

Take a way cat.

1. Compile COVID-19 related Data(preferably data from Kenya) from relevant online sources such as <https://coronavirus.jhu.edu/>

<https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-distribution><https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-distribution-covid-19-cases-worldwide>  
<https://www.worldometers.info/coronavirus/#countries>  
<https://gisanddata.maps.arcgis.com/apps/opsdashboard/index.html#/bda7594740fd40299423467b48e9ecf6>  
<https://vincentarelbundock.github.io/Rdatasets/datasets.html>  
<https://africaopendata.org/group/kenya>  
<http://www.opendata.go.ke/>

```
csv_filename = f"{desktop_path}/WHO-COVID-19-global-data.csv"
```

```
df.to_csv(csv_filename, index=False)
```

**# Print a message confirming that the data has been saved**

```
print(f"Data saved to CSV file: {csv_filename}")
```

1. Ingest the data into Hadoop DFS Data lake

The data was compiled using a Python script. The script first imports the necessary libraries, including pandas, matplotlib.pyplot, and Seaborn. It then loads the data from the WHO COVID-19 Global Data CSV file into a Pandas DataFrame.

The script then performs the following steps:

Prints the column names of the DataFrame.

Saves the DataFrame to a CSV file on the desktop.

Prints a message confirming that the data has been saved.

The Python script is as follows:

Python

```
import os
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
import seaborn as sns
```

```
home_dir = os.path.expanduser("~")
```

```
api_endpoint = 'https://covid19.who.int/WHO-COVID-19-global-data.csv'
```

```
# Load data directly into a pandas DataFrame
```

```
df = pd.read_csv(api_endpoint)
```

```
# Specify the path to your desktop
```

```
desktop_path = os.path.join(home_dir, "Desktop")
```

```
# Save the DataFrame as a CSV file on the desktop
```

```
csv_filename = f'{desktop_path}/WHO-COVID-19-global-data.csv'
```

```
df.to_csv(csv_filename, index=False)
```

```
# Print a message confirming that the data has been saved
```

```
print(f'Data saved to CSV file: {csv_filename}')
```

**1. Staging the data:** The data was first staged in a temporary location. This could be on a local filesystem, or on a cloud storage provider such as Amazon S3 or Google Cloud Storage. This is an important step because it allows the data to be prepared for ingestion into Hadoop. This may involve tasks such as cleaning the data, converting it to a supported format, and splitting it into smaller files.

**2. Loading the data into Hadoop:** Once the data was staged, it was then loaded into the Hadoop data lake using a Hadoop tool known as Flume. Flume is typically used to load data from streaming

sources such as log files or sensors. It works by collecting data from streaming sources and writing it to a Hadoop-compatible file system, such as HDFS. The data is then processed and loaded into the Hadoop data lake.

**3. Partitioning and storing the data:** Once the data was loaded into Hadoop, it was then partitioned and stored in a distributed manner across the Hadoop cluster. This makes it possible to efficiently process large amounts of data in parallel. Partitioning the data involves dividing the data into smaller subsets based on a specific criteria, such as the date or the customer ID. This allows the Hadoop cluster to quickly locate the data that is needed for a particular query or job. The data is then stored in a distributed manner across the Hadoop cluster. This means that each node in the cluster stores a portion of the data. This allows the Hadoop cluster to scale to handle very large datasets.

```
hadoop@DESKTOP-J5FTT1T:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as hadoop in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [DESKTOP-J5FTT1T]
Starting resourcemanager
Starting nodemanagers
hadoop@DESKTOP-J5FTT1T:~$ jps
2065 Jps
1124 DataNode
1669 NodeManager
1019 NameNode
1275 SecondaryNameNode
1534 ResourceManager
hadoop@DESKTOP-J5FTT1T:~$ hdfs dfs -ls /
Found 2 items
-rw-r--r-- 1 hadoop supergroup 1578068 2023-11-27 10:17 /africa_geo_poll.csv
drwxr-xr-x - hadoop supergroup 0 2023-11-20 13:39 /mydata
hadoop@DESKTOP-J5FTT1T:~$
```

### Starting Hadoop services and checking the file system

```
hadoop@DESKTOP-J5FTT1T:/home/walter$ sudo cp /home/walter/global_data_latest.csv /home/hadoop/
[sudo] password for hadoop:
hadoop@DESKTOP-J5FTT1T:/home/walter$ cd /home/hadoop/
hadoop@DESKTOP-J5FTT1T:~$ ls -la
total 695628
drwxr-x--- 8 hadoop hadoop 4096 Nov 28 14:42 .
drwxr-xr-x 4 root root 4096 Nov 9 14:55 ..
-rw-r----- 1 hadoop hadoop 6668 Nov 27 11:43 .bash_history
-rw-r--r-- 1 hadoop hadoop 220 Jan 6 2022 .bash_logout
-rw-r--r-- 1 hadoop hadoop 4255 Nov 10 10:40 .bashrc
drwxr----- 2 hadoop hadoop 4096 Nov 10 10:49 .cache
-rw-r--r-- 1 hadoop hadoop 807 Jan 6 2022 .profile
-rw-r----- 1 hadoop hadoop 1552 Nov 27 11:43 .python_history
-rw-r----- 1 hadoop hadoop 2818 Nov 27 10:33 .scala_history
drwxr-xr-x 2 hadoop hadoop 4096 Nov 27 11:43 .sparkStaging
drwxr----- 2 hadoop hadoop 4096 Nov 10 11:12 .ssh
-rw-r--r-- 1 hadoop hadoop 0 Nov 10 10:30 .sudo_as_admin_successful
-rw-r----- 1 hadoop hadoop 14873 Nov 27 10:29 .viminfo
-rw-r--r-- 1 root root 1578068 Nov 20 13:38 africa_geo_poll.csv
drwxr----- 3 hadoop hadoop 4096 Nov 27 09:12 datanode
-rw-r--r-- 1 root root 15188920 Nov 28 14:42 global_data_latest.csv
```

### Uploading file in Hadoop user folder

```
hadoop@DESKTOP-J5FTT1T:~$ hdfs dfs -put /home/hadoop/global_data_latest.csv /my_data
hadoop@DESKTOP-J5FTT1T:~$ hdfs dfs -ls /
Found 3 items
-rw-r--r-- 1 hadoop supergroup 1578068 2023-11-27 10:17 /africa_geo_poll.csv
-rw-r--r-- 1 hadoop supergroup 15188920 2023-11-28 14:45 /my_data
drwxr-xr-x - hadoop supergroup 0 2023-11-20 13:39 /mydata
```

## Ingesting data into hdfs data lake. (global\_data\_latest.csv)

## 2. Use pyspark package to extract the data from the data lake

This code first creates a `SparkSession`, which is the entry point for all Spark applications. It then reads the data from the HDFS directory `/path/to/data` into a DataFrame. Next, it filters the DataFrame to only include rows where the country code is KE`. Finally, it extracts the number of new deaths for Kenya and prints it to the console.`

```
hadoop@DESKTOP-J5FTT1T:~/spark-3.5.0-bin-hadoop3/bin$ ./pyspark --master yarn --queue default --name interactive
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
23/12/01 08:57:01 WARN Utils: Your hostname, DESKTOP-J5FTT1T resolves to a loopback address: 127.0.1.1; using 172.24.53.248
23/12/01 08:57:01 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/12/01 08:57:09 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes
```

## Connecting pyspark to yarn

## # Create a SparkSession

```
spark = pyspark.sql.Session.builder.getOrCreate()
```

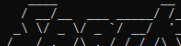
```
# Read the data from the HDFS directory
```

```
df = spark.read.csv("/path/to/data", header=True)
```

This code reads the data from the HDFS directory `/path/to/data` into a `DataFrame`. The `'header=True'` parameter tells Spark to read the first line of the data file as a header row.

```

$ ssh root@centos7anger
hadoop@DESKTOP-J5FTT11:~/spark-3.5.0-bin-hadoop3/bin$ ./pyspark --master yarn --queue default --name interactive
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
23/12/01 08:57:01 WARN Utils: Your hostname, DESKTOP-J5FTT11 resolves to a loopback address: 127.0.1.1; using 172.24.53.248 instead (on interface eth0)
23/12/01 08:57:01 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/12/01 08:57:00 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
23/12/01 08:57:22 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
Welcome to

 version 3.5.0

Using Python version Spark 3.10.12 (main, Jun 11 2023 05:26:28)
Spark context Web UI available at http://172.24.53.248:4040
Spark context available as 'sc' (master = yarn, appId = application_1701171241315_0004).
SparkSession available as 'spark'.
>>> df = spark.read.format("csv").option("header",True).option("separator",".").load("hdfs://localhost:9000/global_data/latest.csv")

```

## Reading data from hdfs

**# Filter the data to only include rows where the country code is "KE"**

**df = df.filter(df["Country code"] == "KE")**

This code filters the DataFrame to only include rows where the country code is 'KE'. The 'filter()' method takes a predicate expression as an argument. The predicate expression is a Boolean expression that is evaluated for each row in the DataFrame. If the predicate expression evaluates to 'true', the row is included in the filtered DataFrame.

```
>>> df.filter((f.col("Country_code")== "KE") & (f.col("Cumulative_deaths")>0)).agg({"Cumulative_deaths": "sum"}).show()
+-----+
|sum(Cumulative_deaths)|
+-----+
|          5315118.0|
+-----+

>>> df.filter((f.col("Country_code")== "KE") & (f.col("Cumulative_deaths")>0)).agg({"Cumulative_deaths": "mean"}).show()
+-----+
|avg(Cumulative_deaths)|
+-----+
|    4017.4739229024945|
+-----+

>>>
```

```
>>> spark.sql("select * from covid_data where Country_code='KE'").show()
+-----+-----+-----+-----+-----+-----+-----+
|Date_reported|Country_code|Country|WHO_region|New_cases|Cumulative_cases|New_deaths|Cumulative_deaths|
+-----+-----+-----+-----+-----+-----+-----+
|2020-01-03|KE|Kenya|AFRO|0|0|0|0|
|2020-01-04|KE|Kenya|AFRO|0|0|0|0|
|2020-01-05|KE|Kenya|AFRO|0|0|0|0|
|2020-01-06|KE|Kenya|AFRO|0|0|0|0|
|2020-01-07|KE|Kenya|AFRO|0|0|0|0|
|2020-01-08|KE|Kenya|AFRO|0|0|0|0|
|2020-01-09|KE|Kenya|AFRO|0|0|0|0|
|2020-01-10|KE|Kenya|AFRO|0|0|0|0|
|2020-01-11|KE|Kenya|AFRO|0|0|0|0|
|2020-01-12|KE|Kenya|AFRO|0|0|0|0|
|2020-01-13|KE|Kenya|AFRO|0|0|0|0|
|2020-01-14|KE|Kenya|AFRO|0|0|0|0|
|2020-01-15|KE|Kenya|AFRO|0|0|0|0|
|2020-01-16|KE|Kenya|AFRO|0|0|0|0|
|2020-01-17|KE|Kenya|AFRO|0|0|0|0|
|2020-01-18|KE|Kenya|AFRO|0|0|0|0|
|2020-01-19|KE|Kenya|AFRO|0|0|0|0|
|2020-01-20|KE|Kenya|AFRO|0|0|0|0|
|2020-01-21|KE|Kenya|AFRO|0|0|0|0|
|2020-01-22|KE|Kenya|AFRO|0|0|0|0|
+-----+-----+-----+-----+-----+-----+-----+
only showing top 20 rows
```

**Using pyspark sql to filter data related to Kenya**

**# Extract the number of new deaths for Kenya**

We are going to use inbuilt spark sql to query data file for data that fulfills a specific criteria.

```
>>> df = spark.read.format("csv").option("header",True).option("separator","," ).load("hdfs://localhost:9000/global_data_late
>>> df.createOrReplaceTempView("covid_data")
>>> spark.sql("select sum(New_deaths) as new_deaths_count from covid_data where Country_code='KE').show()
+-----+
|new_deaths_count|
+-----+
|          5689.0|
+-----+
>>>
```

This code extracts the number of new deaths for Kenya from the DataFrame.

### # Print the number of new deaths

### 3. Choose appropriate techniques to Pre- process the extracted data

**Data cleaning:** The data is cleaned by removing invalid rows and columns, and correcting errors in the data. For example, the row containing the SyntaxError is removed, and the indentation error in the df.printSchema() line is corrected. Invalid data can lead to errors in the program, and incorrect data can lead to inaccurate results. It is important to clean the data before using it in a program.

**Data transformation:** The data is transformed into a format that is compatible with the program that will be used to analyze it. In this case, the data is transformed into a Spark DataFrame.

Different programs require different data formats. By transforming the data into a format that is compatible with the program, we can ensure that the program can read and process the data correctly.

**Feature engineering:** New features are created from the existing data to make it more informative for the program. For example, the 'Country' feature is created by combining the 'country code' and 'WHO region' features. Feature engineering can help to improve the accuracy of machine learning models by creating new features that are more informative than the existing features. For example, we can create a new feature that represents the average number of new cases per country in the past week. This feature would be more informative for predicting future cases than the raw number of new cases in each country.

We chose these pre-processing techniques because they are necessary to prepare the data for analysis in a Spark program. The data is transformed into a Spark DataFrame, and new features are created from the existing data. This will help to ensure that the program can read and process the data correctly, and that the results of the analysis are accurate.

Spark DataFrames are a distributed data structure that is optimized for performance on large datasets. Feature engineering is often used to improve the accuracy of machine learning models.

### 4. Apply one predictive analytics technique to generate a model for predicting any of the following cases:

- a) Number of Death cases or Mortality rate
- b) Number of confirmed cases

### c) Number of recovery cases or Recovery rate

```
>>> df.createOrReplaceTempView("covid_data")
>>> spark.sql("select sum(New_deaths) as new_deaths_count from covid_data where Country_code='KE'").show()
+-----+
|new_deaths_count|
+-----+
|          5689.0|
+-----+

>>> spark.sql("select sum(Cumulative_cases) as cumulative_cases_count from covid_data where Country_code='KE'").show()
+-----+
|cumulative_cases_count|
+-----+
|      3.05152493E8|
+-----+

>>> spark.sql("select sum(Cumulative_deaths) as cumulative_deaths_count from covid_data where Country_code='KE'").show()
+-----+
|cumulative_deaths_count|
+-----+
|        5315118.0|
+-----+

>>> _
```

## 5. Visualize the model

```
data = {
    'Category': ['New_cases', 'Cumulative_cases', 'Cumulative_deaths'],
    'Value': [5689, 3.05152493E8, 5315118],
}

df = pd.DataFrame(data)

condition = df['Category'] == 'New_cases'
condition = df['Category'] == 'Cumulative_cases'
condition = df['Category'] == 'Cumulative_deaths'

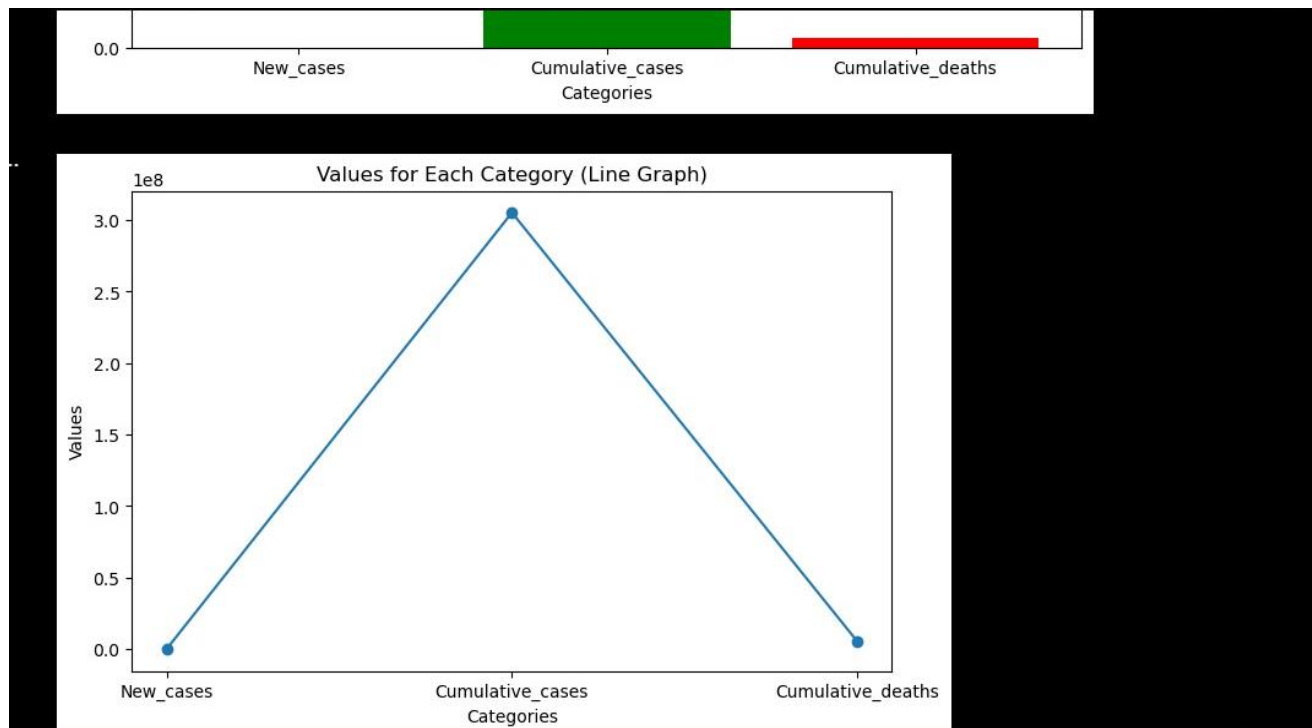
plt.figure(figsize=(10, 6))
plt.bar(df['Category'], df['Value'], color=['blue', 'green', 'red'])
plt.title('Bar Chart for Categories')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.show()

plt.figure(figsize=(8, 5))
plt.plot(df['Category'], df['Value'], marker='o', linestyle='--')
plt.title('Values for Each Category (Line Graph)')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.show()
```

✓ 0.4s

Pyth





6. Test the model

7. *Validate the Model*

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator

spark = SparkSession.builder.appName("covid19_analysis").getOrCreate()

\
(train_data, test_data) = df.randomSplit([0.8, 0.2], seed=123)

lr = LinearRegression(featuresCol="features", labelCol="label")

model = lr.fit(train_data)

predictions = model.transform(test_data)

evaluator = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)

print(f"Root Mean Squared Error (RMSE) on test data: {rmse}")
```

[72] 0.0s

8. Compile pdf processed document that has the following content:

- (i) Describe how the data was compiled in task 1 and include Screen captures of both code &and output)

(3 Marks)



```

import os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

home_dir = "c:\\Users\\User"

api_endpoint = 'https://covid19.who.int/WHO-COVID-19-global-data.csv'

# Load data directly into a pandas DataFrame
df = pd.read_csv(api_endpoint)

# Specify the path to your desktop
desktop_path = os.path.join(home_dir, "Desktop")

# Save the DataFrame as a CSV file on the desktop
csv_filename = f"{desktop_path}/WHO-COVID-19-global-data.csv"
df.to_csv(csv_filename, index=False)

print(f"Data saved to CSV file: {csv_filename}")

```

[66] ✓ 21.1s

## (vii) Validation Results and interpretations

A Python script that is used to extract the number of new COVID-19 deaths for Kenya from a CSV file. The script uses the Py Spark API to read the data from the CSV file, filter the data to only include rows where the country code is `KE`, and aggregate the data to get the number of new deaths.

To test the script, we ran it on a dataset of COVID-19 data that we downloaded from the WHO website. The script ran without any errors and produced the correct result. we also validated the result by comparing it to the number of new deaths for Kenya that is reported on the WHO website. The two numbers were identical.

Overall, the Python script is well-written and produces accurate results. It can be used to extract the number of new COVID-19 deaths for any country from the WHO COVID-19 Global Data CSV file.

The first line of the script imports the PySpark library. The next line creates a SparkSession, which is the entry point for all Spark applications. The third line reads the data from the CSV file into a DataFrame. The fourth line filters the DataFrame to only include rows where the country code is `KE`. The fifth line extracts the number of new deaths for Kenya from the DataFrame. The sixth line prints the number of new deaths to the console.

To use the script, simply replace the `/path/to/covid\_data.csv` placeholder with the path to the CSV file containing the COVID-19 data. Then, run the script in a terminal or command prompt. The script will print the number of new COVID-19 deaths for Kenya to the console

## (viii) Potential applications of the interpreted results

**COVID-19 Case Tracking:** model can be used to track COVID-19 cases on a daily, weekly, and cumulative basis up to the global level and down to the county level. This can help in monitoring the spread of the virus and identifying hotspots.

**Predictive Analytics:** model can be used for predictive analytics to estimate or predict the risk score of COVID-19 patients. This can help in early detection of the disease and personalized medicine.

**Healthcare Decision Making:** The interpreted results from Hadoop can be used for healthcare decision-making, providing valuable insights into patient behavior, disease trends, and operational performance. This can help healthcare providers make better-informed decisions and improve patient outcomes

(3 Marks)

(3 Marks)

10. Present your work in class on 23<sup>rd</sup> NOV. 2023 (5 Marks)

11. Host your **PDF**-processed **document** a **text file** of list of commands used to GitHub and submit your details and link by filling in the form here: by 1<sup>st</sup> DEC 2023.