DATABASE MANAGEMENT SYSTEMS

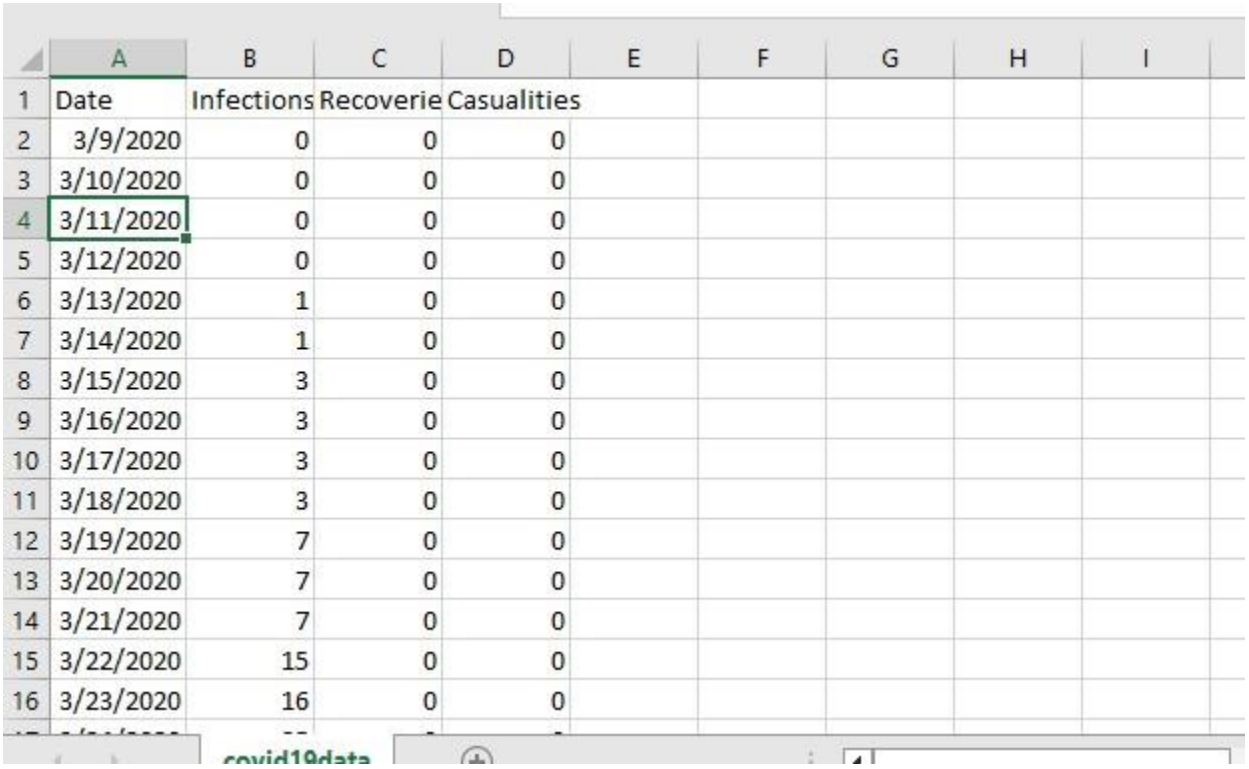
TAKE AWAY CAT

GROUP 3

MEMBER NAMES	**REG.NO**
Jane Njoroge	SCT221-0442/2021
Cecilia Maingi	SCT221-0095/2021
Evan Mwaura	SCT221-0110/2021
Jacinta Ngugi	SCT221-0262/2021
Steve Matara	SCT221-0488/2021

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

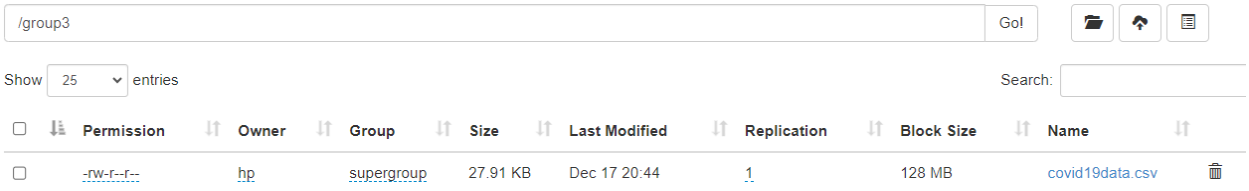
(i) Describe how the data was compiled in task 1 and include Screen captures of both code &and output)



	A	B	C	D
1	Date	Infections	Recoverie	Casualities
2	3/9/2020	0	0	0
3	3/10/2020	0	0	0
4	3/11/2020	0	0	0
5	3/12/2020	0	0	0
6	3/13/2020	1	0	0
7	3/14/2020	1	0	0
8	3/15/2020	3	0	0
9	3/16/2020	3	0	0
10	3/17/2020	3	0	0
11	3/18/2020	3	0	0
12	3/19/2020	7	0	0
13	3/20/2020	7	0	0
14	3/21/2020	7	0	0
15	3/22/2020	15	0	0
16	3/23/2020	16	0	0

Initially, we identified the primary data source where we gathered the information we required. Employing the comprehensive Africa-based COVID-19 dataset, we applied a filter to extract exclusively the data pertaining to Kenya, we then narrowed it to only the columns bearing the essential information for our analysis. This refined dataset which we then complied into a new work book.

(ii) Describe how the data was ingested into Hadoop data lake and include screen shots.




```
C:\hadoop\hadoop-2.9.0\bin>hdfs dfs -mkdir /group3

C:\hadoop\hadoop-2.9.0\bin>hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - hp supergroup          0 2023-12-17 20:42 /group3

C:\hadoop\hadoop-2.9.0\bin>hdfs dfs -put D:\class_units\3.1\AdvDB\TakeAwayCat\covid19data.csv /group3
```

1. Create a new directory named "group3" in the root directory ("/") of HDFS.

hdfs dfs -mkdir /group3

2. Upload a file named "covid19data.csv" from the local file system (in this case, from the path "D:\class_units\3.1\AdvDB\TakeAwayCat") to the HDFS directory "/group3".

hdfs dfs -put D:\class_units\3.1\AdvDB\TakeAwayCat\covid19data.csv /group3

(iii) Describe how data was extracted using pyspark and include associated screen shots

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder.appName("ExtractData").getOrCreate()
>>> data = spark.read.format("csv").option("header","true").load("hdfs://localhost:9000/group3/covid19data.csv")
>>> data.show()
+---------+----------+----------+-----------+
|     Date|Infections|Recoveries|Casualities|
+---------+----------+----------+-----------+
| 3/9/2020|         0|         0|          0|
|3/10/2020|         0|         0|          0|
|3/11/2020|         0|         0|          0|
|3/12/2020|         0|         0|          0|
|3/13/2020|         1|         0|          0|
|3/14/2020|         1|         0|          0|
|3/15/2020|         3|         0|          0|
|3/16/2020|         3|         0|          0|
|3/17/2020|         3|         0|          0|
|3/18/2020|         3|         0|          0|
|3/19/2020|         7|         0|          0|
|3/20/2020|         7|         0|          0|
|3/21/2020|         7|         0|          0|
|3/22/2020|        15|         0|          0|
|3/23/2020|        16|         0|          0|
|3/24/2020|        25|         0|          0|
|3/25/2020|        28|         1|          0|
|3/26/2020|        31|         1|          1|
|3/27/2020|        31|         1|          1|
|3/28/2020|        38|         1|          1|
+---------+----------+----------+-----------+
only showing top 20 rows
```

1. Import the PySpark library, which is a Python library for Apache Spark, a distributed data processing framework.

from pyspark.sql import SparkSession

2. Create a Spark session. SparkSession is the entry point to any Spark functionality. It is used to configure and initiate the functionality of Spark.

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

3. Read data from a CSV file into a Spark DataFrame. In this case, it reads a CSV file located at hdfs://localhost:9000/group3/covid19data.csv

data = spark.read.format("csv").option("header", "true").load("hdfs://localhost:9000/group3/covid19data.csv")

4. Display the contents of the DataFrame using the show() method.

data.show()

(iv) Describe pre-processing tasks/techniques used to prepare the data (include screen shots) and give reason (s) to justify your choices.

```
>>> cleaned_data = data.filter((data["Infections"] != 0) | (data["Recoveries"] != 0) | (data["Casualities"] != 0))
>>> cleaned_data.show()
+---------+----------+----------+-----------+
|     Date|Infections|Recoveries|Casualities|
+---------+----------+----------+-----------+
|3/13/2020|         1|         0|          0|
|3/14/2020|         1|         0|          0|
|3/15/2020|         3|         0|          0|
|3/16/2020|         3|         0|          0|
|3/17/2020|         3|         0|          0|
|3/18/2020|         3|         0|          0|
|3/19/2020|         7|         0|          0|
|3/20/2020|         7|         0|          0|
|3/21/2020|         7|         0|          0|
|3/22/2020|        15|         0|          0|
|3/23/2020|        16|         0|          0|
|3/24/2020|        25|         0|          0|
|3/25/2020|        28|         1|          0|
|3/26/2020|        31|         1|          1|
|3/27/2020|        31|         1|          1|
|3/28/2020|        38|         1|          1|
|3/29/2020|        42|         1|          1|
|3/30/2020|        50|         1|          1|
|3/31/2020|        59|         1|          1|
| 4/1/2020|        81|         3|          1|
+---------+----------+----------+-----------+
only showing top 20 rows
```

1. Filter data in a way that is eliminating zeros. The data is filtered based on the conditions

cleaned_data = data.filter((data["Infections"] != 0) | (data["Recoveries"] != 0)

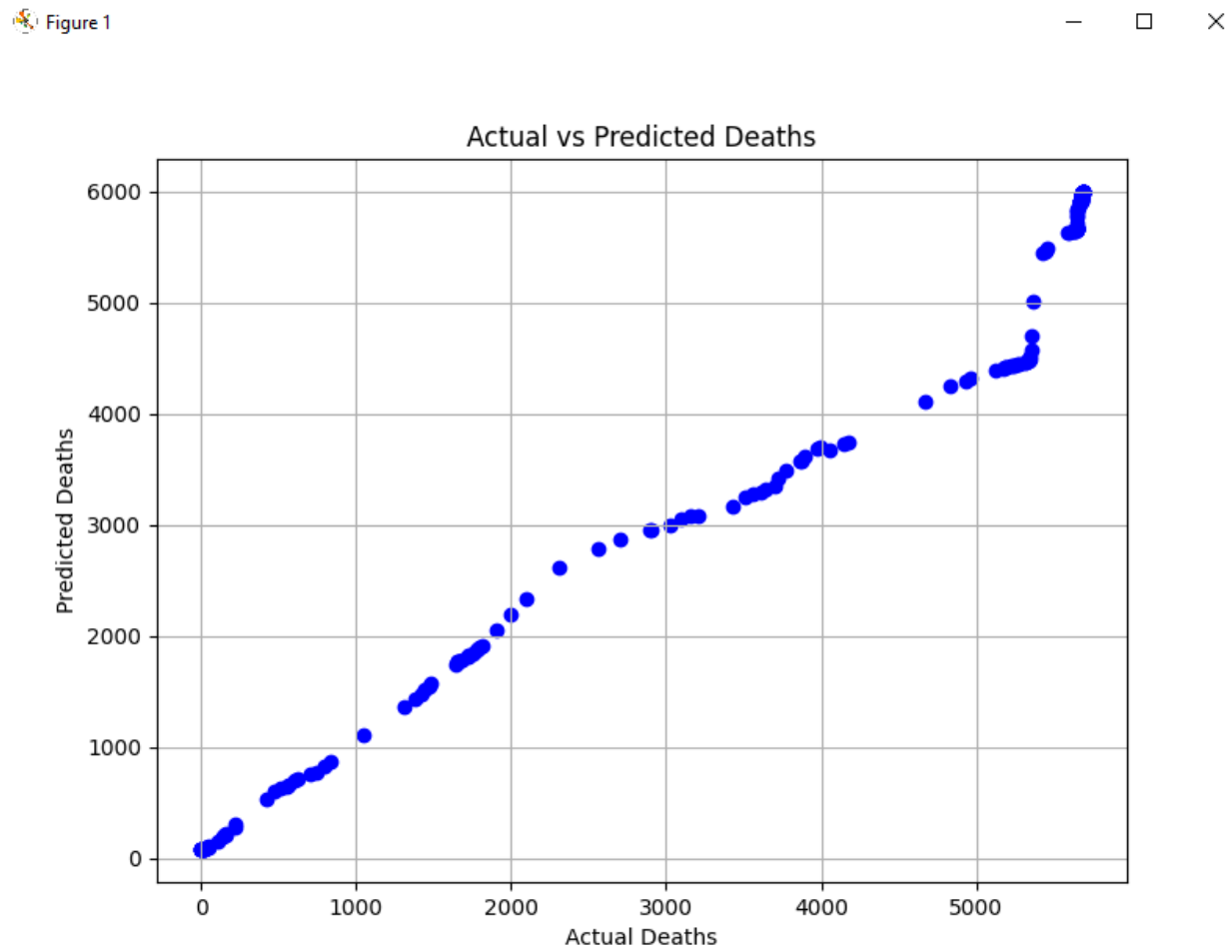
2. Show filtered data

cleaned_data.show() - method displays the first 20 rows of the filtered DataFrame in a tabular format.

(vi) Test results and interpretations

```
>>> from pyspark.sql import SparkSession
>>> from pyspark.ml.feature import VectorAssembler
>>> from pyspark.ml.regression import LinearRegression
>>> spark = SparkSession.builder.appName("DeathPrediction").getOrCreate()
>>> data = spark.read.csv("hdfs://localhost:9000/group3/covid19data.csv", header=True, inferSchema=True)
>>> feature_cols = ['Infections', 'Recoveries']
>>> data_assembled = assembler.transform(data).select('Features', 'Casualities')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'assembler' is not defined
>>> assembler = VectorAssembler(inputCols=feature_cols, outputCol='Features')
>>> data_assembled = assembler.transform(data).select('Features', 'Casualities')
>>> train_data, test_data = data_assembled.randomSplit([0.8, 0.2], seed=42)
>>> lr = LinearRegression(featuresCol='Features', labelCol='Casualities')
>>> lr_model = lr.fit(train_data)
23/12/17 21:48:18 WARN Instrumentation: [3781f24c] regParam is zero, which might cause numerical instability and overfitting.
23/12/17 21:48:20 WARN InstanceBuilder$NativeBLAS: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
23/12/17 21:48:20 WARN InstanceBuilder$NativeBLAS: Failed to load implementation from:dev.ludovic.netlib.blas.ForeignLinkerBLAS
23/12/17 21:48:20 WARN InstanceBuilder$NativeLAPACK: Failed to load implementation from:dev.ludovic.netlib.lapack.JNILAPACK
>>> from pyspark.ml.evaluation import RegressionEvaluator
>>> evaluator = RegressionEvaluator(labelCol='Casualities', predictionCol='prediction', metricName='rmse')
>>> rmse = evaluator.evaluate(predictions)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'predictions' is not defined
>>> rmse = evaluator.evaluate(prediction)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'prediction' is not defined
>>> predictions = lr_model.transform(test_data)
>>> rmse = evaluator.evaluate(predictions)
>>> r2 = evaluator.evaluate(predictions, {evaluator.metricName: "r2"})
>>> print(f"Root Mean Squared Error (RMSE): {rmse}")
Root Mean Squared Error (RMSE): 334.8099187005787
>>> print(f"R-squared (R2): {r2}")
R-squared (R2): 0.9764618107486417
>>>
```

```
>>> import matplotlib.pyplot as plt
>>> import pandas as pd
>>> predictions_pd = predictions.select('Casualities', 'prediction').toPandas()
>>> plt.figure(figsize=(8, 6))
<Figure size 800x600 with 0 Axes>
>>> plt.scatter(predictions_pd['Casualities'], predictions_pd['prediction'], color='blue')
<matplotlib.collections.PathCollection object at 0x000001FF82699090>
>>> plt.title('Actual vs Predicted Deaths')
Text(0.5, 1.0, 'Actual vs Predicted Deaths')
>>> plt.xlabel('Actual Deaths')
Text(0.5, 0, 'Actual Deaths')
>>> plt.ylabel('Predicted Deaths')
Text(0, 0.5, 'Predicted Deaths')
>>> plt.grid(True)
>>> plt.show()
```



Root Mean Squared Error (RMSE): 334.8099187005787 -- The RMSE is approximately 334.81, suggesting that, on average, the model's predictions are off by around 334.81 units with respect to the target variable (Casualities).

R-squared (R2): 0.9764618107486417 -- The R-squared value is approximately 0.9765, indicating that your model explains about 97.65% of the variance in the Casualities variable. This is a high R-squared value and suggests that the model is a good fit for the data.

(vii) Validation Results and interpretations

```
>>> test_predictions = lr_model.transform(test_data)
>>> test_rmse = evaluator.evaluate(test_predictions)
>>> test_r2 = evaluator.evaluate(test_predictions, {evaluator.metricName: "r2"})
>>> print(f"Test Root Mean Squared Error (RMSE): {test_rmse}")
Test Root Mean Squared Error (RMSE): 334.8099187005787
>>> print(f"Test R-squared (R2): {test_r2}")
Test R-squared (R2): 0.9764618107486417
>>>
```

Test Root Mean Squared Error (RMSE): 334.8099187005787-In this case, the RMSE of 334.81 suggests that, on average, the model's predictions are off by 334.809918700578781 units. A lower RMSE indicates better model performance, so a value of 334.81 suggests relatively accurate predictions.

Test R-squared (R2): 0.9764618107486417-A value of 0.9765 means that the model explains approximately 97.65% of the variability in the number of casualties on the test dataset. A higher R-squared value indicates a better fit of the model to the test data.

(viii) Potential applications of the interpreted results

- Identifying factors that contribute to COVID-19 deaths: By analyzing the data further, you may be able to identify factors that contribute to COVID-19 deaths, such as underlying health conditions, age, or socioeconomic status. This information could be used to target interventions to high-risk groups and reduce the overall number of deaths.
- Monitoring the effectiveness of COVID-19 interventions: The interpreted results could be used to monitor the effectiveness of COVID-19 interventions, such as vaccination campaigns or social distancing measures. By comparing the predicted and actual deaths over time, it may be possible to assess the impact of these interventions and make adjustments as needed.
- Allocating resources: The results could be used to allocate resources such as hospital beds, ventilators, and personal protective equipment (PPE) to areas where they are most needed. By predicting where the highest number of deaths is likely to occur, healthcare providers can ensure that they have the resources in place to care for sick patients.
- Raising awareness: The results could be used to raise awareness about the risks of COVID-19 and the importance of taking steps to prevent the spread of the virus. By sharing the data with the public, you can help people to understand the severity of the pandemic and encourage them to take steps to protect themselves and others.