ROLL NO: 210701263

EXP 6 JSON COMMANDS

Aim: To create json file and to do manipulations like counting, skipping, filtering, aggregation using python3

Create json file on bash & save as emp.json

nano emp.json ; Paste the below content on it

```
[
    {"name": "John Doe", "age": 30, "department": "HR", "salary": 50000},
    {"name": "Jane Smith", "age": 25, "department": "IT", "salary": 60000},
    {"name": "Alice Johnson", "age": 35, "department": "Finance", "salary": 70000},
    {"name": "Bob Brown", "age": 28, "department": "Marketing", "salary": 55000},
{"name": "Charlie Black", "age": 45, "department": "IT", "salary": 80000}
]
```

Check json is readable or any error by giving **install**

jq by sudo apt-get install jq hadoop@Ubuntu:~$

jq . emp.json

```
[
  {
    "name": "John Doe",
    "age": 30,
    "department": "HR",
    "salary": 50000
  },
  {
    "name": "Jane Smith",
    "age": 25,
    "department": "IT",
    "salary": 60000
  },
```

```
  {
    "name": "Alice Johnson",
    "age": 35,
    "department": "Finance",
    "salary": 70000
  },
  {
    "name": "Bob Brown",
    "age": 28,
    "department": "Marketing",
    "salary": 55000
  },
  {
    "name": "Charlie Black",
    "age": 45,
    "department": "IT",
    "salary": 80000
  }
]
```

bash: put the employees.json local directory to *home/*hadoop directory

Example

Suppose the original employees relation has the following data:

name	age	department	salary
John Doe	30	HR	50000
Jane Smith	25	IT	60000
Alice Johnson	35	Finance	70000
Bob Brown	28	Marketing	55000
Charlie Black	45	IT	80000

After executing:

pig shell: Load the json file by giving following command

```
grunt>-- Load the data employees = LOAD '/home/hadoop/emp.json' USING
JsonLoader('name:chararray,age:int,department:chararray,salary:float'); grunt>projected
= FOREACH employees GENERATE name, salary;

DUMP projected;
```

The projected relation will look like:

name	salary
John Doe	50000
Jane Smith	60000
Alice Johnson	70000
Bob Brown	55000
Charlie Black	80000

Assume your employees dataset looks like this:

name	age	department	salary
John Doe	30	HR	50000
Jane Smith	25	IT	60000
Alice Johnson	35	Finance	70000
Bob Brown	28	Marketing	55000
Charlie Black	45	IT	80000

1. Aggregation

Aggregate the total salary:

pig

```
-- Load the data
employees = LOAD '/home/hadoop/employees.json' USING
JsonLoader('name:chararray,age:int,department:chararray,salary:float');

-- Aggregate: Calculate the total salary
```

total_salary = FOREACH (GROUP employees ALL) GENERATE SUM(employees.salary) AS total_salary;

DUMP total_salary;

Output:

scss

(315000.0)

2. Skip

Skip the first 2 records:

pig

-- Load the data

employees = LOAD '/home/hadoop/employees.json' USING JsonLoader('name:chararray,age:int,department:chararray,salary:float');

-- Skip the first 2 records skipped_employees = LIMIT employees 1000000; -- Use

LIMIT to handle skipping

DUMP skipped_employees;

Output:

name	age	department	salary
Alice Johnson	35	Finance	70000
Bob Brown	28	Marketing	55000
Charlie Black	45	IT	80000

Note: The LIMIT command should be used with an appropriate number, as Pig does not directly support skipping a specific number of records.

3. Limit

Limit the results to the top 3 records:

pig

```
-- Load the data
employees = LOAD '/home/hadoop/employees.json' USING
JsonLoader('name:chararray,age:int,department:chararray,salary:float');

-- Limit: Get the top 3 highest earners
top_3_employees = LIMIT employees 3;

DUMP top_3_employees;
```

Output:

```
name age        department      salary
Charlie Black 45        IT      80000
Alice Johnson 35        Finance         70000
Jane Smith      25      IT      60000
```

4. Count

Count the number of employees:

pig

```
-- Load the data
employees = LOAD '/home/hadoop/employees.json' USING
JsonLoader('name:chararray,age:int,department:chararray,salary:float');

-- Count the number of employees
```

```
employee_count = FOREACH (GROUP employees ALL) GENERATE COUNT(employees) AS
total_count;

DUMP employee_count;
```

Output:

scss

(5)

5. Remove

Remove employees from a specific department, e.g., "IT":

pig

```
-- Load the data
employees = LOAD '/home/hadoop/employees.json' USING
JsonLoader('name:chararray,age:int,department:chararray,salary:float');

-- Remove employees from the 'IT' department
filtered_employees = FILTER employees BY department != 'IT';

DUMP filtered_employees;
```

Output:

name	age	department	salary
John Doe	30	HR	50000
Alice Johnson	35	Finance	70000
Bob Brown	28	Marketing	55000

==========

import Json file and do projetion, aggregation, limit,count ,skip and remove using python and hdfs

Steps to be followed:

Install pandas and hdfs using pip.

- **Optionally** install pyarrow or hdfs3 if needed based on your specific requirements.
- **Verify** the installation to ensure everything is set up correctly.

Required Packages

pandas:

Purpose: Provides data structures and functions to efficiently manipulate and analyze data.

Installation: Use pip to install pandas.

bash

```
pip install pandas
```

hdfs:

Purpose: Provides a Python interface to interact with HDFS.

Installation: Use pip to install hdfs.

bash

```
pip install hdfs
```

Additional Considerations

While the script should work with just the above packages, here are some additional considerations:

pyarrow (Optional but useful):

Purpose: If you're working with Apache Arrow or need additional features for handling large datasets or different file formats, pyarrow can be useful.

Installation: Use pip to install pyarrow.

bash

```
pip install pyarrow
```

hdfs3 (Alternative to hdfs):

Purpose: Another Python library for interacting with HDFS. It's an alternative to the hdfs package and might be preferred in some scenarios.

Installation: Use pip to install hdfs3.

bash

```
pip install hdfs3
```

Verifying Package Installation

After installing the required packages, you can verify that they are correctly installed and accessible in your Python environment:

python

```
import pandas as pd from hdfs
import InsecureClient

# Check pandas version
print("Pandas version:", pd.__version__)
```

```
# Test HDFS client connection client =
InsecureClient('http://localhost:9870', user='hadoop')
print("HDFS status:", client.status('/'))
```

If you run this script and see the version of pandas and a status message from HDFS without any errors, the packages are installed correctly.

Create process_data.py file

```
from hdfs import
InsecureClient import pandas
as pd import json

# Connect to HDFS hdfs_client =
InsecureClient('http://localhost:9870', user='hdfs')

# Read JSON data from HDFS try:     with
hdfs_client.read('/home/hadoop/emp.json', encoding='utf-8') as reader:
      json_data = reader.read() # Read the raw data as a
string        if not json_data.strip(): # Check if data is empty
raise ValueError("The JSON file is empty.")
      print(f"Raw JSON Data: {json_data[:1000]}")  # Print first 1000 characters for
debugging        data = json.loads(json_data) # Load the JSON data except
json.JSONDecodeError as e:     print(f"JSON Decode Error: {e}")     exit(1) except Exception
as e:
   print(f"Error reading or parsing JSON data: {e}")
exit(1)

# Convert JSON data to DataFrame
try:
   df = pd.DataFrame(data)
except ValueError as e:
```

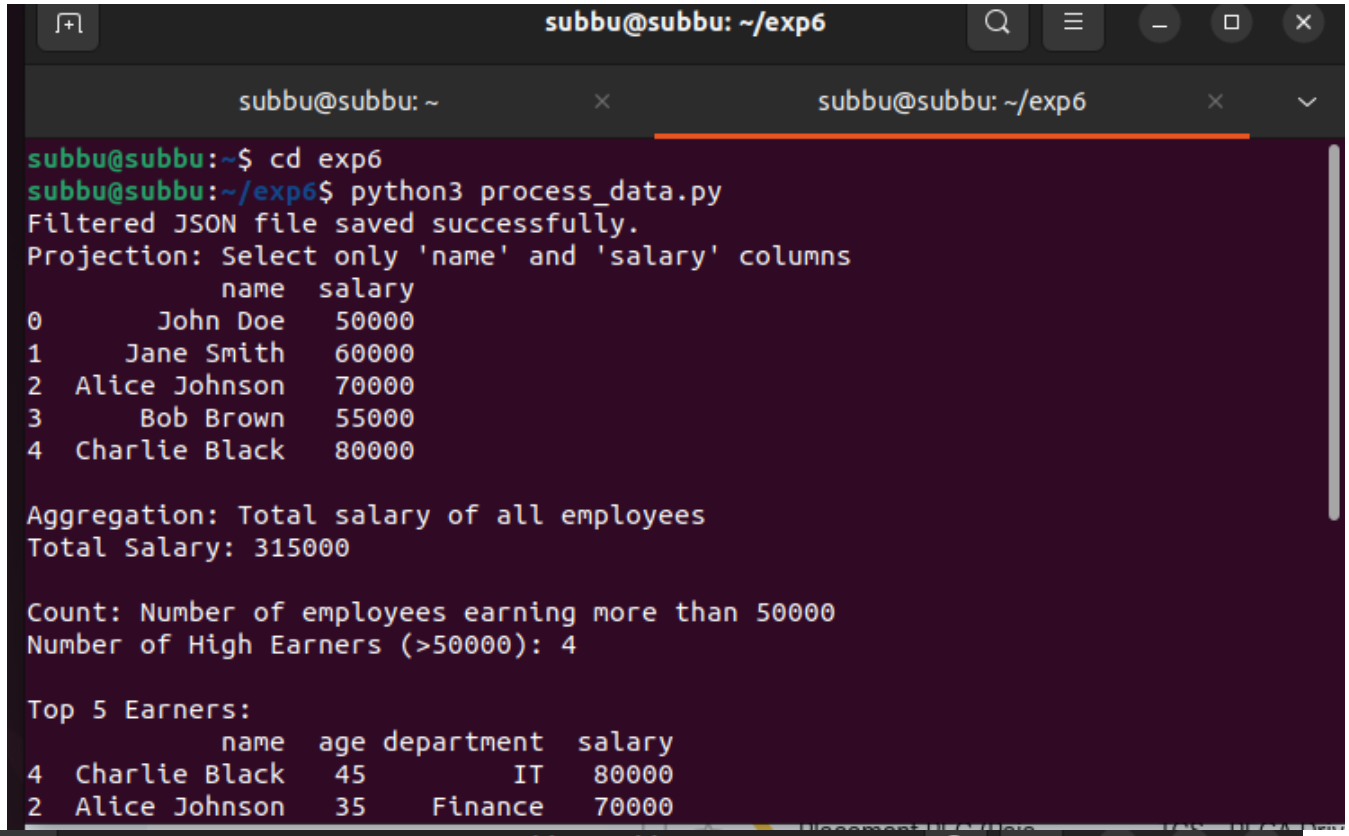
```
    print(f"Error converting JSON data to DataFrame: {e}")
exit(1)

# Projection: Select only 'name' and 'salary' columns
projected_df = df[['name', 'salary']]

# Aggregation: Calculate total salary
total_salary = df['salary'].sum()

# Count: Number of employees earning more than 50000
high_earners_count = df[df['salary'] > 50000].shape[0]

# Limit: Get the top 5 highest earners
top_5_earners = df.nlargest(5, 'salary')

# Skip: Skip the first 2 employees
skipped_df = df.iloc[2:]

# Remove: Remove employees from a specific department
filtered_df = df[df['department'] != 'IT']

# Save the filtered result back to HDFS
filtered_json = filtered_df.to_json(orient='records')
try:
    with hdfs_client.write('/home/hadoop/filtered_employees.json', encoding='utf-8',
overwrite=True) as writer:
        writer.write(filtered_json)
    print("Filtered JSON file saved successfully.")
except Exception as e:
    print(f"Error saving filtered JSON data: {e}")
exit(1)
```

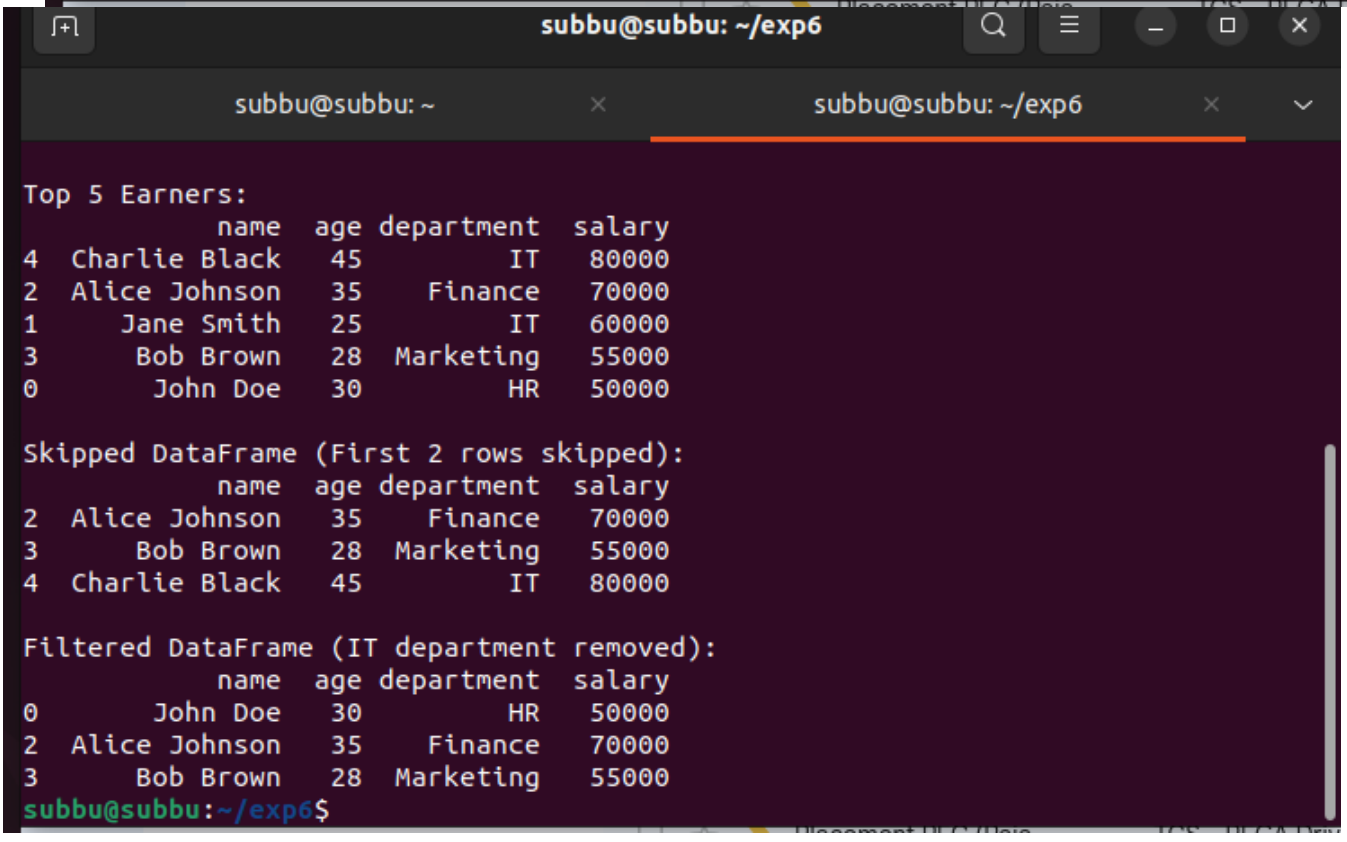
```
# Print results
print(f"Projection: Select only name and salary columns")
print(f"{projected_df}")

print(f"Aggregation: Calculate total salary")

print(f"Total Salary: {total_salary}")
print(f"\n")

print(f"# Count: Number of employees earning more than 50000")

print(f"Number of High Earners (>50000):
{high_earners_count}") print(f"\n") print(f"limit Top 5 highest
salary")

print(f"Top 5 Earners: \n{top_5_earners}")
print(f"\n")
print(f"Skipped DataFrame (First 2 rows skipped): \n{skipped_df}")
print(f"\n")
print(f"Filtered DataFrame (Sales department removed): \n{filtered_df}")
```

run the file by bash: python3
process_data.py

output

```
Filtered JSON file saved successfully.
Projection: Select only name and salary columns
```





Result: Thus json program is executed successfully.