

Exp.No: 6

Import a JSON file from the command line. Apply the following actions with the data present in the JSON file where, projection, aggregation, remove, count, limit, skip and sort

AIM:

To import a JSON file from the command line and apply the following actions with the data present in the JSON file where, projection, aggregation, remove, count, limit, skip and sort using jq tool.

PROCEDURE:

- Create a json file 'emp.json' and provide data in it.

```
[
  {
    "name" : "Anu",
    "age":12,
    "dept": "Computer",
    "salary":10000
  },
  {
    "name" : "Bob",
    "age" :14,
    "dept" : "HR",
    "salary":15000
  },
  {
    "name": "Jane Smith",
    "age": 25,
    "department": "IT",
    "salary": 60000
  },
  {
    "name": "Alice Johnson",
    "age": 35,
    "department": "Finance",
```

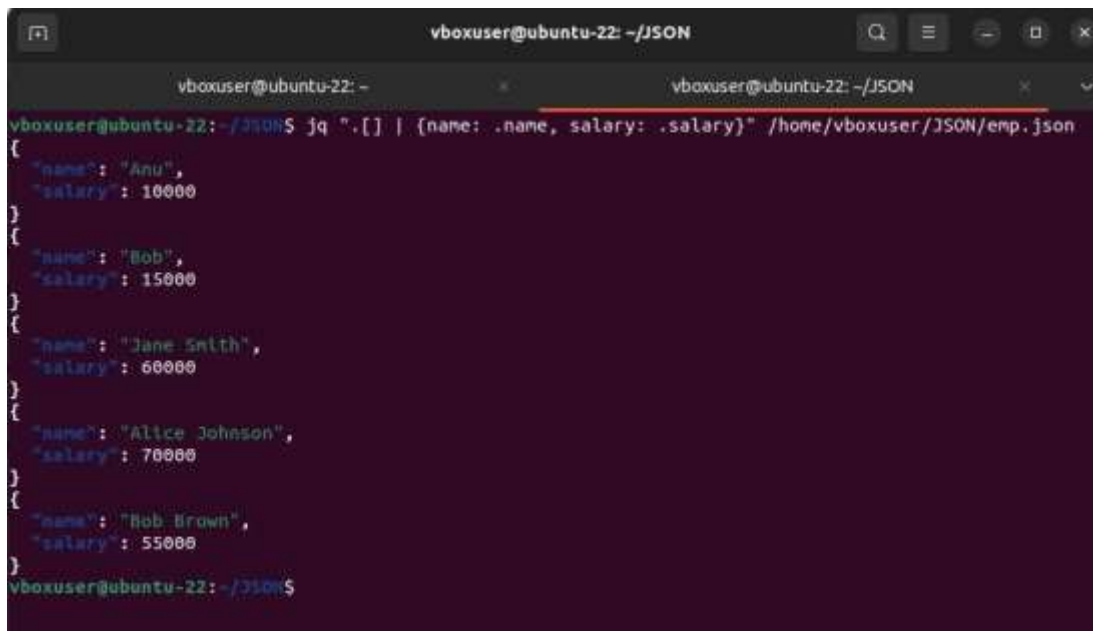
```
"salary": 70000
},
{
  "name": "Bob Brown",
  "age": 28,
  "department": "Marketing",
  "salary": 55000
}
]
```

- Open the command prompt.
- Navigate to the folder where emp.json is stored.
- Load and view the JSON data with jq.
- Use the jq commands for projection, aggregation, removal, counting, limiting, and sorting operations.

OUTPUT:

Running jq queries:

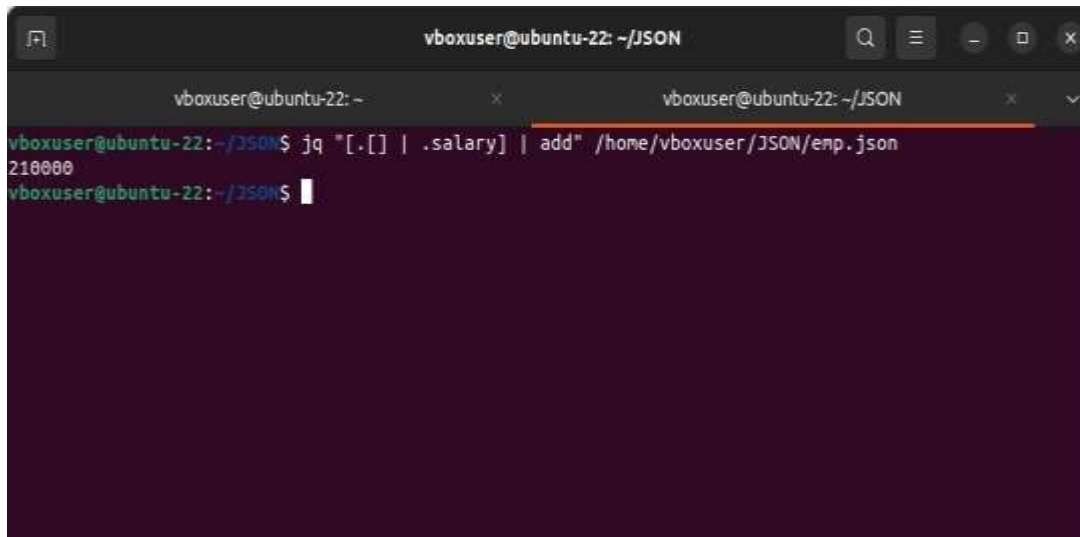
1. Projection:



A terminal window titled 'vboxuser@ubuntu-22: ~/JSON' shows a jq command being executed. The command is 'jq ".[]" | {name: .name, salary: .salary}" /home/vboxuser/JSON/emp.json'. The output is a JSON array of objects, each containing 'name' and 'salary' fields. The objects are: {'name': 'Anu', 'salary': 10000}, {'name': 'Bob', 'salary': 15000}, {'name': 'Jane Smith', 'salary': 60000}, {'name': 'Alice Johnson', 'salary': 70000}, and {'name': 'Bob Brown', 'salary': 55000}.

```
vboxuser@ubuntu-22: ~/JSON$ jq ".[]" | {name: .name, salary: .salary}" /home/vboxuser/JSON/emp.json
[
  {
    "name": "Anu",
    "salary": 10000
  },
  {
    "name": "Bob",
    "salary": 15000
  },
  {
    "name": "Jane Smith",
    "salary": 60000
  },
  {
    "name": "Alice Johnson",
    "salary": 70000
  },
  {
    "name": "Bob Brown",
    "salary": 55000
  }
]
vboxuser@ubuntu-22: ~/JSON$
```

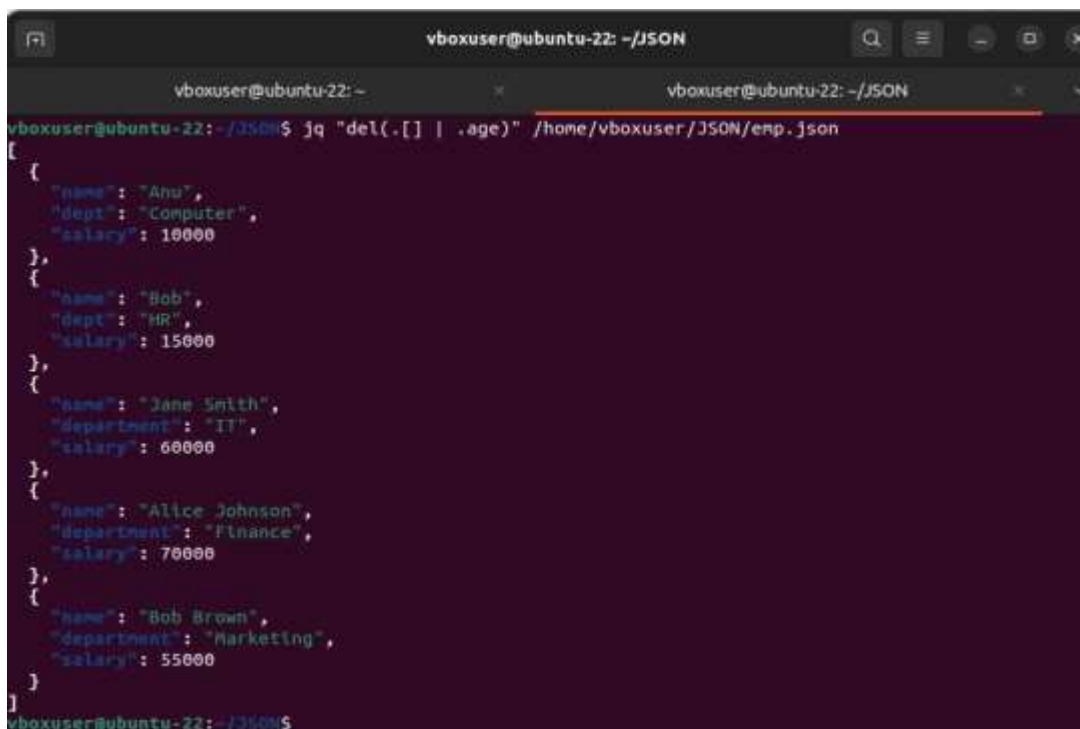
2. Aggregation:



A terminal window titled 'vboxuser@ubuntu-22: ~/JSON' showing a jq command being executed. The command is `jq "[.[] | .salary] | add" /home/vboxuser/JSON/emp.json`. The output of the command is `210000`.

```
vboxuser@ubuntu-22: ~/JSON
vboxuser@ubuntu-22: ~
vboxuser@ubuntu-22: ~/JSON$ jq "[.[] | .salary] | add" /home/vboxuser/JSON/emp.json
210000
vboxuser@ubuntu-22: ~/JSON$
```

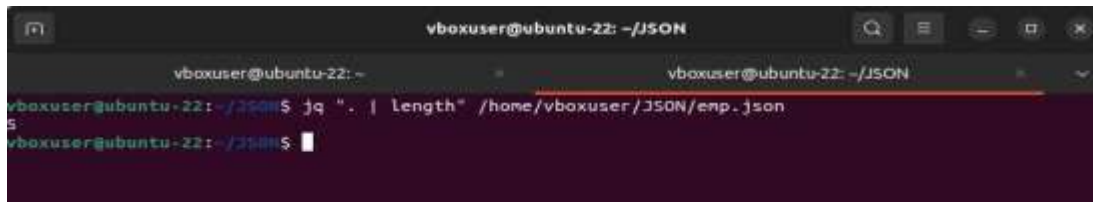
3. Remove:



A terminal window titled 'vboxuser@ubuntu-22: ~/JSON' showing a jq command being executed. The command is `jq "del(.[] | .age)" /home/vboxuser/JSON/emp.json`. The output is a JSON array of employee objects, each with 'name', 'dept', and 'salary' fields.

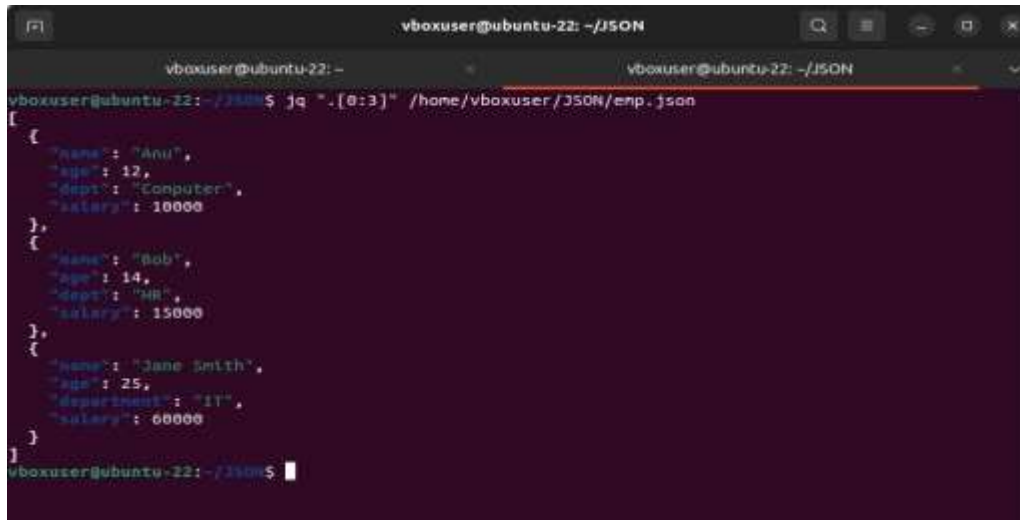
```
vboxuser@ubuntu-22: ~/JSON
vboxuser@ubuntu-22: ~
vboxuser@ubuntu-22: ~/JSON$ jq "del(.[] | .age)" /home/vboxuser/JSON/emp.json
[
  {
    "name": "Anu",
    "dept": "Computer",
    "salary": 10000
  },
  {
    "name": "Bob",
    "dept": "HR",
    "salary": 15000
  },
  {
    "name": "Jane Smith",
    "department": "IT",
    "salary": 60000
  },
  {
    "name": "Alice Johnson",
    "department": "Finance",
    "salary": 70000
  },
  {
    "name": "Bob Brown",
    "department": "Marketing",
    "salary": 55000
  }
]
vboxuser@ubuntu-22: ~/JSON$
```

4. Count:



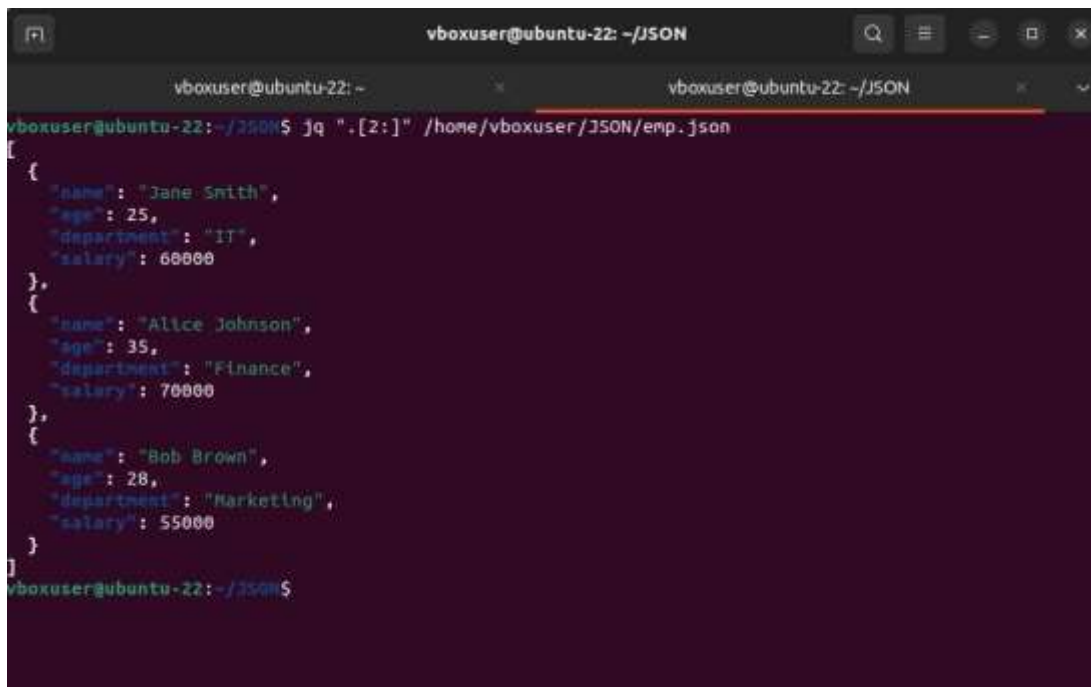
```
vboxuser@ubuntu-22: ~/JSON
vboxuser@ubuntu-22: ~
vboxuser@ubuntu-22:~/JSON$ jq ". | length" /home/vboxuser/JSON/emp.json
5
vboxuser@ubuntu-22:~/JSON$
```

5. Limit:



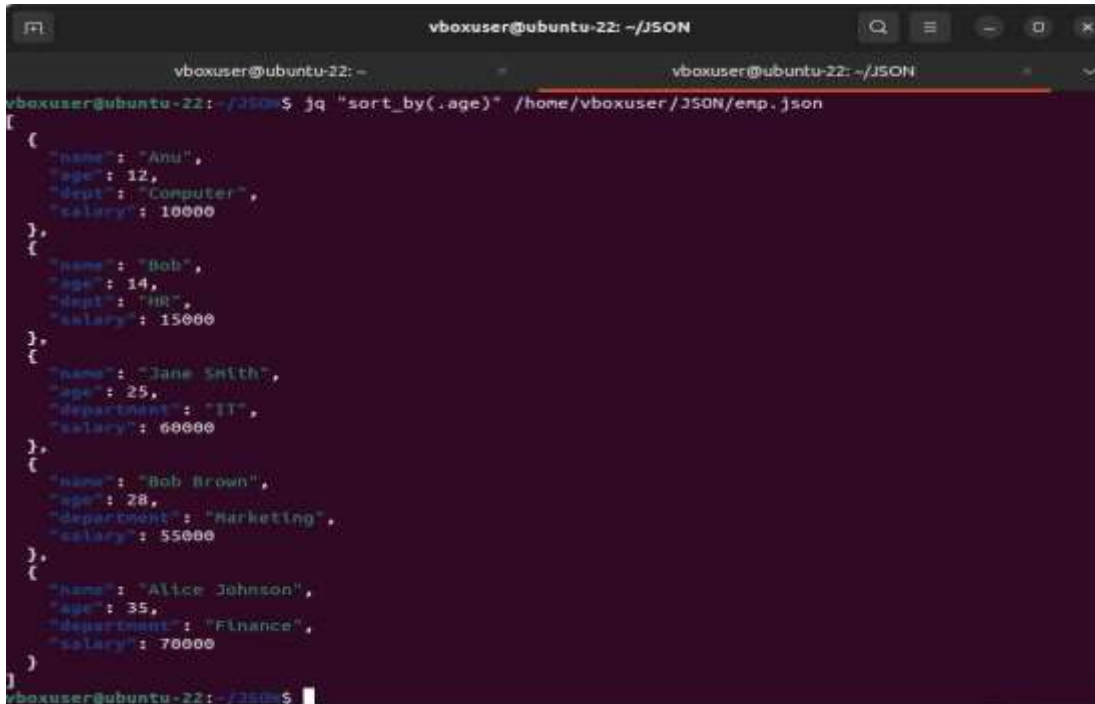
```
vboxuser@ubuntu-22: ~/JSON
vboxuser@ubuntu-22: ~
vboxuser@ubuntu-22:~/JSON$ jq ".[0:3]" /home/vboxuser/JSON/emp.json
[
  {
    "name": "Anu",
    "age": 12,
    "dept": "Computer",
    "salary": 10000
  },
  {
    "name": "Bob",
    "age": 14,
    "dept": "HR",
    "salary": 15000
  },
  {
    "name": "Jane Smith",
    "age": 25,
    "department": "IT",
    "salary": 60000
  }
]
vboxuser@ubuntu-22:~/JSON$
```

6. Skip:



```
vboxuser@ubuntu-22: ~/JSON
vboxuser@ubuntu-22: ~
vboxuser@ubuntu-22:~/JSON$ jq ".[2:]" /home/vboxuser/JSON/emp.json
[
  {
    "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
  }
]
vboxuser@ubuntu-22:~/JSON$
```

7. Sort:



```
vboxuser@ubuntu-22: ~/JSON
vboxuser@ubuntu-22: ~
vboxuser@ubuntu-22: ~/JSON
vboxuser@ubuntu-22: ~/JSON$ jq "sort_by(.age)" /home/vboxuser/JSON/emp.json
[
  {
    "name": "Anu",
    "age": 12,
    "dept": "Computer",
    "salary": 10000
  },
  {
    "name": "Bob",
    "age": 14,
    "dept": "HR",
    "salary": 15000
  },
  {
    "name": "Jane Smith",
    "age": 25,
    "department": "IT",
    "salary": 60000
  },
  {
    "name": "Bob Brown",
    "age": 28,
    "department": "Marketing",
    "salary": 55000
  },
  {
    "name": "Alice Johnson",
    "age": 35,
    "department": "Finance",
    "salary": 70000
  }
]
```

8. Analyzing json data with python

1. Load emp.json into hdfs root folder
2. Run the below python code with root privileges

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

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

# Read JSON data from HDFS
try:
    with hdfs_client.read('/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('/exp6/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")
```

```

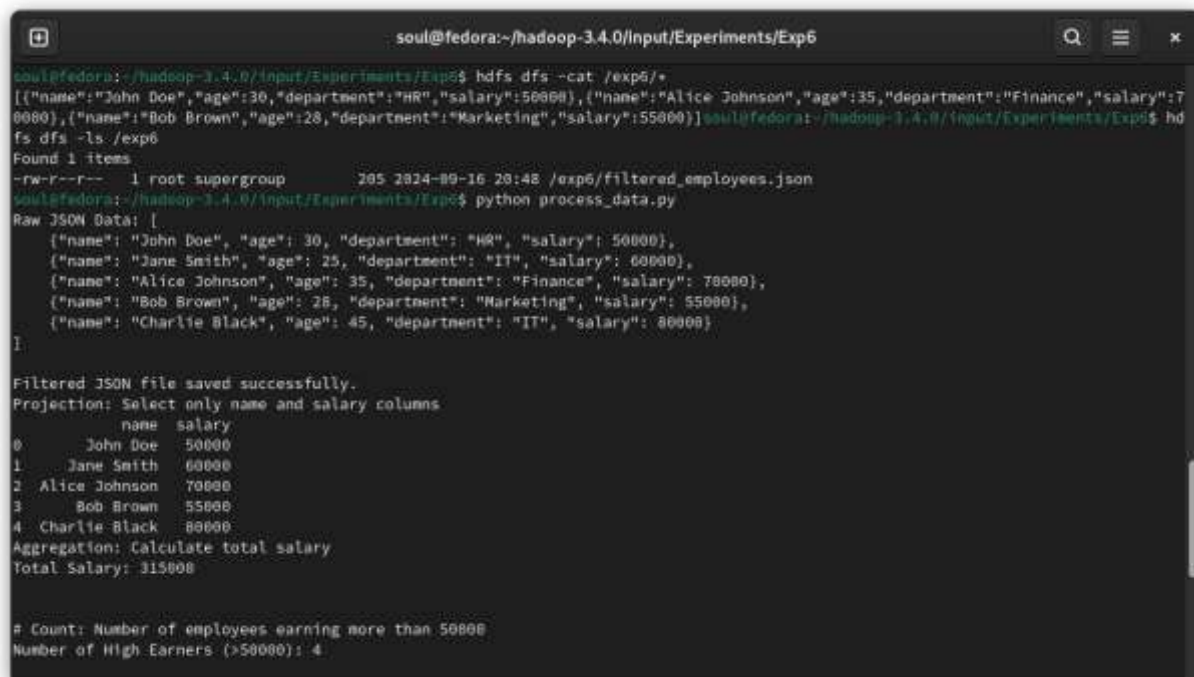
print(f"# Count: Number of employees earning more than 50000")

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}")

```

Execution - a



```

soul@fedora:~/hadoop-3.4.0/input/Experiments/Exp6
soul@fedora:~/hadoop-3.4.0/input/Experiments/Exp6$ hdfs dfs -cat /exp6/*
[{"name":"John Doe","age":30,"department":"HR","salary":50000}, {"name":"Alice Johnson","age":35,"department":"Finance","salary":70000}, {"name":"Bob Brown","age":28,"department":"Marketing","salary":55000}]
soul@fedora:~/hadoop-3.4.0/input/Experiments/Exp6$ hdfs dfs -ls /exp6
Found 1 items
-rw-r--r-- 1 root supergroup 205 2024-09-16 20:48 /exp6/filtered_employees.json
soul@fedora:~/hadoop-3.4.0/input/Experiments/Exp6$ python process_data.py
Raw JSON Data: [
  {"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}
]

Filtered JSON file saved successfully.
Projection: Select only name and salary columns
  name  salary
0   John Doe   50000
1   Jane Smith   60000
2   Alice Johnson   70000
3    Bob Brown   55000
4  Charlie Black   80000
Aggregation: Calculate total salary
Total Salary: 315000

# Count: Number of employees earning more than 50000
Number of High Earners (>50000): 4

```

Execution – b

```
soul@fedora:~/hadoop-3.4.0/Input/Experiments/Exp5
Top 5 Earners:
  name age department salary
4 Charlie Black 45 IT 80000
2 Alice Johnson 35 Finance 70000
1 Jane Smith 25 IT 60000
3 Bob Brown 28 Marketing 55000
0 John Doe 30 HR 50000

Skipped DataFrame (First 2 rows skipped):
  name age department salary
2 Alice Johnson 35 Finance 70000
3 Bob Brown 28 Marketing 55000
4 Charlie Black 45 IT 80000

Filtered DataFrame (Sales department removed):
  name age department salary
0 John Doe 30 HR 50000
2 Alice Johnson 35 Finance 70000
3 Bob Brown 28 Marketing 55000
soul@fedora:~/hadoop-3.4.0/Input/Experiments/Exp5$ cat 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}
]
soul@fedora:~/hadoop-3.4.0/Input/Experiments/Exp5$
```

RESULT:

Thus to import a JSON file from the command line and apply the following actions with the data present in the JSON file where, projection, aggregation, remove, count, limit, skip and sort using jq tool is completed successfully