Part 1: Database and Jupyter Notebook Set Up

Explanation:

1. **Import Libraries:**
    - From pymongo import MongoClient: Imports the MongoClient class from the PyMongo library, which is used to connect to MongoDB.
    - From pprint import pprint: Imports the pprint function from the pprint library for nicely formatted printing of Python data structures, particularly useful for MongoDB documents.
2. **Create MongoClient Instance:**
    - mongo = MongoClient(port=27017): Creates a MongoClient instance and connects to the MongoDB server running on the default port 27017.
3. **Create Database:**
    - **Define Database:** The most critical change is using mongo['uk_food'] in cell [3] to create/access the correct database immediately.
    - **Define Collection:** By moving the creation of establishments_collection to after db is correctly assigned, we ensure the collection resides in the correct uk_food database.
    - **Encoding Issues:** When you open a file for reading without specifying an encoding, Python tries to use a default encoding based on your system settings. Sometimes that default encoding is cp1252 or something similar, and when it encounters characters not part of that encoding (e.g., some UTF-8 characters), it raises a UnicodeDecodeError.
    - **UTF-8:** UTF-8 is a versatile character encoding that can represent a very broad range of characters from various languages. It's the standard for text on the web and commonly used with JSON.
    - **Explicit Encoding:** By explicitly stating encoding='utf-8', you're forcing Python to use the UTF-8 encoding when reading the JSON file, thus preventing the UnicodeDecodeError.
4. **Confirm Database Creation:**
    - print(mongo.list_database_names()): Lists all the databases in the MongoDB server, helping to confirm that uk_food is created successfully.
5. **Assign the Database:**
    - db = mongo['uk_food']: Assigns the uk_food database to a variable db for easy access.
6. **Review Collections:**
    - print(db.list_collection_names()): Lists all the collections within the uk_food database, confirming that the establishments collection is there.

7. **Review a Document:**
    - pprint(db.establishments.find_one()): Fetches the first document in the establishments collection using find_one() and displays it using pprint() to have a nicely formatted output.
8. **Assign the Collection to a Variable:**
    - establishments = db['establishments']: Assigns the establishments collection to a variable establishments for easier use in subsequent operations.
9. **Markdown Cell Placeholder**
    - The terminal text mongoimport --type json -d uk_food -c establishments --drop --jsonArray establishments.json that imports the json file into mongo should be added in place of YOUR IMPORT TEXT HERE into the markdown cell.

Part 2: Update the Database

Explanation:

1. **Insert New Restaurant:**
    - Creates a Python dictionary new_restaurant with the restaurant details.
    - Uses establishments.insert_one(new_restaurant) to add the document to the collection.
2. **Verify that the New Restaurant was Inserted Correctly:**
    - **Capture Insert Result:**
        - The establishments.insert_one(new_restaurant) line is now assigned to the variable insert_result.
        - The insert_result object provides details about the insertion operation, including if it was acknowledged and the ID of the inserted document.
    - **Check Insertion Acknowledgment:**
        - The code uses if insert_result.acknowledged to confirm the insert was acknowledged. If an insert operation has been acknowledged then the database has received the request, and the operation was successful, or there has been an error.
        - If insert_result.acknowledged is True proceed with retrieval. Otherwise, print an error message.

- **Retrieve the Inserted Document:**
    - The code retrieves the inserted document by its _id using establishments.find_one({"_id": inserted_id}). The inserted_id comes from the insert_result.inserted_id variable that was returned from establishments.insert_one().
- If the document was successfully inserted then it is printed using pprint().
- **Error Handling:**
    - If the inserted document cannot be retrieved using find_one() then an error message is printed.

3. **Find BusinessTypeID:**
    - Sets a query to search for the document with BusinessType: "Restaurant/Cafe/Canteen".
    - Sets a projection to only return the fields BusinessTypeID and BusinessType, and exclude the _id field.
    - Uses establishments.find_one() to retrieve a single document matching the query.
    - Extracts the BusinessTypeID value from the returned document into business_type_id.
4. **Update New Restaurant with BusinessTypeID:**
    - Uses establishments.update_one() to update the document matching "BusinessName": "Penang Flavours" to set its BusinessTypeID to the value found in the previous step.
    - Prints the updated record using pprint for review.
5. **Remove Dover Establishments:**
    - Uses establishments.count_documents({"LocalAuthorityName": "Dover"}) to count how many establishments are in the Dover authority.
    - Uses establishments.delete_many({"LocalAuthorityName": "Dover"}) to delete all documents from the collection with a Local Authority name of "Dover".
    - Uses establishments.find_one({"LocalAuthorityName": "Dover"}) to verify that there are no documents with a Local Authority Name of "Dover" remaining.
    - Uses establishments.count_documents({"LocalAuthorityName": "Dover"}) to confirm the expected number of documents have been deleted.
6. **Convert Data Types:**
    - **Set RatingValue to None if non-numeric**: The first code block uses a $in operator to convert all of the rating values which are not valid numbers to None, so that they will not cause a conversion error.

- **Conditional Type Conversions:** The revised update_many uses a $cond operator, which allows for conditional logic:
    - For each field (geocode.longitude, geocode.latitude, RatingValue):
      * It first checks using the $ne (not equal) operator, if the value is None.
      * If it's *not* None, it converts the value to the appropriate type.
      * If it *is* None then the value will remain None, and no conversion will be attempted.
    - Uses establishments.update_many() with an aggregation pipeline to:
        - Convert geocode.longitude and geocode.latitude from strings to double.
        - Convert RatingValue from strings to integers.
    - Prints a sample document with pprint() to verify the data types have been converted correctly.

Part 3: Exploratory Analysis

Explanation:

1. **Setup:**
    - Imports pymongo for database interaction, pprint for formatted printing, and pandas for data analysis using DataFrames.
    - Creates a MongoClient to connect to the local MongoDB instance.
    - Assigns the uk_food database to the db variable.
    - Confirms that the establishments collection is in the database, and assigns it to the variable establishments.
2. **Query 1: Establishments with Hygiene Score of 20:**
    - **Query:** {"scores.Hygiene": 20}: Selects documents where the Hygiene score within the nested scores object is exactly 20.
    - **Results:** The code finds that **41** establishments have a hygiene score of 20, displays the first of these documents using pprint, and then presents the data in a Pandas DataFrame, which has **41** rows.

3. **Query 2: Establishments in London with RatingValue >= 4:**
    - **Query:**
        - {"LocalAuthorityName": {"$regex": "London"}}: Uses a regular expression to select documents where the LocalAuthorityName field contains the word "London" (this allows to return the City of London documents).
        - "RatingValue": {"$gte": 4}: Selects documents where RatingValue is greater than or equal to 4.
    - **Results:** The code finds that **33** establishments in London (including the City of London) have a RatingValue of 4 or greater. The first matching document is printed using pprint, and the code returns a Pandas DataFrame with **33** rows.
4. **Query 3: Top 5 Establishments Near "Penang Flavours" with RatingValue = 5 (Sorted by Hygiene):**
    - **Query:**
        - "RatingValue": 5: Selects restaurants with a RatingValue equal to 5.
        - The geocode.latitude and geocode.longitude are used with $gte and $lte to define a bounding box of 0.01 degrees around the latitude and longitude of "Penang Flavours".
    - **Sorting & Limiting:**
        - sort = [("scores.Hygiene", 1)] sorts the results by scores.Hygiene in ascending order (lowest first).
        - limit = 5 limits the output to the top 5 results.
    - **Results:** The code outputs the top 5 establishments nearest to "Penang Flavours" with a RatingValue of 5, sorted by their hygiene scores. The code then displays the results in a Pandas DataFrame with **5** rows.
5. **Query 4: Establishments with Hygiene Score of 0 per Local Authority:**
    - **Aggregation Pipeline:**
        - {"$match": {"scores.Hygiene": 0}}: Filters documents with a hygiene score of 0.
        - {"$group": {"_id": "$LocalAuthorityName", "count": {"$sum": 1}}}: Groups the results by LocalAuthorityName and counts the number of establishments in each group (using $sum on a field set to 1 to count individual records).
        - {"$sort": {"count": -1}}: Sorts the results by count in descending order (highest count first).
        - {"$limit": 10}: Limits the results to 10

- **Results:** The code prints out that there are **55** local authorities with establishments that have a hygiene score of 0. The code then prints the top 10 local authorities with a hygiene score of 0, and a Pandas DataFrame with **10** rows. The pprint method for this section is modified to use an enumerate() method so that the output is limited to the first ten records.

General Notes:

- Each analysis step clearly prints the number of results using count_documents or len().
- The first document from each query is printed using pprint.
- The data is converted to a Pandas DataFrame and the number of rows of the DataFrame are printed using len().
- The first ten rows of each DataFrame are displayed using head().