across datasets.
7. Integration and Harmonization
Key Harmonization: Ensure that keys used to join different datasets are consistent, including formats and value ranges. This might involve mapping product IDs to a standard format across transaction and product datasets.
Categorization: Standardize and categorize data, especially when integrating datasets. For example, ensure that product categories are consistent across different sources for accurate analysis.

3. How would you merge and summarize the data to help answer the business question? You are encouraged to write out a simple query in SQL, R, Python or any other type of code or language.

Basic

1. Historical Sales
Approach: Calculate the average sales per product/store during the previous holiday seasons. This method assumes that sales patterns will be similar to the past.
Pros: Easy to implement and understand.
Cons: May not account for trends, changes in consumer behavior, or unique circumstances of the current

2. Weighted Historical Averages
Approach: apply more weight to more recent sales data, assuming that recent behavior is more indicative of future sales than older data. This can help capture trends without explicitly modeling them.
Implementation: Calculate the weighted average of product sales, where sales closer to the prediction period (e.g., last year's Christmas season) have higher weights.

3. Error Correction Model
Approach: Use an error correction approach to adjust future sales predictions based on the discrepancies between past predictions and actual sales.
Implementation: After making initial predictions, compare them with actual sales data as it becomes available during the early part of the holiday season. Use the differences to adjust predictions for the remainder of the season.

Advanced

1.Time Series Analysis (e.g., ARIMA, Seasonal Decomposition)
Approach: Apply time series analysis techniques like ARIMA (AutoRegressive Integrated Moving Average) to predict future sales based on past patterns, seasonality, and trends.
Pros: More accurately captures seasonal effects and trends.
Cons: Requires more sophisticated data preparation and analysis skills.

2. Machine Learning Models (e.g., Random Forest, XGBoost)
Approach: Train machine learning models on historical sales data along with other variables such as store location, product category. These models can capture complex non-linear relationships.
Pros: Can incorporate a wide range of factors and learn complex patterns.
Cons: Needs time in model tuning and validation.

3. Deep Learning Models (e.g., LSTM)
Approach: Use LSTM (Long Short-Term Memory) networks, a type of recurrent neural network, to model sales data sequences. This approach can capture long-term dependencies and patterns in sales data.
Pros: Highly effective for sequential data with long-term dependencies.
Cons: Complex to implement, requires a lot of data, and is computationally intensive.

4. Ensemble Methods
Approach: Combine predictions from multiple models (e.g., time series, machine learning) to improve accuracy. This could involve weighting predictions based on historical performance.
Pros: Can leverage the strengths of various models to improve overall predictions.
Cons: Increases complexity in model management and interpretation.

5. Demand Forecasting with External Data
Approach: Integrate external data sources such as weather forecasts, economic indicators, or events to improve prediction accuracy. This acknowledges that sales might be influenced by external factors beyond historical sales data.
Pros: Provides a more holistic view of potential demand drivers.

```sql
WITH time_windows AS (
    SELECT
        '2023-12-15' AS current_christmas_start,
        '2024-01-08' AS current_christmas_end,
        '2022-12-15' AS previous_christmas_start,
        '2023-01-08' AS previous_christmas_end,
        '2023-11-01' AS collection_start,
        '2024-01-08' AS collection_end
),
active_products AS (
    SELECT DISTINCT
        product_id
    FROM product_data
    WHERE DeactivationDate IS NULL
),
active_stores_products AS (
    SELECT DISTINCT
        t.store_id,
        ap.product_id
    FROM transactions t
    CROSS JOIN active_products ap
    WHERE DATE(t.transaction_begin_date) BETWEEN (SELECT collection_start FROM time_windows) AND (SELECT collection_end FROM time_windows)
       OR DATE(t.transaction_begin_date) BETWEEN (SELECT previous_christmas_start FROM time_windows) AND (SELECT previous_christmas_end FROM time_windows)
),
relevant_dates AS (
    SELECT calendar_date
    FROM calendar
    WHERE calendar_date BETWEEN (SELECT current_christmas_start FROM time_windows) AND (SELECT current_christmas_end FROM time_windows)
),
comprehensive_combinations AS (
    SELECT
        rd.calendar_date,
        asp.store_id,
        asp.product_id
    FROM relevant_dates rd
    CROSS JOIN active_stores_products asp
),
current_year_transactions AS (
    SELECT
        store_id,
        product_id,
        SUM(COALESCE(sale_quantity, 0)) AS sale_quantity_sum,
        DATE(transaction_begin_date) AS transaction_date
    FROM transactions
    WHERE DATE(transaction_begin_date) BETWEEN (SELECT current_christmas_start FROM time_windows) AND (SELECT current_christmas_end FROM time_windows)
    AND Transaction_Status <> 'Voided'
    GROUP BY store_id, product_id, DATE(transaction_begin_date)
),
previous_year_transactions AS (
    SELECT
        store_id,
        product_id,
        SUM(COALESCE(sale_quantity, 0)) AS sale_quantity_sum,
        DATE(transaction_begin_date) AS transaction_date
    FROM transactions
    WHERE DATE(transaction_begin_date) BETWEEN (SELECT previous_christmas_start FROM time_windows) AND (SELECT previous_christmas_end FROM time_windows)
    AND Transaction_Status <> 'Voided'
    GROUP BY store_id, product_id, DATE(transaction_begin_date)
),
aggregated_transactions AS (
    SELECT
        cc.store_id,
        cc.product_id,
        cc.calendar_date,
        pd.product_category,
        pd.product_name,
        pd.unit_of_measure,
        pd.supplier,
        cyt.transaction_date,
        cyt.sale_quantity_sum AS sale_quantity_sum_current_year,
        pyt.sale_quantity_sum AS sale_quantity_sum_previous_year,
        SUM(cyt.sale_quantity_sum) OVER(PARTITION BY cc.store_id, cc.product_id ORDER BY cc.calendar_date) AS sales_cumulative_current_year,
        SUM(pyt.sale_quantity_sum) OVER(PARTITION BY cc.store_id, cc.product_id ORDER BY cc.calendar_date) AS sales_cumulative_sum_previous_year

    FROM comprehensive_combinations cc
    LEFT JOIN current_year_transactions cyt ON cc.store_id = cyt.store_id AND cc.product_id = cyt.product_id AND cc.calendar_date = cyt.transaction_date
    LEFT JOIN previous_year_transactions pyt ON cc.store_id = pyt.store_id AND cc.product_id = pyt.product_id AND cc.calendar_date = DATE_ADD(pyt.transaction_date, INTERVAL 1 YEAR)
    LEFT JOIN product_data pd ON cc.product_id = pd.product_id
)
SELECT *
FROM aggregated_transactions
ORDER BY store_id, product_id, calendar_date;
```

4. How would you optimize this database query for better performance? How would you troubleshoot performance issues?

Optimizing
Indexing:
· Ensure that indexes are created on columns that are frequently used in WHERE clauses, JOIN conditions, and as part of the ORDER BY and GROUP BY clauses. However, over-indexing can slow down write operations.
· Index Maintenance

Query Refactoring:
Simplify Queries: Break complex queries into simpler subqueries that can be executed efficiently.
Avoid Selecting Unnecessary Columns
Optimize Joins: Ensure that joins are made on indexed columns and reconsider the join order to minimize the size of intermediate result sets.

Use of Query Caching:
Implement caching mechanisms to store the results of frequently executed queries, reducing the need to access the database for repeated requests.

Partitioning:
For large tables

Optimize Data Models:
Normalize, but also consider denormalization where necessary to reduce complex joins in read-heavy operations.

Troubleshooting Performance Issues
Detailed logging
Benchmarking: execution time measurement
Debugging and profiling tools
EXPLAIN Plans: Run EXPLAIN (or equivalent) on your query to understand the execution plan, which shows how the database engine plans to execute your query. Look for full table scans, large joins without indexes, and inefficient operations.
Monitoring Tools: Utilize database monitoring tools to identify slow queries, resource-intensive operations

Configuration Tuning:
Database Configuration: adjust parallelism settings, to align with your workload and hardware capabilities.

Regular Maintenance:
Perform routine maintenance tasks such as updating statistics, rebuilding indexes, and cleaning up fragmented data to keep the database performing optimally.

What considerations would you put in place to ensure data accuracy and clear data versioning?

Ensuring Data Accuracy

Validation at Point of Entry:
Strict validation rules for data entering the system, including checks for data types, formats, and range constraints. This can prevent incorrect data from being stored at the source.

Data Quality Framework:
Establish a comprehensive data quality management framework that includes routine checks for accuracy, completeness, consistency, and timeliness of data. Use automated tools where possible to scan for anomalies and outliers.

Standardization and Cleansing:
Apply standardization rules to ensure data uniformity across systems. Use data cleansing processes to correct known inaccuracies, like incorrect dates, etc.

Use of Master Data Management
Implement MDM practices to maintain a single source of truth for critical data entities across the organization. This ensures that all systems use consistent, accurate data.

Regular Audits and Reviews:
Schedule regular audits of the data and its handling processes to identify areas where inaccuracies may arise. Include both automated and manual review processes.

Training

Implementing Clear Data Versioning

Version Control Systems:
Utilize version control systems for data models, schemas, and ETL scripts, similar to how software code is managed.

Data Immutability:
Where applicable, adopt an immutable approach to data storage, where updates and deletions create new records rather than overwrite existing ones. This allows for tracking changes over time.

Timestamping and Metadata:
Ensure each data record includes timestamp fields for creation and last modification. Use metadata to track the version of data schemas, definitions, and transformation logic applied.

Snapshotting and Archiving:
Take regular snapshots of critical datasets for archival purposes. This creates historical versions that can be referenced or restored if needed.

Access Controls and Audit Trails