KUMARAGURU
college of technology
character is life

DATA COLLECTION AND DATA MANAGEMENT LAB WORKBOOK

Subject Code & Name : U18AII3205- Data Collection and Data Management

Regulations : R18

Class : II B.Tech- AI&DS

Certificate

This is to certify that it is a bonafide record of practical work done by Sri/Kum. _____ bearing the Roll No. _____ of _____ class_____ branch in the _____ Laboratory during the academic year _____ under our supervision.

Faculty in charge

Internal Examiner

Table of Contents

S.No	Date	Name of the experiment	Program (10)	Execution (10)	Viva (10)	Total (30)	Staff sign
1.		Manipulating Data using MySQL					
2.		Processing CSV Data using Python					
3.		Processing JSON Data using Python					
4.		Processing XLS Data using Python					
5.		Implementation of Mongo Client using Python					
6.		Implementation of Map and Filter					
7.		Implementation of List Comprehension					
8.		File Management tasks in Hadoop					

Ex No:1	Manipulating Data using MySQL

Aim:

To execute the following instructions using MySQL.

DDL:

Data Definition Language (DDL) is used to create and modify the structure of objects in a database using predefined commands and a specific syntax. These database objects include tables, sequences, locations, aliases, schemas, and indexes.

DDL Commands:

- CREATE

CREATE TABLE <table-name>

(<column name><data type>[<size>]

(<column name><data type>[<size>]……

);

- ALTER

ALTER TABLE <table-name> ADD <column-name><data type><size>;

- DROP

DROP TABLE table-name;

- TRUNCATE

TRUNCATE TABLE table-name;

DML:

DML is an abbreviation for **Data Manipulation Language**. Represents a collection of programming languages explicitly used to make changes to the database, such as: CRUD operations to create, read, update, and delete data. Using INSERT, SELECT, UPDATE, and DELETE commands.

DML Commands:

- INSERT

INSERT INTO table_name (column1, column2, column3, ...)VALUES (value1, value2, value3);

- SELECT

SELECT column1, column2, ...

FROM table_name;

- UPDATE

UPDATE table_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

- DELETE

DELETE FROM table_name WHERE condition;

Creating Table and Inserting Values into the Table:

Program:

```
CREATE TABLE Employee (
  Employeeid INTEGER UNIQUE,
  firstname VARCHAR(20) NOT NULL,
  lastname VARCHAR(20) NOT NULL,
  joiningdate DATE NOT NULL,
  salary INTEGER NOT NULL,
  department VARCHAR(30) NOT NULL
);
CREATE TABLE Incentive (
Employeeid INTEGER PRIMARY KEY,
IncentiveDate DATE NOT NULL,
IncentiveAmount INTEGER NOT NULL
);


INSERT INTO Employee values(1001,"Tony","Stark",'2020-06-01',30000,"Insurance");
INSERT INTO Employee values(1002,"Kevin","Feige",'2020-07-01',28000,"Insurance");
INSERT INTO Employee values(1003,"Thomas","Shelby",'2020-08-01',26000,"Insurance");
INSERT INTO Employee values(1004,"Steve","Rogers",'2020-09-01',24000,"Insurance");
INSERT INTO Employee values(1005,"Hugh","Jackman",'2020-10-01',22000,"Insurance");
INSERT INTO Incentive VALUES (1001,'2021-06-01',1000);
INSERT INTO Incentive VALUES (1002,'2021-07-01',1200);
INSERT INTO Incentive VALUES (1003,'2021-08-01',1300);
INSERT INTO Incentive VALUES (1004,'2021-09-01',1400);
INSERT INTO Incentive VALUES (1005,'2021-10-01',900);
```

1.Get all the Employee details from Employee Table:

Program:

```
#1
SELECT * FROM Employee;
```

Output:

Output

```
Employeeid      firstname       lastname
joiningdate     salary  department
1001    Tony    Stark   2020-06-01      30000
Insurance
1002    Kevin   Feige   2020-07-01      28000
Insurance
1003    Thomas  Shelby  2020-08-01      26000   Finance
1004    Steve   Rogers  2020-09-01      24000
Insurance
1005    Hugh    Jackman 2020-10-01      22000   Help
Desk
```

2.Get First name Last name with Salary:

Program:

```
#2
SELECT firstname,lastname,salary FROM Employee;
```

Output:

Output

```
firstname       lastname        salary
Tony    Stark   30000
Kevin   Feige   28000
Thomas  Shelby  26000
Steve   Rogers  24000
Hugh    Jackman 22000

[Execution complete with exit code 0]
```

3.Get First Name from Employee Table in Uppercase.

Program:

```
#3
SELECT UPPER(firstname) FROM Employee;
```

Output:

Output

```
UPPER(firstname)
TONY
KEVIN
THOMAS
STEVE
HUGH

[Execution complete with exit code 0]
```

4.Get Unique department from Employee Table.

Program:

```
#4
SELECT DISTINCT department FROM Employee;
```

Output:

Output

```
department
Insurance
Finance
Help Desk

[Execution complete with exit code 0]
```

5.Display the First 3 characters of Last name from Employee Table.

Program:

```
#5
SELECT SUBSTRING(lastname,1,3) FROM Employee;
```

Output:

Output

```
SUBSTRING(lastname,1,3)
Sta
Fei
She
Rog
Jac

[Execution complete with exit code 0]
```

6.Get First name from Employee Table after replacing ‘O’ with ‘#’.

Program:

```
#6
SELECT replace(firstname,'o','#') FROM Employee;
```

Output:

Output

```
replace(firstname,'o','#')
T#ny
Kevin
Th#mas
Steve
Hugh

[Execution complete with exit code 0]
```

7.Display First name Last name as single column separated by underscore.
Program:

```
#7
SELECT CONCAT(firstname,'_',lastname) FROM Employee;
```

Output:

Output

```
CONCAT(firstname,'_',lastname)
Tony_Stark
Kevin_Feige
Thomas_Shelby
Steve_Rogers
Hugh_Jackman

[Execution complete with exit code 0]
```

8.Display First name and Last name of Employee with their respective Incentive amount.

Program:

```
#8
SELECT Employee.firstname,Employee.lastname,Incentive.IncentiveAmount
FROM Employee,Incentive WHERE Employee.Employeeid=Incentive.Employeeid
```

Output:

Output

```
firstname       lastname        IncentiveAmount
Tony    Stark   1000
Kevin   Feige   1200
Thomas  Shelby  1300
Steve   Rogers  1400
Hugh    Jackman 900

[Execution complete with exit code 0]
```

9.Calculate the time duration between Joining date and Incentive date.

Program:

```
#9
SELECT DATEDIFF(IncentiveDate,joiningdate)  as days from Employee,
Incentive WHERE Employee.Employeeid=Incentive.Employeeid ;
```

Output:

Output

```
days
365
365
365
365
365

[Execution complete with exit code 0]
```

10.Display Incentive date and amount for the department Insurance.

Program:

```
#10
SELECT Employeeid,IncentiveDate,IncentiveAmount FROM Incentive
WHERE Employeeid IN (SELECT Employeeid FROM Employee WHERE Department = 'Insurance');
```

Output:

Output

```
Employeeid      IncentiveDate   IncentiveAmount
1001    2021-06-01      1000
1002    2021-07-01      1200
1004    2021-09-01      1400

[Execution complete with exit code 0]
```

Result:

The programs were executed successfully, and the result was verified.

Ex No:2	Processing CSV Data using Python

Aim:

To process the given CSV file using Python.

Program:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
cd/content/drive/My Drive
```

Importing necessary libraries:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Reading CSV File:

```
data=pd.read_csv("heart.csv")
```

Performing EDA:

data.head()

```
data.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

data.shape

```
[5]  data.shape

(918, 12)
```

data.duplicated().sum()

```
[6]  data.duplicated().sum()

0
```

data.describe()

```
data.describe()
```

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.553377
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.497414
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.000000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.000000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.000000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.000000

data.info()

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

data.columns

```
[9] data.columns
```

```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

data.nunique()

```
[10] data.nunique()
```

```
Age               50
Sex                2
ChestPainType      4
RestingBP         67
Cholesterol      222
FastingBS          2
RestingECG         3
MaxHR            119
ExerciseAngina     2
Oldpeak           53
ST_Slope           3
HeartDisease       2
dtype: int64
```

data.isnull().sum()

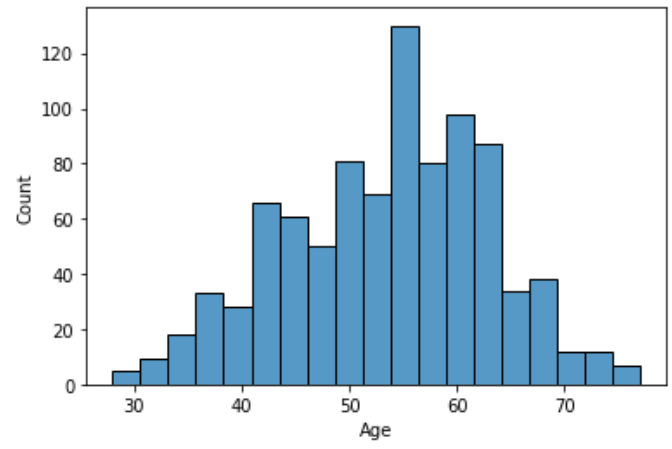
```
data.isnull().sum()
```

```
Age              0
Sex              0
ChestPainType    0
RestingBP        0
Cholesterol      0
FastingBS        0
RestingECG       0
MaxHR            0
ExerciseAngina   0
Oldpeak          0
ST_Slope         0
HeartDisease     0
dtype: int64
```

sns.histplot(x='Age', data=data,)

plt.show()

```
sns.histplot(x='Age', data=data, )
plt.show()
```



Result:

Thus, the given csv file is processed using python.

Exp no: 3	Processing JSON data using python

Aim:

To process the given JSON data file using python.

Theory:

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight data-interchange format
- JSON is plain text written in JavaScript object notation
- JSON is used to send data between computers
- JSON is language independent

Example syntax:

```
[
  {
    "name": "Molecule Man",
    "age": 29,
    "secretIdentity": "Dan Jukes",
    "powers": ["Radiation resistance", "Turning tiny", "Radiation blast"]
  },
  {
    "name": "Madame Uppercut",
    "age": 39,
    "secretIdentity": "Jane Wilson",
    "powers": [
      "Million tonne punch",
      "Damage resistance",
      "Superhuman reflexes"
    ]
  }
]
```

Program:

```
country = '{"name": "United States", "population": 331002651}'

print(type(country))
```

```
import json
country = '{"name": "United States", "population": 331002651}'
country_dict = json.loads(country)
print(type(country))
print(type(country_dict))


print(country_dict['name'])


countries = '["United States", "Canada"]'
counties_list= json.loads(countries)
print(type(counties_list))


import json
bool_string = 'true'
bool_type = json.loads(bool_string)
print(bool_type)


import json
# Data to be written
dictionary = {
  "name": "United States",
  "population": 331002651,
  "capital": "Washington D.C.",
  "languages": [
    "English",
    "Spanish"
  ]
}

# Serializing json
```

```
json_object = json.dumps(dictionary, indent=4)
# Writing to sample.json
with open("united_states.json", "w") as outfile:
    outfile.write(json_object)



import json
with open('united_states.json') as f:
  data = json.load(f)
print(type(data))


print(data.keys())


data['name']


import json
languages = ["English","French"]
country = {
    "name": "Canada",
    "population": 37742154,
    "languages": languages,
    "president": None,
}
country_string = json.dumps(country)
print(country_string)
```

```
import json
languages = ["English", "French"]
languages_string = json.dumps(languages)
print(languages_string)
```

```
import json
# Tuple is encoded to JSON array.
languages = ("English", "French")
# Dictionary is encoded to JSON object.
country = {
    "name": "Canada",
    "population": 37742154,
    "languages": languages,
    "president": None,
}
with open('countries_exported.json', 'w') as f:
    json.dump(country, f, indent=4)
```

```
import json
class Country:
    def __init__(self, name, population, languages):
        self.name = name
        self.population = population
        self.languages = languages
canada = Country("Canada", 37742154, ["English", "French"])
class CountryEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, Country):
```

```
        # JSON object would be a dictionary.
        return {
                "name" : o.name,
                "population": o.population,
                "languages": o.languages
               }
    else:
        # Base class will raise the TypeError.
        return super().default(o)
print(json.dumps(canada, cls=CountryEncoder))
```

Output:

Converting JSON string to Python Object

```
[ ] country = '{"name": "United States", "population": 331002651}'
    print(type(country))
```

```
<class 'str'>
```

```
import json

country = '{"name": "United States", "population": 331002651}'
country_dict = json.loads(country)

print(type(country))
print(type(country_dict))
```

```
<class 'str'>
<class 'dict'>
```

```
[ ] print(country_dict['name'])
```

```
United States
```

```
[ ] countries = '["United States", "Canada"]'
    counties_list= json.loads(countries)

    print(type(counties_list))
```

```
<class 'list'>
```

```
[ ] import json

    bool_string = 'true'
    bool_type = json.loads(bool_string)
    print(bool_type)
```

```
True
```

Converting JSON file to Python object

Save the following JSON data as a new file and name it united_states.json:

```
import json

# Data to be written
dictionary = {
   "name": "United States",
   "population": 331002651,
   "capital": "Washington D.C.",
   "languages": [
      "English",
      "Spanish"
   ]
}

# Serializing json
json_object = json.dumps(dictionary, indent=4)

# Writing to sample.json
with open("united_states.json", "w") as outfile:
    outfile.write(json_object)
```

```
[ ] import json

    with open('united_states.json') as f:
      data = json.load(f)

    print(type(data))
```

```
<class 'dict'>
```

```
[ ] print(data.keys())
```

```
dict_keys(['name', 'population', 'capital', 'languages'])
```

```
data['name']
```

```
'United States'
```

```
[ ] import json

    languages = ["English","French"]
    country = {
        "name": "Canada",
        "population": 37742154,
        "languages": languages,
        "president": None,
    }

    country_string = json.dumps(country)
    print(country_string)
```

```
{"name": "Canada", "population": 37742154, "languages": ["English", "French"], "president": null}
```

```
import json

languages = ["English", "French"]

languages_string = json.dumps(languages)
print(languages_string)
```

```
["English", "French"]
```

+ Code + Text

Writing Python object to a JSON file

```
[ ] import json

    # Tuple is encoded to JSON array.
    languages = ("English", "French")
    # Dictionary is encoded to JSON object.
    country = {
        "name": "Canada",
        "population": 37742154,
        "languages": languages,
        "president": None,
    }

    with open('countries_exported.json', 'w') as f:
        json.dump(country, f, indent=4)
```

output of the above code part been written to a json file countries_exported.json in the files section

Converting custom Python objects to JSON objects

```
import json

class Country:
    def __init__(self, name, population, languages):
        self.name = name
        self.population = population
        self.languages = languages


canada = Country("Canada", 37742154, ["English", "French"])

class CountryEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, Country):
           # JSON object would be a dictionary.
            return {
                      "name" : o.name,
                      "population": o.population,
                      "languages": o.languages
                    }
        else:
            # Base class will raise the TypeError.
            return super().default(o)
print(json.dumps(canada, cls=CountryEncoder))
```

```
{"name": "Canada", "population": 37742154, "languages": ["English", "French"]}
```

Result:

Thus the given JSON file has been processed using python.

Exp no: 4	Processing XLS Data using Python

Aim:

To process the given XLS file using python.

Theory:

The XLS file extension is used for files saved as Microsoft Excel worksheets. Excel is a popular spreadsheet program used with data like numbers and formulas, text, and drawing shapes. Excel is part of the Microsoft Office Suite of software.

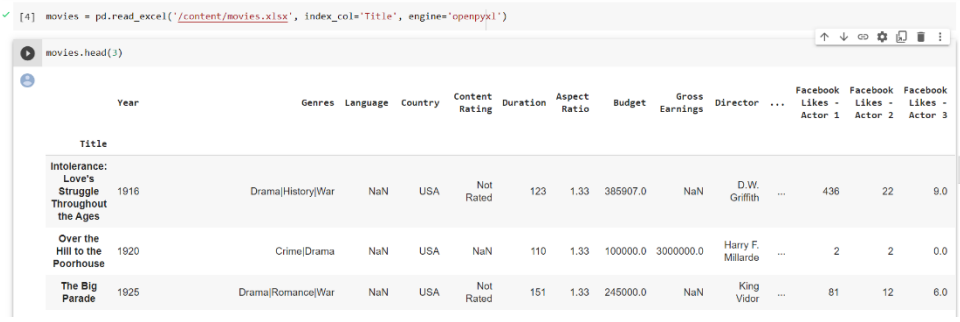
XLS was created by Microsoft for use with Microsoft Excel and is also known as Binary Interchange File Format (BIFF). This file type was introduced for the first time by making it part of Excel for Windows in 1987.

Program:

```
movies = pd.read_excel('/content/movies.xlsx', index_col='Title', engine='openpyxl')


movies.head(3)


len(movies)


movies.keys()


dfs = pd.read_excel('/content/movies.xlsx', sheet_name=None, engine='openpyxl')




movies_1900 = dfs['1900s']
 movies_1900.head()
```

```
movies_2000 = pd.read_excel('/content/movies.xlsx', sheet_name='2000s',
engine='openpyxl')
movies_2000.head()


book = pd.ExcelFile('/content/movies.xlsx')


book.sheet_names


 book.parse
 df3 = book.parse('2000s')
 df3.head()


df4 = book.parse('2000s', index_col=0, usecols=['Title', 'Year', 'Director', 'Budget'])
df4.head()
```

Output:



```
[4] movies = pd.read_excel('/content/movies.xlsx', index_col='Title', engine='openpyxl')

movies.head(3)
```

Title	Year	Genres	Language	Country	Content Rating	Duration	Aspect Ratio	Budget	Gross Earnings	Director	...	Facebook Likes - Actor 1	Facebook Likes - Actor 2	Facebook Likes - Actor 3
Intolerance: Love's Struggle Throughout the Ages	1916	Drama|History|War	NaN	USA	Not Rated	123	1.33	385907.0	NaN	D.W. Griffith	...	436	22	9.0
Over the Hill to the Poorhouse	1920	Crime|Drama	NaN	USA	NaN	110	1.33	100000.0	3000000.0	Harry F. Millarde	...	2	2	0.0
The Big Parade	1925	Drama|Romance|War	NaN	USA	Not Rated	151	1.33	245000.0	NaN	King Vidor	...	81	12	6.0

```
[ ] len(movies)
```

```
1338
```

```
movies.keys()
```

```
Index(['Year', 'Genres', 'Language', 'Country', 'Content Rating', 'Duration',
       'Aspect Ratio', 'Budget', 'Gross Earnings', 'Director', 'Actor 1',
       'Actor 2', 'Actor 3', 'Facebook Likes - Director',
       'Facebook Likes - Actor 1', 'Facebook Likes - Actor 2',
       'Facebook Likes - Actor 3', 'Facebook Likes - cast Total',
       'Facebook likes - Movie', 'Facenumber in posters', 'User Votes',
       'Reviews by Users', 'Reviews by Crtiics', 'IMDB Score'],
      dtype='object')
```

```
[ ] dfs = pd.read_excel('/content/movies.xlsx', sheet_name=None, engine='openpyxl')
```

```
movies_1900 = dfs['1900s']

movies_1900.head()
```

	Title	Year	Genres	Language	Country	Content Rating	Duration	Aspect Ratio	Budget	Gross Earnings	...	Facebook Likes - Actor 1	Facebook Likes - Actor 2	Facebook Likes - Actor 3	Facebook Likes - cast Total	Facebook likes - Movie	Fac in
0	Intolerance: Love's Struggle Throughout the Ages	1916	Drama|History|War	NaN	USA	Not Rated	123	1.33	385907.0	NaN	...	436	22	9.0	481	691	
1	Over the Hill to the Poorhouse	1920	Crime|Drama	NaN	USA	NaN	110	1.33	100000.0	3000000.0	...	2	2	0.0	4	0	
2	The Big Parade	1925	Drama|Romance|War	NaN	USA	Not Rated	151	1.33	245000.0	NaN	...	81	12	6.0	108	226	
3	Metropolis	1927	Drama|Sci-Fi	German	Germany	Not Rated	145	1.33	6000000.0	26435.0	...	136	23	18.0	203	12000	
4	Pandora's Box	1929	Crime|Drama|Romance	German	Germany	Not Rated	110	1.33	NaN	9950.0	...	426	20	3.0	455	926	

select specific worksheet

```
movies_2000 = pd.read_excel('/content/movies.xlsx', sheet_name='2000s', engine='openpyxl')

movies_2000.head()
```

	Title	Year	Genres	Language	Country	Content Rating	Duration	Aspect Ratio	Budget	Gross Earnings	...	Facebook Likes - Actor 1	Facebook Likes - Actor 2	Facebook Likes - Actor 3	Facebook Likes - cast Total	Facebook likes - Movie
0	102 Dalmatians	2000	Adventure|Comedy|Family	English	USA	G	100.0	1.85	85000000.0	66941559.0	...	2000.0	795.0	439.0	4182	372
1	28 Days	2000	Comedy|Drama	English	USA	PG-13	103.0	1.37	43000000.0	37035515.0	...	12000.0	10000.0	664.0	23864	0
2	3 Strikes	2000	Comedy	English	USA	R	82.0	1.85	6000000.0	9821335.0	...	939.0	706.0	585.0	3354	118
3	Aberdeen	2000	Drama	English	UK	NaN	106.0	1.85	6500000.0	64148.0	...	844.0	2.0	0.0	846	260
4	All the Pretty Horses	2000	Drama|Romance|Western	English	USA	PG-13	220.0	2.35	57000000.0	15527125.0	...	13000.0	861.0	820.0	15006	652

5 rows × 25 columns

```
[ ]  book = pd.ExcelFile('/content/movies.xlsx')
```

To display list of all worksheets

```
book.sheet_names
```

```
['1900s', '2000s', '2010s', '3000s']
```

```
[ ]  ?book.parse

     df3 = book.parse('2000s')

     df3.head()
```

	Title	Year	Genres	Language	Country	Content Rating	Duration	Aspect Ratio	Budget	Gross Earnings	...	Facebook Likes - Actor 1	Facebook Likes - Actor 2	Facebook Likes - Actor 3	Facebook Likes - cast Total	Facebook likes - Movie
0	102 Dalmatians	2000	Adventure|Comedy|Family	English	USA	G	100.0	1.85	85000000.0	66941559.0	...	2000.0	795.0	439.0	4182	372
1	28 Days	2000	Comedy|Drama	English	USA	PG-13	103.0	1.37	43000000.0	37035515.0	...	12000.0	10000.0	664.0	23864	0
2	3 Strikes	2000	Comedy	English	USA	R	82.0	1.85	6000000.0	9821335.0	...	939.0	706.0	585.0	3354	118
3	Aberdeen	2000	Drama	English	UK	NaN	106.0	1.85	6500000.0	64148.0	...	844.0	2.0	0.0	846	260
4	All the Pretty Horses	2000	Drama|Romance|Western	English	USA	PG-13	220.0	2.35	57000000.0	15527125.0	...	13000.0	861.0	820.0	15006	652

5 rows × 25 columns

parse supports most of the parameters available for read_excel

```
df4 = book.parse('2000s', index_col=0, usecols=['Title', 'Year', 'Director', 'Budget'])

df4.head()
```

Title	Year	Budget	Director
102 Dalmatians	2000	85000000.0	Kevin Lima
28 Days	2000	43000000.0	Betty Thomas
3 Strikes	2000	6000000.0	DJ Pooh
Aberdeen	2000	6500000.0	Hans Petter Moland
All the Pretty Horses	2000	57000000.0	Billy Bob Thornton

Result:

Thus ,the XLS file is processed using Python.

Exp no: 5	Implementation of MongoClient Using Python

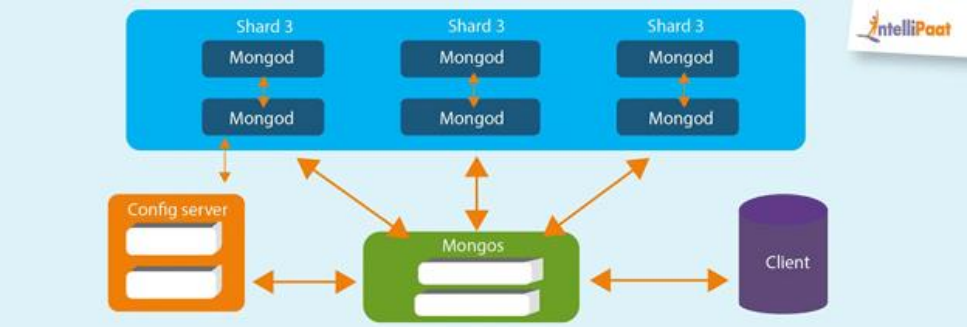
Aim:

To implement the Mongo client using python.

Theory:

MongoDB is an open-source document-oriented database that is designed to store a large scale of data and also allows you to work with that data very efficiently. It is categorized under the NoSQL (Not only SQL) database because the storage and retrieval of data in the MongoDB are not in the form of tables.

The **MongoClient** is an API that allows for making Connections to MongoDB. It takes in an URI to the database in MongoDB Atlas and returns a mutable reference to the database object.



Program:

```
pip install pymongo

from pymongo import MongoClient

client=MongoClient()

from pymongo import MongoClient

client = MongoClient()

print(client)
```

Output:

```
In [25]: pip install pymongo

Requirement already satisfied: pymongo in c:\users\siddhartha devan v\anaconda3\lib\site-packages (4.3.3)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in c:\users\siddhartha devan v\anaconda3\lib\site-packages (from pymong
o) (2.2.1)
Note: you may need to restart the kernel to use updated packages.
```

```
In [26]: from pymongo import MongoClient
```

```
In [27]: client=MongoClient()
```

```
In [29]: from pymongo import MongoClient
client = MongoClient()
print(client)

MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True)
```

Result:

Thus, a Mongo Client is connected to the MongoDB server.

Exp no: 6	Implementation of Map and Filter

Aim:

To implement Maps and Filters in Python.

Theory:

lambda():

While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword. Hence, anonymous functions are also called lambda functions.

Lambda function can use any quantity of parameter, but only have one expression

Syntax: lambda argument: manipulate(argument)

MAP():

map() function

simple syntax: map(func, iterable)

parameter: func is an function that map() pass to the every elements in the iterable object, the iterable is an object that has iter attribute, so every elements can execute the function.

return value: a map object

filter():

filter() function is using a function to "filter" the sequence, the function is going to examinate every elements in the sequence is True or False

filter() syntax: filter(func, iterable)

Parameter: func test iterable sequances' elements is True or False, iterable is the iterable sequances that been filter

Return value: an iterable sequance that every elements is True to the filter function func

Layman term: filter() is to filter a set of data based on the given conditions

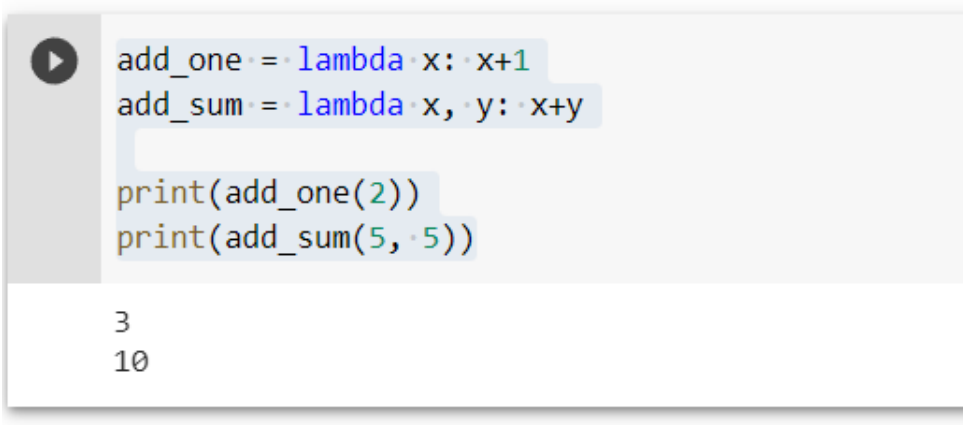
Program:

```
add_one = lambda x: x+1
add_sum = lambda x, y: x+y
print(add_one(2))
print(add_sum(5, 5))

numbers = [1, 2, 3, 4, 5]
for i in range(0, len(numbers)):
    numbers[i] += 1
print(numbers)



def add_one(n):
    return n+1
numbers = [1, 2, 3, 4, 5]
result = map(add_one, numbers)
print(result)
print(type(result))
print(list(result))



numbers = (1, 2, 3, 4, 5)
print(f"Is tuple numbers iterable? Answer: {hasattr(numbers, '__iter__')}")
result = map(add_one, numbers)
print(result)
print(type(result))
print(tuple(result))

numbers = (1, 2, 3, 4, 5)
result = map(lambda x: x + 1, numbers)
print(tuple(result))
```

```
numbers = [3, '23', 42]
print(list(map(list, numbers)))

university = [{'name': 'NYU',
               'city': 'New York'},
              {'name': 'NUS',
               'city': "Singapore"}]
names = list(map(lambda x: x['name'], university))
print(names)



# filter vowel
def func(variable):
    letters = ['a', 'e', 'i', 'o', 'u']
    if (variable.lower() in letters):
        return True
    else:
        return False
# given sequance
sequance = ['I', 'l', 'o', 'v', 'e', 'p', 'y', 't', 'h', 'o', 'n']
filtered = list(filter(func, sequance))
print(f"The vowel in the sequance is {filtered}")



# odd number
odd_number = filter(lambda x: x % 2, numbers)
print(f"The odd number is {list(odd_number)}.")
# even number
even_number = filter(lambda x: x % 2 == 0, numbers)
```

```
print(f"The even number is {list(even_number)}.")
# positive number
positive_number = filter(lambda x: x > 0, numbers)
print(f"The positive number is {list(positive_number)}.")


university = [{'name': 'NYU',
               'city': 'New York'},
              {'name': 'NUS',
               'city': "Singapore"}]
names = list(filter(lambda x: x['name'] == 'NUS', university))
print(names)
```

Output:



```
numbers = [1, 2, 3, 4, 5]
for i in range(0, len(numbers)):
    numbers[i] += 1
print(numbers)
```

```
[2, 3, 4, 5, 6]
```

After map() function:

```
def add_one(n):
    return n+1

numbers = [1, 2, 3, 4, 5]
result = map(add_one, numbers)
print(result)
print(type(result))
print(list(result))
```

```
<map object at 0x7fe88d3bbc10>
<class 'map'>
[2, 3, 4, 5, 6]
```

```
from sys import getsizeof
print(f'The size of map object in memory is {getsizeof(result)} bytes')
print(f'Convert it into list: {getsizeof(list(result))} bytes')
```

```
The size of map object in memory is 48 bytes
Convert it into list: 56 bytes
```

The requirement of object to passed in map() function is iterable so as long as the object has attribute of **iter** it works, **not only list, but also tuple**, such as:

```
numbers = (1, 2, 3, 4, 5)
print(f"Is tuple numbers iterable? Answer: {hasattr(numbers, '__iter__')}")

result = map(add_one, numbers)
print(result)
print(type(result))
print(tuple(result))
```

```
Is tuple numbers iterable? Answer: True
<map object at 0x7fe891eece50>
<class 'map'>
(2, 3, 4, 5, 6)
```

```
numbers = (1, 2, 3, 4, 5)
result = map(lambda x: x + 1, numbers)
print(tuple(result))
```

```
(2, 3, 4, 5, 6)
```

```
# list of strings
words = ['Singapore', 'Guangzhou', 'Tokyo']

# convert every elements in the array into List
converted = list(map(list, words))
print(converted)
print(f"The type of converted: {type(converted)}")
print(f"The lenght of converted: {len(converted)}")
```

```
[['S', 'i', 'n', 'g', 'a', 'p', 'o', 'r', 'e'], ['G', 'u', 'a', 'n', 'g', 'z', 'h', 'o', 'u'], ['T', 'o', 'k', 'y', 'o']]
The type of converted: <class 'list'>
The lenght of converted: 3
```

can be avoided by the following way

```
[ ]  numbers = [3, '23', 42]
     print(list(map(float, numbers)))
```

```
[3.0, 23.0, 42.0]
```

```
[ ]  university = [{'name': 'NYU',
                    'city': 'New York'},
                   {'name': 'NUS',
                    'city': "Singapore"}]

     names = list(map(lambda x: x['name'], university))
     print(names)
```

```
['NYU', 'NUS']
```

```
[ ]  # filter vowel
     def func(variable):
         letters = ['a', 'e', 'i', 'o', 'u']
         if (variable.lower() in letters):
             return True
         else:
             return False

     # given sequance
     sequance = ['I', 'l', 'o', 'v', 'e', 'p', 'y', 't', 'h', 'o', 'n']

     filtered = list(filter(func, sequance))
     print(f"The vowel in the sequance is {filtered}")
```

```
The vowel in the sequance is ['I', 'o', 'e', 'o']
```

```
[ ]  # odd number
     odd_number = filter(lambda x: x % 2, numbers)
     print(f"The odd number is {list(odd_number)}.")

     # even number
     even_number = filter(lambda x: x % 2 == 0, numbers)
     print(f"The even number is {list(even_number)}.")

     # positive number
     positive_number = filter(lambda x: x > 0, numbers)
     print(f"The positive number is {list(positive_number)}.")
```

```
The odd number is [1, -3, 5, 9].
The even number is [-20, 0, 12].
The positive number is [1, 5, 9, 12].
```

```
[ ]  university = [{'name': 'NYU',
                    'city': 'New York'},
                   {'name': 'NUS',
                    'city': "Singapore"}]

     names = list(filter(lambda x: x['name'] == 'NUS', university))
     print(names)
```

```
[{'name': 'NUS', 'city': 'Singapore'}]
```

Result:

Thus, the maps and filters are successfully implemented using python.

Exp no: 7	Implementation of List Comprehension

Aim:

To implement the various list comprehension techniques in python.

Theory:

A Python list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element in the Python list. Python List comprehension provides a much shorter syntax for creating a new list based on the values of an existing list.

Advantages of List Comprehension:

- More time-efficient and space-efficient than loops.
- Require fewer lines of code.
- Transforms iterative statement into a formula.

Program:

```
rv = """Once upon a midnight dreary, while I pondered, weak and weary,
   Over many a quaint and curious volume of forgotten lore,
   While I nodded, nearly napping, suddenly there came a tapping,
   As of some one gently rapping, rapping at my chamber door.
   'Tis some visitor, I muttered, tapping at my chamber door;
   Only this and nothing more."""

# Write your code here!
count = 0
for char in rv:
   count += 1
num_chars = count
print(num_chars)

nums = [4, 2, 8, 23.4, 8, 9, 545, 9, 1, 234.001, 5, 49, 8, 9 , 34, 52, 1, -2, 9.1, 4]
nums = nums[1:4] + nums[5:]
```

```
print(nums)

julia = ("Julia", "Roger", 1967, "Duplicity", 2009, "Actress", "Atlantic")
print(julia[2])
print(julia[2:6])

print(len(julia))

julia = julia[:3] + ("Eat pray Love", 2010) + julia[5:]
print(julia)
```

NESTED DATA

```
nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
print(nested1[0])
print(len(nested1))
nested1.append(['i'])
print("-------")
for L in nested1:
    print(L)



nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
y = nested1[1]
print(y)
print(y[0])

print([10, 20, 30][1])
print(nested1[1][0])
```

```
nested2 = [{'a': 1, 'b': 3}, {'a': 5, 'c': 90, 5: 50}, {'b': 3, 'c': "yes"}]

#write code to print the value associated with key 'c' in the second dictionary (90)

print(nested2[1]['c'])

#write code to print the value associated with key 'b' in the third dictionary

print(nested2[2]['b'])

#add a fourth dictionary add the end of the list; print something to check your work.

nested2.append({'d':5, 'e':9,'r':56})
print(nested2)

#change the value associated with 'c' in the third dictionary from "yes" to "no"; print
something to check your work

nested2[2]['c'] = 'no'
print(nested2)



def square(x):
    return x*x

L = [square, abs, lambda x: x+1]

print("****names****")
for f in L:
    print(f)
```

```
print("****call each of them****")
for f in L:
    print(f(-2))


print("****just the first one in the list****")
print(L[0])
print(L[0](3))



animals = [['cat', 'dog', 'mouse'], ['horse', 'cow', 'goat'], ['cheetah', 'giraffe', 'rhino']]


idx1 = animals[1][0]


data = ['bagel', 'cream cheese', 'breakfast', 'grits', 'eggs', 'bacon', [34, 9, 73, []], [['willow',
'birch', 'elm'], 'apple', 'peach', 'cherry']]


plant = data[7][0][0]
print(plant)


d = {'key1': {'a': 5, 'c': 90, 5: 50}, 'key2':{'b': 3, 'c': "yes"}}



d[5] = {1:2,3:4}



d['key1']['d'] = d['key2']
print(d)



info = {'personal_data':
```

```
        {'name': 'Lauren',
         'age': 20,
         'major': 'Information Science',
         'physical_features':
            {'color': {'eye': 'blue',
                       'hair': 'brown'},
             'height': "5'8"}
        },
       'other':
        {'favorite_colors': ['purple', 'green', 'blue'],
         'interested_in': ['social media', 'intellectual property', 'copyright', 'music', 'books']
        }
      }
color = info['personal_data']['physical_features']['color']
print(color)
```

NESTED ITERATIONS

```
nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
for x in nested1:
    print("level1: ")
    for y in x:
        print("     level2: " + y)


def count_leaves(n):
    count = 0
    for L in n:
        count += 1
```

```
    return count


info = [['Tina', 'Turner', 1939, 'singer'], ['Matt', 'Damon', 1970, 'actor'],
        ['Kristen', 'Wiig', 1973, 'comedian'], ['Michael', 'Phelps', 1985, 'swimmer'],
        ['Barack', 'Obama', 1961, 'president']]

last_names = []
for lst in info:
    last_names.append(lst[1])
last_names


# 3. Below, we have provided a list of lists named L. Use nested iteration to save every
# string containing “b” into a new list named b_strings.


L = [['apples', 'bananas', 'oranges', 'blueberries', 'lemons'],
     ['carrots', 'peas', 'cucumbers', 'green beans'],
     ['root beer', 'smoothies', 'cranberry juice']]

b_strings = []
for lst in L:
    for word in lst:
        if 'b' in word:
            b_strings.append(word)
b_strings
```

STRUCTURING NESTED DATA

```
nested1 = [1, 2, ['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
for x in nested1:
    print("level1: ")
    for y in x:
        print("     level2: {}".format(y))


# We can solve this with special casing, a conditional that checks the type.

nested1 = [1, 2, ['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
for x in nested1:
    print("level1: ")
    if type(x) is list:
        for y in x:
            print("     level2: {}".format(y))
    else:
        print(x)
```

DEEP AND SHALLOW COPIES

```
original = [['dogs', 'puppies'], ['cats', "kittens"]]
copied_version = original[:]
print(copied_version)
print(copied_version is original)
print(copied_version == original)
original[0].append(["canines"])
print(original)
print("-------- Now look at the copied version ----------")
print(copied_version)
```

```
original = [['dogs', 'puppies'], ['cats', "kittens"]]
copied_outer_list = []
for inner_list in original:
    copied_inner_list = []
    for item in inner_list:
        copied_inner_list.append(item)
    copied_outer_list.append(copied_inner_list)
print(copied_outer_list)
original[0].append(["canines"])
print(original)
print("-------- Now look at the copied version -----------")
print(copied_outer_list)

# Or , equivalently, you could take advantage of the slice operator to do the copying
# of the inner list.

original = [['dogs', 'puppies'], ['cats', "kittens"]]
copied_outer_list = []
for inner_list in original:
    copied_inner_list = inner_list[:]
    copied_outer_list.append(copied_inner_list)
print(copied_outer_list)
original[0].append(["canines"])
print(original)
print("-------- Now look at the copied version -----------")
print(copied_outer_list)
```

```
import copy
x = [0, 1, [2, 3]]
x_assign = x
x_copy = x.copy()
x_deepcopy = copy.deepcopy(x)
x[1] = 100
x[2][0] = 200
print(x)
print(x_assign)
print(x_copy)
print(x_deepcopy)

def square(x):
    return x*x

L = [square, abs, lambda x: x+1]

print("****names****")
for f in L:
    print(f)

print("****call each of them****")
for f in L:
    print(f(-2))

print("****just the first one in the list****")
print(L[0])
print(L[0](3))
```

Output:

```
rv = """Once upon a midnight dreary, while I pondered, weak and weary,
    Over many a quaint and curious volume of forgotten lore,
    While I nodded, nearly napping, suddenly there came a tapping,
    As of some one gently rapping, rapping at my chamber door.
    'Tis some visitor, I muttered, tapping at my chamber door;
    Only this and nothing more."""

# Write your code here!
count = 0
for char in rv:
    count += 1
num_chars = count
print(num_chars)
```

```
348
```

```
nums = [4, 2, 8, 23.4, 8, 9, 545, 9, 1, 234.001, 5, 49, 8, 9 , 34, 52, 1, -2, 9.1, 4]
nums = nums[1:4] + nums[5:]
print(nums)
```

```
[2, 8, 23.4, 9, 545, 9, 1, 234.001, 5, 49, 8, 9, 34, 52, 1, -2, 9.1, 4]
```

```
julia = ("Julia", "Roger", 1967, "Duplicity", 2009, "Actress", "Atlantic")
print(julia[2])
print(julia[2:6])

print(len(julia))

julia = julia[:3] + ("Eat pray Love", 2010) + julia[5:]
print(julia)
```

```
1967
(1967, 'Duplicity', 2009, 'Actress')
7
('Julia', 'Roger', 1967, 'Eat pray Love', 2010, 'Actress', 'Atlantic')
```

NESTED DATA

```
nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
print(nested1[0])
print(len(nested1))
nested1.append(['i'])
print("-------")
for L in nested1:
    print(L)
```

```
['a', 'b', 'c']
3
-------
['a', 'b', 'c']
['d', 'e']
['f', 'g', 'h']
['i']
```

```
nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
y = nested1[1]
print(y)
print(y[0])

print([10, 20, 30][1])
print(nested1[1][0])
```

```
['d', 'e']
d
20
d
```

```
nested2 = [{'a': 1, 'b': 3}, {'a': 5, 'c': 90, 5: 50}, {'b': 3, 'c': "yes"}]

#write code to print the value associated with key 'c' in the second dictionary (90)

print(nested2[1]['c'])

#write code to print the value associated with key 'b' in the third dictionary
```

```
print(nested2[2]['b'])

#add a fourth dictionary add the end of the list; print something to check your work.

nested2.append({'d':5, 'e':9,'r':56})
print(nested2)

#change the value associated with 'c' in the third dictionary from "yes" to "no"; print something to check your work

nested2[2]['c'] = 'no'
print(nested2)
```

```
90
3
[{'a': 1, 'b': 3}, {'a': 5, 'c': 90, 5: 50}, {'b': 3, 'c': 'yes'}, {'d': 5, 'e': 9, 'r': 56}]
[{'a': 1, 'b': 3}, {'a': 5, 'c': 90, 5: 50}, {'b': 3, 'c': 'no'}, {'d': 5, 'e': 9, 'r': 56}]
```

```
def square(x):
    return x*x

L = [square, abs, lambda x: x+1]

print("****names****")
for f in L:
    print(f)

print("****call each of them****")
for f in L:
    print(f(-2))

print("****just the first one in the list****")
print(L[0])
print(L[0](3))
```

```
****names****
<function square at 0x7f20b8ebae50>
<built-in function abs>
<function <lambda> at 0x7f20b8ebac10>
****call each of them****
4
2
-1
****just the first one in the list****
<function square at 0x7f20b8ebae50>
9
```

```
animals = [['cat', 'dog', 'mouse'], ['horse', 'cow', 'goat'], ['cheetah', 'giraffe', 'rhino']]

idx1 = animals[1][0]


data = ['bagel', 'cream cheese', 'breakfast', 'grits', 'eggs', 'bacon', [34, 9, 73, []], [['willow', 'birch', 'elm'], 'apple', 'peach',

plant = data[7][0][0]
print(plant)
```

```
willow
```

```
d = {'key1': {'a': 5, 'c': 90, 5: 50}, 'key2':{'b': 3, 'c': "yes"}}




d[5] = {1:2,3:4}



d['key1']['d'] = d['key2']
print(d)
```

```
{'key1': {'a': 5, 'c': 90, 5: 50, 'd': {'b': 3, 'c': 'yes'}}, 'key2': {'b': 3, 'c': 'yes'}, 5: {1: 2, 3: 4}}
```

```
info = {'personal_data':
         {'name': 'Lauren',
          'age': 20,
          'major': 'Information Science',
```

```
            'physical_features':
               {'color': {'eye': 'blue',
                          'hair': 'brown'},
                'height': "5'8"}
            },
          'other':
            {'favorite_colors': ['purple', 'green', 'blue'],
             'interested_in': ['social media', 'intellectual property', 'copyright', 'music', 'books']
            }
        }
color = info['personal_data']['physical_features']['color']
print(color)
```

```
{'eye': 'blue', 'hair': 'brown'}
```

NESTED ITERATIONS

```
nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
for x in nested1:
    print("level1: ")
    for y in x:
        print("     level2: " + y)
```

```
level1:
     level2: a
     level2: b
     level2: c
level1:
     level2: d
     level2: e
level1:
     level2: f
     level2: g
     level2: h
```

```
def count_leaves(n):
    count = 0
    for L in n:
        count += 1
    return count
```

```
info = [['Tina', 'Turner', 1939, 'singer'], ['Matt', 'Damon', 1970, 'actor'],
        ['Kristen', 'Wiig', 1973, 'comedian'], ['Michael', 'Phelps', 1985, 'swimmer'],
        ['Barack', 'Obama', 1961, 'president']]

last_names = []
for lst in info:
    last_names.append(lst[1])
last_names
```

```
['Turner', 'Damon', 'Wiig', 'Phelps', 'Obama']
```

```
# 3. Below, we have provided a list of lists named L. Use nested iteration to save every
# string containing “b” into a new list named b_strings.
```

```
L = [['apples', 'bananas', 'oranges', 'blueberries', 'lemons'],
     ['carrots', 'peas', 'cucumbers', 'green beans'],
     ['root beer', 'smoothies', 'cranberry juice']]

b_strings = []
for lst in L:
    for word in lst:
        if 'b' in word:
            b_strings.append(word)
b_strings
```

```
['bananas',
 'blueberries',
 'cucumbers',
 'green beans',
 'root beer',
 'cranberry juice']
```

STRUCTURING NESTED DATA

```
nested1 = [1, 2, ['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
for x in nested1:
    print("level1: ")
    for y in x:
        print("     level2: {}".format(y))
```

```
level1:
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-78cadb62132a> in <module>
      2 for x in nested1:
      3     print("level1: ")
----> 4     for y in x:
      5         print("     level2: {}".format(y))

TypeError: 'int' object is not iterable

SEARCH STACK OVERFLOW
```

```
# We can solve this with special casing, a conditional that checks the type.

nested1 = [1, 2, ['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
for x in nested1:
    print("level1: ")
    if type(x) is list:
        for y in x:
            print("     level2: {}".format(y))
    else:
        print(x)
```

```
level1:
1
level1:
2
level1:
     level2: a
     level2: b
     level2: c
level1:
     level2: d
     level2: e
level1:
     level2: f
     level2: g
     level2: h
```

DEEP AND SHALLOW COPIES

```
original = [['dogs', 'puppies'], ['cats', "kittens"]]
copied_version = original[:]
print(copied_version)
print(copied_version is original)
print(copied_version == original)
original[0].append(["canines"])
print(original)
print("-------- Now look at the copied version -----------")
print(copied_version)
```

```
[['dogs', 'puppies'], ['cats', 'kittens']]
False
True
[['dogs', 'puppies', ['canines']], ['cats', 'kittens']]
-------- Now look at the copied version -----------
[['dogs', 'puppies', ['canines']], ['cats', 'kittens']]
```

```
original = [['dogs', 'puppies'], ['cats', "kittens"]]
copied_outer_list = []
for inner_list in original:
    copied_inner_list = []
    for item in inner_list:
        copied_inner_list.append(item)
    copied_outer_list.append(copied_inner_list)
print(copied_outer_list)
original[0].append(["canines"])
print(original)
```

```
print("-------- Now look at the copied version -----------")
print(copied_outer_list)
```

```
[['dogs', 'puppies'], ['cats', 'kittens']]
[['dogs', 'puppies', ['canines']], ['cats', 'kittens']]
-------- Now look at the copied version -----------
[['dogs', 'puppies'], ['cats', 'kittens']]
```

```
# Or , equivalently, you could take advantage of the slice operator to do the copying
# of the inner list.

original = [['dogs', 'puppies'], ['cats', "kittens"]]
copied_outer_list = []
for inner_list in original:
    copied_inner_list = inner_list[:]
    copied_outer_list.append(copied_inner_list)
print(copied_outer_list)
original[0].append(["canines"])
print(original)
print("-------- Now look at the copied version -----------")
print(copied_outer_list)
```

```
[['dogs', 'puppies'], ['cats', 'kittens']]
[['dogs', 'puppies', ['canines']], ['cats', 'kittens']]
-------- Now look at the copied version -----------
[['dogs', 'puppies'], ['cats', 'kittens']]
```

```
import copy
x = [0, 1, [2, 3]]
x_assign = x
x_copy = x.copy()
x_deepcopy = copy.deepcopy(x)
x[1] = 100
x[2][0] = 200
print(x)
print(x_assign)
print(x_copy)
print(x_deepcopy)
```

```
[0, 100, [200, 3]]
[0, 100, [200, 3]]
[0, 1, [200, 3]]
[0, 1, [2, 3]]
```

```
def square(x):
    return x*x

L = [square, abs, lambda x: x+1]

print("****names****")
for f in L:
    print(f)

print("****call each of them****")
for f in L:
    print(f(-2))

print("****just the first one in the list****")
print(L[0])
print(L[0](3))
```

```
****names****
<function square at 0x7f20b8eba0d0>
<built-in function abs>
<function <lambda> at 0x7f20b8eba820>
****call each of them****
4
2
-1
****just the first one in the list****
<function square at 0x7f20b8eba0d0>
9
```

Result:

Thus, List Comprehension techniques are successfully executed.

Exp no: 8	File Management Tasks in Hadoop

Aim:

To install and configure Hadoop in windows system

Procedure:

1. Prerequisites

First, we need to make sure that the following prerequisites are installed:

1. Java 8 runtime environment (JRE): Hadoop 3 requires a Java 8 installation. I prefer using the offline installer.

2. Java 8 development Kit (JDK)

3. To unzip downloaded Hadoop binaries, we should install 7zip.

4. I will create a folder "E:\hadoop-env" on my local machine to store downloaded files.

2. Download Hadoop binaries

The first step is to download Hadoop binaries from the official website. The binary package size is about 342 MB.

Figure 1 — Hadoop binaries download link

After finishing the file download, we should unpack the package using 7zip int two steps. First, we should extract the hadoop-3.2.1.tar.gz library, and then, we should unpack the extracted tar file:

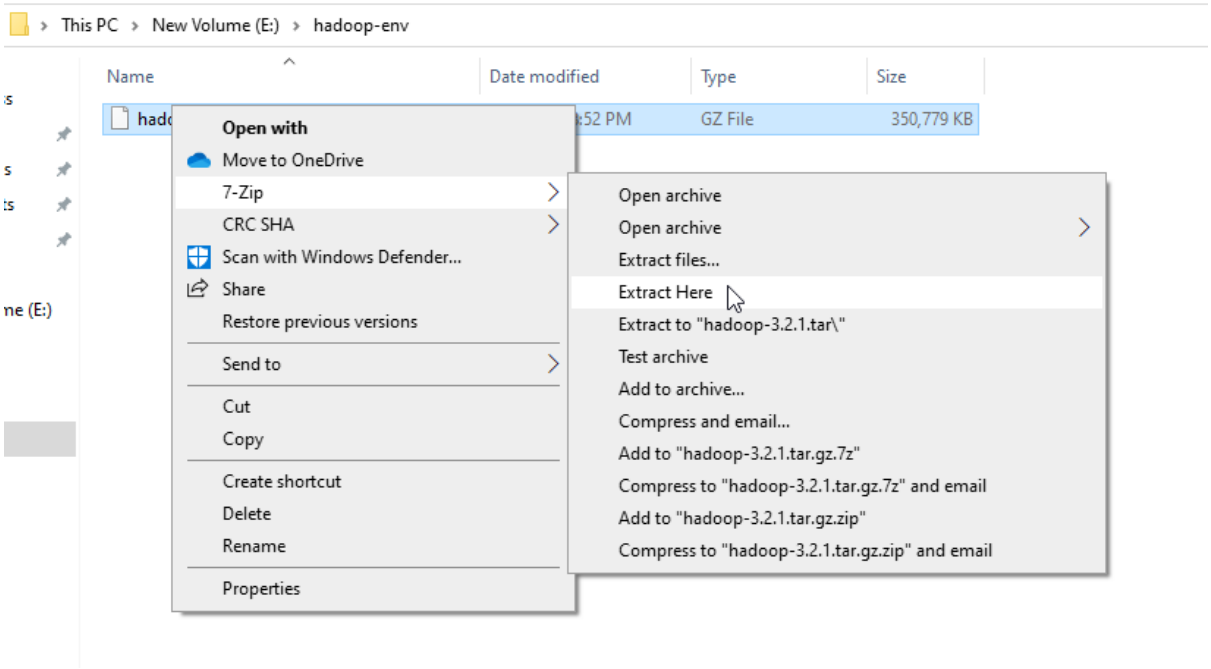


Figure 2 — Extracting hadoop-3.2.1.tar.gz package using 7zip

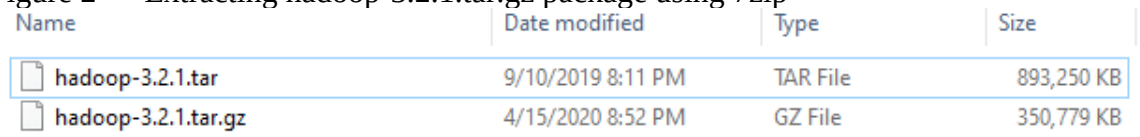


Figure 3 — Extracted hadoop-3.2.1.tar file

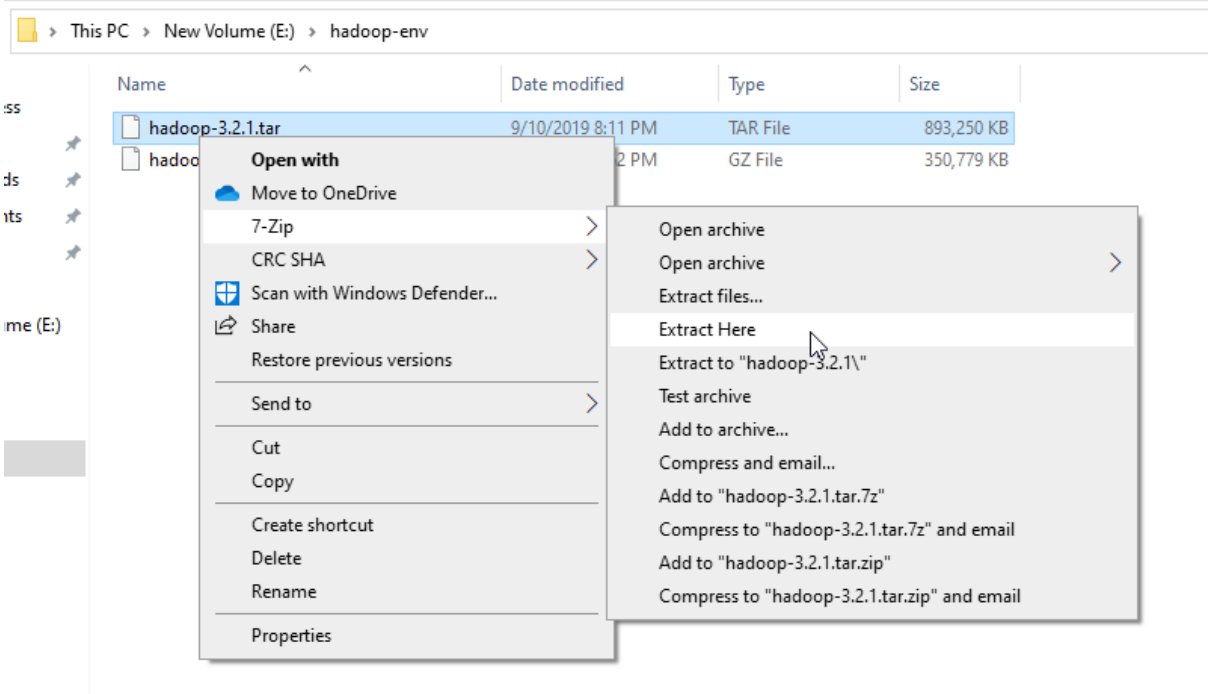


Figure 4 — Extracting the hadoop-3.2.1.tar file

The tar file extraction may take some minutes to finish. In the end, you may see some warnings about symbolic link creation. Just ignore these warnings since they are not related to windows.

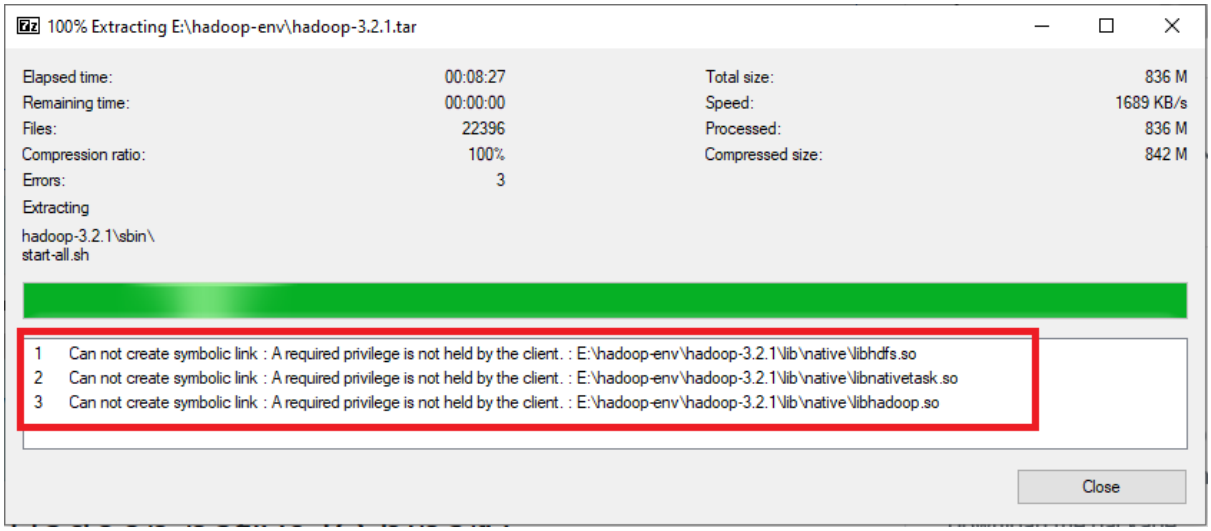


Figure 5 — Symbolic link warnings

After unpacking the package, we should add the Hadoop native IO libraries, which can be found in the following GitHub repository: https://github.com/cdarlint/winutils.

Since we are installing Hadoop 3.2.1, we should download the files located in https://github.com/cdarlint/winutils/tree/master/hadoop-3.2.1/bin and copy them into the "hadoop-3.2.1\bin" directory.

3. Setting up environment variables

After installing Hadoop and its prerequisites, we should configure the environment variables to define Hadoop and Java default paths.
To edit environment variables, go to Control Panel > System and Security > System (or right-click > properties on My Computer icon) and click on the "Advanced system settings" link.

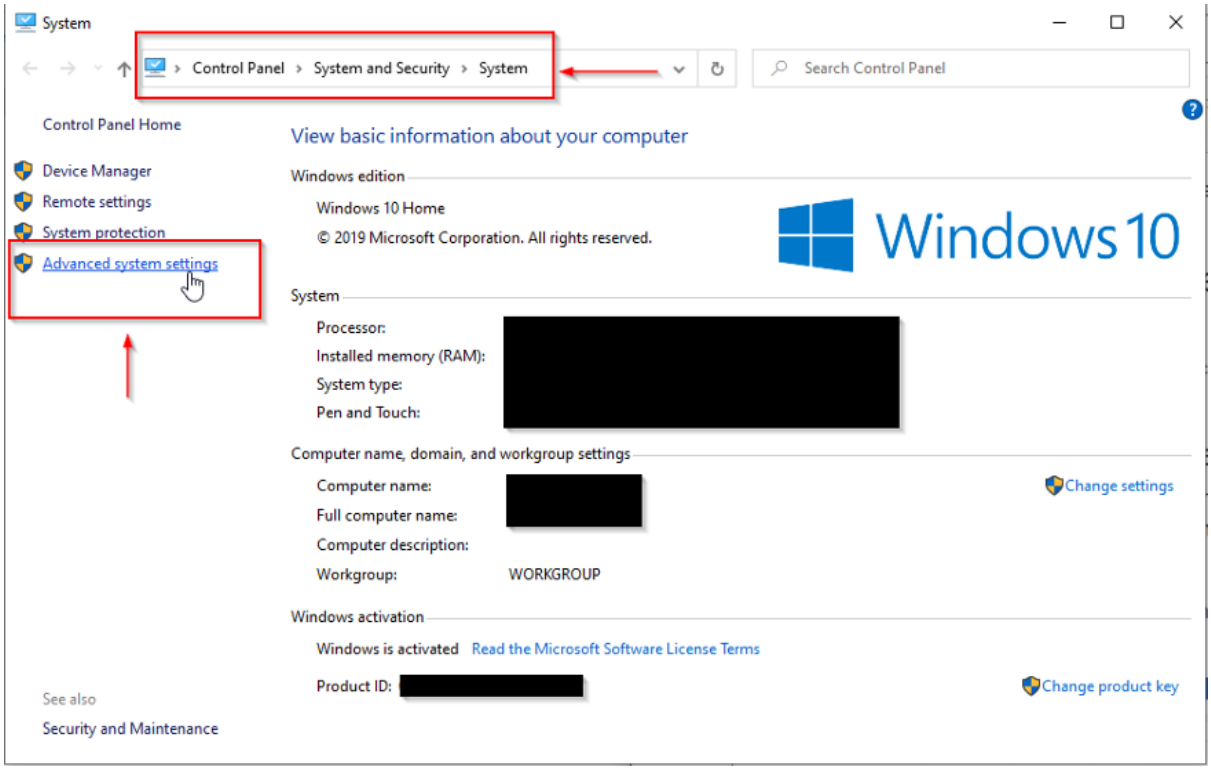


Figure 6 — Opening advanced system settings

When the “Advanced system settings” dialog appears, go to the “Advanced” tab and click on the “Environment variables” button located on the bottom of the dialog.

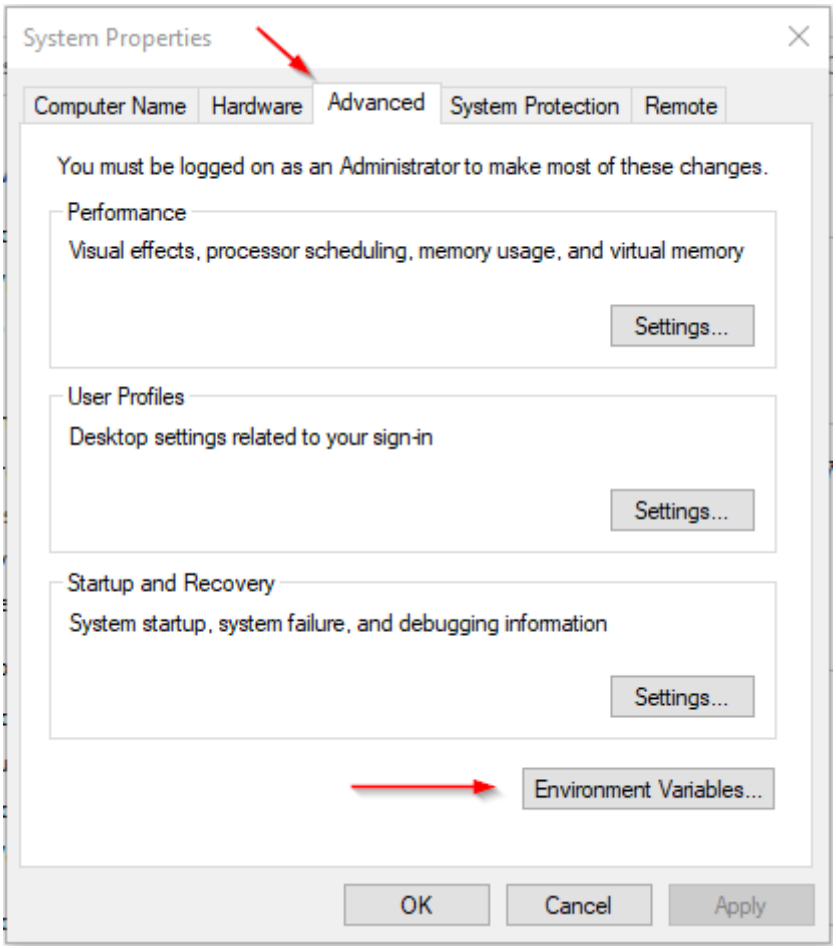


Figure 7 — Advanced system settings dialog

In the “Environment Variables” dialog, press the “New” button to add a new variable.
Note: In this guide, we will add user variables since we are configuring Hadoop for a single user. If you are looking to configure Hadoop for multiple users, you can define System variables instead.
There are two variables to define:

1. JAVA_HOME: JDK installation folder path
2. HADOOP_HOME: Hadoop installation folder path

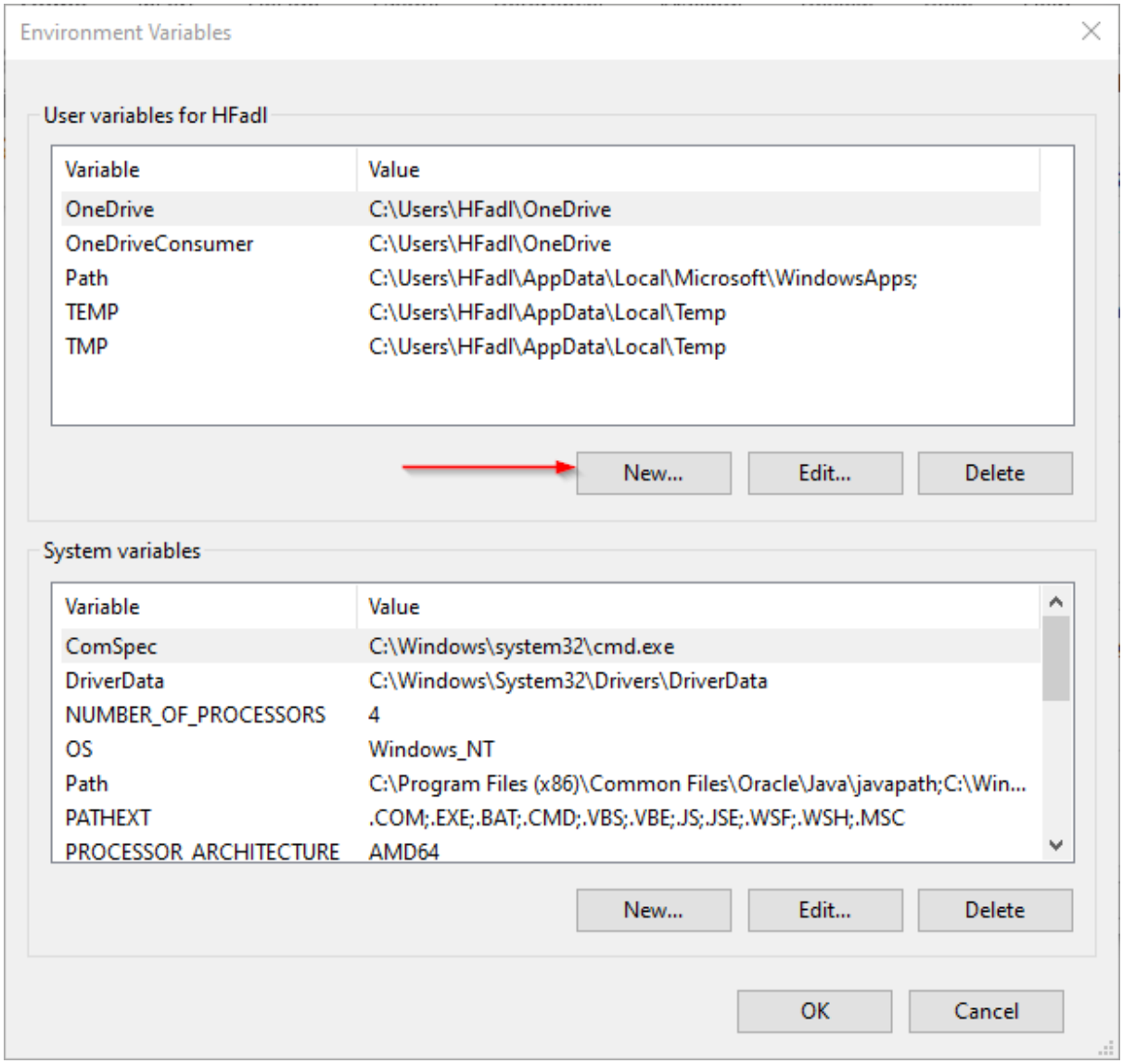


Figure 8 — Adding JAVA_HOME variable

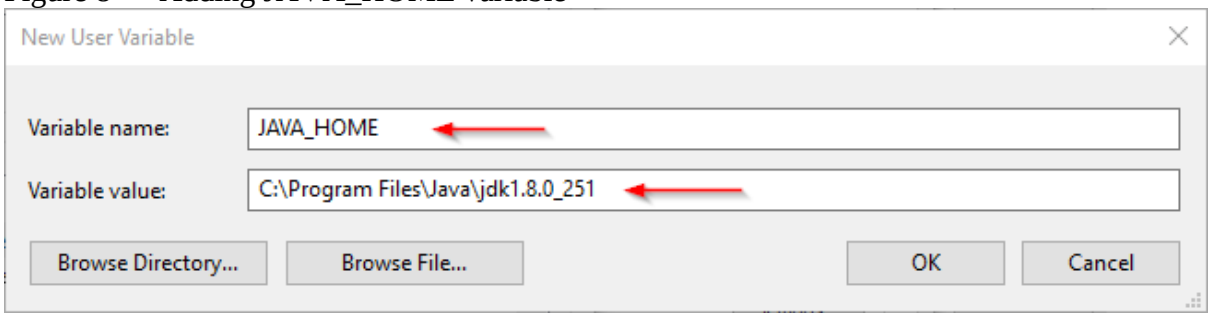


Figure 9 — Adding HADOOP_HOME variable

Now, we should edit the PATH variable to add the Java and Hadoop binaries paths as shown in the following screenshots.

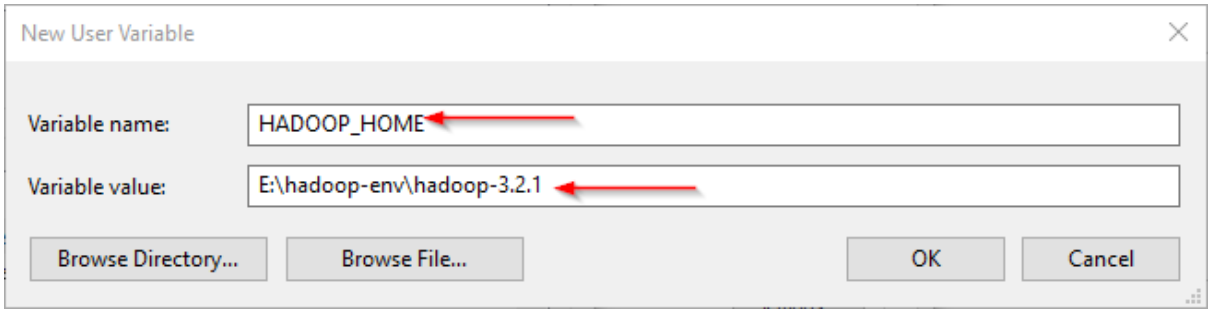


Figure 10 — Editing the PATH variable

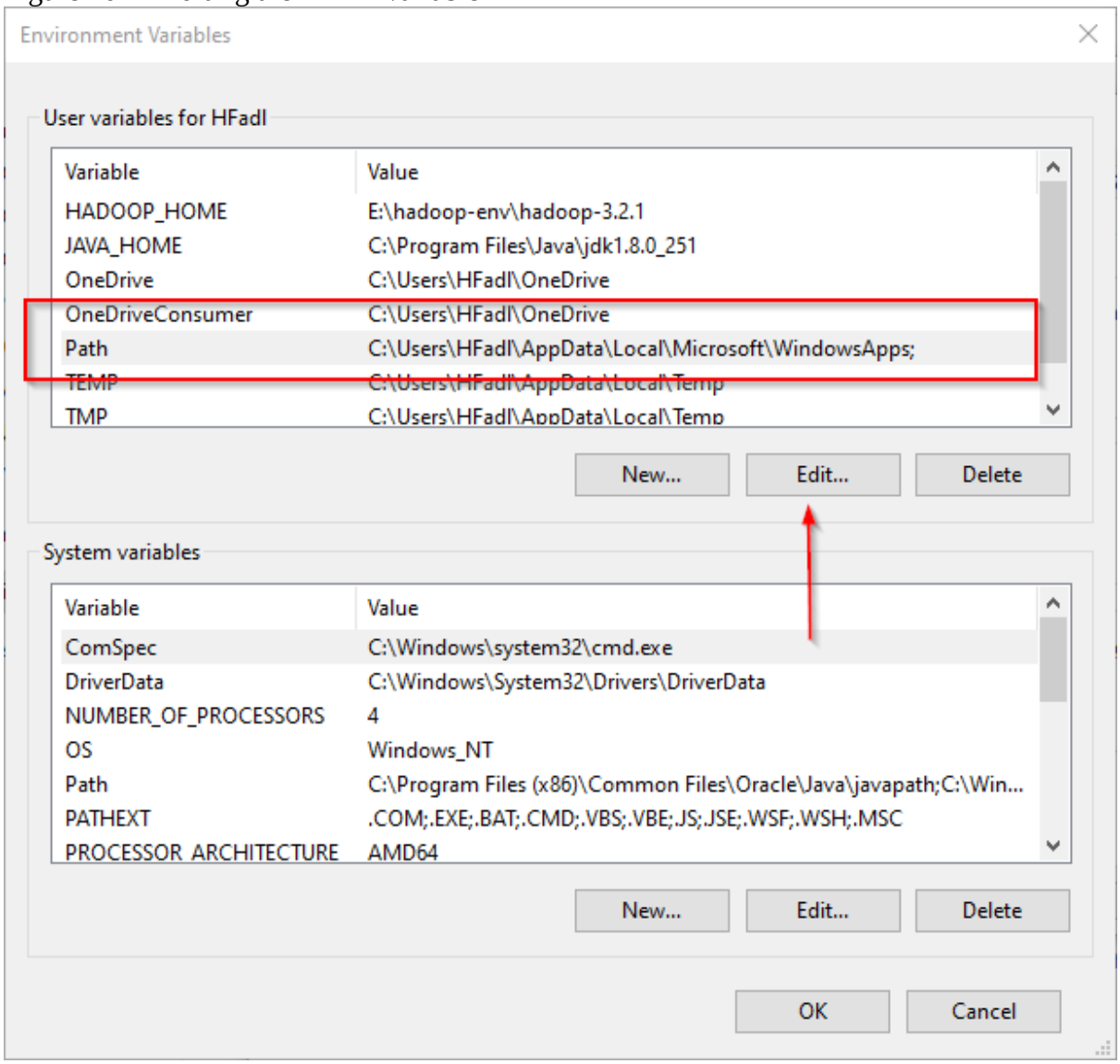


Figure 11 — Editing PATH variable

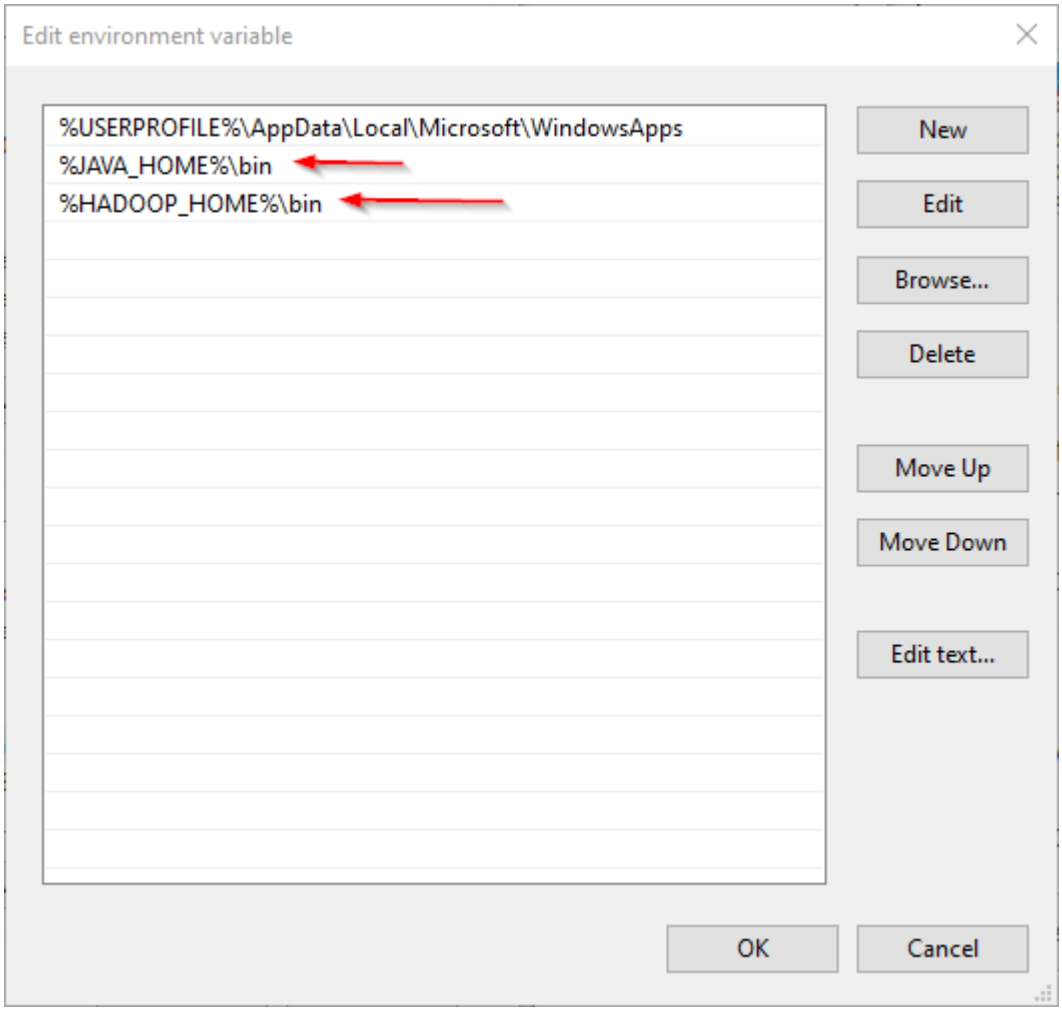


Figure 12— Adding new paths to the PATH variable

3.1. JAVA_HOME is incorrectly set error

Now, let's open PowerShell and try to run the following command:
hadoop -version
In this example, since the JAVA_HOME path contains spaces, I received the following error:
JAVA_HOME is incorrectly set

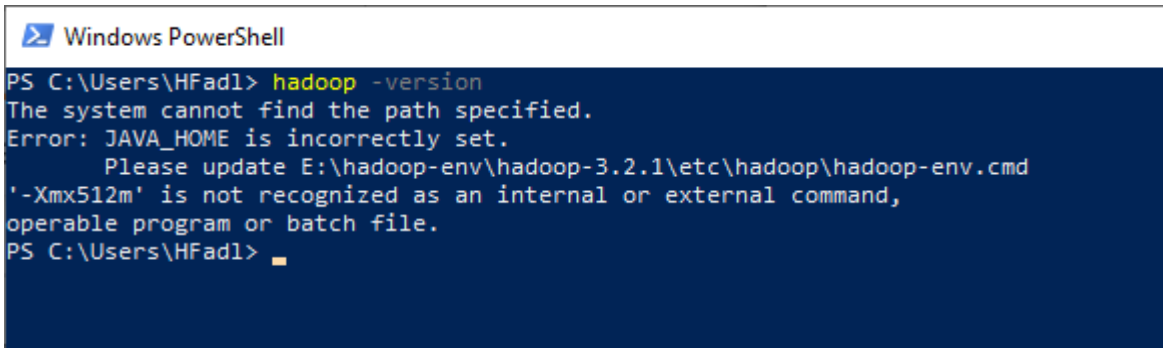


Figure 13 — JAVA_HOME error
To solve this issue, we should use the windows 8.3 path instead. As an example:

- Use "Progra~1" instead of "Program Files"
- Use "Progra~2" instead of "Program Files(x86)"

After replacing “Program Files” with “Progra~1”, we closed and reopened PowerShell and tried the same command. As shown in the screenshot below, it runs without errors.

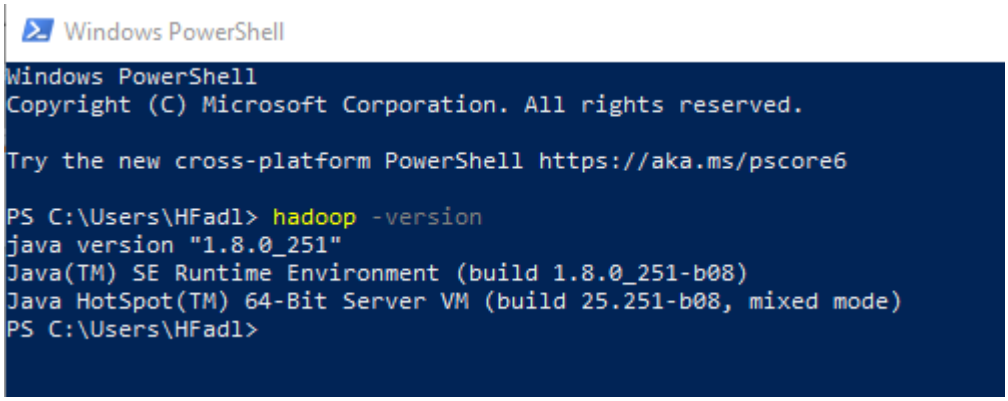


Figure 14 — hadoop -version command executed successfully

4. Configuring Hadoop cluster

There are four files we should alter to configure Hadoop cluster:

1. %HADOOP_HOME%\etc\hadoop\hdfs-site.xml
2. %HADOOP_HOME%\etc\hadoop\core-site.xml
3. %HADOOP_HOME%\etc\hadoop\mapred-site.xml
4. %HADOOP_HOME%\etc\hadoop\yarn-site.xml

4.1. HDFS site configuration

As we know, Hadoop is built using a master-slave paradigm. Before altering the HDFS configuration file, we should create a directory to store all master node (name node) data and another one to store data (data node). In this example, we created the following directories:

- E:\hadoop-env\hadoop-3.2.1\data\dfs\namenode
- E:\hadoop-env\hadoop-3.2.1\data\dfs\datanode

Now, let’s open “hdfs-site.xml” file located in “%HADOOP_HOME%\etc\hadoop” directory, and we should add the following properties within the <configuration></configuration> element:

<property><name>dfs.replication</name><value>1</value></property><property><name>df
s.namenode.name.dir</name><value>file:///E:/hadoop-env/hadoop-
3.2.1/data/dfs/namenode</value></property><property><name>dfs.datanode.data.dir</name>
<value>file:///E:/hadoop-env/hadoop-3.2.1/data/dfs/datanode</value></property>

Note that we have set the replication factor to 1 since we are creating a single node cluster.

4.2. Core site configuration

Now, we should configure the name node URL adding the following XML code into the <configuration></configuration> element within "core-site.xml":

<property><name>fs.default.name</name><value>hdfs://localhost:9820</value></property>

4.3. Map Reduce site configuration

Now, we should add the following XML code into the <configuration></configuration> element within "mapred-site.xml":

<property><name>mapreduce.framework.name</name><value>yarn</value><description>M
apReduce framework name</description></property>

4.4. Yarn site configuration

Now, we should add the following XML code into the <configuration></configuration> element within "yarn-site.xml":

<property><name>yarn.nodemanager.aux-
services</name><value>mapreduce_shuffle</value><description>Yarn Node Manager Aux
Service</description></property>

5. Formatting Name node

After finishing the configuration, let's try to format the name node using the following command:

hdfs namenode -format

Due to a bug in the Hadoop 3.2.1 release, you will receive the following error:

2020–04–17 22:04:01,503 ERROR namenode.NameNode: Failed to start namenode.java.lang.UnsupportedOperationExceptionat java.nio.file.Files.setPosixFilePermissions(Files.java:2044)at org.apache.hadoop.hdfs.server.common.Storage$StorageDirectory.clearDirectory(Storage.java:452)at org.apache.hadoop.hdfs.server.namenode.NNStorage.format(NNStorage.java:591)at org.apache.hadoop.hdfs.server.namenode.NNStorage.format(NNStorage.java:613)at org.apache.hadoop.hdfs.server.namenode.FSImage.format(FSImage.java:188)at org.apache.hadoop.hdfs.server.namenode.NameNode.format(NameNode.java:1206)at org.apache.hadoop.hdfs.server.namenode.NameNode.createNameNode(NameNode.java:1649)at org.apache.hadoop.hdfs.server.namenode.NameNode.main(NameNode.java:1759)2020–04–17 22:04:01,511 INFO util.ExitUtil: Exiting with status 1: java.lang.UnsupportedOperationException2020–04–17 22:04:01,518 INFO namenode.NameNode: SHUTDOWN_MSG:

This issue will be solved within the next release. For now, you can fix it temporarily using the following steps (reference):

1. Download hadoop-hdfs-3.2.1.jar file from the following link.
2. Rename the file name hadoop-hdfs-3.2.1.jar to hadoop-hdfs-3.2.1.bak in folder %HADOOP_HOME%\share\hadoop\hdfs
3. Copy the downloaded hadoop-hdfs-3.2.1.jar to folder %HADOOP_HOME%\share\hadoop\hdfs

Now, if we try to re-execute the format command (Run the command prompt or PowerShell as administrator), you need to approve file system format.

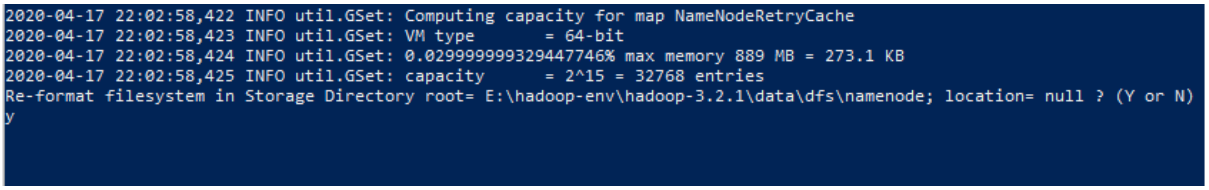


Figure 15 — File system format approval

And the command is executed successfully:

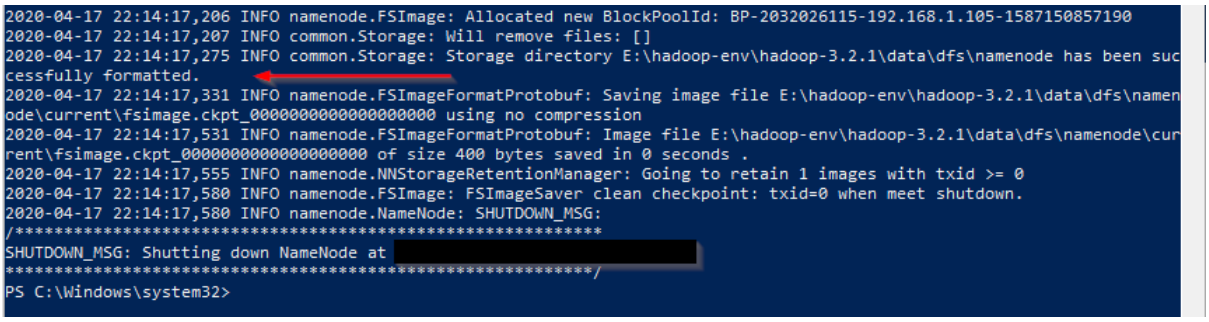


Figure 16 — Command executed successfully

6. Starting Hadoop services

Now, we will open PowerShell, and navigate to “%HADOOP_HOME%\sbin” directory. Then we will run the following command to start the Hadoop no des:

.\start-dfs.cmd

Figure 17 — StartingHadoop nodes

Two command prompt windows will open (one for the name node and one for the data node) as follows:

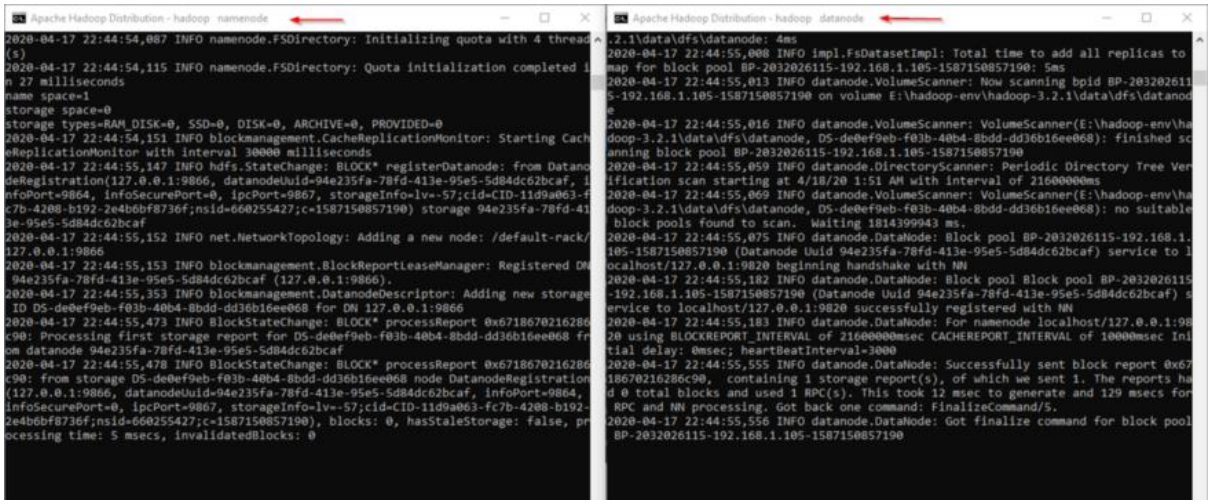


Figure 18 — Hadoop nodes command prompt windows

Next, we must start the Hadoop Yarn service using the following command:

```
./start-yarn.cmd
```

Figure 19 — Starting Hadoop Yarn services

Two command prompt windows will open (one for the resource manager and one for the node manager) as follows:

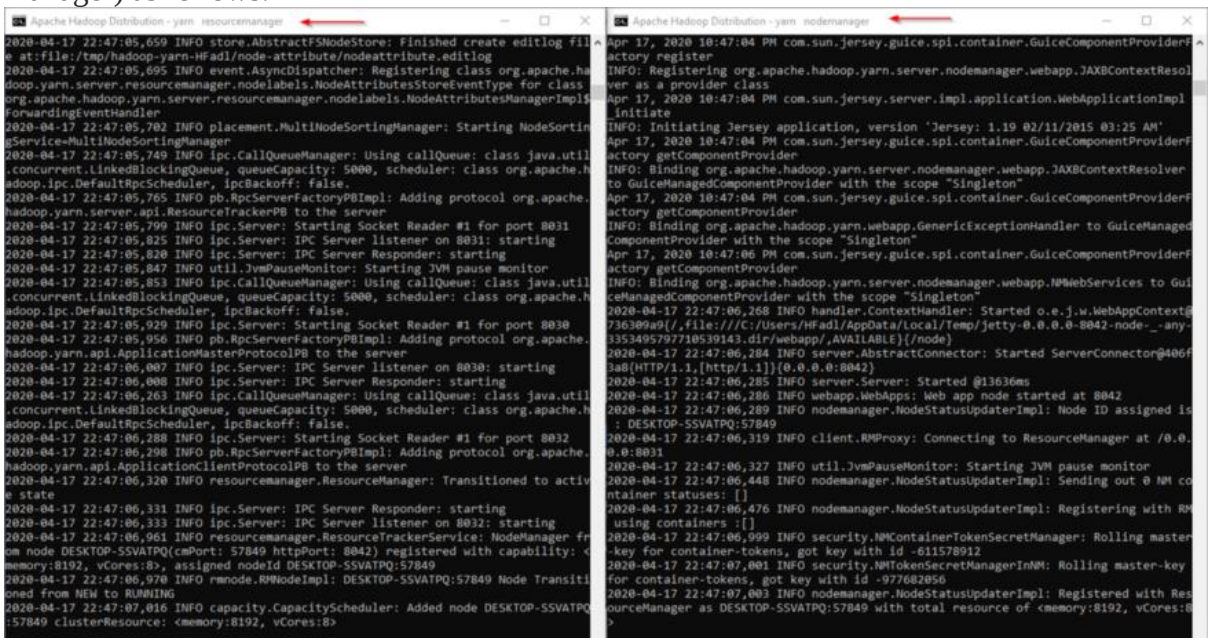


Figure 20— Node manager and Resource manager command prompt windows

To make sure that all services started successfully, we can run the following command:

```
jps
```

It should display the following services:

```
14560 DataNode
4960 ResourceManager
5936 NameNode
768 NodeManager
14636 Jps
```

```
PS E:\hadoop-env\hadoop-3.2.1\sbin> jps
14560 DataNode
4960 ResourceManager
5936 NameNode
768 NodeManager
14636 Jps
PS E:\hadoop-env\hadoop-3.2.1\sbin>
```

Figure 21 — Executing jps command

7. Hadoop Web UI

There are three web user interfaces to be used:

- Name node web page: http://localhost:9870/dfshealth.html

Figure 22 — Name node web page

- Data node web page: http://localhost:9864/datanode.html

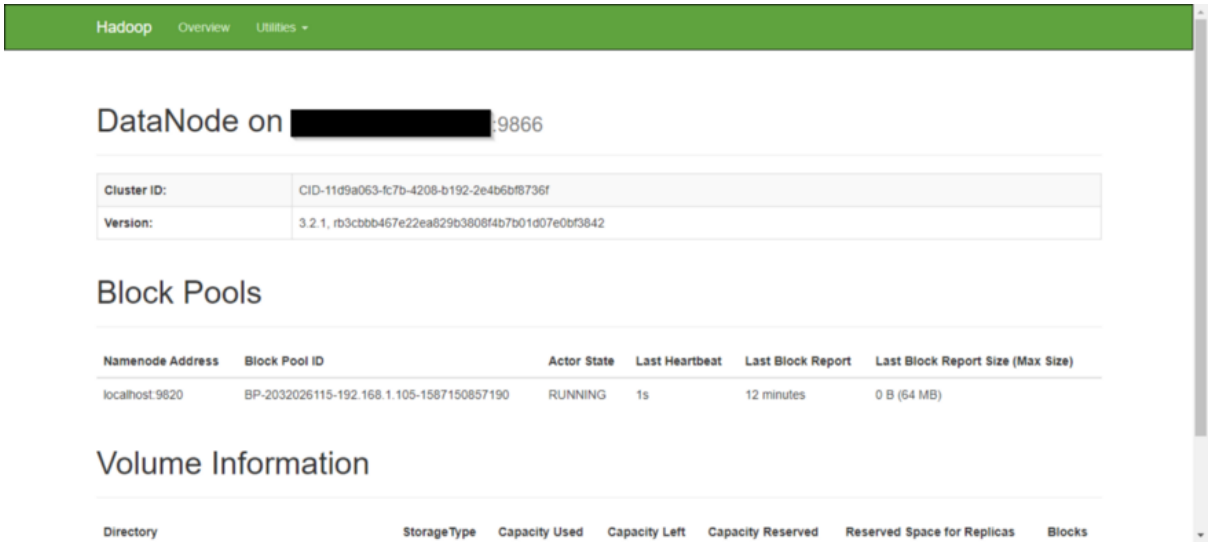


Figure 23 — Data node web page

- Yarn web page: http://localhost:8088/cluster

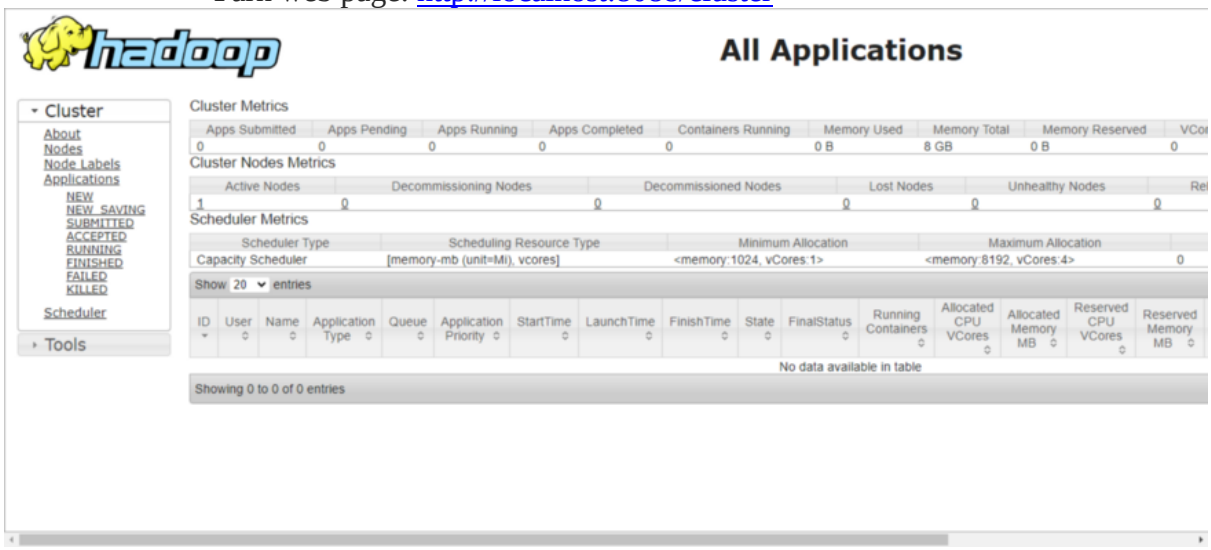


Figure 24 — Yarn web page

Result:

Thus, Hadoop is installed and configured successfully.