**Summary Notes - Day 7**

**Name : Podutur Lahari - DE126**

**Date : 13-11-2024**

**Python Collections for Data Engineering**

### **1. Get Unique Values from List Using Map and Set :**

* Objective: Obtain unique items from a list.
* Approach: Use set() to remove duplicates from the list, and map() to apply a function to each element if needed.
* Example: unique\_list = list(set(original\_list)) removes duplicates in original\_list.

### **2. Sort Python Lists Using Key :**

* Objective: Sort lists based on a custom sorting rule.
* Approach: Use the sort() method with the key parameter, or sorted() to return a sorted list.
* Example: list.sort(key=lambda x: x.property) sorts based on a specified attribute or transformation.

### **3. Overview of JSON Strings and Files :**

* Objective: Work with JSON data (JavaScript Object Notation) in Python.
* JSON Basics: JSON is a lightweight format for storing and transporting data.
* Structure: JSON data typically consists of key-value pairs (like a Python dictionary) and can store arrays or nested objects.

### **4. Read JSON Strings to Python Dicts or Lists :**

* Objective: Convert JSON strings into Python dictionaries or lists.
* Method: Use json.loads() to parse JSON strings and return Python objects.
* Example: python\_dict = json.loads(json\_string).

### **5. Read JSON Schemas from File to Python Dicts :**

* Objective: Load JSON schemas (structure definitions for JSON data) from files.
* Method: Use json.load() to read JSON from files and convert them to Python dicts.
* Example: with open('schema.json') as f: schema\_dict = json.load(f).

### **6. Processing JSON Data Using Python :**

* Objective: Manipulate JSON data using Python's built-in features.
* Typical Steps: Loading, accessing elements, updating values, and saving modified data.
* Tools: json library methods like json.dumps() (to JSON string) and json.dump() (to file).

### **7. Extract Details from Complex JSON Arrays Using Python :**

* Objective: Navigate nested or complex JSON structures to retrieve specific values.
* Approach: Use loops or recursive functions to traverse nested lists and dictionaries.
* Example: Access nested elements using data['key1']['key2'].

### **8. Sort Data in JSON Arrays Using Python :**

* Objective: Sort elements within a JSON array based on specific fields.
* Approach: Use sorted() with a key function tailored to the JSON object’s field.
* Example: sorted(data, key=lambda x: x['field']).

### **9. Create Function to Get Column Details from Schemas JSON File :**

* Objective: Extract schema column details (like column names, types) from a JSON file.
* Function Structure: Design a function that reads the schema JSON file, extracts relevant column data, and returns it in a dictionary or list format.

### **10. Lists and Tuples in Python :**

* Lists: Mutable, ordered sequences that allow item reassignment.
* Tuples: Immutable, ordered sequences that are fixed after creation.
* Common Operations: Slicing, indexing, appending (only for lists), and tuple unpacking.

### **11. Enriching Data Using Numpy and Pandas :**

* Numpy: Used for numerical operations on large arrays/matrices, faster than Python lists.
* Pandas: Provides data structures like DataFrames for data manipulation, handling missing data, and performing complex operations.
* Data Enrichment: Use Pandas for aggregating, cleaning, joining, and analyzing datasets, and Numpy for numerical computations.

**JSON FILES**

**1. Pandas for Data Processing :**

* Objective: Use Pandas to manipulate and analyze data effectively.
* Core Features: Provides structures like Series and DataFrames for handling data, enabling data cleaning, filtering, aggregations, and reshaping operations.

**2. Reading CSV Data Using Pandas :**

* Objective: Load CSV data into Pandas for processing.
* Method: Use pd.read\_csv('file.csv') to read CSV files into a DataFrame, with options to specify delimiters, column headers, data types, etc.

**3. Read Data from CSV Files to Pandas DataFrames :**

* Objective: Store CSV data in a DataFrame for further manipulation.
* Typical Steps: Load data with pd.read\_csv(), then examine it using methods like .head(), .info(), and .describe().

**4. Filter Data in Pandas DataFrame Using Query :**

* Objective: Filter data based on specific conditions.
* Method: Use .query('condition') for quick and readable filtering, or .loc[] for more complex filters.
* Example: df.query('status == "Active" & age > 25').

**5. Get Count by Status Using Pandas DataFrame APIs :**

* Objective: Count occurrences based on a column's values.
* Method: Use .value\_counts() on a specific column, or .groupby('status').size() to group and count by status.

**6. Get Count by Month and Status Using Pandas DataFrame APIs :**

* Objective: Get counts grouped by month and status.
* Method: Use .groupby(['month', 'status']).size() or .groupby(['month', 'status']).count() to get counts based on multiple columns.

**7. Create DataFrames Using Dynamic Column List on CSV Data :**

* Objective: Load a subset of columns from CSV into a DataFrame.
* Method: Specify columns using the usecols parameter in pd.read\_csv().
* Example: pd.read\_csv('file.csv', usecols=['name', 'status', 'date']).

**8. Performing Inner Join Between Pandas DataFrames :**

* Objective: Merge two DataFrames based on common keys.
* Method: Use pd.merge(df1, df2, on='key') for an inner join on a common column.
* Example: pd.merge(df1, df2, on='id') to join by id.

**9. Perform Aggregations on Join Results :**

* Objective: Apply aggregations like sum, mean, or count after a join.
* Method: After merging, use .groupby() and .agg() on the result DataFrame to perform aggregations on specific columns.

**10. Sort Data in Pandas DataFrames :**

* Objective: Sort DataFrame rows by one or more columns.
* Method: Use .sort\_values(by='column\_name') to sort by column in ascending or descending order.
* Example: df.sort\_values(by=['date'], ascending=False).

**11. Writing Pandas DataFrames to Files :**

* Objective: Export DataFrames to various file formats.
* Method: Use .to\_csv('output.csv') for CSV files, or .to\_excel('output.xlsx') for Excel files.

**12. Write Pandas DataFrames to JSON Files :**

* Objective: Save DataFrames as JSON for interoperability.
* Method: Use .to\_json('file.json') to export data in JSON format.
* Example: df.to\_json('output.json', orient='records')