**Design Decisions**

1. **Logging**:
   * You use Python’s logging module to log various levels of messages (info, error) which helps in tracking the progress and diagnosing issues.
2. **Schema Definition**:
   * You define schemas for nested JSON structures which ensures that Spark can correctly interpret the incoming data format.
3. **Data Processing**:
   * You fetch data from multiple URLs, aggregate it, and handle it as a JSON string. This approach allows you to work with data from multiple sources in a unified way.
4. **Data Transformation**:
   * **Exploding Arrays**: You explode arrays to normalize nested structures.
   * **Column Extraction and Renaming**: You select and rename columns to make the DataFrame schema more intuitive.
   * **Postal Code Mapping**: You create a UDF to map postal codes to provinces, which adds useful geographic context.
   * **One-Hot Encoding**: You handle categorical data in the handoverServices array by creating one-hot encoded columns.
5. **Anonymization**:
   * Sensitive data is anonymized using hashing, which is a good practice for protecting personal information.
6. **Data Partitioning**:
   * Data is saved as Parquet files and partitioned by province, optimizing for performance when querying by province.

**Introduction**

* **Purpose**: The assignment involves extracting, processing, and storing data from multiple API endpoints related to various brands. The goal is to transform and manage this data efficiently using Apache Spark, ensuring that it is both accessible and anonymized where necessary.
* **APIs Used**:
  + **API Endpoints**:
    - https://ecgplacesmw.colruytgroup.com/ecgplacesmw/v3/nl/places/filter/clp-places
    - https://ecgplacesmw.colruytgroup.com/ecgplacesmw/v3/nl/places/filter/okay-places
    - https://ecgplacesmw.colruytgroup.com/ecgplacesmw/v3/nl/places/filter/spar-places
    - https://ecgplacesmw.colruytgroup.com/ecgplacesmw/v3/nl/places/filter/dats-places
    - https://ecgplacesmw.colruytgroup.com/ecgplacesmw/v3/nl/places/filter/cogo-colpnts
* **Expected Outcome**: The expected outcome is to produce two Parquet files (unauthorized\_data.parquet and authorized\_data.parquet) containing processed data with sensitive information anonymized and data partitioned by province for optimized querying.

**2. Setup and Configuration**

* **Initializing Spark Session**:
  + You start by initializing a Spark session with spark = SparkSession.builder.appName("API Data Extraction").getOrCreate(). This sets up the environment needed for Spark operations.
* **Setting Up Logging**:
  + Logging is configured to capture and record information:
    - **Logger Creation**: logger = logging.getLogger('API\_Extraction\_Logger')
    - **Log Level**: Set to INFO with logger.setLevel(logging.INFO).
    - **File Handler**: Logs are written to assignment.log with file\_handler = FileHandler('assignment.log').
    - **Formatter**: Defines log format with timestamps and message details using formatter = Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s').
    - **Adding Handler**: The file handler is added to the logger.

**3. Data Fetching**

* **API Endpoints**:
  + A list of API URLs is provided, each of which returns data related to specific brands.
* **Data Fetching Process**:
  + **Function**: fetch\_json\_data(url) handles the fetching of data from each URL.
  + **Error Handling**: Uses try-except blocks to log errors if fetching fails or exceptions occur.
  + **Response Handling**: Checks HTTP status codes and logs success or failure. Data is assumed to be in JSON format and is processed accordingly.

**4. Schema Definitions**

* **Schemas**:
  + **Address Schema**: Defines fields for address components.
  + **Ensign Schema**: Defines fields for ensign information.
  + **Geo Coordinates Schema**: Contains latitude and longitude fields.
  + **Opening Hours Schema**: Details for opening and closing times.
  + **Place Type Schema**: Information about place types.
  + **Main Schema**: The main\_schema combines these nested schemas to represent the complete structure of the JSON data, ensuring Spark can parse and process the data correctly.

**5. Data Processing**

* **Transformations**:
  + **Exploding Arrays**: Uses explode(col("placeSearchOpeningHours")) to normalize the placeSearchOpeningHours array into individual rows.
  + **Column Extraction and Renaming**: Selects and renames columns to flatten the structure and make it more manageable.
  + **Adding Calculated Fields**: Applies UDFs to derive additional fields, such as mapping postal codes to provinces.

**6. Anonymization**

* **Process**:
  + **Hashing Sensitive Information**: Uses SHA-256 hashing (hash\_sensitive\_info function) to anonymize sensitive fields such as streetName and houseNumber.
  + **Importance**: Anonymization protects sensitive data, ensuring compliance with privacy regulations and safeguarding personal information from unauthorized access.

**7. Data Saving**

* **Saving Data**:
  + **Format**: Data is saved in Parquet format, which is efficient for both storage and querying.
  + **Partitioning**: Data is partitioned by province to optimize performance and storage. This is done with .partitionBy("province").
  + **Modes**: Uses mode("overwrite") to replace existing files.

**8. Conclusion**

* **Results**: The script processes data from multiple APIs, handles transformations, anonymizes sensitive information, and saves the results into partitioned Parquet files.
* **Challenges**:
  + Handling various JSON structures and potential inconsistencies in the data.
  + Ensuring that anonymization processes are robust and protect sensitive information.
* **Potential Improvements**:
  + **Error Handling**: Enhance error handling and logging to capture more detailed information.
  + **Testing**: Implement thorough testing with different data samples and edge cases.

**"Comprehensive Analysis of Geographic Distances Using Apache Spark: API Data Extraction, Haversine Distance Calculation, and Metrics Enrichment"**

 **Introduction**

* **Purpose**: To extend the analysis by calculating and adding distance-related metrics.
* **Expected Outcome**: To enrich the dataset with average distances to customers and competitors.

 **Define and Use Haversine Function**

* **Purpose**: To compute distances between locations based on latitude and longitude.
* **Implementation**: Detailed explanation of the Haversine formula and its integration into Spark using UDF.

 **DataFrame Selection and Aliasing**

* **Purpose**: To prepare and organize data for distance calculations.
* **Implementation**: Explanation of column selection and DataFrame aliasing for cross join operations.

 **Distance Calculation**

* **Purpose**: To compute distances between all place pairs and handle self-joins.
* **Implementation**: Details on cross join, distance calculation, and filtering of self-joins.

 **Caching for Optimization**

* **Purpose**: To improve performance by caching intermediate results.
* **Implementation**: Use of .cache() to store the distance DataFrame.

 **Calculate Average Distances**

* **Purpose**: To derive meaningful metrics on average distances.
* **Implementation**: Steps to calculate and round average distances to customers and competitors.

 **Enrich Original DataFrame**

* **Purpose**: To merge calculated metrics with the original dataset.
* **Implementation**: Joining average distance metrics back to the original DataFrame.

 **Data Cleaning**

* **Purpose**: To ensure the final DataFrame is clean and usable.
* **Implementation**: Filling missing values and removing duplicates.

 **Conclusion**

* **Summary**: The enhanced DataFrame now includes calculated average distances, providing additional insights.
* **Challenges**: Handling large-scale distance calculations and optimizing performance.
* **Potential Improvements**: Further optimizations, additional metrics, or visualizations.