Big Data Analytics Lab

Ex - 1 - Linux commands

1. **Display information about files in the current directory**

```
ls -l
```

2. **Display the current working directory**

```
pwd
```

3. **Create a new directory**

```
mkdir <directory_name>
```

4. **Navigate between different folders**

```
cd <path_to_directory>    # Change to another directory
cd ..                     # Move up one directory level
cd                        # Go to the home directory
```

5. **Remove empty directories from the directory lists**

```
rmdir <directory_name>
```

6. **Copy files**

   - **a. Copy files from one directory to the same directory**

     ```
     cp <filename> <new_filename>
     ```

   - **b. Copy files from one directory to another directory**

```
cp <filename> <path_to_target_directory>
```

7. **Rename and move files**
   - **a. Rename a filename to another name**

     ```
     mv <old_filename> <new_filename>
     ```

   - **b. Move a file from one directory to another**

     ```
     mv <filename> <path_to_target_directory>
     ```

8. **Delete files and directories**
   - **a. Delete individual files from a directory**

     ```
     rm <filename>
     ```

   - **b. Delete an entire directory which contains files**

     ```
     rm -r <directory_name>
     ```

9. **Get basic information about the OS**

   ```
   uname -a
   ```

10. **Find a file in the directory**

    ```
    find <directory_path> -name <filename>
    ```

11. **Create empty files**

    ```
    touch <filename>
    ```

12. **Display file contents on terminal**

```
cat <filename>           # Display full contents
head <filename>          # Display first 10 lines
tail <filename>          # Display last 10 lines
```

13. **Clear terminal**

```
clear
```

14. **Display the processes in terminal**

```
ps
```

15. **Access manual for all Linux commands**

```
man <command>
```

16. **Search for a specific string in an output**

```
grep <search_string> <filename>
```

17. **Display active processes on the terminal**

```
top
```

18. **Download files from the internet**

```
wget <URL>
```

19. **Create or update passwords for existing users**

```
passwd <username>
```

20. **View the exact location of any tool/software installed**

```
which <command_name>
```

21. **Check the details of the file system**

```
df -h
```

22. **Check lines, word count, and characters in a file using different options**

```
wc <filename>           # Show lines, words, characters
wc -l <filename>        # Show only lines
wc -w <filename>        # Show only words
wc -c <filename>        # Show only characters
```

Ex - 2 - Hadoop Installation

http://localhost:9870/

Ex - 3 - Implementing MapReduce1

Main User -

```
which sshd # /user/sbin/sshd
sudo //user/sbin/sshd # No directory - /run/sshd
sudo mkdir -p /run/sshd
ssh localhost
```

Hadoop User -

inputFile.txt

```
MapReduce processes large datasets
Hadoop is designed for big data
Data processing with MapReduce is efficient
Big data analytics is powered by Hadoop
```

mapper.py

```
#!/usr/bin/env python3
import sys

for line in sys.stdin:
    words = line.strip().split()
    for word in words:
        print(f"{word}\t1")
```

reducer.py

```
#!/usr/bin/env python3
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
    word, count = line.strip().split('\t')
    count = int(count)

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count

if current_word == word:
    print(f"{current_word}\t{current_count}")
```

shell commands

```
chmod +x mapper.py
chmod +x reducer.py

start-dfs.sh
start-yarn.sh

hdfs dfs -mkdir -p /BigDataLab/ex3/input # Create directory
hdfs dfs -put inputFile.txt /BigDataLab/ex3/input

ls /usr/local/hadoop/share/tools/lib/hadoop-streaming-3.3.6.jar

hadoop jar
/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.6
    -input /BigDataLab/ex3/input
    -output /BigDataLab/ex3/output
    -mapper /home/hd_user/LabEx/ex3/mapper.py
    -reducer /home/hd_user/LabEx/ex3/reducer.py

hdfs dfs -rm -r /BigDataLab/ex3/output
# to remove output in case of errors

hdfs dfs -cat /BigDataLab/ex3/output/part-00000

stop-dfs.sh
stop-yarn.sh
```

```
hd_user@DESKTOP-D5H3PKE:~/LabEx/ex3$ hdfs dfs -cat /BigDataLab/ex3/output/part-00000
Big     1
Data    1
Hadoop  3
MapReduce       3
a       1
analytics       1
big     1
by      1
data    2
datasets        1
designed        1
efficient       1
for     1
framework       1
is      4
large   1
powered 1
processes       1
processing      1
with    1
```

Ex - 4(i) - Implementing MapReduce2

mapper.py

```
#!/usr/bin/env python3
import sys

for line in sys.stdin:
        parts = line.strip().split()
        date = parts[1]
        month = date[4:6]
        min_temp = parts[5]
        max_temp = parts[6]
        print(f'{month}\t{min_temp}\t{max_temp}')
```

reducer.py

```
#!/usr/bin/env python3
import sys
```

```
curr_month = None
min_temp, max_temp = float('inf'), float('-inf')

for line in sys.stdin:
        month, min_temp_str, max_temp_str =
            line.strip().split('\t')
        min_temp_val, max_temp_val =
            float(min_temp_str), float(max_temp_str)

        if curr_month == month:
                min_temp = min(min_temp, min_temp_val)
                max_temp = max(max_temp, max_temp_val)
        else:
                if curr_month is not None:
                        print(f'{curr_month}\t{min_temp}\t
                            {max_temp}')
                curr_month = month
                min_temp = min_temp_val
                max_temp = max_temp_val

if curr_month is not None:
        print(f'{curr_month}\t{min_temp}\t{max_temp}')
```

```
hd_user@DESKTOP-D5H3PKE:~/LabEx/ex4_1$ hdfs dfs -cat /BigDataLab/ex4_1/output/part-00000
01      -22.3   0.9
02      -18.4   0.6
03      -12.8   0.9
04      -8.8    4.0
05      1.3     5.6
06      10.2    10.2
07      10.2    12.5
08      10.6    11.7
09      -9999.0 10.6
```

Ex - 4(ii) - Implementing MapReduce2

mapper.py

```
#!/usr/bin/env python3
import sys
from itertools import combinations

k = int(sys.argv[1])

for line in sys.stdin:
    items = line.strip().split(',')
    items = sorted(items)
    for item in combinations(items, k):
        print(f'{",".join(item)}\t1')
```

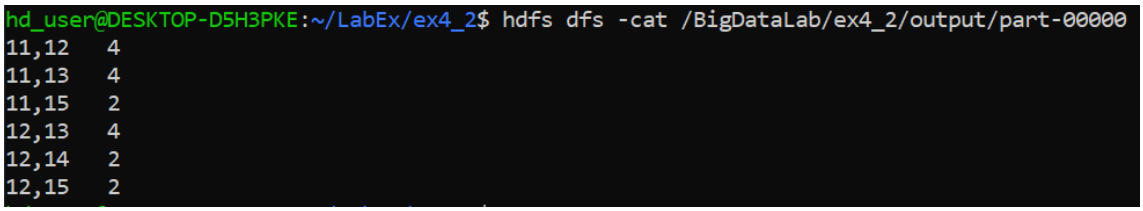
reducer.py

```
#!/usr/bin/env python3
import sys

min_support = 2
cur_item = None
cur_count = 0

for line in sys.stdin:
    item, count = line.strip().split('\t')
    count = int(count)

    if cur_item == item:
        cur_count += count
    else:
        if cur_item and cur_count >= min_support:
            print(f'{cur_item}\t{cur_count}')
        cur_item = item
        cur_count = count
```

```
if cur_item and cur_count >= min_support:
    print(f'{cur_item}\t{cur_count}')
```

shell command

```
hadoop jar
    /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3
  -input /BigDataLab/ex4_2/input/data.txt \
  -output /BigDataLab/ex4_2/output \
  -mapper "/home/hd_user/LabEx/ex4_2/mapper.py 2" \
  -reducer "/home/hd_user/LabEx/ex4_2/reducer.py" \
  -file /home/hd_user/LabEx/ex4_2/mapper.py \
  -file /home/hd_user/LabEx/ex4_2/reducer.py
```



Ex - 5(i) - PySpark WordCount

inputFile.txt

```
MapReduce processes large datasets
Hadoop is designed for big data
Data processing with MapReduce is efficient
Big data analytics is powered by Hadoop
```

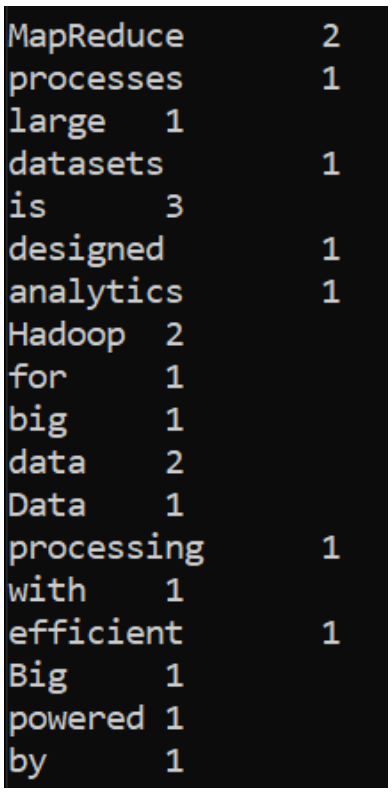
wordCount.py

```
from pyspark import SparkContext

sc = SparkContext(appName="WordCount")
```

```
input_file = "inputFile.txt"
lines = sc.textFile(input_file)

word_counts = lines.flatMap(lambda line : line.split())
        .map(lambda word : (word,1)).reduceByKey(lambda a,b: a+b

for word, count in word_counts.collect():
        print(f"{word}\t{count}")

sc.stop()
```

shell command

```
~/BigData/spark-3.5.3-bin-hadoop3/bin/spark-submit wordCount.py
# navigate to directory where spark-submit is present - spark-3
# this command to be typed in /BigData/ex5_1 not in spark-3....
```



Ex - 5(ii) - PySpark WordCount

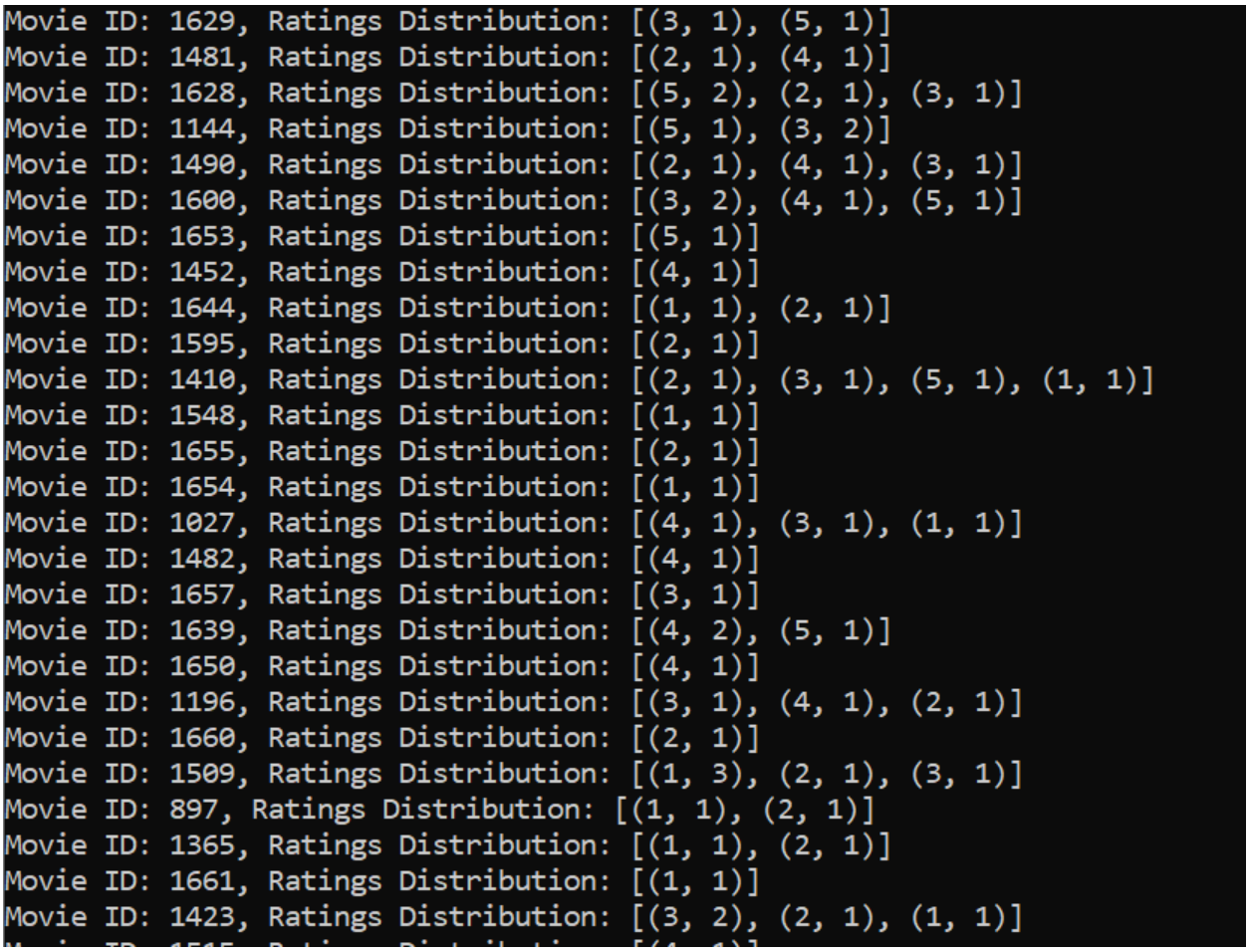
inputFile.txt - MovieRatings Dataset

movie_ratings.py

```
from pyspark import SparkContext

sc = SparkContext("local", "Movie Ratings Distribution")

text_file = sc.textFile("inputFile.txt")

movie_ratings = text_file.map(lambda line: line.split("\t"))
                         .map(lambda fields: (fields[1],
                                 int(fields[2])))

ratings_distribution = movie_ratings.map(lambda x:((x[0],x[1]),1
                                    .reduceByKey(lambda a,b:a+b)
                                    .map(lambda x: (x[0][0],
                                        (x[0][1], x[1])))
                                    .groupByKey()
                                    .mapValues(list)

output = ratings_distribution.collect()
for (movie_id, ratings) in output:
    print(f"Movie ID:{movie_id}, Ratings Distribution:{ratings}"

sc.stop()
```

shell command

```
~/BigData/spark-3.5.3-bin-hadoop3/bin/spark-submit
    movie_ratings.py
```



Ex - 6(i) - Mapvalue function of RDD

Data - friends_test.csv (Col - ID, Name, Age, Num of frnds)

Find avg number of friends for each unique age

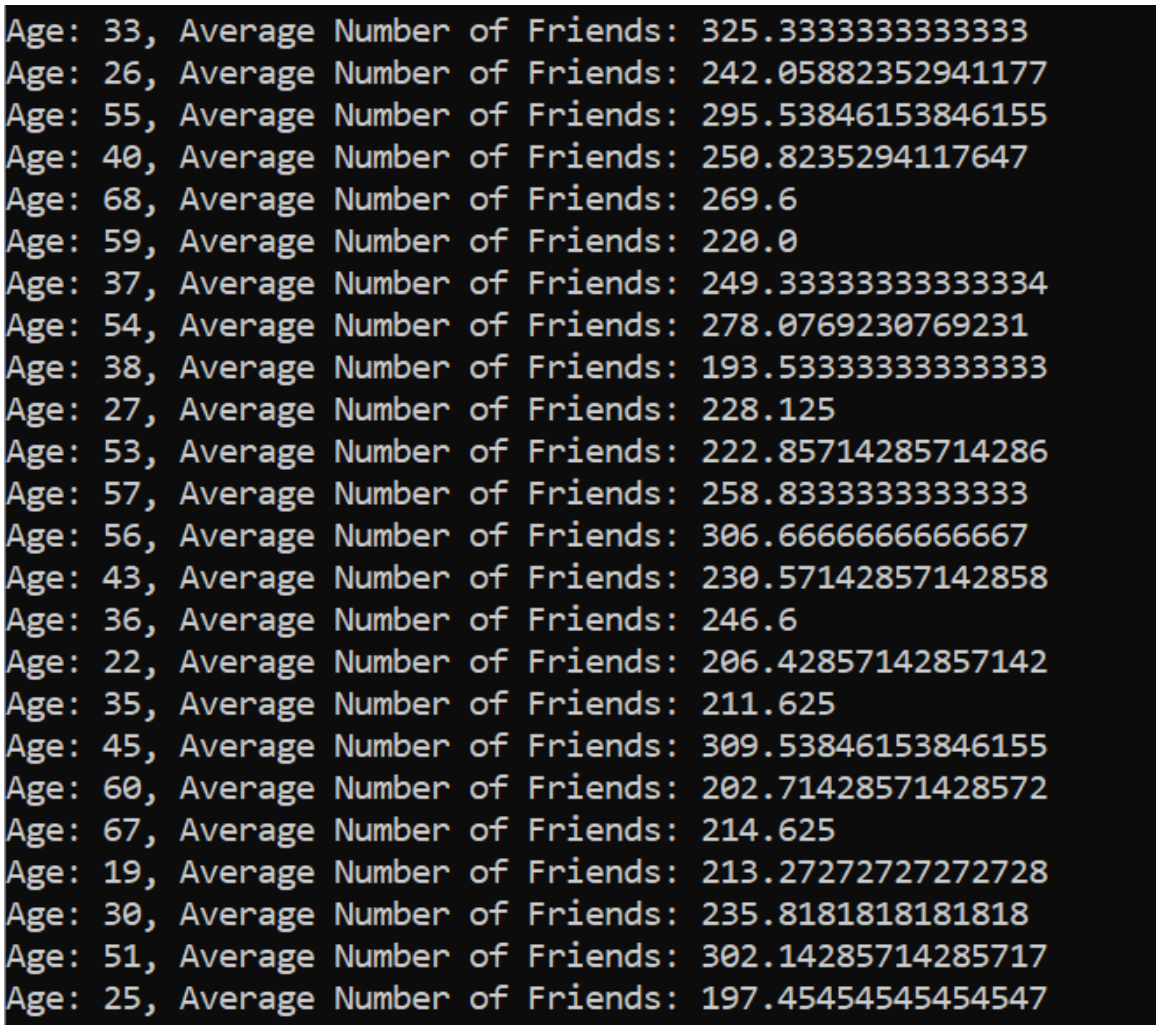
num_frnds.py

```
from pyspark import SparkContext

sc = SparkContext("local", "Friends Average")

data = sc.textFile("friends_test.csv").map(lambda line:
                line.split(","))
age_friends_rdd = data.map(lambda x: (int(x[2]), int(x[3])))
```

```
combined = age_friends_rdd.combineByKey(
    lambda friends: (friends, 1),
    lambda acc, friends: (acc[0] + friends, acc[1] + 1),
    lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])
)

average_friends_rdd = combined.mapValues(lambda x: x[0] / x[1])

results = average_friends_rdd.collect()
for age, avg_friends in results:
    print(f"Age: {age}, Average Number of Friends:{avg_friends}"

sc.stop()
```

shell command -

```
~/BigData/spark-3.5.3-bin-hadoop3/bin/spark-submit num_frnds.py
```



Ex - 6(ii)

Data - temp.csv

temp.py

```
from pyspark import SparkContext

sc = SparkContext("local", "Temperature Analysis")

data = sc.textFile("temp.csv").zipWithIndex()
    .filter(lambda x: x[1] != 0).map(lambda x: x[0].split(","))

temp_rdd = data.map(lambda x: (x[0], x[1], x[2], int(x[3])))
```

```
tmin_rdd = temp_rdd.filter(lambda x: x[2] == "TMIN")

overall_min_temp = tmin_rdd.map(lambda x: x[3]).min()
print(f"Overall Minimum Temperature: {overall_min_temp}")

min_temp_per_itemID = tmin_rdd.map(lambda x: (x[0], x[3]))
                              .reduceByKey(lambda a, b: min(a, b))
print("Minimum Temperature for each ItemID:")
for itemID, min_temp in min_temp_per_itemID.collect():
    print(f"ItemID: {itemID}, Minimum Temperature: {min_temp}")

min_temp_per_stationID = tmin_rdd.map(lambda x: (x[1], x[3]))
                              .reduceByKey(lambda a, b: min(a, b))
print("Minimum Temperature for each StationID:")
for stationID, min_temp in min_temp_per_stationID.collect():
    print(f"StationID:{stationID}, Minimum Temperature:{min_temp

sc.stop()
```

shell command

```
~/BigData/spark-3.5.3-bin-hadoop3/bin/spark-submit temp.py
```

Ex - 6(iii)

1. Use the “friends_test” dataset. Col1 is ID, Col2 is name, Col 3 is Age, Col 4 is num of friends. Understand **mapvalues function of RDD** in spark and find the average number of friends for each unique age present in the dataset.

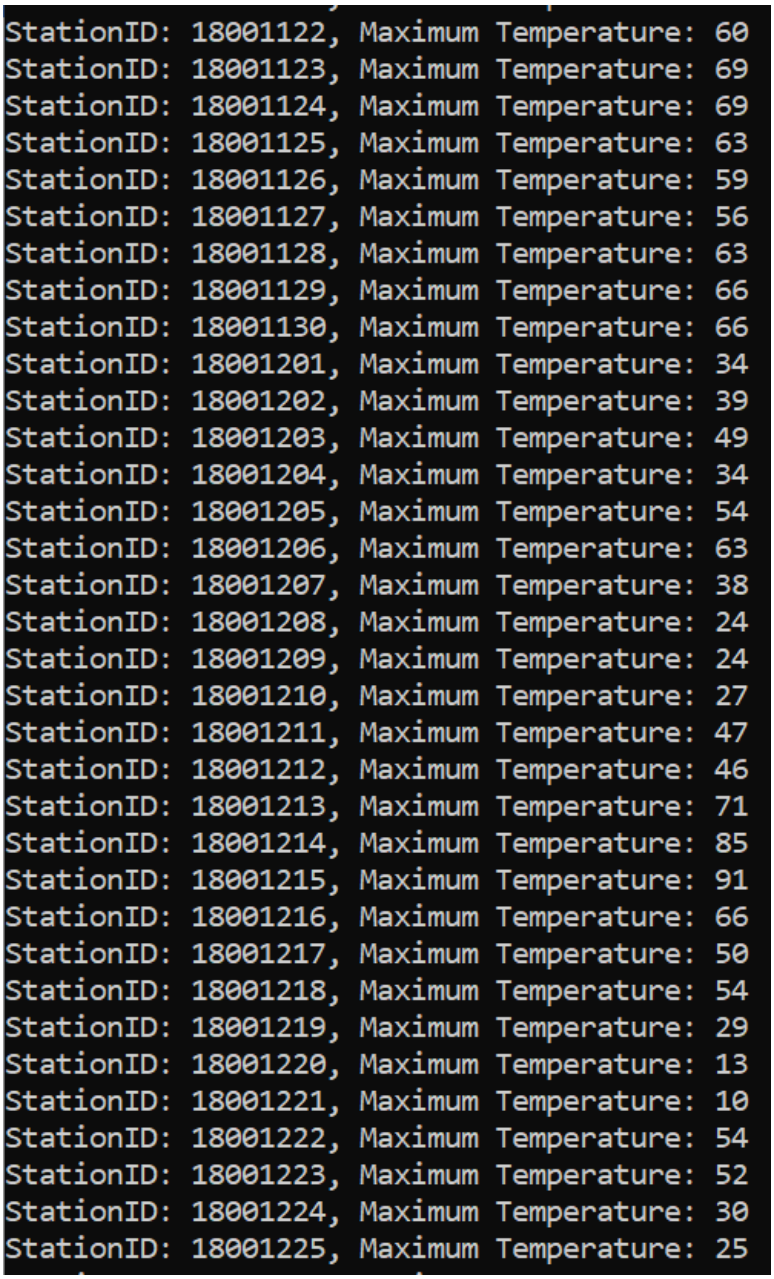
2. Use the “temp.csv” dataset. Column headers are present in the dataset. Understand filter operations and filter out only the “TMIN” values from the “desc” column. With the resultant data (RDD) find the following:

    a. Minimum temperature (overall)
    b. Minimum temperature for every ItemID
    c. Minimum temperature for every StationID.

3. Use the same dataset, filter only “TMAX” column and find the maximum temperatures just like the ones mentioned above.

```
from pyspark import SparkContext

sc = SparkContext("local", "Temperature Analysis")

data = sc.textFile("temp.csv").zipWithIndex()
        .filter(lambda x: x[1] != 0)
        .map(lambda x: x[0].split(","))
temp_rdd = data.map(lambda x: (x[0], x[1], x[2], int(x[3])))
tmax_rdd = temp_rdd.filter(lambda x: x[2] == "TMAX")

overall_max_temp = tmax_rdd.map(lambda x: x[3]).max()
print(f"Overall Maximum Temperature: {overall_max_temp}")

max_temp_per_itemID = tmax_rdd.map(lambda x: (x[0], x[3]))
                    .reduceByKey(lambda a, b: max(a, b))
print("Maximum Temperature for each ItemID:")
for itemID, max_temp in max_temp_per_itemID.collect():
    print(f"ItemID: {itemID}, Maximum Temperature: {max_temp}")

max_temp_per_stationID = tmax_rdd.map(lambda x: (x[1], x[3]))
                        .reduceByKey(lambda a, b: max(a, b))
```

```
print("Maximum Temperature for each StationID:")
for stationID, max_temp in max_temp_per_stationID.collect():
    print(f"StationID: {stationID}, Maximum Temperature: {max_te

sc.stop()
```



AWS

Ex - 1 - EC2

1. **Login to AWS Console**
   - Access your AWS account.
2. **Select VPC Service**
   - Choose the **VPC** service.
   - Ensure you are in the **Mumbai** region.
3. **Delete Existing VPC**
   - Remove any pre-existing VPCs to start with a clean setup.
4. **Create a Custom VPC and Components**
   - Create a new VPC with the following settings:
     - **VPC Name**: `snu-vpc`
     - **CIDR**: `192.168.0.0/16`
   - Create a **public subnet**:
     - **Subnet Name**: `public-subnet`
     - **CIDR**: `192.168.1.0/24`
   - Set up an **Internet Gateway**:
     - **IGW Name**: `snu-igw`
     - Attach the Internet Gateway to `snu-vpc`.
5. **Configure the Route Table**
   - Go to **Route Tables**.
   - Click on the Route Table ID associated with `snu-vpc`.
   - Edit the routes:
     - **Add route**: Destination `0.0.0.0/0` with target `snu-igw`.
6. **Allocate Elastic IPs**

- Go to **Elastic IPs**.
- Allocate two elastic IP addresses.

7. **Launch EC2 Instances**
   - Go to **EC2** service and select **Launch Instance**.
   - Configure two instances:
     - **Number of Instances**: 2
     - **Instance Type**: Select as required
     - **Key Pair**: No key pair (for practice environment)
   - Name the instances:
     - **Instance 1**: Web Server
     - **Instance 2**: Web Client
8. **Attach Elastic IPs to Instances**
   - Associate the previously allocated elastic IPs:
     - Go to **Elastic IPs**, select an IP, and associate it with **Web Server**.
     - Repeat for the **Web Client** instance.
9. **Connect to EC2 Instances**
   - Connect to **Web Server** and **Web Client** individually via **EC2 Instance Connect**.
10. **Install Apache on Web Server**
    - On the Web Server, run:

```
ping 8.8.8.8
sudo apt update
sudo apt install apache2 -y
sudo service apache2 status
```

    - Verify Apache service is running.
11. **Install Links on Web Client**

- On the Web Client, run:

```
ping 8.8.8.8
sudo apt update
sudo apt install links -y
```

12. **Configure Security Group for HTTP Access**
    - Go to **Security Groups** of the **Web Server**.
    - Edit inbound rules:
        - Add an HTTP rule:
            - **Type**: HTTP
            - **Source**: Anywhere (0.0.0.0/0)
13. **Modify Network ACL to Allow SSH and HTTP**
    - Check **Network ACLs** associated with `snu-vpc`.
    - Edit the rules:
        - Add rule:
            - **Rule number**: 100
            - **Type**: All Traffic
            - **Source**: 0.0.0.0/0
            - **Action**: Allow
14. **Test Web Access from Web Client**
    - On the Web Client, run:

```
links http://<Web Server IP>
```

    - Replace `<Web Server IP>` with the elastic IP address of the Web Server.

Ex 2 - Testing Route 53 Service with Custom Domain

1. **Create a Hosted Zone in Route 53**
    - Go to **Route 53** in the AWS Console.
    - Create a new **hosted zone** with the domain name:
        - **Domain**: `21011101122.ngaws.xyz`
    - AWS will automatically generate Name Servers (NS) for this hosted zone.
2. **Login to GoDaddy**
    - Go to https://www.godaddy.com/.
    - Login with the credentials:
        - **Username**: `aws-ng`
        - **Password**: `Welcome1!`
3. **Update Name Server Records in GoDaddy**
    - In GoDaddy, navigate to **DNS Management** for `21011101122.ngaws.xyz`.
    - Add a new **NS (Name Server) record** with the following:
        - **Name**: `21011101122`
        - **Type**: NS
        - **Value**: Paste the name server information from Route 53's hosted zone.
4. **Create a Record in Route 53**
    - Go back to **Route 53** and open the hosted zone `21011101122.ngaws.xyz`.
    - Create a new record:
        - **Name**: `www`
        - **Type**: A (IPv4 Address)
        - **Value**: Enter the **IP address of your Web Server instance**.
        - **Routing Policy**: Simple
5. **Test Domain Reachability**
    - From the **Web Client** instance, verify domain reachability:

- Test with the `links` browser:

```
links www.21011101122.ngaws.xyz
```

- Run an **nslookup** command to check the DNS resolution:

```
nslookup www.21011101122.ngaws.xyz
```

Ex - 3 - Setting Up IAM Users with Console Access and Permissions

1. **Access IAM Service**
   - Search for **IAM** in the AWS Console and open it.
2. **Create an IAM User**
   - Go to **Users** in the IAM dashboard.
   - Click **Create user**.
   - Enter a **username** (e.g., `example_user`).
   - Enable **AWS Management Console access**.
   - Set the console password or choose an auto-generated one.
   - Click **Next** and complete the user creation process.
3. **Copy User Sign-In Details**
   - After creating the user, copy the sign-in link, username, and password.
   - Use these details to log in as the new user in a separate tab.
4. **Assign Permissions to the User**
   - In the root account, go to **IAM**, then **Users**, and select the newly created user.
   - Click on **Add permissions**.
     - Choose **Attach policies directly** and select **EC2 Full Access** policy.

- Alternatively, create a **User Group** with permissions:
  - Go to **User groups** and click **Create group**.
  - Enter a **Group name** (e.g., `EC2_Admins`).
  - Attach the **EC2 Full Access** policy.
  - Add the user to the group by selecting the user from the list.

5. **Create Another IAM User**
   - Repeat the process to create another IAM user with similar or different permissions as needed.

Creating Custom VPC, EC2 Instance and working on SG & NACL

1. Login into your AWS account.
2. Choose VPC Service
3. Choose the region Mumbai
4. Delete the existing VPC
5. Setup custom VPC and its components - Create VPC (snu-vpc: 192.168.0.0/16, public-subnet: 192.168.1.0/24, snu-igw: Attach to VPC)
6. Route Table - Click on Route Table ID, Routes - Edit Routes - Add Routes - 0.0.0.0/0 - internet gateway
7. Get 2 elastic public IP - attach elastic IP
8. Create two EC2 instances(search for ec2) - instances, launch instance (2, VMs, quick start, key pair name: no key pair name)
9. EC2 Instances - Name VM1 as Web Server & VM2 as Web Client
10. Attach the public IP address - Elastic IP Addresses, click on IP, associate IP, web server/client instance
11. Instances - Server/Client(one by one) - Connect to the instance via EC2 instance connect

12. Install Apache (web service) in Web Server - ping 8.8.8.8, sudo apt update, sudo apt install apache, sudo apt install apache2, service apache2 status
13. Install Links (web client) in Web Client - ping 8.8.8.8, sudo apt update, sudo apt install links
14. In the Security Group of Web Server, add rule to allow HTTP access : Instances - Click Instance ID of Server - Security - Security Groups - Edit inbound rules - add rules - HTTP, Anywhere, 0.0.0.0/0
15. Allow SSH & HTTP on the NACL - Instances, Web Server, Security - Check Network ACLs - Click on ID, Edit, Remove, Add - 100, all traffic, 0.0.0.0/0, Allow
16. Test the web access from the web client using links app - copy IP of web server and links <IP> on client, links http://<IP>

Testing Route53 Service

1. Create a hosted zone in AWS Route 53 service(search) – 21011101122.ngaws.xyz
2. Login to https://www.godaddy.com/, aws-ng – Welcome1!
3. Get the name server information from Route 53 dashboard(Value/Route traffic to) and update NS record in GoDaddy portal – add new record, NS, 21011101122, paste the server info
4. AWS Route 53 – Create a record in hosted zone – www, IP address of web server(take it from instances), simple
5. Check reachability - Web client – links www.21011101122.ngaws.xyz, nslookup www.21011101122.ngaws.xyz

IAM Service(Search)

Users - Create user, user_name, give console access, i want to create IAM user

Copy user details - sign in using these details in separate tab

Root account - IAM, click on the user, add permissions, attach policy(ec2 full access) or User Groups, create user group, user group name, select users, attach permissions

create another user