Data Analytics with Python

Listing 1 – Python script (COVID-19 Data Analysis with Pandas and Matplotlib)

```
"""
COVID-19 Data Analysis with Python
Author:  Jens Kolby
Course: MSc Cyber Security, unit 5 Data Analytics with Python and Data Storage
reflection
Python version: 3.13.5
Date: September 2025

Instructions for use:
1. Download the dataset "owid-covid-data.csv" from Our World in Data:
   https://covid.ourworldindata.org/data/owid-covid-data.csv
2. Save the file as "covid19_data.csv" in the same folder as this script.
3. Install required libraries if not already installed:
   pip install pandas matplotlib
4. Run the script:
   python Unit_5__Data_Analytics_with_Python.py


Pseudo-code for Unit 5: Data Analytics with Python

1: Import libraries:
- Pandas (data handling)
- Matplotlib (charts)

2: Load dataset:
- Read COVID-19 CSV file
- Parse date column into proper date format (datetime64 in Pandas), not as a plain
string.
  Dates cannot be sorted correctly as strings, and Pandas time-series functions will
not work without parsing.

3: Clean dataset:
- Detect non-numeric columns (Kalita, Bhattacharyya & Roy, 2023, ch. 13.3.1) to
identify data that needs encoding before analysis.
- Encode non-numeric columns (Kalita, Bhattacharyya & Roy, 2023, ch. 13.3.2; Geron,
2022, ch. 2 Handling Text and Categorical Attributes).
- Detect missing values using isnull() (Kalita, Bhattacharyya & Roy, 2023, ch.
13.3.3; Geron, 2019, ch. 2 Data Cleaning).
- Handle missing values with dropna() (Geron, 2019, ch. 2; Huyen, 2022, ch. 3).
dropna() removes rows or columns with NaN.

4: Basic analysis:
- Descriptive statistics: mean, median, std for key columns. Mean = average; Median
= middle value when sorted;
  std (standard deviation) is high when values fluctuate a lot, and low when values
are close to the mean.
- Time series: calculate 7-day rolling averages for new_cases and new_deaths per
country to reduce noise and show the underlying trend.
- Per-100k metrics: calculate total cases and total deaths per 100,000 population.
  Formula: cases_per_100k = (total_cases / population) * 100000. This allows fair
comparison across countries of different sizes.
```

```
- Correlation matrix for selected variables: new_cases, new_deaths,
stringency_index, people_vaccinated_per_hundred,
  people_fully_vaccinated_per_hundred. Values range from -1 (negative) to +1
(positive).
  Note: stringency_index is a composite measure (0-100) of government response
strictness (school closures, workplace restrictions, travel bans).

5: Visualisations:
- Line chart: 7-day rolling average of new_cases for selected countries over time.
Add title, axis labels (X = Date, Y = new_cases 7-day avg), and legend.
- Bar chart: total cases per 100k per country. Add title and axis labels.

"""

# 1 : Import libraries
import pandas as pd                    # data handling (Kalita et al., 2023, ch.
1.2))
import matplotlib.pyplot as plt
import os, sys

print("CWD:", os.getcwd())
print("__file__:", os.path.abspath(__file__))

print(pd.__version__)           # print pandas version)
print("Pandas is imported and ready to use.")

# 2: Load dataset
df = pd.read_csv("covid19_data.csv") # load COVID-19 dataset
df["date"] = pd.to_datetime(df["date"], errors="coerce")  # parse date column to
datetime format and invalid values become NaT

print(df.shape) # print dataset shape (rows, columns)
print(df.dtypes) # print data types of each column)
print(df.head()) # print first 5 rows of the dataset
# 3: Clean dataset


# Detect missing values (Kalita et al., 2023, ch. 13.3.3; Geron, 2019, ch. 2, Data
Cleaning))
print(df.isnull().sum()) # print count of missing values per column)

# Ensure numeric types for key columns used later
df["new_cases"]  = pd.to_numeric(df["new_cases"], errors="coerce")   # invalid
values become NaN
df["new_deaths"] = pd.to_numeric(df["new_deaths"], errors="coerce")


# Handle missing values: drop rows with NaN (Huyen, 2022)
#  dropna(): keep rows only if essential fields for time series exist

df = df.dropna(subset=["location", "date", "new_cases", "new_deaths"])

# Remove negative values in key metrics (Kalita et al., 2023, ch. 13.3.1))
df = df[(df["new_cases"] >= 0) & (df["new_deaths"] >= 0)]

# One-hot encode continent AFTER cleaning (only if column exists)
if "continent" in df.columns:
```

```
    df = pd.get_dummies(df, columns=["continent"], drop_first=True)  # 1 if in
continent, else 0

# Sort before rolling/plotting
df = df.sort_values(["location", "date"])  # ensure correct order for rolling
windows


# 4: Basic analysis
# Descriptive statistics: mean, median, std for key columns (Kalita et al., 2023,
ch. 1.2))
print("Mean new_cases:", df["new_cases"].mean())
print("Median new_cases:", df["new_cases"].median())
print("Std new_cases:", df["new_cases"].std())

# Time series: 7-day rolling average for new_cases and new_deaths per country
(Kalita et al., 2023, ch. 12.4.2))
# First sort by location and date to ensure correct rolling calculation

df = df.sort_values(by=["location", "date"]) # sort by location and date

# Compute rolling means per country
for country in df["location"].unique(): # loop through each unique country
    country_mask = df["location"] == country # create a mask for the current country

    # 7-day rolling mean for new_cases
    # looping through each country to calculate rolling mean
    # saving the results into two new columns: new_cases_7d and new_deaths_7d
    df.loc[country_mask, "new_cases_7d"] = df.loc[country_mask,
"new_cases"].rolling(7).mean().values
    # 7-day rolling mean for new_deaths
    df.loc[country_mask, "new_deaths_7d"] = df.loc[country_mask,
"new_deaths"].rolling(7).mean().values

# Per 100K population metrics (Huyen, 2022)
df["cases_per_100K"] = (df["total_cases"] / df["population"]) * 100000      # cases
per 100K population
df["deaths_per_100K"] = (df["total_deaths"] / df["population"]) * 100000    # deaths
per 100K population

# Correlation matrix for selected variables (Geron, 2019, ch. 2))
# Correlation matrix (only use columns that actually exist)
cols = [
    "new_cases",
    "new_deaths",
    "stringency_index",
    "people_vaccinated_per_hundred",
    "people_fully_vaccinated_per_hundred",
]

present = [c for c in cols if c in df.columns]
missing  = [c for c in cols if c not in df.columns]
if missing:
    print("Missing columns (skipped):", missing)

# Ensure numeric (coerce non-numeric to NaN) before corr
corr = df[present].apply(pd.to_numeric, errors="coerce").corr()
print(corr)
```

```python
# 5: Visualisations

# Line chart: 7-day rolling average of new_cases for selected countries over time
(Kalita et al., 2023, ch. 12.4.2))
wanted = ["Denmark", "United Kingdom", "Saudi Arabia", "South Africa", "United
States"]
present_countries = [c for c in wanted if c in df["location"].unique()]
if not present_countries:
    raise ValueError("None of the selected countries are present in the dataset.")

# Ensure rolling columns exist (if not already created)
for col, base in [("new_cases_7d", "new_cases"), ("new_deaths_7d", "new_deaths")]:
    if col not in df.columns:
        df[col] = (
            df.groupby("location", observed=True)[base]
              .rolling(window=7, min_periods=1)
              .mean()
              .reset_index(level=0, drop=True)
        )

# --- LINE: new_cases ---
plt.figure(figsize=(12, 6))
plotted_any = False
for country in present_countries:
    cd = df[df["location"] == country]
    y = cd["new_cases_7d"]
    if y.notna().any():
        plt.plot(cd["date"], y, label=country)
        plotted_any = True
    else:
        print(f"[INFO] No valid new_cases_7d values for {country} - skipped.")

if not plotted_any:
    raise ValueError("No lines plotted for new_cases (all values were NaN).")

plt.title("7-Day Rolling Average of New COVID-19 Cases")
plt.xlabel("Date")
plt.ylabel("New Cases (7-day avg)")
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig("line_chart_new_cases.png")
plt.show()
plt.close()

# --- LINE: new_deaths ---
plt.figure(figsize=(12, 6))
plotted_any = False
for country in present_countries:
    cd = df[df["location"] == country]
    y = cd["new_deaths_7d"]
    if y.notna().any():
        plt.plot(cd["date"], y, label=country)
        plotted_any = True
    else:
        print(f"[INFO] No valid new_deaths_7d values for {country} - skipped.")

if not plotted_any:
```

```
    raise ValueError("No lines plotted for new_deaths (all values were NaN).")

plt.title("7-Day Rolling Average of New COVID-19 Deaths")
plt.xlabel("Date")
plt.ylabel("New Deaths (7-day avg)")
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig("line_chart_new_deaths.png")
plt.show()
plt.close()

# --- BAR: cases per 100K for selected countries ---
if "cases_per_100K" in df.columns:
    summary = (
        df[df["location"].isin(present_countries)]
        .groupby("location", observed=True)["cases_per_100K"]
        .max()
        .sort_values(ascending=False)
    )
    plt.figure(figsize=(10, 6))
    summary.plot(kind="bar")
    plt.title("Total COVID-19 Cases per 100K Population (Selected Countries)")
    plt.xlabel("Country")
    plt.ylabel("Cases per 100K")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig("bar_chart_cases_per_100K_selected.png")
    plt.show()
    plt.close()
else:
    print("Column 'cases_per_100K' not found - skipping bar chart.")
```

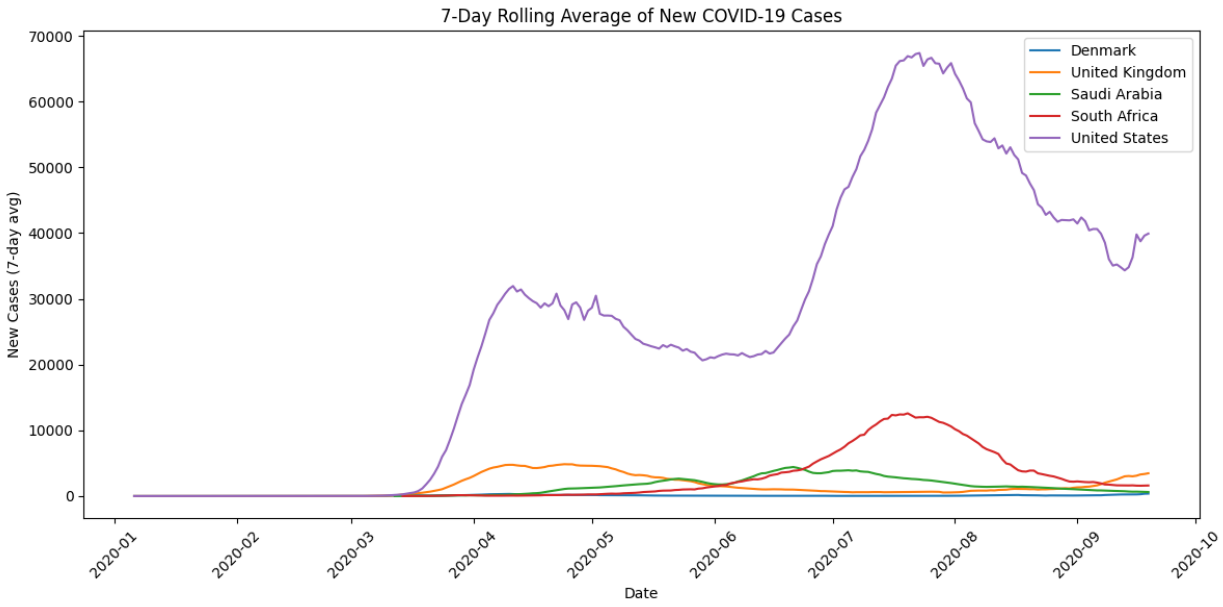
Use of AI Tools

I created my own draft code first, but encountered issues with syntax, encoding, and structuring the visualization section. I used ChatGPT as a support tool for debugging and refactoring. It suggested improvements, which I implemented to make the code runnable.

Visual Output

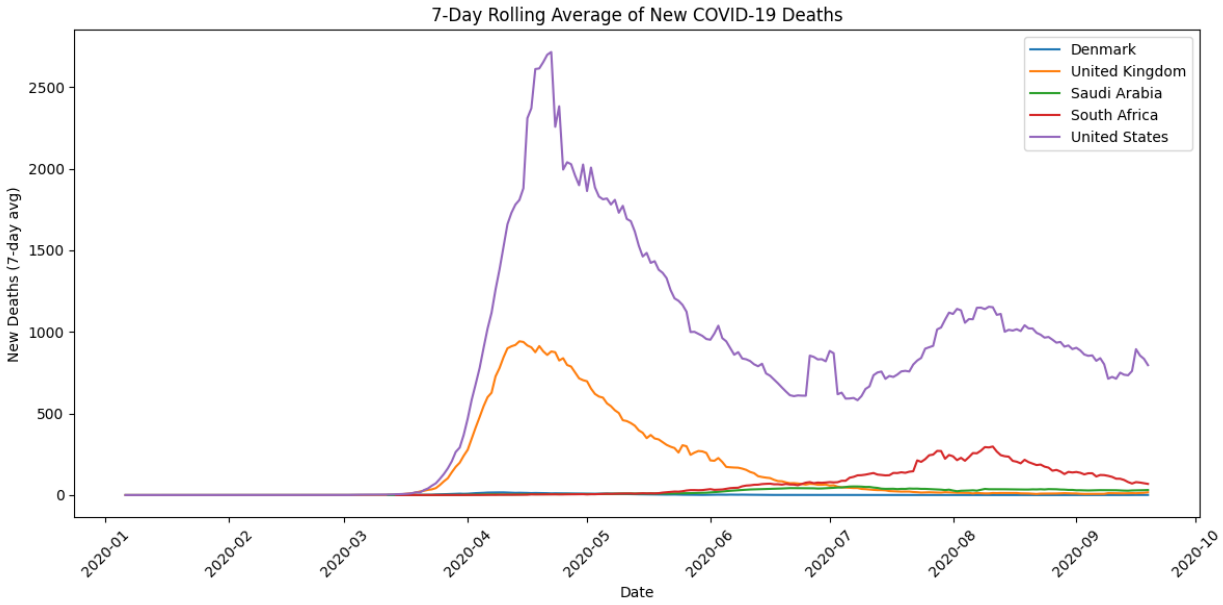
The following output is presented in graphical form, generated using the Matplotlib library. Two line charts display the 7-day rolling averages of new COVID-19 cases and new deaths for a selection of countries. In addition, a bar chart illustrates the total number of cases per 100K population across the same countries.

Figure 1 – 7 Day Rolling Average of New COVID-19 Cases



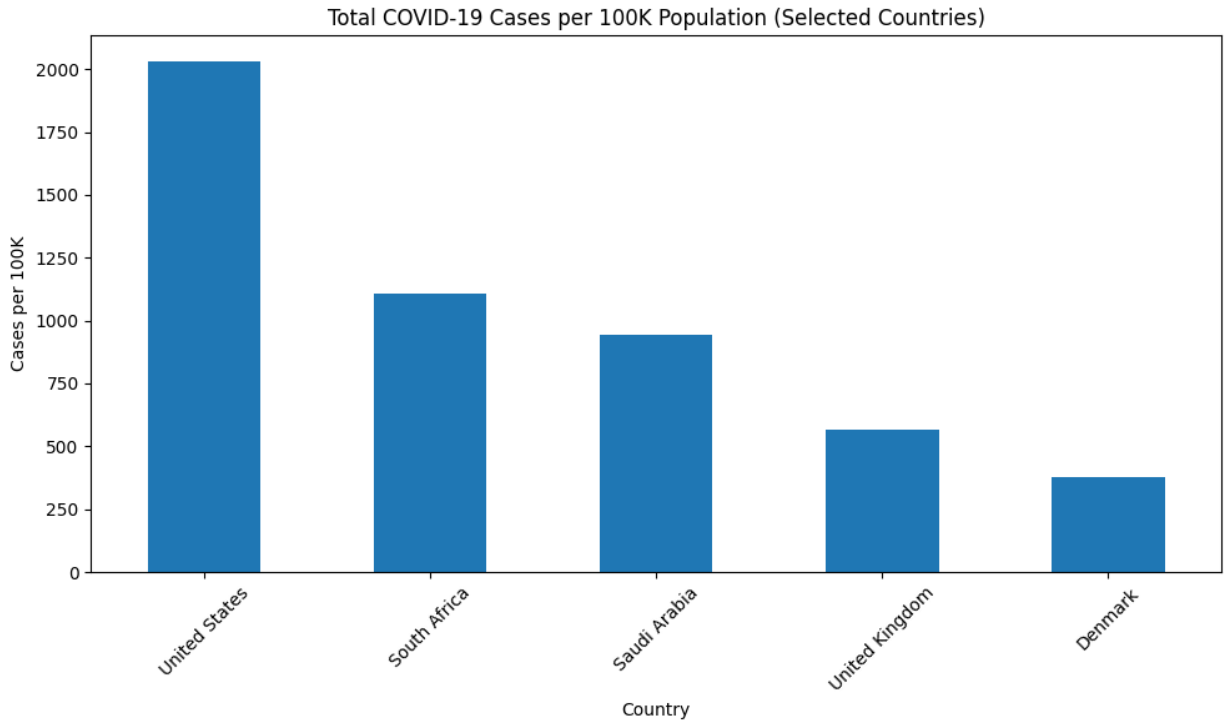
The figure shows that the US experienced the highest number of new cases during the period. There was a sharp increase in mid-2020. The UK, South Africa, and South Africa also show clear peaks in the same period, while Denmark remains at a lower level throughout.

Figure 2 – 7-Day Rolling Average of New COVID-19 Deaths



The figure shows that the US had by far the highest number of daily deaths during the period, with a steep rise in March 2020 followed by multiple peaks later in the year. The UK also experienced a significant peak in April 2020, while South Africa shows clear peaks in mid-2020. In contrast, Denmark and Saudi Arabia remained at low levels throughout.

Figure 3 – Total COVID-19 Cases per 100K Population



The bar chart shows that the US had the highest number of cases per 100K population, significantly higher than Denmark and the UK. South Africa and Saudi Arabia recorded intermediate levels.

Reflection – SQL vs NoSQL

When working with datasets such as the COVID-19 dataset, it is crucial to select the right storage model, as this choice has a significant impact on the efficiency, scalability, and the type of analysis that can be performed. A relational database, such as SQL, MySQL, or PostgreSQL, stores data in structured tables with well-defined schemas. This approach fits the COVID-19 dataset particularly well, since it contains clearly defined attributes such as date, location, new_cases, new_deaths, and population. Queries can be executed efficiently using joins, aggregations, and filters. Moreover, SQL ensures data integrity through constraints, which is critical when handling time-series data across multiple countries. This is especially important because even small inconsistencies, such as missing dates, negative values, or duplicated entries, can distort rolling averages, correlations, and long-term trends.

NoSQL databases, such as MongoDB, CouchDB, Redis, Cassandra, and others, are designed to handle huge volumes of data with greater flexibility. They allow records to be stored in JSON-like documents, making it easier to adapt when data structures change or new variables are added. This flexibility and high scalability make NoSQL well-suited for applications that require rapid processing and analysis of evolving datasets (Khan et al., 2019). However, NoSQL sacrifices strict consistency, which makes it less ideal for structured statistical analysis of the COVID-19 dataset.

References

Géron, A., 2019. *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd ed. Sebastopol: O'Reilly Media.

Huyen, C., 2022*. Designing Machine Learning Systems*. Sebastopol: O'Reilly Media.

Kalita, J.K., Bhattacharyya, D.K., and Roy, S. 2023. *Fundamentals of Data Science: Theory and Practice.* Cham: Springer.

Khan, S., Liu, X., Ali, S.A. and Alam, M., 2019. *Storage solutions for big data systems: A qualitative study and comparison.* Available at: https://arxiv.org/pdf/1904.11498 [Accessed 1 September 2025].