

Lukas saidim - SCT222-0137/2020

Nelson kamau - Sct222-0301/2020

Kahura Gedion karanja - Sct222-0155/2020

Kibet collins kemboi - SCT222-0147/2020

Francis Peter Thuo - Sct222-0126/2020

DBMS CAT

Question 1

Covid-19 is a contagious disease caused by Virus known as Sars-Cov-2, which hit the entire world in the year 2020.

This task aims to study the Covid-19 Statistics with respect to active cases, confirmed cases, total deaths and so on across the world. Further, it analyses the countries which have been badly hit by the virus and some important ratios to find out about the deaths caused by Covid-19 virus.

The Dataset comprises of six .csv files which contain information about the Covid-19 status across different countries and across the world. The data has been collected from the World Health Organization website and from several other government sources and then compiled thereafter. The six data files are explained below:

- 'country_wise.csv': Raw data comprising of Covid-19 statistics (total cases, active cases, recovered cases, total deaths) across different countries/regions as defined by WHO.
- 'day_wise_country.csv': Day-wise and country-wise Covid-19 statistics.
- 'day_wise_worldwide.csv': Day-wise Covid-19 Statistics for from January-2020 to July-2020 across the world.
- 'east_africa_covid_data.csv': cumulative number of coronavirus (COVID-19) cases in East Africa.
- 'kenya_county_cases.csv': Cumulative number of confirmed coronavirus (COVID-19) cases in Kenya counties.
- 'kenya_daily_cases.csv': Total confirmed cases daily in Kenya.

Data for COVID-19 is compiled by downloading information from various sources such as Worldometer, Africa Open Data, and Kenya Open Data. The data is be obtained in different formats, including XMLs then later converted to CSVs.

In the Jupyter Notebook environment, we compile and examine the data to prepare for analysis. The initial steps involve setting up the notebook on a local machine and importing the necessary modules.

```

import pandas as pd
import matplotlib.pyplot as plt

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import pandas as pd
import random
import math
import time
from sklearn.linear_model import LinearRegression, BayesianRidge
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error
import datetime
import operator
%matplotlib inline
import warnings

```

NB

1. ***import pandas as pd***: This imports the Pandas library and aliases it as `pd`. Pandas is a powerful data manipulation and analysis library for Python. It provides data structures like DataFrame for efficient data handling.
2. ***import matplotlib.pyplot as plt***: This imports the `pyplot` module from the Matplotlib library and aliases it as `plt`. Matplotlib is a popular plotting library in Python, and `pyplot` provides a convenient interface for creating various types of plots.
3. ***import numpy as np***: This imports the NumPy library and aliases it as `np`. NumPy is a numerical computing library for Python, providing support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these.
4. ***import matplotlib.colors as mcolors***: This imports the `colors` module from Matplotlib and aliases it as `mcolors`. It provides a collection of named colors and color maps that can be used in plotting.
6. ***import random***: This imports the `random` module, which provides functions for generating random numbers and performing random operations.
7. ***import math***: This imports the built-in `math` module, providing mathematical functions and constants.
8. ***import time***: This imports the built-in `time` module, which provides various time-related functions.

9. `from sklearn.linear_model import LinearRegression, BayesianRidge`: This imports specific regression models from the scikit-learn library. Scikit-learn is a machine learning library for Python, and here you're importing linear regression and Bayesian Ridge regression models.
10. `from sklearn.model_selection import RandomizedSearchCV, train_test_split`: This imports classes for model selection, including `RandomizedSearchCV` for hyperparameter tuning and `train_test_split` for splitting data into training and testing sets.
11. `from sklearn.preprocessing import PolynomialFeatures`: This imports a class for generating polynomial features. It is often used in combination with linear regression to model non-linear relationships.
12. `from sklearn.svm import SVR`: This imports the Support Vector Regression (SVR) model from scikit-learn, which is a regression algorithm based on support vector machines.
13. `from sklearn.metrics import mean_squared_error, mean_absolute_error`: This imports functions for evaluating regression model performance, including mean squared error and mean absolute error.
14. `import datetime`: This imports the built-in `datetime` module, which provides classes for working with dates and times.
15. `import operator`: This imports the `operator` module, which provides functions that implement built-in operations as functions.
16. `%matplotlib inline`: This is a Jupyter Notebook magic command that ensures that Matplotlib plots are displayed inline in the notebook rather than in a separate window.

By using the Pandas library to read a CSV file named 'country_wise.csv' from the specified path, and then it prints the first few rows of the loaded data.

```
[8]: # Access dataset
path = r'C:\Users\lukas\Covid-19'
country_wise = read_data(path, 'country_wise.csv')
# Print the first few rows of the data
country_wise.head()
```

[8]:

	Country/Region	Confirmed	Deaths	Recovered	Active	New cases	New deaths	New recovered	Deaths / 100 Cases	Recovered / 100 Cases	Deaths / 100 Recovered	Confirmed last week	1 week change	1 week % increase	WHO Region
0	Afghanistan	36263	1269	25198	9796	106	10	18	3.50	69.49	5.04	35526	737	2.07	Eastern Mediterranean
1	Albania	4880	144	2745	1991	117	6	63	2.95	56.25	5.25	4171	709	17.00	Europe
2	Algeria	27973	1163	18837	7973	616	8	749	4.16	67.34	6.17	23691	4282	18.07	Africa
3	Andorra	907	52	803	52	10	0	0	5.73	88.53	6.48	884	23	2.60	Europe
4	Angola	950	41	242	667	18	1	0	4.32	25.47	16.94	749	201	26.84	Africa

The `country_wise.shape` command is used to retrieve the dimensions of the DataFrame `country_wise`. Specifically, it returns a tuple representing the number of rows and columns in the DataFrame.

```
[9]: country_wise.shape
```

```
[9]: (187, 15)
```

```
[10]: day_wise_worldwide = read_data(path,'day-wise-worldwide.csv')
      day_wise_worldwide.head()
```

```
[10]:
```

	Date	Confirmed	Deaths	Recovered	Active	New cases	New deaths	New recovered	Deaths / 100 Cases	Recovered / 100 Cases	Deaths / 100 Recovered	No. of countries
0	22/01/2020	555	17	28	510	0	0	0	3.06	5.05	60.71	6
1	23/01/2020	654	18	30	606	99	1	2	2.75	4.59	60.00	8
2	24/01/2020	941	26	36	879	287	8	6	2.76	3.83	72.22	9
3	25/01/2020	1434	42	39	1353	493	16	3	2.93	2.72	107.69	11
4	26/01/2020	2118	56	52	2010	684	14	13	2.64	2.46	107.69	13

Day wise (New cases, deaths, recovered, deaths/100, recovered/100)

This line uses the `read_data()` function from the Pandas library to read data from a CSV file. The path variable represents the file path or URL from which the CSV file is read, and 'day-wise-worldwide.csv' is the name of the CSV file.

```
: day_wise_worldwide = read_data(path,'day-wise-worldwide.csv')
  day_wise_worldwide.head()
```

	Date	Confirmed	Deaths	Recovered	Active	New cases	New deaths	New recovered	Deaths / 100 Cases	Recovered / 100 Cases	Deaths / 100 Recovered	No. of countries
0	22/01/2020	555	17	28	510	0	0	0	3.06	5.05	60.71	6
1	23/01/2020	654	18	30	606	99	1	2	2.75	4.59	60.00	8
2	24/01/2020	941	26	36	879	287	8	6	2.76	3.83	72.22	9
3	25/01/2020	1434	42	39	1353	493	16	3	2.93	2.72	107.69	11
4	26/01/2020	2118	56	52	2010	684	14	13	2.64	2.46	107.69	13

Day wise for each country

We are going to read another dataset named 'day-wise-country.csv'. Each row represents a specific observation at a particular date for a given country or region.

```
: day_wise_country = read_data(path,'day-wise-country.csv')
  day_wise_country.head()
```

	Date	Country/Region	Confirmed	Deaths	Recovered	Active	New cases	New deaths	New recovered	WHO Region
0	2020-01-22	Afghanistan	0	0	0	0	0	0	0	Eastern Mediterranean
1	2020-01-22	Albania	0	0	0	0	0	0	0	Europe
2	2020-01-22	Algeria	0	0	0	0	0	0	0	Africa
3	2020-01-22	Andorra	0	0	0	0	0	0	0	Europe
4	2020-01-22	Angola	0	0	0	0	0	0	0	Africa

```
day_wise_country.shape
```

```
(35156, 10)
```

East African countries

We identify and filter data for East African countries from a DataFrame named country wise.

```
] : #Identify East African countries.
east_africa_countries = ['Burundi', 'Djibouti', 'Eritrea', 'Ethiopia',
                        'Kenya', 'Rwanda', 'Somalia',
                        'South Sudan', 'Sudan', 'Tanzania', 'Uganda']

# Filtering data for East African countries
east_africa_data = country_wise[country_wise['Country/Region'].isin(east_africa_countries)]
east_africa_data.head(11)
```

	Country/Region	Confirmed	Deaths	Recovered	Active	New cases	New deaths	New recovered	Deaths / 100 Cases	Recovered / 100 Cases	Deaths / 100 Recovered	Confirmed last week	1 week change	1 week % increase	WHO Region
28	Burundi	378	1	301	76	17	0	22	0.26	79.63	0.33	322	56	17.39	Africa
48	Djibouti	5059	58	4977	24	9	0	11	1.15	98.38	1.17	5020	39	0.78	Eastern Mediterranean
55	Eritrea	265	0	191	74	2	0	2	0.00	72.08	0.00	251	14	5.58	Africa
58	Ethiopia	14547	228	6386	7933	579	5	170	1.57	43.90	3.57	10207	4340	42.52	Africa
90	Kenya	17975	285	7833	9857	372	5	90	1.59	43.58	3.64	13771	4204	30.53	Africa
139	Rwanda	1879	5	975	899	58	0	57	0.27	51.89	0.51	1629	250	15.35	Africa
153	Somalia	3196	93	1543	1560	18	0	22	2.91	48.28	6.03	3130	66	2.11	Eastern Mediterranean
156	South Sudan	2305	46	1175	1084	43	1	0	2.00	50.98	3.91	2211	94	4.25	Africa
159	Sudan	11424	720	5939	4765	39	3	49	6.30	51.99	12.12	10992	432	3.93	Eastern Mediterranean
166	Tanzania	509	21	183	305	0	0	0	4.13	35.95	11.48	509	0	0.00	Africa
174	Uganda	1128	2	986	140	13	0	4	0.18	87.41	0.20	1069	59	5.52	Africa

Then save the filtered data for East African countries to a new CSV file named 'east-africa-covid-data.csv'.

```
] : # Save the filtered data to a new CSV file
east_africa_data.to_csv('east-africa-covid-data.csv', index=False)
```

Kenya cases in counties

```
: kenya_county_cases = read_data(path, 'kenya-county-cases-2022.csv')
kenya_county_cases.head()
```

```
:   County  Cases
0 Nairobi City 129,123
1 Kiambu      19,778
2 Mombasa     17,794
3 Nakuru      16,717
4 Uasin Gishu 10,349
```

```
: kenya_county_cases.shape
```

```
: (11, 2)
```

The data provides information on the number of COVID-19 cases reported in various counties in Kenya. Each row corresponds to a specific county, and the 'Cases' column indicates the reported number of cases in that county. The 'County' column holds the names of the counties.

Kenya daily cases

The resulting `kenya_daily_cases` DataFrame should have the same columns as `day_wise_country`, and it will include only the rows where the country is 'Kenya'. To specifically get information on daily COVID-19 metrics like Confirmed, Deaths, Recovered, Active, New cases, New deaths, and New recovered, you can look at the relevant columns in the DataFrame. The structure of these columns should represent the daily changes in those metrics for Kenya.

	Date	Country/Region	Confirmed	Deaths	Recovered	Active	New cases	New deaths	New recovered	WHO Region
90	2020-01-22	Kenya	0	0	0	0	0	0	0	Africa
277	2020-01-23	Kenya	0	0	0	0	0	0	0	Africa
464	2020-01-24	Kenya	0	0	0	0	0	0	0	Africa
651	2020-01-25	Kenya	0	0	0	0	0	0	0	Africa

Save to a new csv file

```
# Save the filtered data to a new CSV file
kenya_daily_cases.to_csv('kenya-daily-cases.csv', index=False)
```

```
kenya_daily_cases.shape
```

```
(188, 10)
```

Question 2 – ingest data into Hadoop

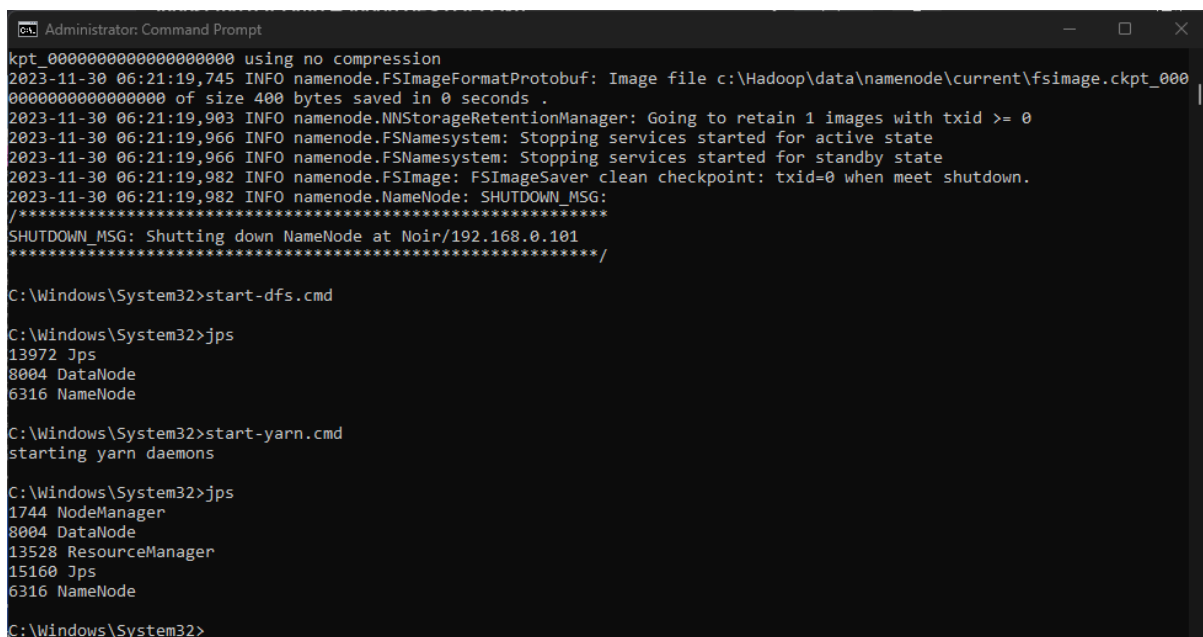
Starting Hadoop involves initiating the various Hadoop daemons and services. Hadoop is typically deployed in a distributed environment, and starting it requires a set of commands.

Start Hadoop Distributed File System (HDFS):

- Open a terminal
- Use the following command to start HDFS: *on windows*
- start-dfs.cmd
- This command starts the NameNode and DataNode daemons.

Start Yet Another Resource Negotiator (YARN):

- start-yarn.cmd
- This command starts ResourceManager and NodeManager daemons.
- Verify Hadoop Processes:
- After starting HDFS and YARN, you can check the status of the Hadoop processes by accessing the Hadoop web interfaces.



```
Administrator: Command Prompt
kpt_000000000000000000 using no compression
2023-11-30 06:21:19,745 INFO namenode.FSImageFormatProtobuf: Image file c:\Hadoop\data\namenode\current\fsimage.ckpt_000
0000000000000000 of size 400 bytes saved in 0 seconds .
2023-11-30 06:21:19,903 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2023-11-30 06:21:19,966 INFO namenode.FSNamesystem: Stopping services started for active state
2023-11-30 06:21:19,966 INFO namenode.FSNamesystem: Stopping services started for standby state
2023-11-30 06:21:19,982 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.
2023-11-30 06:21:19,982 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at Noir/192.168.0.101
*****/

C:\Windows\System32>start-dfs.cmd

C:\Windows\System32>jps
13972 Jps
8004 DataNode
6316 NameNode

C:\Windows\System32>start-yarn.cmd
starting yarn daemons

C:\Windows\System32>jps
1744 NodeManager
8004 DataNode
13528 ResourceManager
15160 Jps
6316 NameNode

C:\Windows\System32>
```

- Open a web browser and navigate to the following URLs:
- HDFS NameNode: <http://localhost:5007> instead we used: 9201 due to conflict with Jupyter notebook
- YARN ResourceManager: <http://localhost:8088>

Hadoop	Overview	Datanodes	Datanode Volume Failures	Snapshot	Startup Progress	Utilities ▾
--------	----------	-----------	--------------------------	----------	------------------	-------------

Overview 'localhost:9500' (✓active)

Started:	Mon Dec 04 18:59:28 +0300 2023
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 11:22:00 +0300 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-c076f1b5-7dcf-4017-b73f-5c7bc6b57121
Block Pool ID:	BP-1576700280-192.168.0.101-1701314479286

Summary

Security is off.

Safemode is off.

11 files and directories, 8 blocks (8 replicated blocks, 0 erasure coded block groups) = 19 total filesystem object(s).

Heap Memory used 157.92 MB of 336.5 MB Heap Memory. Max Heap Memory is 889 MB.



Cluster

About

Nodes

Node Labels

Applications

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	
0	0	0	0	0	<memory:0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes
1	0	0

Scheduler Metrics

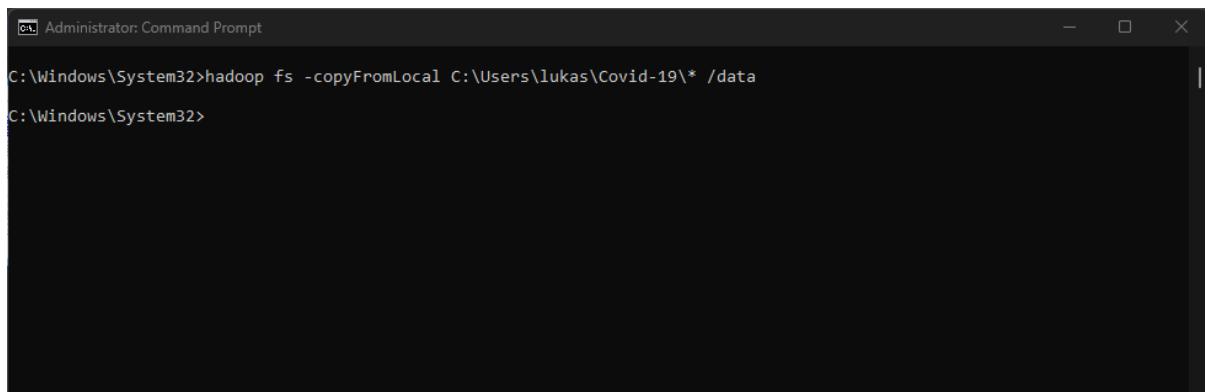
Scheduler Type	Scheduling Resource Type	Minimum Allocation
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>

Show 20 ▾ entries

ID ▾	User ▾	Name ▾	Application Type ▾	Application Tags ▾	Queue ▾	Application Priority ▾	StartTime ▾	LaunchTime ▾	FinishTime ▾	State ▾
------	--------	--------	--------------------	--------------------	---------	------------------------	-------------	--------------	--------------	---------

Showing 0 to 0 of 0 entries

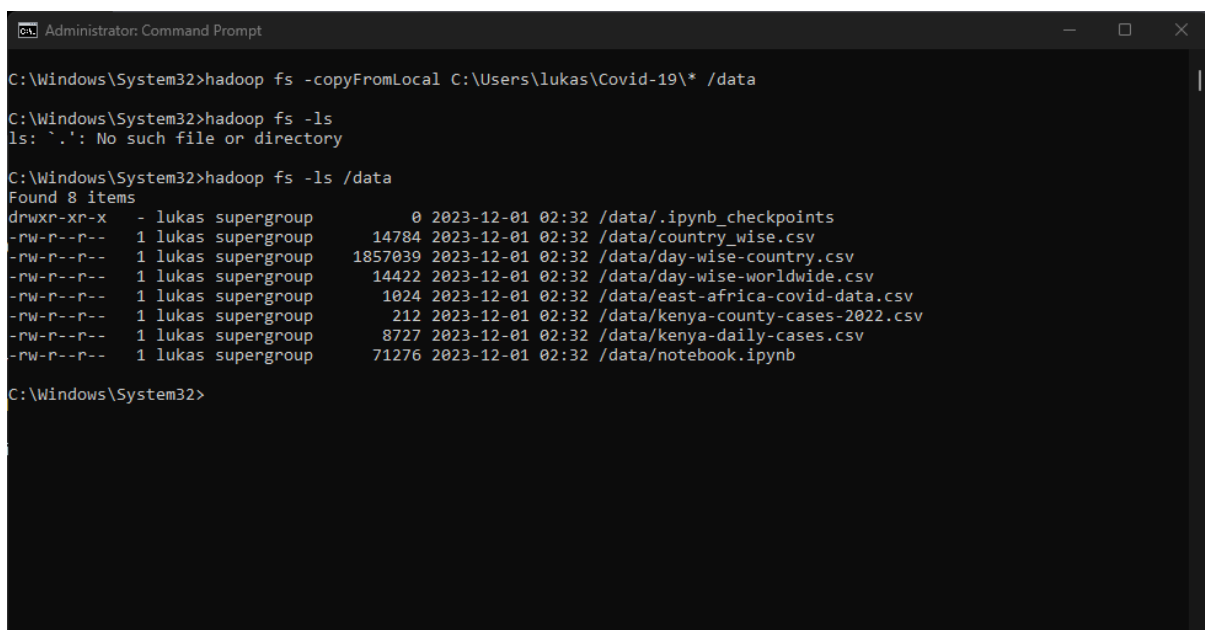
Ingesting data into Hadoop involves transferring data from external sources into the Hadoop Distributed File System (HDFS) so that it can be processed and analyzed by Hadoop-based applications.



```
Administrator: Command Prompt
C:\Windows\System32>hadoop fs -copyFromLocal C:\Users\lukas\Covid-19\* /data
C:\Windows\System32>
```

The Hadoop command-line interface provides commands to interact with HDFS, including copying files.

After ingesting data, you can use Hadoop commands or web interfaces to check if the data has been successfully loaded into HDFS.



```
Administrator: Command Prompt
C:\Windows\System32>hadoop fs -copyFromLocal C:\Users\lukas\Covid-19\* /data
C:\Windows\System32>hadoop fs -ls
ls: `.`: No such file or directory
C:\Windows\System32>hadoop fs -ls /data
Found 8 items
drwxr-xr-x  - lukas supergroup      0 2023-12-01 02:32 /data/.ipynb_checkpoints
-rw-r--r--  1 lukas supergroup    14784 2023-12-01 02:32 /data/country_wise.csv
-rw-r--r--  1 lukas supergroup  1857039 2023-12-01 02:32 /data/day-wise-country.csv
-rw-r--r--  1 lukas supergroup    14422 2023-12-01 02:32 /data/day-wise-worldwide.csv
-rw-r--r--  1 lukas supergroup     1024 2023-12-01 02:32 /data/east-africa-covid-data.csv
-rw-r--r--  1 lukas supergroup      212 2023-12-01 02:32 /data/kenya-county-cases-2022.csv
-rw-r--r--  1 lukas supergroup     8727 2023-12-01 02:32 /data/kenya-daily-cases.csv
-rw-r--r--  1 lukas supergroup    71276 2023-12-01 02:32 /data/notebook.ipynb
C:\Windows\System32>
```

Show 25 entries

Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	drwxr-xr-x	lukas	supergroup	0 B	Dec 01 02:32	0	0 B	.ipynb_checkpoints	
<input type="checkbox"/>	-rw-r--r--	lukas	supergroup	14.44 KB	Dec 01 02:32	1	128 MB	country_wise.csv	
<input type="checkbox"/>	-rw-r--r--	lukas	supergroup	1.77 MB	Dec 01 02:32	1	128 MB	day-wise-country.csv	
<input type="checkbox"/>	-rw-r--r--	lukas	supergroup	14.08 KB	Dec 01 02:32	1	128 MB	day-wise-worldwide.csv	
<input type="checkbox"/>	-rw-r--r--	lukas	supergroup	1 KB	Dec 01 02:32	1	128 MB	east-africa-covid-data.csv	
<input type="checkbox"/>	-rw-r--r--	lukas	supergroup	212 B	Dec 01 02:32	1	128 MB	kenya-county-cases-2022.csv	
<input type="checkbox"/>	-rw-r--r--	lukas	supergroup	8.52 KB	Dec 01 02:32	1	128 MB	kenya-daily-cases.csv	
<input type="checkbox"/>	-rw-r--r--	lukas	supergroup	69.61 KB	Dec 01 02:32	1	128 MB	notebook.ipynb	

Showing 1 to 8 of 8 entries

Previous
1
Next

Use pyspark package to extract the data from the data lake

```
]: %pip install pyspark
%pip install seaborn
%pip install plotly
```

```
]: import pandas as pd
import matplotlib.pyplot as plt

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import pandas as pd
import random
import math
import time
from sklearn.linear_model import LinearRegression, BayesianRidge
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error
import datetime
import operator
%matplotlib inline
import warnings
```

These commands will install the specified Python packages (pyspark, seaborn, and plotly) using the Python package manager, pip.

```
: from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("HadoopDataRetrieval").enableHiveSupport().getOrCreate()
```

`spark = SparkSession.builder.appName("HadoopDataRetrieval").enableHiveSupport().getOrCreate():`
 This line creates a Spark session. Here's what each part of the chain does:

SparkSession.builder: Starts the process of building a SparkSession.

appName("HadoopDataRetrieval"): Sets a user-defined name for the application, which will be displayed in the Spark web UI.

enableHiveSupport(): Enables Hive support, allowing you to interact with Hive using Spark.

getOrCreate(): Gets an existing SparkSession or creates a new one if it doesn't exist.

```
hdfs_path = "hdfs://localhost:9500/data/country_wise.csv"

# Example for reading a CSV file
df = spark.read.csv(hdfs_path, header=True, inferSchema=True, samplingRatio=0.1)
```

```
# Show the first few rows of the DataFrame
df.show()
```

Using PySpark to read a CSV file from HDFS (Hadoop Distributed File System) into a DataFrame named df.

`hdfs_path = "hdfs://localhost:9500/data/country_wise.csv"`

This line defines the HDFS path to the CSV file you want to read. The path is specified as a URL starting with "hdfs://" followed by the hostname and port number of the HDFS Namenode, and then the path to the file.

Country/Region	Confirmed	Deaths	Recovered	Active	New cases	New deaths	New recovered	Deaths / 100 Cases	Recovered / 100 Cases	Deaths / 100 Recovered
Confirmed last week	1 week change	1 week % increase	WHO Region							
Afghanistan	36263	1269	25198	9796	106	10	18	3.5	69.49	5.04
Albania	4880	144	2745	1991	117	6	63	2.95	56.25	5.25
Algeria	27973	1163	18837	7973	616	8	749	4.16	67.34	6.17
Andorra	907	52	803	52	10	0	0	5.73	88.53	6.48
Angola	950	41	242	667	18	1	0	4.32	25.47	16.94
Antigua and Barbuda	86	3	65	18	4	0	5	3.49	75.58	4.62
Argentina	167416	3059	72575	91782	4890	120	2057	1.83	43.35	4.21
Armenia	37390	711	26665	10014	73	6	187	1.9	71.32	2.67

```
[80]: # Now, read the full dataset
full_df = spark.read.option("header", "true").csv(hdfs_path, header=True, inferSchema=True, samplingRatio=1)

# Save the full DataFrame as a CSV file
output_path = "dataset"
full_df.write.mode("overwrite").csv(output_path, header=True)
```

This line reads the full CSV dataset into a new PySpark DataFrame called **full_df**.

option("header", "true"): Specifies that the first row of the CSV file contains column names.

samplingRatio=1: Reads the entire dataset for schema inference since the sampling ratio is set to 1.

Choose appropriate techniques to Pre- process the extracted data

Handling Missing Values:

Check for missing values and decide how to handle them. Common methods include dropping rows with missing values, filling missing values with the mean or median, or using more advanced imputation techniques.

```
: # Check for missing values
missing_values = country_wise.isnull().sum()

# Display missing values
print("Missing Values:")
print(missing_values)

# Drop rows with missing values
country_wise = country_wise.dropna()

# Alternatively, fill missing values with the mean
# df = df.fillna(df.mean())
```

```
Missing Values:
Country/Region      0
Confirmed           0
Deaths              0
Recovered           0
Active              0
New cases           0
New deaths          0
New recovered       0
Deaths / 100 Cases  0
Recovered / 100 Cases  0
Deaths / 100 Recovered  0
Confirmed last week  0
1 week change       0
1 week % increase   0
WHO Region          0
dtype: int64
```

Data Types Conversion

Ensure that columns have the correct data types. Convert columns to appropriate types if necessary.

```
: # Convert a column to datetime type
day_wise_country['Date'] = pd.to_datetime(day_wise_country['Date'])
```

Exploratory Data Analysis (EDA)

Explore the data to gain insights and identify patterns. Use visualizations such as histograms, box plots, and scatter plots.

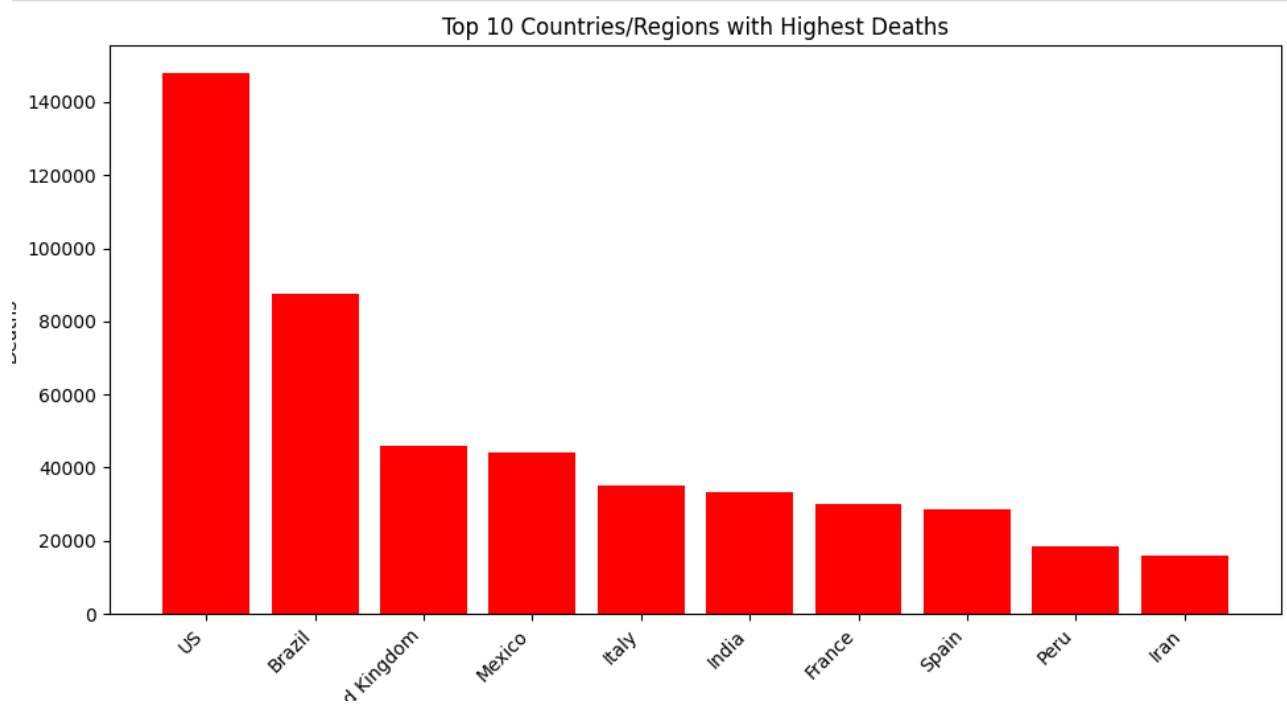
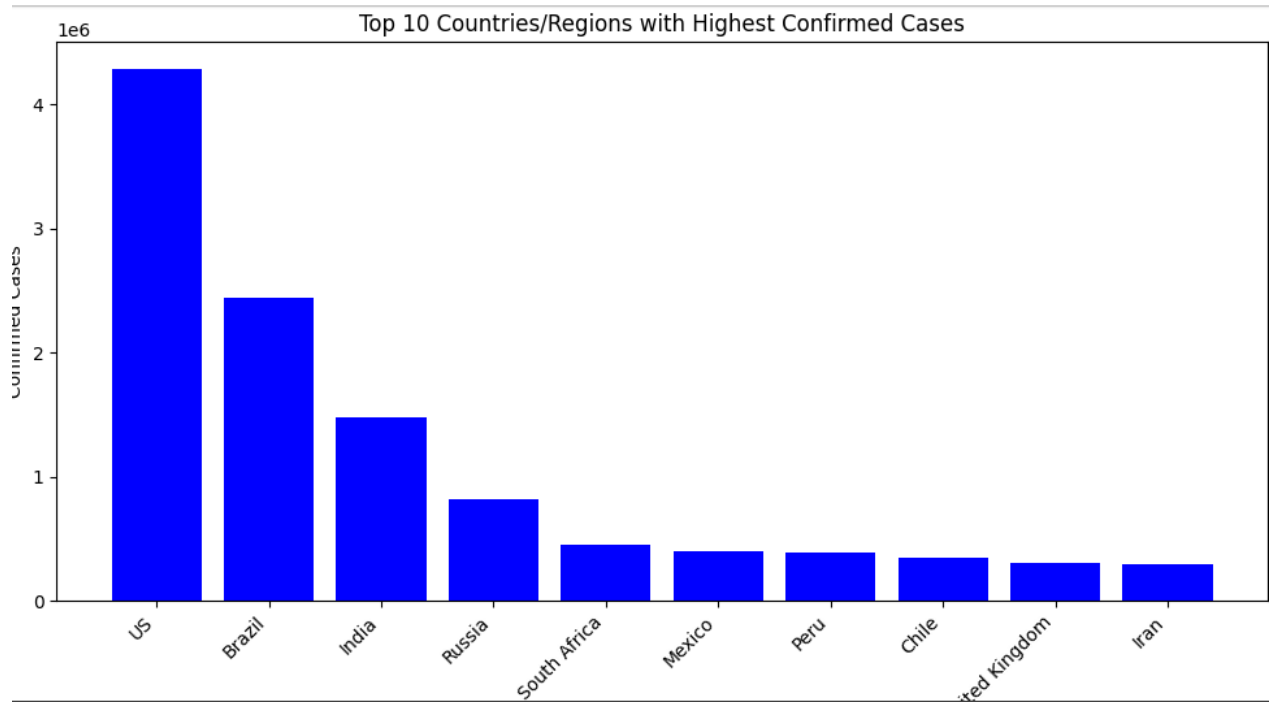
```
# Sort the DataFrame by Confirmed cases and Deaths
df_sorted_confirmed = country_wise.sort_values(by='Confirmed', ascending=False).head(10)
df_sorted_deaths = country_wise.sort_values(by='Deaths', ascending=False).head(10)

# Plotting the bar graph for Confirmed cases
plt.figure(figsize=(10, 6))
plt.bar(df_sorted_confirmed['Country/Region'], df_sorted_confirmed['Confirmed'], color='blue')
plt.title('Top 10 Countries/Regions with Highest Confirmed Cases')
plt.xlabel('Country/Region')
plt.ylabel('Confirmed Cases')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Show the plot
plt.show()

# Plotting the bar graph for Deaths
plt.figure(figsize=(10, 6))
plt.bar(df_sorted_deaths['Country/Region'], df_sorted_deaths['Deaths'], color='red')
plt.title('Top 10 Countries/Regions with Highest Deaths')
plt.xlabel('Country/Region')
plt.ylabel('Deaths')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Show the plot
plt.show()
```



```

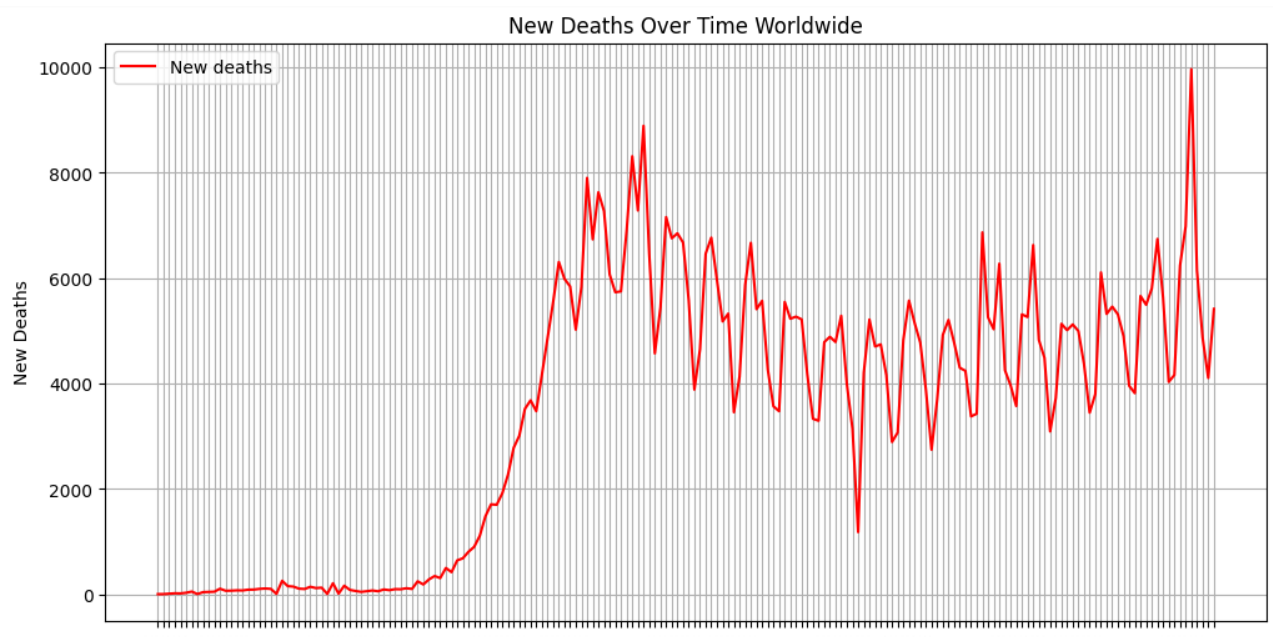
: |# Plotting the graph for New deaths over time
plt.figure(figsize=(12, 6))

# Plotting New deaths
plt.plot(day_wise_worldwide['Date'], day_wise_worldwide['New deaths'], label='New deaths', color='red')

# Set plot properties
plt.title('New Deaths Over Time Worldwide')
plt.xlabel('Date')
plt.ylabel('New Deaths')
plt.legend()
plt.xticks(rotation=45, ha='right')
plt.grid(True)

# Show the plot
plt.show()

```



```

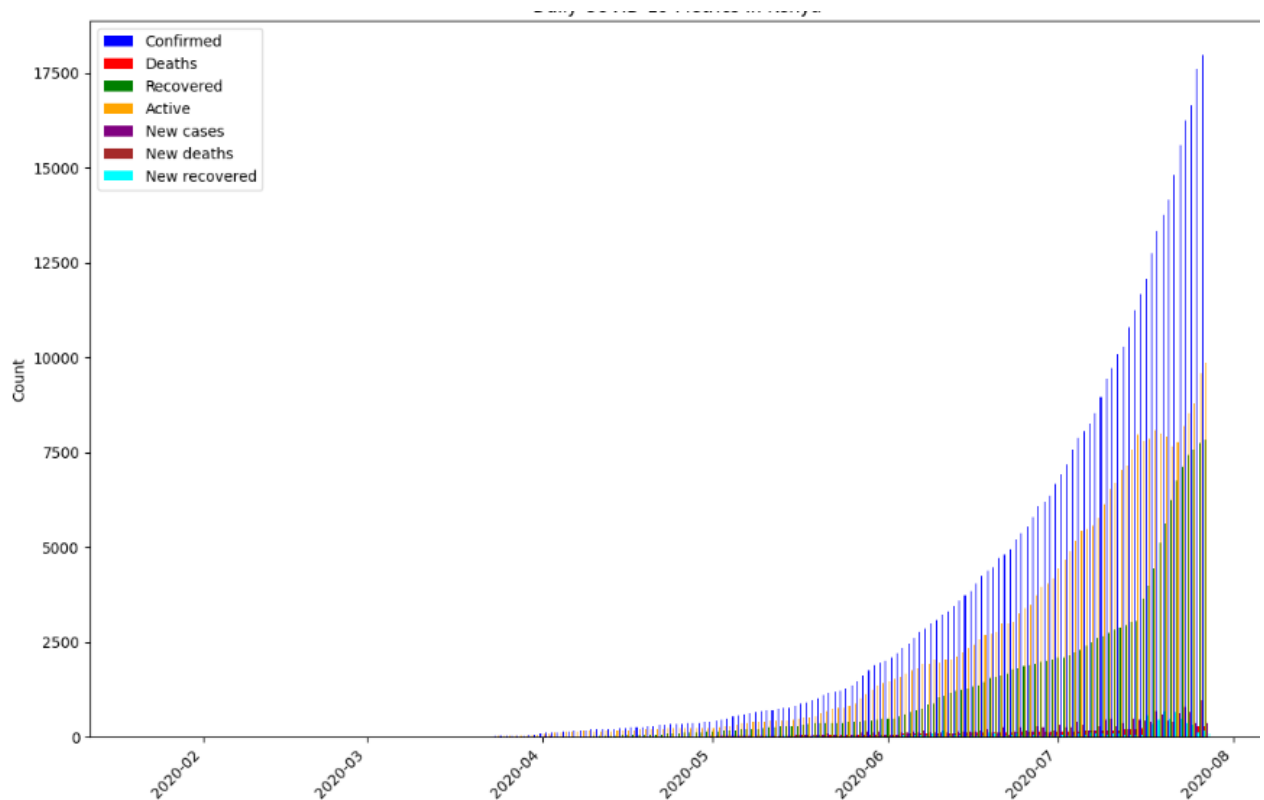
# Plotting the bar graph for daily COVID-19 metrics in Kenya
plt.figure(figsize=(12, 8))

# Bar width for better visibility
bar_width = 0.2

# Plotting Confirmed cases
plt.bar(kenya_daily_cases['Date'] - pd.to_timedelta(bar_width * 2, unit='d'), kenya_daily_cases['Confirmed'], width=bar_width, label='Confirmed', color='blue')
# Plotting Deaths
plt.bar(kenya_daily_cases['Date'] - pd.to_timedelta(bar_width, unit='d'), kenya_daily_cases['Deaths'], width=bar_width, label='Deaths', color='red')
# Plotting Recovered cases
plt.bar(kenya_daily_cases['Date'], kenya_daily_cases['Recovered'], width=bar_width, label='Recovered', color='green')
# Plotting Active cases
plt.bar(kenya_daily_cases['Date'] + pd.to_timedelta(bar_width, unit='d'), kenya_daily_cases['Active'], width=bar_width, label='Active', color='orange')
# Plotting New cases
plt.bar(kenya_daily_cases['Date'] + pd.to_timedelta(bar_width * 2, unit='d'), kenya_daily_cases['New cases'], width=bar_width, label='New cases', color='purple')
# Plotting New deaths
plt.bar(kenya_daily_cases['Date'] + pd.to_timedelta(bar_width * 3, unit='d'), kenya_daily_cases['New deaths'], width=bar_width, label='New deaths', color='brown')
# Plotting New recovered
plt.bar(kenya_daily_cases['Date'] + pd.to_timedelta(bar_width * 4, unit='d'), kenya_daily_cases['New recovered'], width=bar_width, label='New recovered', color='pink')
# Set plot properties
plt.title('Daily COVID-19 Metrics in Kenya')
plt.xlabel('Date')
plt.ylabel('Count')
plt.legend()
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Show the plot
plt.show()

```



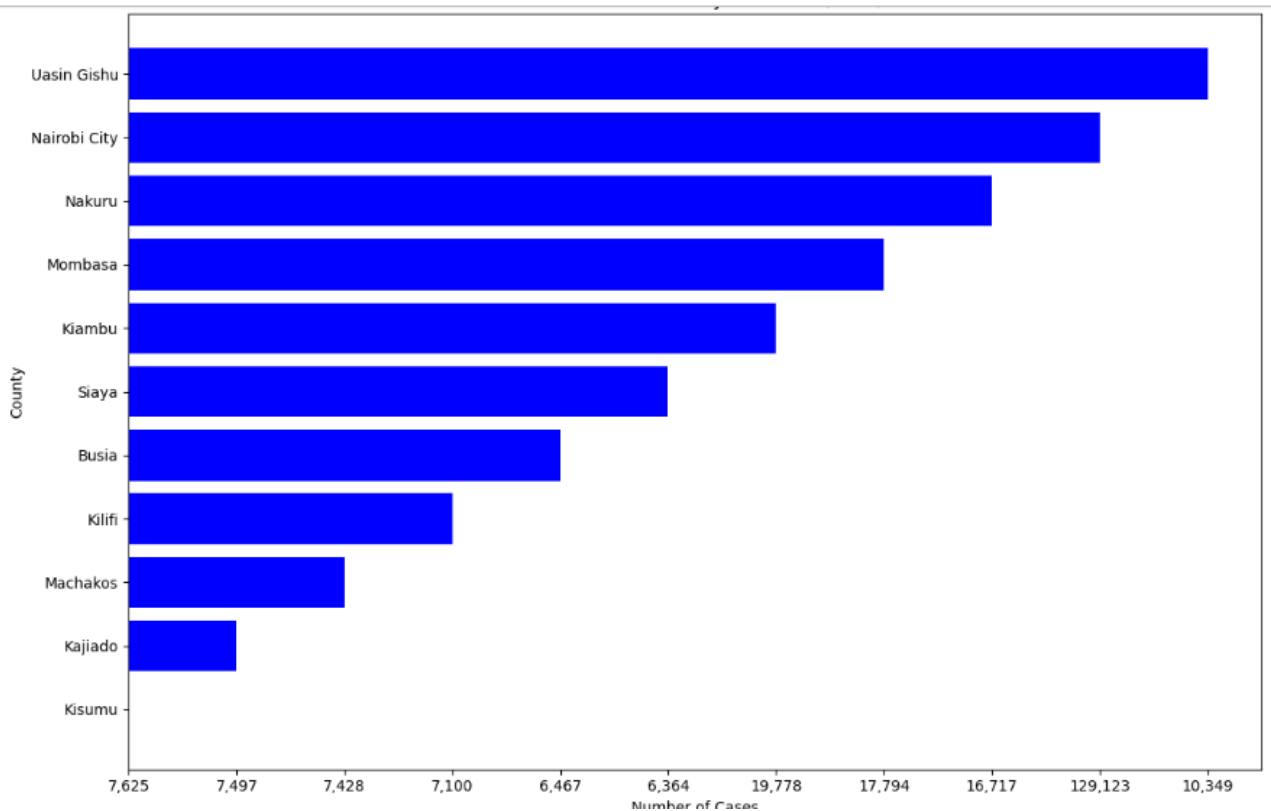
```
# Plotting the horizontal bar graph for COVID-19 cases in Kenya counties
plt.figure(figsize=(12, 8))

# Sort the DataFrame by Cases
kenya_county_cases_sorted = kenya_county_cases.sort_values(by='Cases', ascending=False)

# Plotting the horizontal bar graph
plt.barh(kenya_county_cases_sorted['County'], kenya_county_cases_sorted['Cases'], color='blue')

# Set plot properties
plt.title('COVID-19 Cases in Kenya Counties (2022)')
plt.xlabel('Number of Cases')
plt.ylabel('County')
plt.tight_layout()

# Show the plot
plt.show()
```

```
# Sort the DataFrame by Confirmed cases
east_africa_data_sorted = east_africa_data.sort_values(by='Confirmed', ascending=False)

# Plotting the grouped bar graph
plt.figure(figsize=(12, 8))

bar_width = 0.2
bar_positions = range(1, len(east_africa_data_sorted) * 2 + 1, 2)

# Plotting Confirmed cases
plt.bar(bar_positions, east_africa_data_sorted['Confirmed'], width=bar_width, label='Confirmed', color='blue')

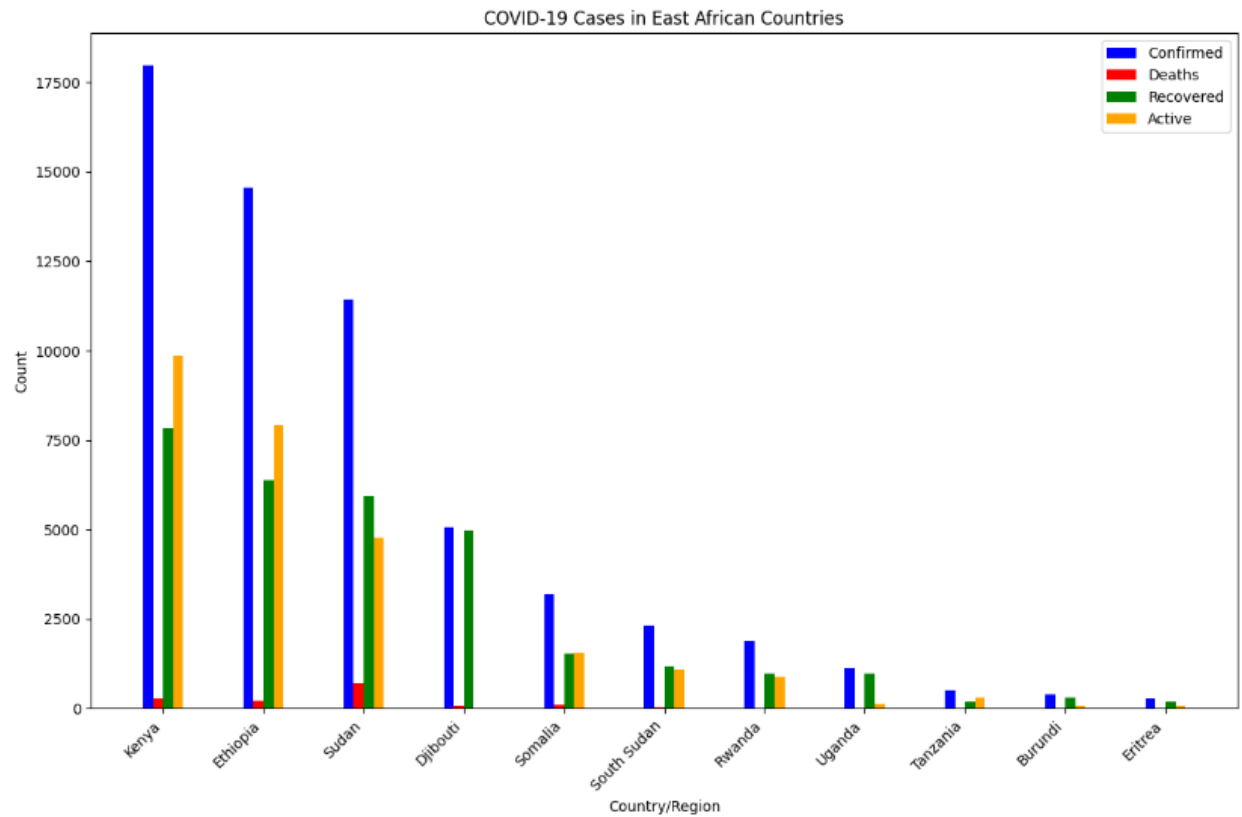
# Plotting Deaths
plt.bar([pos + bar_width for pos in bar_positions], east_africa_data_sorted['Deaths'], width=bar_width, label='Deaths', color='red')

# Plotting Recovered cases
plt.bar([pos + 2 * bar_width for pos in bar_positions], east_africa_data_sorted['Recovered'], width=bar_width, label='Recovered', color='green')

# Plotting Active cases
plt.bar([pos + 3 * bar_width for pos in bar_positions], east_africa_data_sorted['Active'], width=bar_width, label='Active', color='orange')

# Set plot properties
plt.title('COVID-19 Cases in East African Countries')
plt.xlabel('Country/Region')
plt.ylabel('Count')
plt.xticks([pos + 1.5 * bar_width for pos in bar_positions], east_africa_data_sorted['Country/Region'], rotation=45, ha='right')
plt.legend()
plt.tight_layout()

# Show the plot
plt.show()
```



```
# Feature engineering: Extracting day of the year as a numerical feature
kenya_daily_cases['Date'] = pd.to_datetime(kenya_daily_cases['Date'])
kenya_daily_cases['DayOfYear'] = kenya_daily_cases['Date'].dt.dayofyear

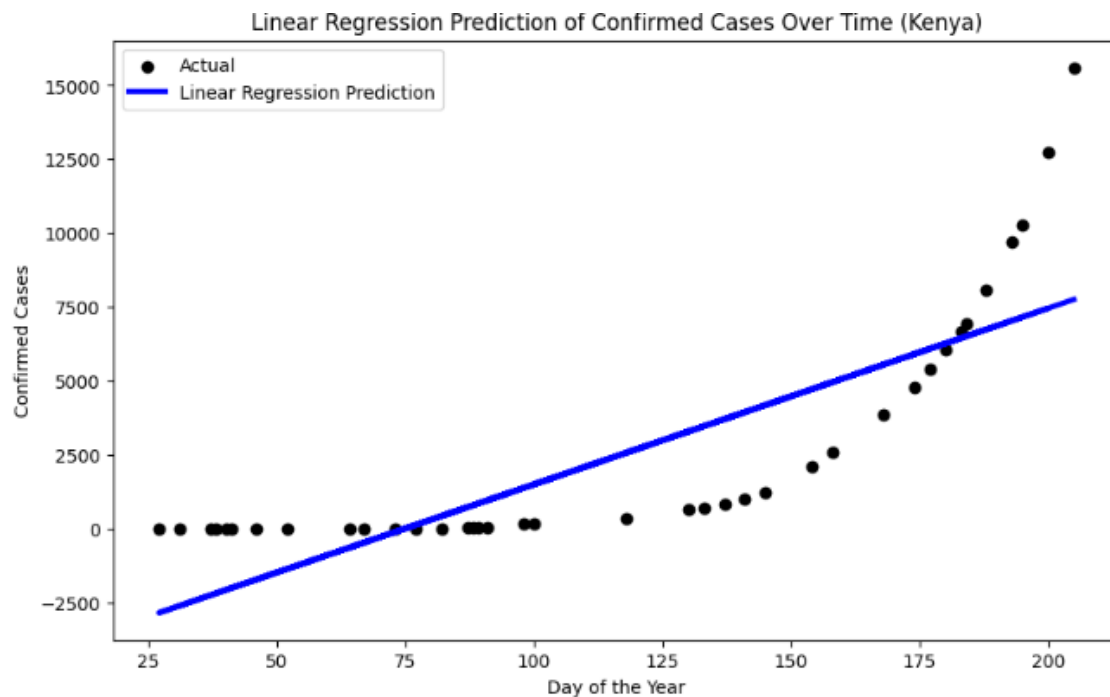
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    kenya_daily_cases[['DayOfYear']],
    kenya_daily_cases['Confirmed'],
    test_size=0.2,
    random_state=42
)

# Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Visualize the predictions
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='black', label='Actual')
plt.plot(X_test, y_pred, color='blue', linewidth=3, label='Linear Regression Prediction')
plt.title('Linear Regression Prediction of Confirmed Cases Over Time (Kenya)')
plt.xlabel('Day of the Year')
plt.ylabel('Confirmed Cases')
plt.legend()
plt.show()
```



This example uses a linear regression model based on the day of the year to predict the number of confirmed cases for Kenya. The code also calculates the Mean Squared Error as an evaluation metric.

Potential applications of the interpreted results

The interpreted results from the time series analysis and predictions of COVID-19 cases in Kenya can be applied in various ways to support decision-making, public health efforts, and resource allocation. Here are potential applications of the interpreted results:

- 1. Resource Planning and Allocation** - Governments and healthcare authorities can use the predictions to plan for and allocate resources such as hospital beds, ventilators, medical personnel, and medications in anticipation of future needs.
- 2. Policy Decision Support** - Policy-makers can use the insights gained from the analysis to make informed decisions on public health measures, lockdowns, travel restrictions, and vaccination campaigns.
- 3. Early Warning Systems** - The predictions can serve as part of an early warning system, helping authorities take proactive measures to prevent or mitigate potential surges in COVID-19 cases.
- 4. Healthcare Capacity Planning** - Hospitals and healthcare facilities can use the predictions to plan for surges in patient numbers, ensuring they have the necessary capacity and resources to handle increased demand.

5. Public Awareness and Communication - Communicating the predictions to the public can help raise awareness and encourage adherence to preventive measures during periods of higher risk.

6. Vaccination Campaign Planning - The predictions can aid in planning and optimizing vaccination campaigns by identifying periods of higher risk and targeting specific demographics or regions.

7. Research and Epidemiological Studies - Researchers can use the interpreted results to further investigate patterns and trends, contributing to epidemiological studies that deepen our understanding of the virus's behaviour.

8. International Collaboration - Sharing the predictions with international health organizations and neighboring countries can facilitate collaboration and information exchange, especially in regions with interconnected healthcare systems.

9. Data-Driven Decision-Making - The results support a data-driven approach to decision-making, enabling stakeholders to make evidence-based choices and respond more effectively to the evolving situation.

10. Public Health Messaging - Tailoring public health messaging based on predicted trends can help communicate risks and preventive measures more effectively to the public.

Predictions are inherently uncertain, and various factors can influence the accuracy of the model. Therefore, the results should be used as a supportive tool alongside other data and expert judgment in the decision-making process. Regular model evaluation and validation against new data are crucial to maintaining the model's relevance and accuracy over time.