FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION

“NATIONAL RESEARCH UNIVERSITY HIGHER

SCHOOL OF ECONOMICS”

MOSCOW INSTITUTE OF ELECTRONICS AND MATHEMATICS

TECHNICAL SPECIFICATION

" Python in Data Science"

Analysis of COVID-19 statistics in different countries

DEVELOPER’s GUIDE

Student:

Mohamed Abdelhafiz Salem Abdelhafiz Youssef

Email: abmokhamed@edu.hse.ru

Phone number: +79067678234

Supervisor: Polyakov Konstantin Lvovich

Content table

List of figures

1. Install Prerequisites

Python 3.10

Check if Python is already installed in CMD:

- python --version on Windows
- python3 --version on Linux/MacOs

Something like **Python 3.10.6** should appear

If not installed:

Download the latest stable version (preferably Python 3.10 at least)

Download it from: https://www.python.org/downloads/

Run the installer and **check the box that says "Add Python to PATH"** before clicking "Install Now"

Anaconda

Check if Anaconda is already installed in CMD:

- conda --version

Something like **conda 24.x.x** should appear

If not installed:

Download it from: https://www.anaconda.com/download

Run the installer and accept all default settings.

MySQL + MySQL Workbench

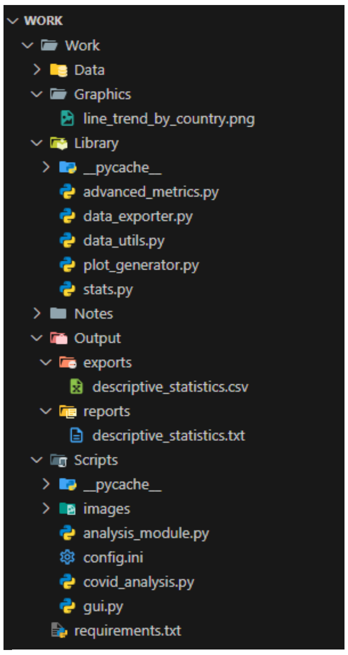
Download it from: https://dev.mysql.com/downloads/

Download the one complete version of the installer (300mb+)

Run the installer and select the **FULL** installation and continue with execute and install, make sure to remember the **password**, the **port** and the **user**.

2. Project

After downloading the project ZIP file, Find the downloaded ZIP file (Work.zip) in your **Downloads** folder or wherever you saved it and extract it. The structure of the folder is the following:



```
WORK
  Work
    > Data
    v Graphics
        line_trend_by_country.png
    v Library
      > __pycache__
        advanced_metrics.py
        data_exporter.py
        data_utils.py
        plot_generator.py
        stats.py
    > Notes
    v Output
      v exports
          descriptive_statistics.csv
      v reports
          descriptive_statistics.txt
    v Scripts
      > __pycache__
      > images
        analysis_module.py
        config.ini
        covid_analysis.py
        gui.py
      requirements.txt
```

Figure 1 - Project Structure

Remarque: The data folder contains the daily reports csv files of all countries during the covid period, one thing you must be sure of is: when using other csv data files, make sure it has the same structure as the csv files inside the folder, also the name of these files must be DD-MM-YYYY.

3. Virtual Environment

After extracting the Zip file, Open **anaconda Prompt,** we will use the **Anaconda Navigator** later.

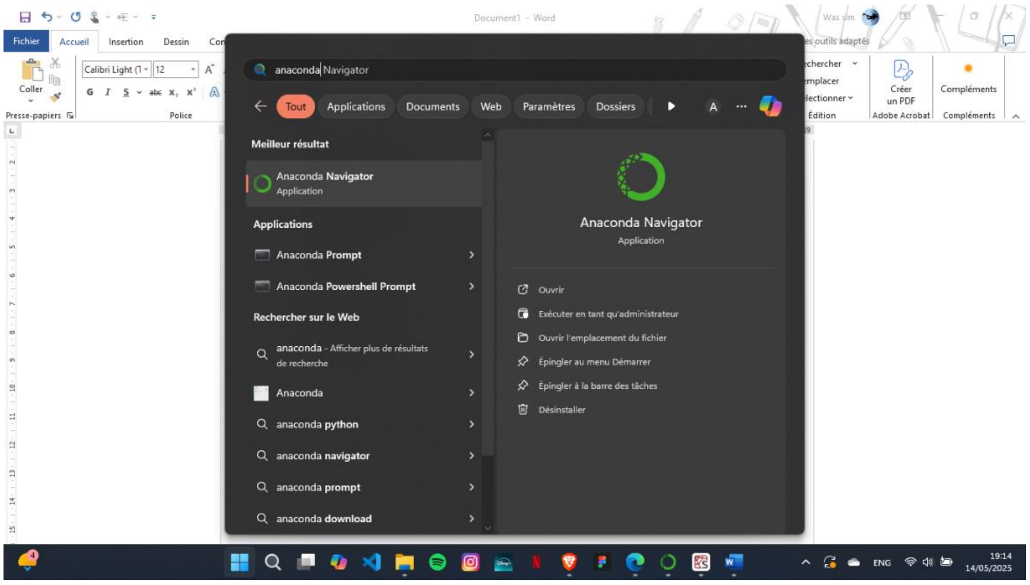
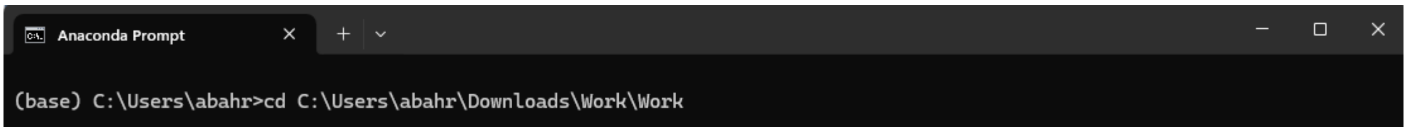


Figure 2 - Anaconda Prompt

Access the path to your extracted Work folder. If the folder was extracted in Downloads, it should be something like: C:\Users\abahr\Downloads\Work\Work

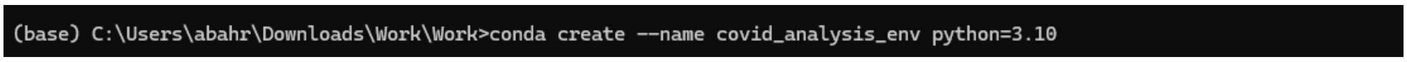
- Access Folder: cd C:\Users\abahr\Downloads\Work\Work



```
Anaconda Prompt

(base) C:\Users\abahr>cd C:\Users\abahr\Downloads\Work\Work
```

Figure 3 - Working Folder

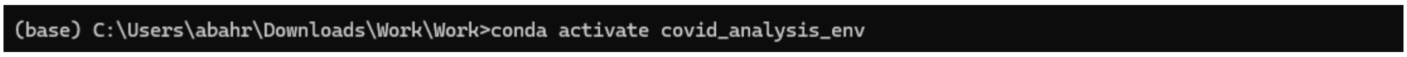
- Create virtual environment: conda create --name covid_analysis_env python=3.10



```
(base) C:\Users\abahr\Downloads\Work\Work>conda create --name covid_analysis_env python=3.10
```

Figure 4 - Virtual environment creation

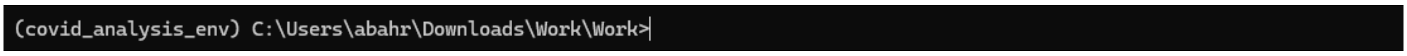
- Activate it: conda activate covid_analysis_env



```
(base) C:\Users\abahr\Downloads\Work\Work>conda activate covid_analysis_env
```

Figure 5 - Environment activation

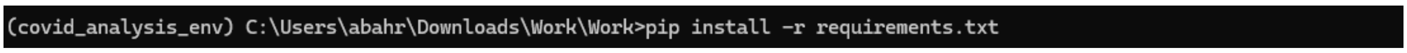
  - **Base** will change into **covid_analysis_env** which means the environment is activated



```
(covid_analysis_env) C:\Users\abahr\Downloads\Work\Work>
```

Figure 6 - Environment activated

- Install all dependencies: pip install-r requirements.txt



```
(covid_analysis_env) C:\Users\abahr\Downloads\Work\Work>pip install -r requirements.txt
```

Figure 7 - Install dependencies

Now Open Anaconda Navigator and choose the environment we just created

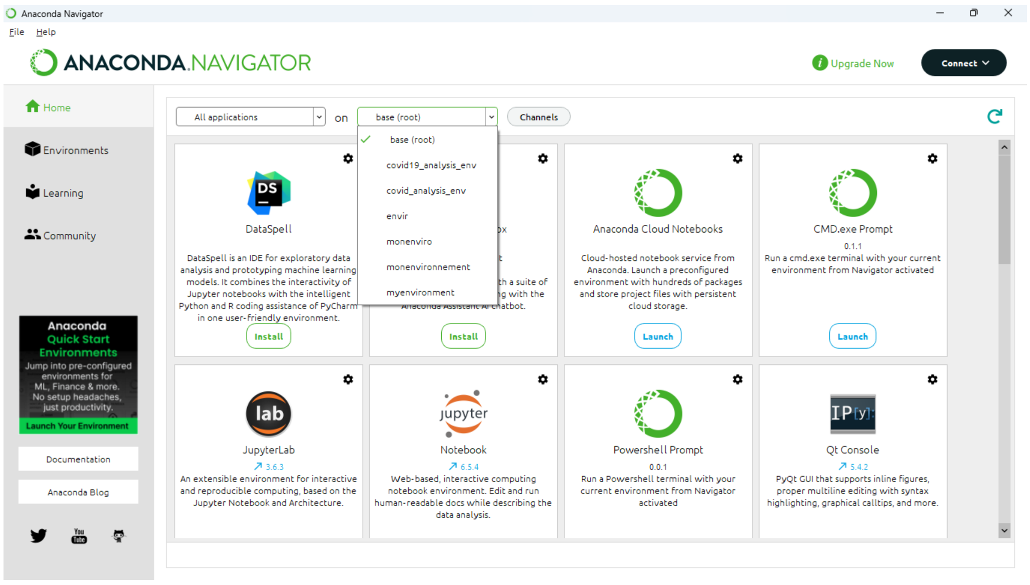


Figure 8 - Environment selection

Search for Spyder, Install It and launch it

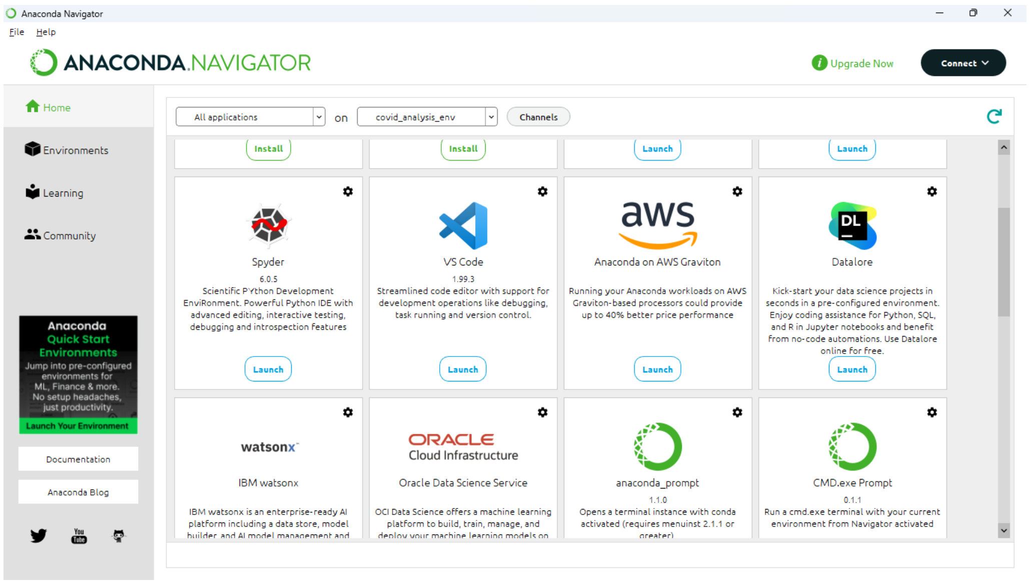
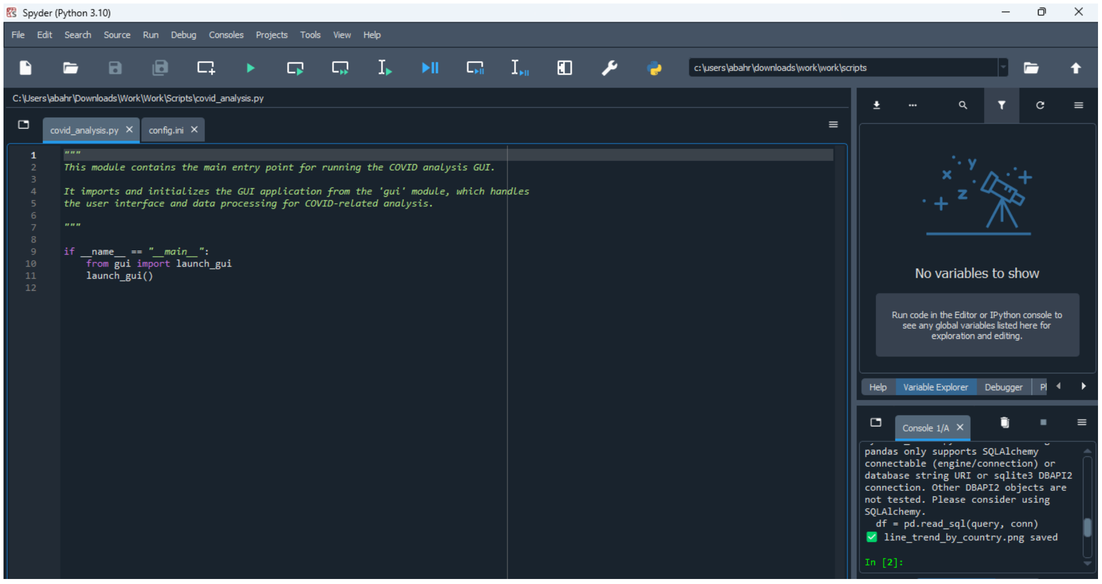


Figure 9 - Spyder installation and launch

Inside Spyder Open the **covid_analysis.py** and the **config.ini** files from **Work/Scripts folder**


Spyder (Python 3.10)
File Edit Search Source Run Debug Consoles Projects Tools View Help
c:\users\abahr\downloads\work\work\scripts
C:\Users\abahr\Downloads\Work\Work\Scripts\covid_analysis.py
covid_analysis.py
config.ini
No variables to show
Run code in the Editor or IPython console to see any global variables listed here for exploration and editing.
Help
Variable Explorer
Debugger
Console 1/A

```python
"""
This module contains the main entry point for running the COVID analysis GUI.

It imports and initializes the GUI application from the 'gui' module, which handles
the user interface and data processing for COVID-related analysis.

"""

if __name__ == "__main__":
    from gui import launch_gui
    launch_gui()
```

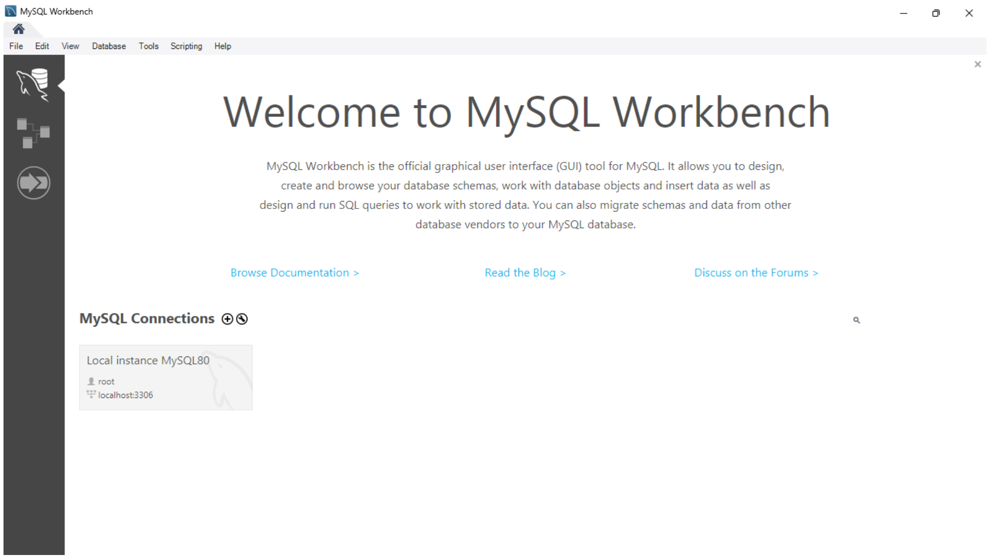
```
pandas only supports SQLAlchemy
connectable (engine/connection) or
database string URI or sqlite3 DBAPI2
connection. Other DBAPI2 objects are
not tested. Please consider using
SQLAlchemy.
  df = pd.read_sql(query, conn)
✅ line_trend_by_country.png saved

In [2]:
```

Figure 10 - Spyder IDE

4. Database

Now after Opening **covid_analysis.py** and **config.ini** in spyder we need to configure the **database** information's so:

- Open MySQL Workbench

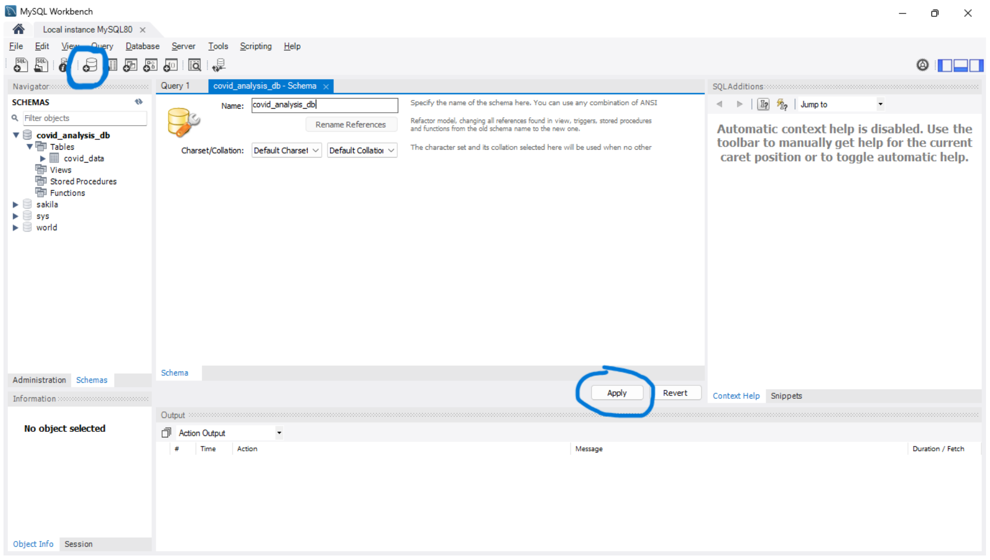


Welcome to MySQL Workbench

MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

Figure 11 - MySQL Workbench

- At the bottom you can see MySQL localhost connection click on it, it will ask for **user** and **password**, the ones you used during installation, and then it will take you into this interface
- Create a new database schema and name it **covid_analysis_db,** it's empty for now.



Name: covid_analysis_db

Specify the name of the schema here. You can use any combination of ANSI

Refactor model, changing all references found in view, triggers, stored procedures and functions from the old schema name to the new one.

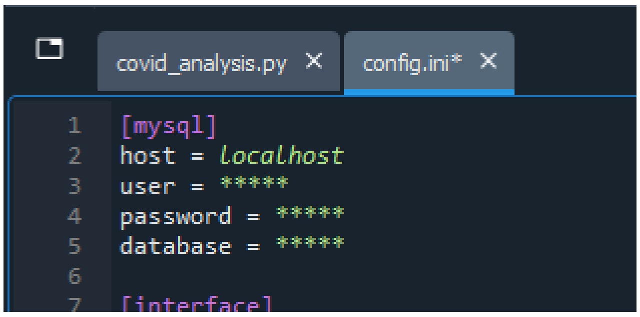
Charset/Collation: Default Charset Default Collation

The character set and its collation selected here will be used when no other

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Figure 12 - Database creation

- Now we go back to spyder to update the **config.ini** file for now it's something like this, fill the variables with the right information's: **password** and **user** used during **installation**, and the **name** of the database we just created

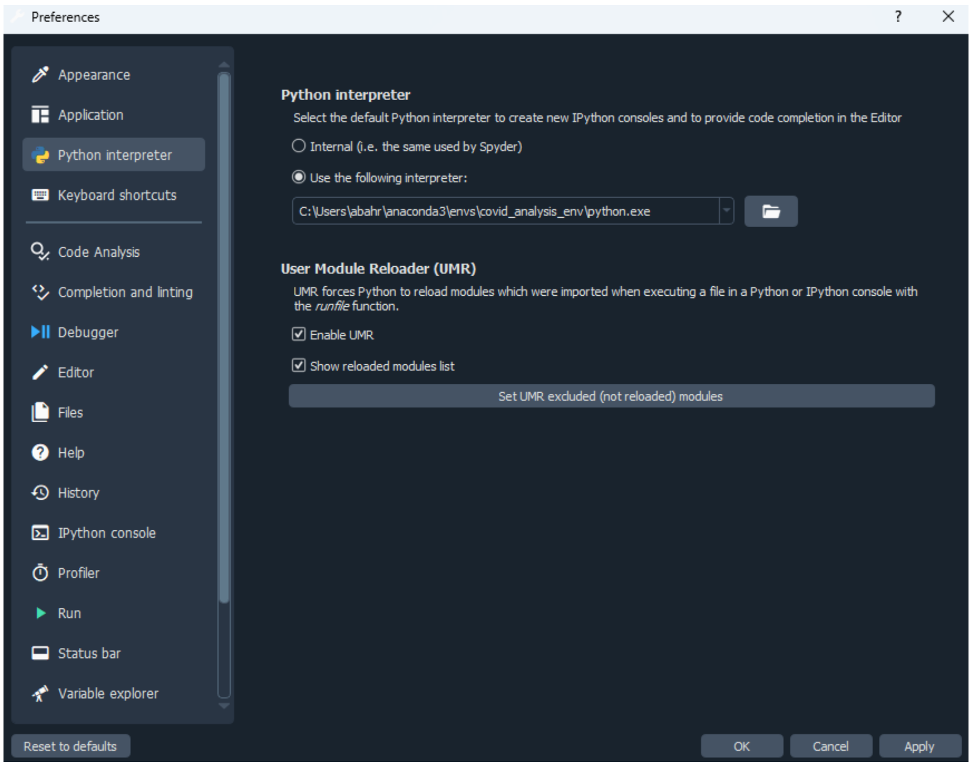


```
[mysql]
host = localhost
user = *****
password = *****
database = *****

[interface]
```

Figure 13 - MySQL configuration

5. Python Interpreter

Inside Spyder in **Tools>Preferences>Python Interpreter** choose the one used by the environment we created previously. (In our example **covid_analysis_env\python.exe)**. Click apply, maybe you will need to restart Spyder for the change to take action



Python interpreter

Select the default Python interpreter to create new IPython consoles and to provide code completion in the Editor

○ Internal (i.e. the same used by Spyder)

● Use the following interpreter:

C:\Users\abahr\anaconda3\envs\covid_analysis_env\python.exe

User Module Reloader (UMR)

UMR forces Python to reload modules which were imported when executing a file in a Python or IPython console with the *runfile* function.

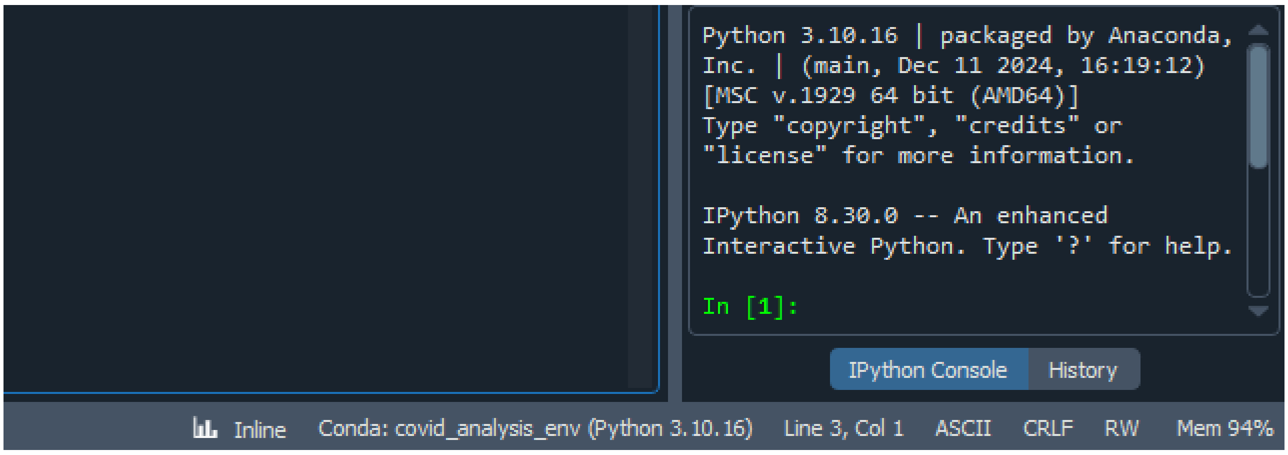
☑ Enable UMR

☑ Show reloaded modules list

Set UMR excluded (not reloaded) modules

Figure 14 - Python interpreter

You should see the environment we are using at the bottom of Spyder, make sure it's the right one:



```
Python 3.10.16 | packaged by Anaconda,
Inc. | (main, Dec 11 2024, 16:19:12)
[MSC v.1929 64 bit (AMD64)]
Type "copyright", "credits" or
"license" for more information.

IPython 8.30.0 -- An enhanced
Interactive Python. Type '?' for help.

In [1]:
```

Inline   Conda: covid_analysis_env (Python 3.10.16)   Line 3, Col 1   ASCII   CRLF   RW   Mem 94%

Figure 15 - Environment check

6. Run the Program

Now go to the covid_analysis.py file opened in Spyder and run it and the GUI must appear

7. Developer's Guide

Work\Work\Scripts\covid_analysis.py

First and foremost, the file from which the program runs is **covid_analysis.py**, by calling the **launch_gui()** function from **gui.py** file within the same folder.

Work\Work\Library\data_utils.py

connect_to_database()

- Establishes and returns a connection to the MySQL database using credentials from the config.ini file.
- This connection is reused throughout the module for database operations.
- Essential for reading from and writing to the covid_data table.

ensure_columns_exist_in_db(conn)

- Creates the covid_data table with predefined columns if it doesn't already exist.
- Ensures that the database schema is ready for data insertion.
- Uses the given MySQL connection to execute the table creation query.

load_all_csvs(folder_path)

- Reads all CSV files from a specified folder, filters by date and country, and processes them into a single DataFrame.
- Only includes data from 2021 to 2023 and for target countries.
- Cleans columns and adds a report_date field from the filename.

clean_data(df)

- Cleans the input DataFrame by filling missing values and converting data types.
- Ensures consistency in numeric and string fields like confirmed, deaths, and recovered.
- Also sorts the data by country and report date for better organization.

insert_data_into_db(df)

- Inserts records from a cleaned DataFrame into the covid_data table in the database.
- Uses a loop to insert rows one by one using parameterized queries.
- Closes all connections after committing the transaction.

load_data_from_db()

- Fetches all records from the covid_data table into a pandas DataFrame.
- Useful for downstream analysis and visualization.
- Ensures a clean connection lifecycle with proper closing.

truncate_covid_data_table()

- Removes all rows from the covid_data table but retains its schema.
- Useful for resetting the database between data reloads or tests.
- Closes all connections cleanly after truncation.

Work\Work\Library\advanced_metrics.py

compare_wave_intensity(df)

- Compares COVID-19 wave intensity across 2021, 2022, and 2023 by country.
- Calculates peak confirmed cases per year and percentage changes between them.
- Returns a DataFrame summarizing yearly trends per country.

calculate_rates(df)

- Calculates average fatality and recovery rates for each country.
- Filters out zero-case entries and computes percentages from totals.
- Returns a summarized DataFrame grouped by country.

describe_cases(df)

- Generates summary statistics for confirmed, death, and recovery cases.
- Includes count, mean, min, max, and quartiles for each metric.
- Returns the statistics in a readable DataFrame format.

generate_pivot(df)

- Creates a pivot table of max confirmed cases per country per year.
- Groups data by country and year using the highest daily totals.
- Returns a wide-format DataFrame suitable for trend comparison.

Work\Work\Library\stats.py

stats_by_country(df)

- Aggregates the final maximum values of confirmed, death, and recovered cases per country.
- Groups data only by country for overall totals.
- Returns a summary DataFrame with country-level stats.

stats_by_month(df)

- Calculates total confirmed, deaths, and recovered cases per country by month.
- Adds a month_year column for grouping.
- Returns a monthly trend summary per country.

stats_by_year(df)

- Summarizes confirmed, death, and recovered cases per country by year.
- Extracts year from the date column for grouping.
- Returns annual statistics grouped by country.

stats_by_date_range(df, start_date, end_date)

- Filters data between two given dates and aggregates max case numbers.
- Groups by country to return pandemic status in a time window.
- Returns a filtered country-level summary DataFrame.

filter_data(df, year, month, country)

- Filters dataset by year, month, and country then aggregates by date.
- Groups by both date and country and computes case totals and location averages.
- Returns a cleaned, filtered dataset with daily granularity.

Work\Work\Library\plot_generator.py

_graphics_output_dir(base_dir)

- Returns the absolute path to the graphics output folder.
- Appends 'Graphics' to the given base directory.
- Used internally by all plotting functions to save the plots inside **Work\Work\Graphics**

correlation_heatmap(df, base_dir)

- Creates a heatmap showing correlation between case types.
- Saves the image as correlation_heatmap.png.
- Highlights relationships among confirmed, deaths, and recovered.

bar_total_cases(df, base_dir)

- Plots total confirmed cases per country in a horizontal bar chart.
- Orders countries by descending number of cases.
- Saves the figure as bar_total_cases.png.

line_trend_by_country(df, base_dir)

- Generates a line chart of monthly case trends by country.
- Groups data by month and country for visualization.
- Saves the output as line_trend_by_country.png.

boxplot_cases_by_month(df, base_dir)

- Creates a boxplot of confirmed case distribution per calendar month.
- Helps visualize monthly spread and outliers.
- Saves the chart as boxplot_cases_by_month.png.

scatter_deaths_vs_cases(df, base_dir)

- Builds a scatter plot comparing deaths and confirmed cases per country.
- Each point represents a country’s totals.
- Saved as scatter_deaths_vs_cases.png.

Work\Work\Library\data_exporter.py

export_to_csv(df, filename, base_dir)

- Exports a DataFrame to a CSV in **Output/exports/**.
- Creates the directory if it doesn't exist.
- Saves the file using the provided filename.

Work\Work\Scripts\analysis_module.py

save_report(df, filename, base_dir, title=None, description=None)

Generates a human-readable text report from a DataFrame.

- Saves to **Output/reports/**.
- Includes optional title/description, timestamp, and summary stats.
- Auto-adds totals and top countries if standard COVID columns exist.

Work\Work\Scripts\gui.py

a. Module Purpose

This module is the Graphical User Interface (GUI) entry point for your COVID-19 Data Analysis app, built using the Tkinter library in Python.

It manages:

- The user interface window
- The visual theming and styling loaded from a config file
- Paths and environment setup for data files, configuration, and output folders
- Imports and connects to core data utilities, statistics, plotting, exporting, and reporting modules that do the actual data work behind the scenes.

b. Imports

Standard Libraries:

- **os & sys:** For file paths and system-level operations.
- **datetime:** To manage and display dates/timestamps.
- tkinter (as tk), ttk, messagebo**x:** Core GUI toolkit and widgets.
- **configparser:** To read and write configuration files (config.ini).
- **PIL (Image, ImageTk):** To handle images inside the GUI (e.g., logos, icons).

c. Path Setup:

sys.path.append(...): Adds the project's root directory to the Python path, so you can import modules from Library and other folders easily.

Library Imports:

- From **Library.data_utils**: Functions to handle database connections, data loading, cleaning, and inserting.
- From **Library.stats**: Functions to compute statistics by country, month, year, and apply data filtering.
- From **Library.advanced_metrics**: Functions for complex COVID-19 metrics, pivots, and wave intensity comparisons.
- From **Library.plot_generator**: Functions that generate various plots like heatmaps, bar charts, line trends, boxplots, and scatterplots.
- From **analysis_module**: The function **save_report** to save tabulated reports.
- From **Library.data_exporter**: The function **export_to_csv** to export DataFrames to CSV files.

d. Key Constants

- **CSV_FOLDER_PATH:** The relative path where your raw COVID-19 daily report CSV files are located (../Data/csse_covid_19_daily_reports/).
- **CONFIG_PATH:** Location of the configuration file (config.ini) that controls GUI appearance and settings.
- **BASE_DIR:** The absolute path to the project's root directory, used for building paths to output folders or other resources.
- **Color & Font Globals**: Variables like FONT_FAMILY, BUTTON_COLOR, SIDEBAR_COLOR etc., are declared but initially set to None. These will be populated by reading from the config file, allowing dynamic customization of the GUI's look and feel.

e. Configuration Management

The GUI supports dynamic theming and interface settings through a .ini config file:

load_config():

- Reads config.ini.
- Ensures there's an [interface] section even if missing (important for default fallback values).
- Returns the entire config object for further processing.

save_config(config):

- Writes changes back to config.ini.
- Used if the user changes theme settings or preferences at runtime.

apply_config():

- Loads the config using load_config().
- Reads specific keys from the [interface] section:
- Font family, font size for main UI and sidebar.
- Colors for report buttons, export buttons, general buttons.
- Sidebar background color and main background color.
- Uses a helper safe_get function to provide default values if keys are missing or empty.
- Assigns all these values to the global variables, so the rest of the GUI code can style widgets accordingly.

f. Output Directory Helper

_graphics_output_dir():

- Constructs and returns an absolute path to the Work/Graphics folder.
- This is where all generated plot images will be saved.
- The function uses relative navigation to go up three directories from the current file's location, then down into the Work/Graphics folder.
- Encapsulating this path logic in a function ensures consistent output location across the app.

g. Main functions

Data_page(page_frame)

This function builds the "Data & Statistics" page of the Tkinter GUI. It provides a user interface to manage the COVID-19 dataset, from raw CSV loading to database insertion.

Layout

- A parent Frame (data_page_frame) is placed within the given page_frame.
- It includes:
  - button_frame: Row of buttons for key data operations.
  - status_label: Displays operation feedback (success/failure).
  - table_frame: Displays the loaded/cleaned data using a ttk.Treeview.

Functionality

- show_table(df): Renders a pandas DataFrame in the Treeview.
- update_status(text, color): Updates the feedback label with messages and colors.

- Button handlers:
    - handle_connect_db: Connects to DB and ensures schema.
    - handle_load_data: Loads all CSVs into memory (data_page.df).
    - handle_clean_data: Cleans the in-memory DataFrame.
    - handle_insert_data: Inserts cleaned data into the database.
    - handle_clear_table: Truncates all rows from the covid_data table.

Data_filtering(page_frame)

This function builds the "Filtered Data" page of the Tkinter GUI. It allows users to:

- Load COVID-19 data from the database.
- Apply filters (Year, Month, Country).
- View the filtered results in a scrollable table.

Layout

- Main layout: filtered_page_frame attached to the given page_frame.
- Contains:
    - filter_frame: Top row with filter controls and buttons.
    - status_label: Operation feedback label.
    - tree_frame: Table display area using a ttk.Treeview with scrollbars.

Key Features

- Data Loading (load_data):
    - Uses load_data_from_db() to fetch all records.
    - Stores the result in data_page.df.
    - Displays feedback via status_label.
- Filtering (apply_filter):
    - Applies selected filter values to data_page.df using filter_data().
    - Replaces Treeview content with the filtered results.
    - Re-formats datetime columns for readability.
- Filter Controls:
    - Year: Dropdown for 2021–2023.
    - Month: Dropdown for 1–12.
    - Country: Dropdown for major predefined countries.

Statistics_page(page_frame)

This function builds the "Statistics" page of the Tkinter GUI. It allows users to:

- Load COVID-19 data from the database.
- Choose from a list of predefined statistical analyses.
- Optionally filter data by a specific date range.
- View results in a scrollable table.
- Export CSV data or generate text-based reports.

Layout

- Main layout: stats_frame attached to the given page_frame.
- Contains:
- filter_frame: Top section with filters and control buttons.
- table_frame: Middle section showing results in a ttk.Treeview.
- button_frame: Displays export/report buttons after generation.
- status_label: Footer feedback line for status updates.

Key Features

- Data Loading (load_data)
    - Uses load_data_from_db() to fetch all records.
    - Saves the result to data_page.df.
    - Displays row count in status_label.
- Statistic Selection (handle_stat_selection): Triggered via the Generate button. Performs the following:
    - Checks for loaded data.
    - Reads selected statistic from stat_var.
    - Runs the corresponding function:
        - Country Stats: stats_by_country()
        - Rates: calculate_rates()
        - Descriptive: describe_cases()
        - Pivot: generate_pivot()
        - Month Stats: stats_by_month()
        - Year Stats: stats_by_year()
        - Wave Intensity: compare_wave_intensity()
        - Date Range: stats_by_date_range() (with date inputs)
    - Results are displayed in the Treeview.
    - Status is updated with the number of generated rows.
    - Export and Report buttons appear dynamically.
- Treeview Table (show_table)
    - Dynamically rebuilt on each statistic generation.
    - Columns set from the DataFrame headers.
    - Values inserted row-by-row.

Filter Controls

- Statistic dropdown: Predefined list of analysis types.
- Date Range inputs: Only shown if "Date Range" is selected.
- Buttons:
    - Load Data: Fetch database records.
    - Generate: Run selected statistical function.
    - Export: Export result to CSV.
    - Generate Report: Save report in TXT format.
- Toggle Behavior
    - The date entry fields are shown/hidden based on dropdown selection using stat_var.trace.

Status Label : Provides real-time feedback for:

- Data loading success.
- Statistic generation results.
- Export/report status.
- Input errors (e.g., invalid date format).

Visualization_page(page_frame)

This function builds the Visualization page of the Tkinter GUI. It enables users to:

- Load COVID-19 data from the database.
- Select from multiple predefined plot types.
- Generate plots and display the resulting images within the GUI.
- View status messages for data loading and plot generation.

Layout

- visualization_page_frame: Main container frame inside the given page_frame, with padding and background color.
- filter_frame: Top row container inside visualization_page_frame for controls.
- status_label: Displays real-time status messages below filters.
- img_label: Dynamically created label to show generated plot images.

Key Features

- Data Loading (load_data)
    - Calls load_data_from_db() to fetch COVID-19 data into a DataFrame.
    - Stores the DataFrame in visualization_page.df for use by plotting functions.
    - Updates status_label with the number of rows loaded.
- Plot Selection and Generation
    - Plot Dropdown (plot_menu): Lets the user select from several predefined plot types:
        - Correlation Heatmap
        - Bar Chart: Total Cases by Country
        - Line Trend by Country
        - Boxplot by Month
        - Scatter: Deaths vs Cases
    - Generate Plot Button (generate_button): Triggers plot generation based on selected plot type.
        - Checks if data is loaded; warns if not.
        - Calls the appropriate plotting function, each saving an image file to a graphics output directory.
        - Updates status_label with confirmation and filename.
        - Loads and displays the saved plot image in img_label inside the page.
        - Handles errors in image loading and file absence with user warnings.

Internal Logic

- Plotting Functions: External functions like correlation_heatmap(), bar_total_cases(), etc., accept the DataFrame and output directory path to generate plots saved as PNG files.
- Image Display: Uses PIL (Pillow) to open the generated image, resizes it with thumbnail for fit, converts to Tkinter-compatible format, and updates/creates the label widget accordingly.
- Error Handling: Uses message boxes for:
    - Data not loaded warning.
    - Invalid plot selection.
    - Image loading/display errors.
    - Missing plot files.

Configuration_page(page_frame)

This function builds the Configuration page of the Tkinter GUI, enabling users to adjust interface preferences such as fonts, colors, and sizes. It supports saving changes which trigger an application restart to apply new settings.

Layout

configuration_page_frame: Main container frame inside the given page_frame with padding and background color.

Fields are arranged in a responsive grid layout spanning up to 6 columns before wrapping to next row.

Each configuration setting includes a label and an input widget (dropdown or text entry).

A prominent Save Configuration button is placed below all fields spanning all columns.

Key Features

- Configuration Loading
    - Loads the current configuration dictionary from load_config().
    - Extracts the "interface" section which holds all relevant UI preferences.
- Settings Controls
    - Fields Supported:
        - Font Family (dropdown)
        - Font Size (dropdown)
        - Sidebar Font Family (dropdown)
        - Sidebar Font Size (dropdown)
        - Report Button Color (dropdown)
        - Export Button Color (dropdown)
        - Button Color (dropdown)
        - Sidebar Color (dropdown)
        - Background Color (dropdown with color names mapped to hex codes)
    - Input Widgets:
        - Uses ttk.Combobox (readonly) for dropdown fields with predefined options.
        - Defaults are set from current config or fallback values.
        - Background Color dropdown shows friendly names but stores hex codes internally.
    - Layout dynamically adds fields across columns and rows (wraps after 6 columns).

- Saving Configuration (on_save)
    - On clicking Save Configuration, reads all field values from inputs.
    - Converts background color names back to hex codes.
    - Updates the interface part of the config dictionary.
    - Calls save_config(config) to persist changes.
    - Shows an info message prompting user about application restart.
    - Restarts the Python application immediately by re-executing the current script.

Internal Logic

- Uses a helper function add_field to create label + input pairs and manage layout positioning.
- Stores each input's StringVar in entries dictionary keyed by config keys for easy retrieval.
- Ensures the GUI layout is flexible with column weights and minimum sizes for consistent appearance.

Launch_gui()

The launch_gui() function initializes and launches the main Tkinter graphical user interface (GUI) for the Covid-19 analysis application. It sets up the main window, sidebar navigation with icons and labels, content area, and provides functionality for sidebar expansion/collapse and page switching.

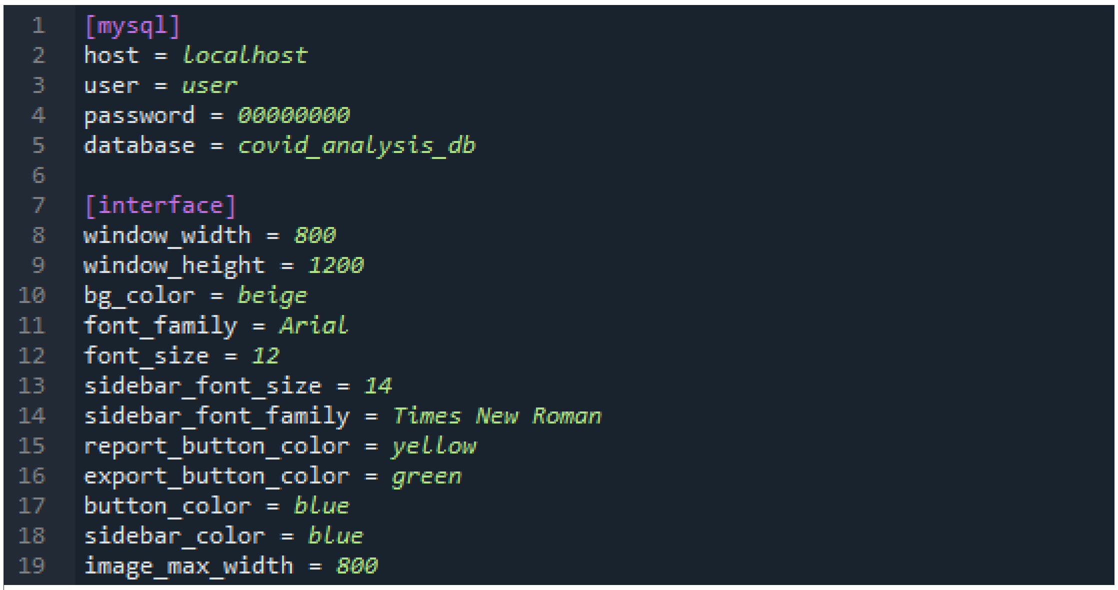
Key Components

- Configuration Application
    - apply_config() is called at the very start to apply saved interface preferences (fonts, colors, sizes, etc.) before the GUI loads.
    - Ensure this function correctly loads and applies user preferences.
- Root Window Initialization
    - root = tk.Tk() creates the main application window.
    - root.geometry('900x600') sets the default size.
    - root.title('Tkinter Covid-19 Analysis') sets the window title.
- Loading Sidebar Icons
    - Multiple tk.PhotoImage instances load icons from the images/ directory.
    - Icons correspond to sidebar actions (toggle menu, data, filtering, statistics, visualization, configuration).
    - All icons should be present in the path images/ relative to the script.
- Sidebar Frame (menu_sidebar_frame)
    - Positioned on the left, background color set by SIDEBAR_COLOR.
    - Initial width is narrow (50 px), expandable to 250 px.
    - Packed with fill=tk.Y and some horizontal padding.
- Page Content Frame (page_frame)
    - Occupies the remaining window space to the right.
    - Positioned with place at (x=30) with full relative width and height.
    - Initially loads the data_page.

- Sidebar Buttons and Labels
    - Sidebar buttons use icon images, placed vertically with fixed positions (x=5, y varies).
    - Each button is paired with:
        - An indicator label (3 px wide) to highlight the active selection.
        - A text label describing the button.
    - Both button and text label are clickable to switch pages.
    - Button background colors change to indicate selection.
- Sidebar Toggle Button
    - At the top, toggles sidebar expansion and collapse.
    - Starts with a "hamburger" icon (toggle_icon).
    - Clicking toggles between expand_menu_sidebar() and shrink_menu_sidebar().
- Important Functions Inside launch_gui()
    - switcher(ind, page, pg)
        - Called on sidebar button clicks or label clicks.
        - Updates background of all indicators to sidebar color.
        - Sets the clicked button's indicator background to white (highlight).
        - Shrinks the sidebar if it is expanded.
        - Clears all children frames in page_frame.
        - Loads the new page by calling page(pg) with the main frame.
    - Sidebar Animation Functions
        - expand_menu(): Gradually increases sidebar width by 10 pixels every 10 ms until 250 px.
        - shrink_menu(): Gradually decreases sidebar width by 10 pixels every 12 ms until 50 px.
        - expand_menu_sidebar(): Initiates expansion and changes toggle button to a "close" icon.
        - shrink_menu_sidebar(): Initiates shrinking and changes toggle button to "open menu" icon.

Work\Work\Scripts\config.ini

Below is an example of the config.ini file used by the application. Make sure to follow the setup instructions above and replace the MySQL configuration values with your own.



```
[mysql]
host = localhost
user = user
password = 00000000
database = covid_analysis_db

[interface]
window_width = 800
window_height = 1200
bg_color = beige
font_family = Arial
font_size = 12
sidebar_font_size = 14
sidebar_font_family = Times New Roman
report_button_color = yellow
export_button_color = green
button_color = blue
sidebar_color = blue
image_max_width = 800
```

Figure 16 - Config.ini Example