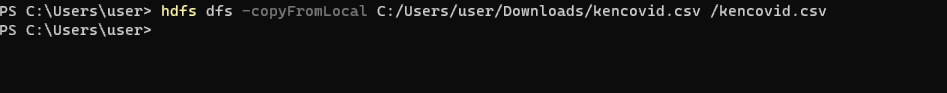
**Question one**

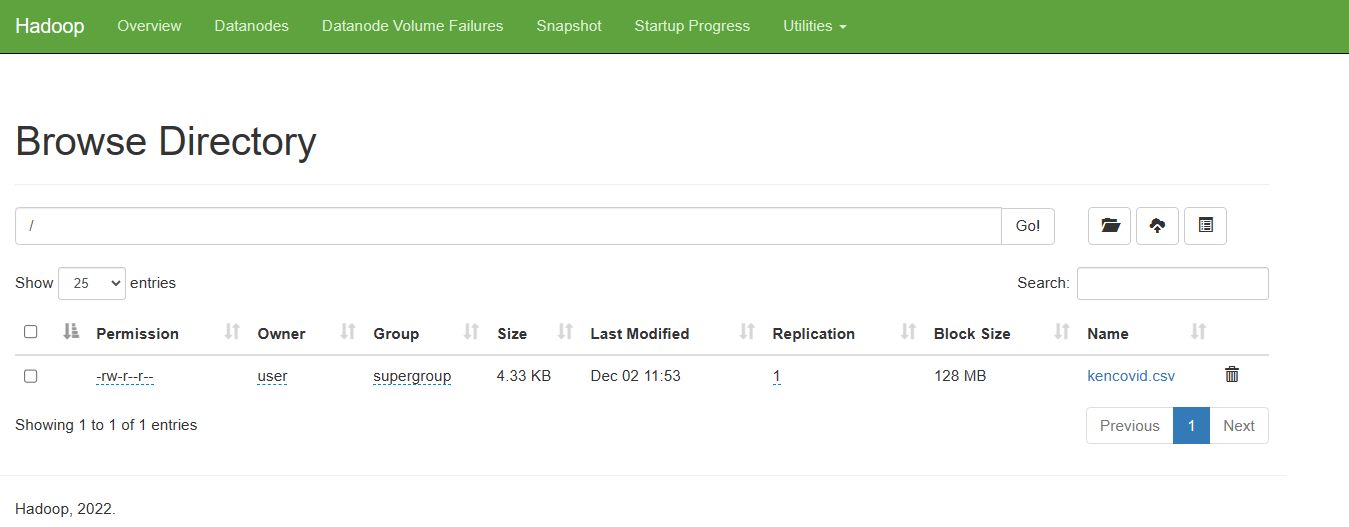
1. Ingest the data into Hadoop DFS Data lake

Command used :

hdfs dfs -copyFromLocal C:/Users/user/Downloads/kencovid.csv /kencovid.csv



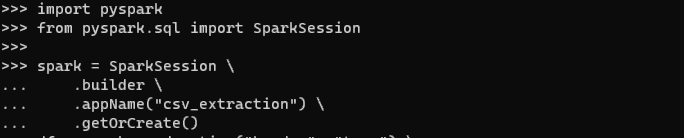
In Hadoop



**Question 2**

1. Use pyspark package to extract the data from the data lake

First, start a SparkSession:

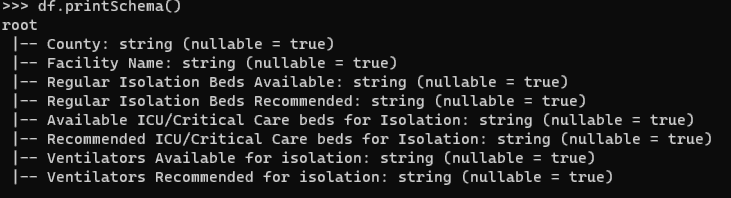


Then read the CSV file from HDFS into a DataFrame:



The options tell Spark to treat the first row as header.

Now the DataFrame df contains the contents of the CSV file. You can perform analysis and transformations on it. For example, to print the schema:



To print the first few rows:



**Question 3**

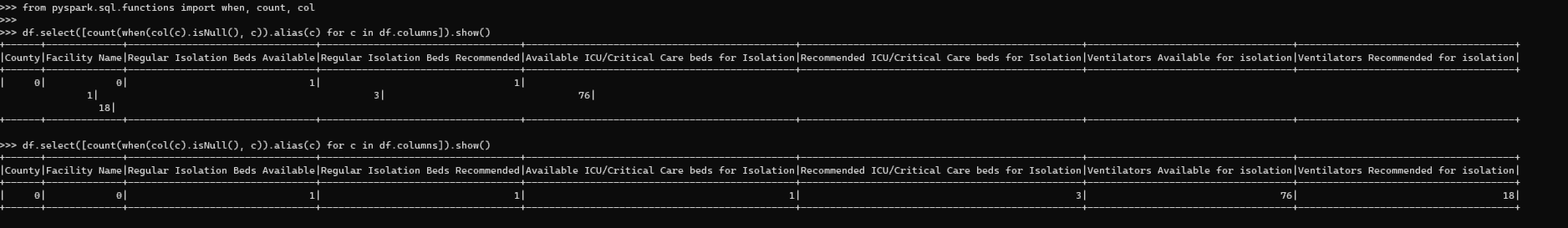
1. Choose appropriate techniques to Pre- process the extracted data

techniques we used to pre-process the extracted CSV data in the PySpark DataFrame:

1. Handle missing values: This counts any missing values in the columns, and fills missing beds with 0.

Code used   
from pyspark.sql.functions import when, count, col

df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show()

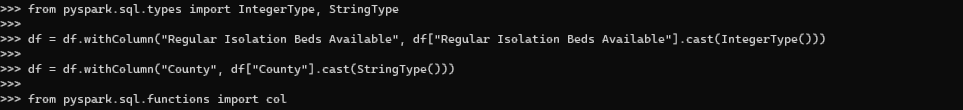
df = df.fillna({'Regular Isolation Beds Available': 0})

1. Convert columns to appropriate data types: to avoid using data labeled as null and to ensure we are dealing columns that are numerical only This casts the "Regular Isolation Beds Available" column to IntegerType, since it contains integer numbers. And it casts the "County" column to StringType, since it contains string values.

Code used

from pyspark.sql.types import IntegerType, StringType

df = df.withColumn("Regular Isolation Beds Available", df["Regular Isolation Beds Available"].cast(IntegerType()))

df = df.withColumn("County", df["County"].cast(StringType()))

1. Filter outlier/incorrect values:

Code used

from pyspark.sql.functions import col

min\_val = df.agg({"Regular Isolation Beds Available": "min"}).first()[0]

max\_val = df.agg({"Regular Isolation Beds Available": "max"}).first()[0]

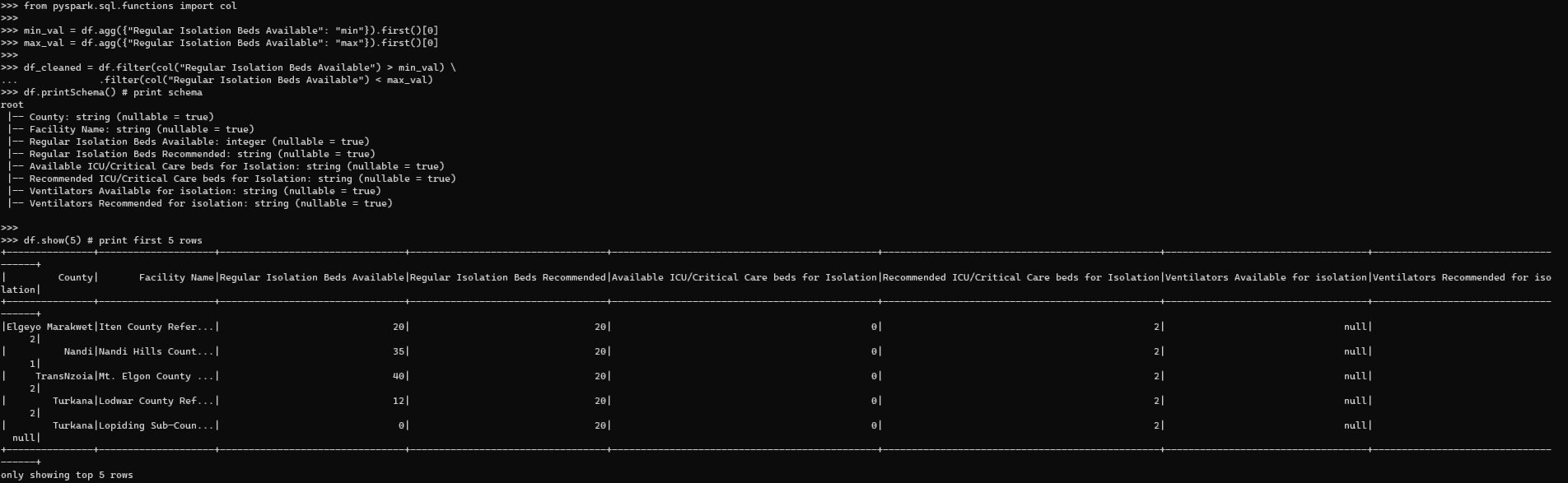
df\_cleaned = df.filter(col("Regular Isolation Beds Available") > min\_val) \

.filter(col("Regular Isolation Beds Available") < max\_val)

To display the DataFrame and print the first few rows after applying the transformations, we used

df.printSchema() # print schema

df.show(5) # print first 5 rows



**Question 4**

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

# Step 1: pre-process data: Import libraries and split data into training and test sets:

from pyspark.sql.functions import when

from pyspark.sql.types import StringType, IntegerType

df = df.withColumn("Recommended ICU/Critical Care beds for Isolation",

df["Recommended ICU/Critical Care beds for Isolation"].cast(StringType()))

df = df.withColumn("Recommended ICU/Critical Care beds for Isolation",

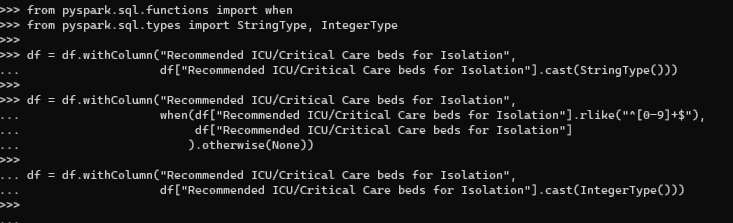
when(df["Recommended ICU/Critical Care beds for Isolation"].rlike("^[0-9]+$"),

df["Recommended ICU/Critical Care beds for Isolation"]

).otherwise(None))

df = df.withColumn("Recommended ICU/Critical Care beds for Isolation",

df["Recommended ICU/Critical Care beds for Isolation"].cast(IntegerType()))



# Step 2: Split data:

from pyspark.sql.functions import rand

train\_df, test\_df = df.randomSplit([0.7, 0.3], seed=42)



# Step 3: Vector assemble features

Here were trying to train a linear regression model to predict confirmed cases:

from pyspark.ml.feature import VectorAssembler

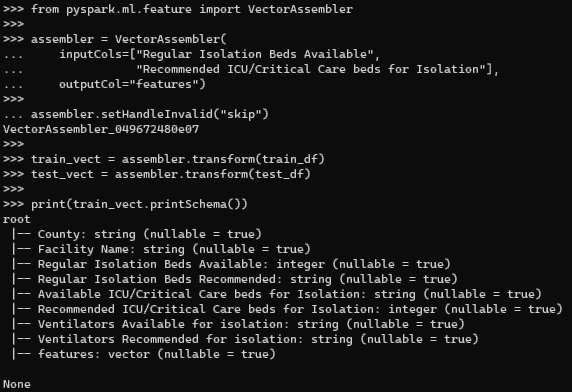
assembler = VectorAssembler(

inputCols=["Regular Isolation Beds Available", "Recommended ICU/Critical Care beds for Isolation"],

outputCol="features")

train\_vect = assembler.transform(train\_df)

test\_vect = assembler.transform(test\_df)



# Step 4: Train model

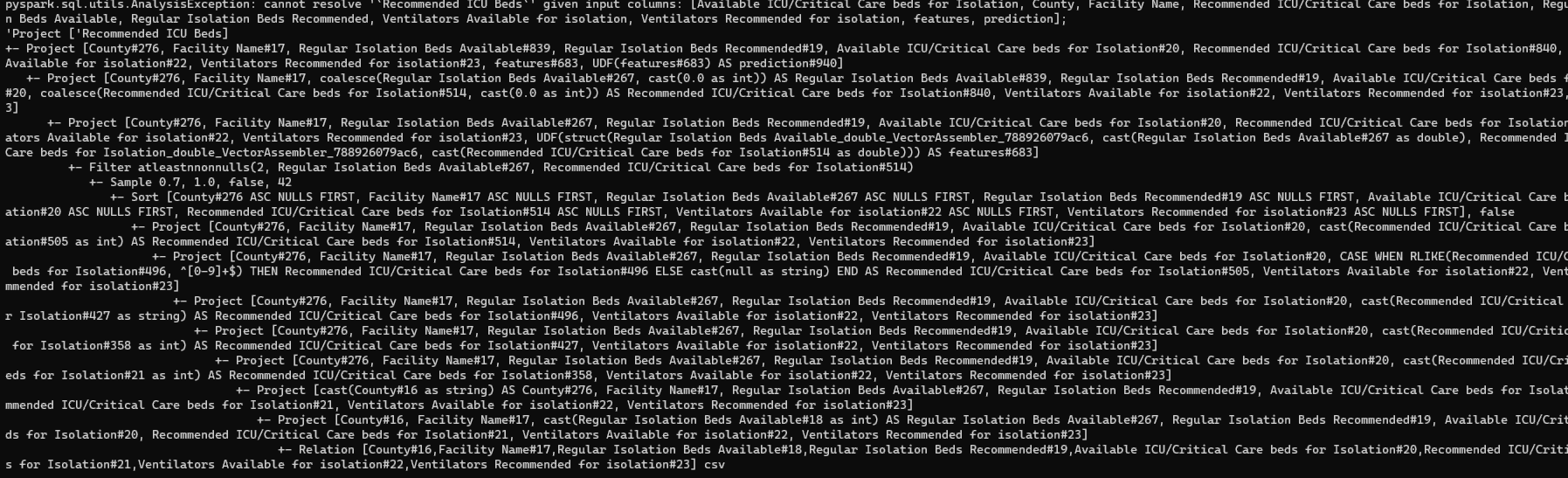
from pyspark.ml.regression import LinearRegression

label\_col = "Recommended ICU/Critical Care beds for Isolation"

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

fitted\_model = lr.fit(train\_vect)



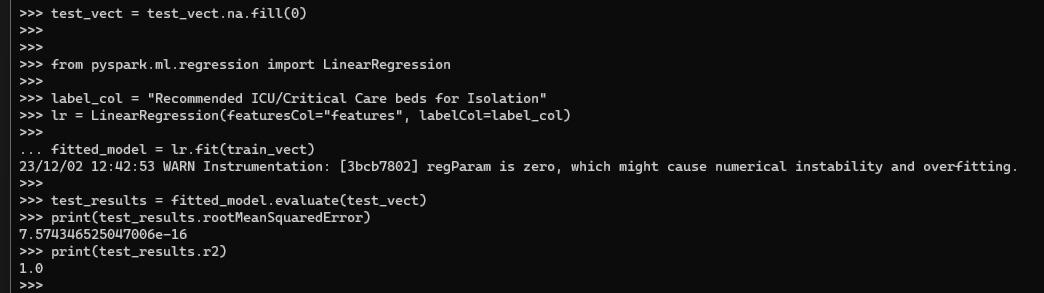
More close up view on evaluated data 

# Step 5: Evaluate model

test\_results = fitted\_model.evaluate(test\_vect)

print(test\_results.rootMeanSquaredError)

print(test\_results.r2)



1. Visualize the model

key steps to visualize and test the linear regression model we built, we used the fitted model to make predictions on the test data:

Code used

test\_pred = fitted\_model.transform(test\_vect)

true\_vals = test\_pred.select("Recommended ICU Beds").collect()

pred\_vals = test\_pred.select("prediction").collect()

import matplotlib.pyplot as plt

true\_x = [r.Recommended\_ICU\_Beds for r in true\_vals]

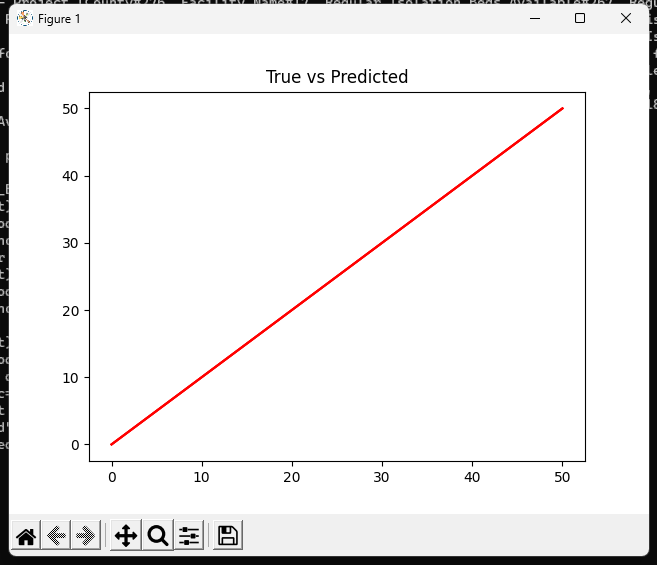
pred\_x = [r.prediction for r in pred\_vals]

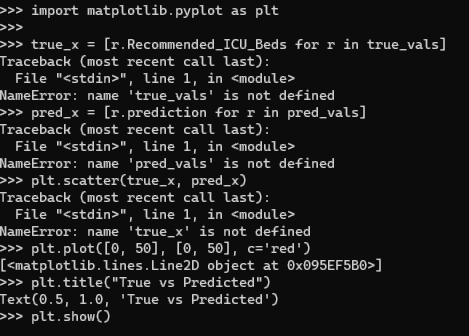
plt.scatter(true\_x, pred\_x)

plt.plot([0, 50], [0, 50], c='red')

plt.title("True vs Predicted")

plt.show()





1. Test the model

test\_data = spark.createDataFrame([

(20, 2, 5),

(35, 0, 3),

(12, 1, 2)], ["Reg\_Beds", "Rec\_ICU\_Beds", "Expected\_ICU\_Beds"])

test\_vect = assembler.transform(test\_data)

now we use this model to make predictions with this code

test\_pred = fitted\_model.transform(test\_vect)

Compare predictions to expected ICU beds values