ΠΟΛΥΤΕΧΝΙΚΉ ΣΧΟΛΉ, ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΉΜΑ ΜΗΧΑΝΙΚΏΝ ΗΛΕΚΤΡΟΝΙΚΏΝ ΥΠΟΛΟΓΙΣΤΏΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΉΣ

Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

Εαρινό Εξάμηνο

Προαιρετική Εργαστηριακή Άσκηση **Python**

Καθηγητές: Ι. Γαροφαλάκης, Σ. Σιούτας, Π. Χατζηδούκας

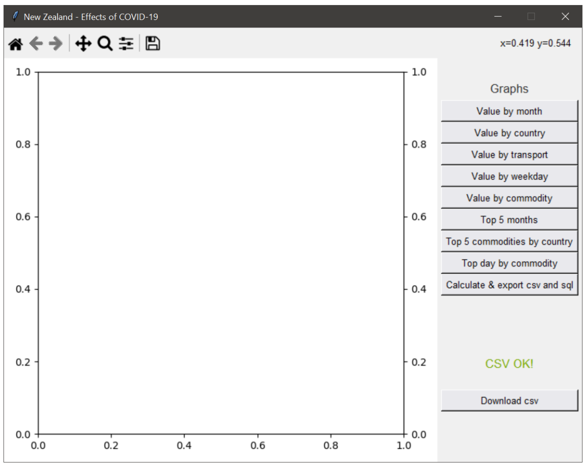
ΑΜ	**Επώνυμο**	**Όνομα**	**Έτος**
1084567	Βιλλιώτης	Αχιλλέας	3^o

ΠΆΤΡΑ, ΙΟΎΝΙΟΣ, 2023

Περιεχόμενα

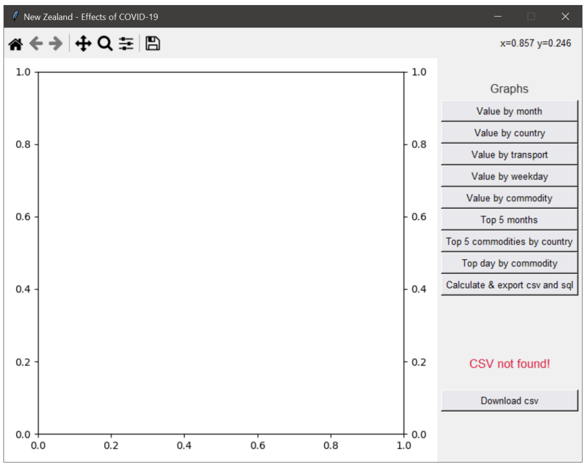
I: Παραδείγματα λειτουργίας

I.a Βασικά σημεία του UI



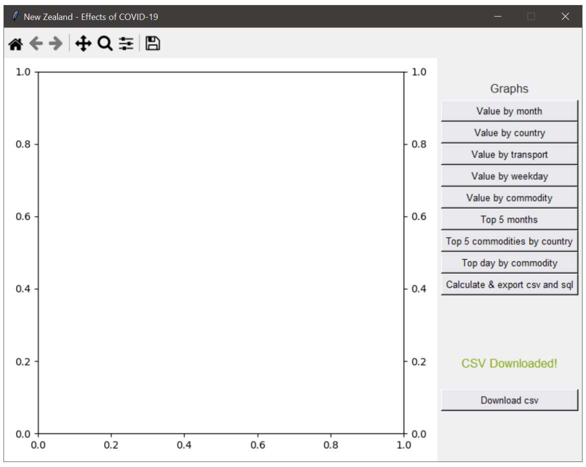
CSV OK!

Βασική οθόνη, csv αρχείο βρέθηκε.



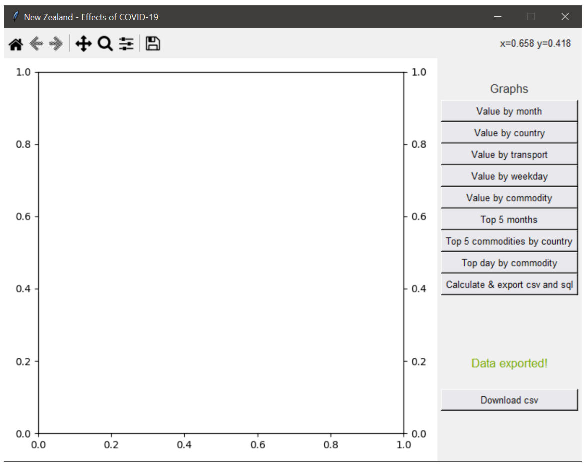
CSV not found!

Το αρχείο csv δεν βρέθηκε.



CSV Downloaded!

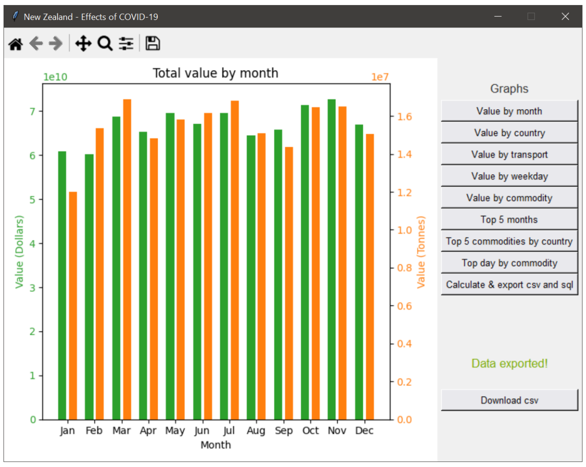
Αφού πατηθεί το κουμπί 'Download csv' εμφανίζεται κατάλληλο μήνυμα.



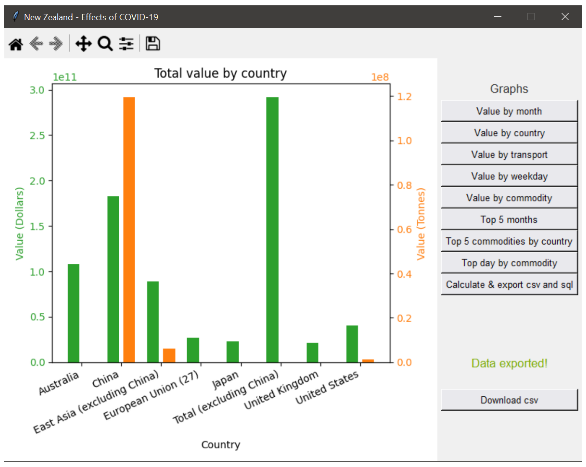
Data exported!

Μήνυμα μετά από την εξαγωγή των δεδομένων.

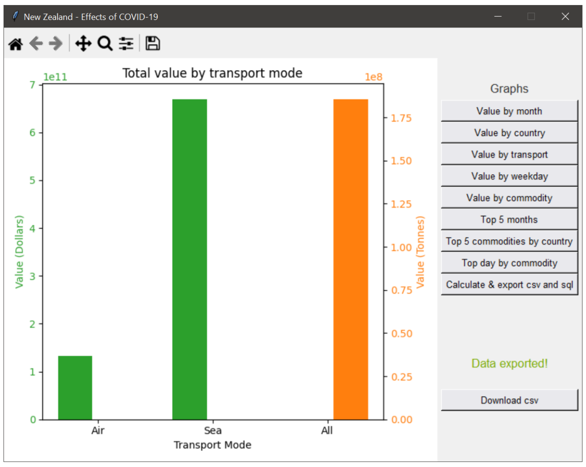
I.b Γραφήματα



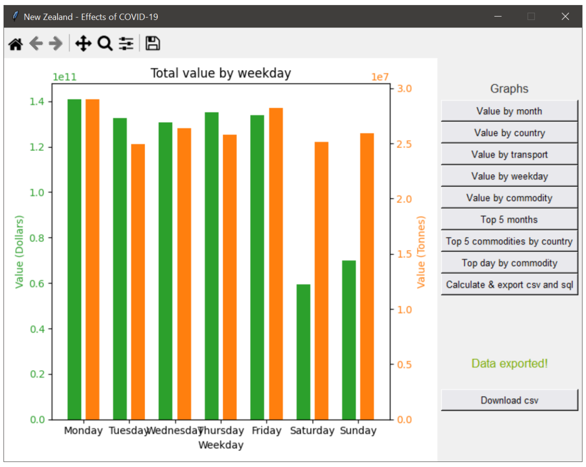
Σύνολο τζίρου ανά μήνα.



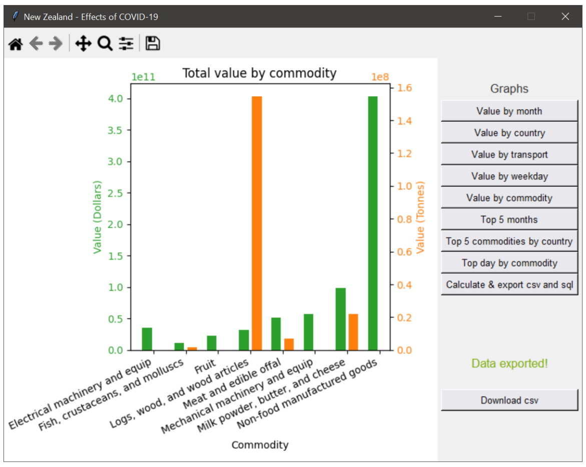
Σύνολο τζίρου ανά χώρα.



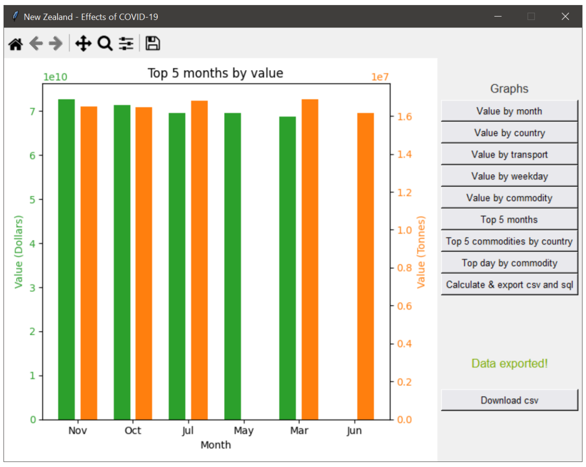
Σύνολο τζίρου ανά μέσο μεταφοράς.



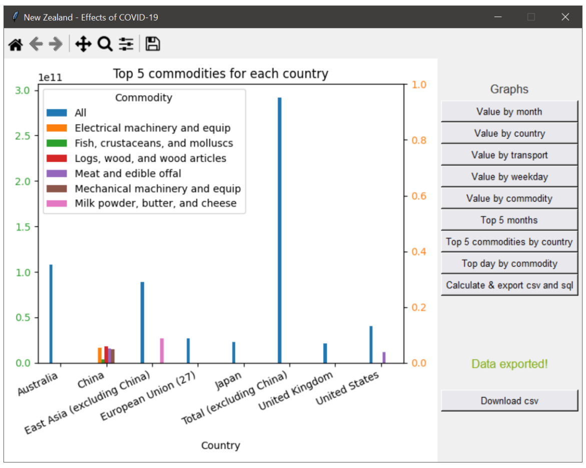
Σύνολο τζίρου ανά μέρα της εβδομάδας.


New Zealand - Effects of COVID-19
Total value by commodity
1e11
1e8
Value (Dollars)
Value (Tonnes)
Electrical machinery and equip
Fish, crustaceans, and molluscs
Fruit
Logs, wood, and wood articles
Meat and edible offal
Mechanical machinery and equip
Milk powder, butter, and cheese
Non-food manufactured goods
Commodity
Graphs
Value by month
Value by country
Value by transport
Value by weekday
Value by commodity
Top 5 months
Top 5 commodities by country
Top day by commodity
Calculate & export csv and sql
Data exported!
Download csv

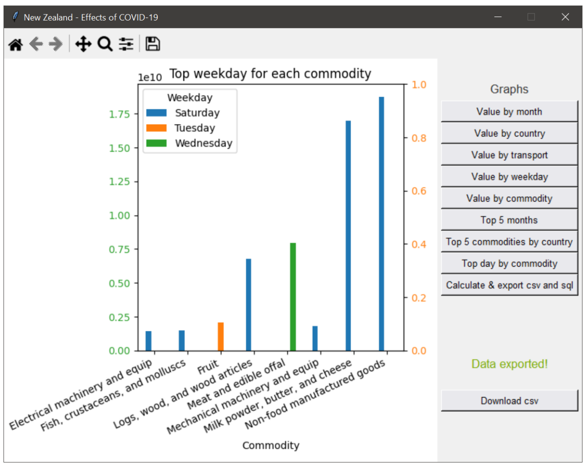
Σύνολο τζίρου ανά κατηγορίας εμπορεύματος.



5 μήνες με τον μεγαλύτερο τζίρο.

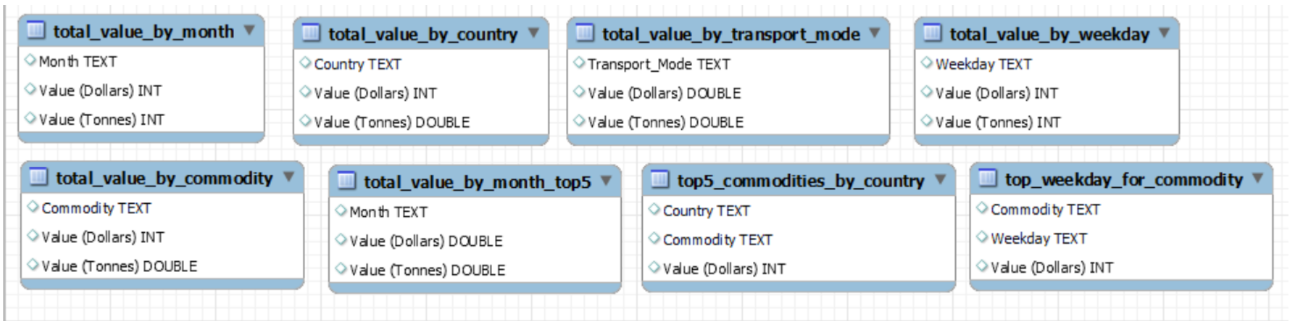

New Zealand - Effects of COVID-19
Top 5 commodities for each country
1e11
Commodity
All
Electrical machinery and equip
Fish, crustaceans, and molluscs
Logs, wood, and wood articles
Meat and edible offal
Mechanical machinery and equip
Milk powder, butter, and cheese
Australia
China
East Asia (excluding China)
European Union (27)
Japan
Total (excluding China)
United Kingdom
United States
Country
Graphs
Value by month
Value by country
Value by transport
Value by weekday
Value by commodity
Top 5 months
Top 5 commodities by country
Top day by commodity
Calculate & export csv and sql
Data exported!
Download csv

5 κατηγορίες εμπορευμάτων με τον μεγαλύτερο τζίρο για κάθε χώρα.


New Zealand - Effects of COVID-19
Top weekday for each commodity
1e10
Weekday
Saturday
Tuesday
Wednesday
Electrical machinery and equip
Fish, crustaceans, and molluscs
Fruit
Logs, wood, and wood articles
Meat and edible offal
Mechanical machinery and equip
Milk powder, butter, and cheese
Non-food manufactured goods
Commodity
Graphs
Value by month
Value by country
Value by transport
Value by weekday
Value by commodity
Top 5 months
Top 5 commodities by country
Top day by commodity
Calculate & export csv and sql
Data exported!
Download csv

Ημέρα με τον μεγαλύτερο τζίρο για κάθε εμπόρευμα.

I.c SQLite Schema



II: Σχόλια-Παραδοχές

II.a Γενικά

Για την εκτέλεση του κώδικα απαιτούνται οι εξωτερικές βιβλιοθήκες `matplotlib, numpy, pandas`. Αναλυτικές απαιτήσεις στο requirements.txt αρχείο.

Υπήρξε προσπάθεια να γίνει σωστό documentation του κώδικα, για χρονικούς λόγους όμως δεν τηρήθηκε μέχρι το πέρας της συγγραφής κώδικα.

Τα γραφήματα δεν παρουσιάζονται με τον βέλτιστο τρόπο, πχ τα γραφήματα 7 και 8 τα οποία δεν είναι κεντραρισμένα στην κάθε χώρα (μικρό οπτικό κατάλοιπο).

Πολλές τιμές δεν εμφανίζονται για τους τόνους διότι το dataset δεν τις περιείχε ενώ στα τελευταία γραφήματα επιλέχθηκε να μην εμφανιστεί για τον ίδιο λόγο. Γενικότερα το dataset περιείχε πολλά κενά.

II.b Οργάνωση κώδικα

Επιπλέον της main, υπάρχουν δύο αρχεία, τα `fun.py` και `config.py`. Στο δεύτερο υπάρχουν μερικές `global` μεταβλητές οι οποίες βοηθούν στην πρόσβαση μεταξύ συναρτήσεων. Στην `fun.py` υπάρχουν όλες οι συναρτήσεις οι οποίες χρειάζονται για την εκτέλεση του κώδικα.

Αρχικά υπάρχουν 8 ζευγάρια συναρτήσεων graph και calculate οι οποίες εμφανίζουν και υπολογίζουν αντίστοιχα τα απαιτούμενα γραφήματα. Είναι σχεδιασμένες με τέτοιο τρόπο έτσι ώστε να υπολογίζονται ακριβώς μια φορά τα DataFrames και μόνο όταν ζητείται απο τον χρήστη.

Εναλλακτικά η συνάρτηση `calculate_all_data_and_export` που καλείται από το αντίστοιχο κουμπί υπολογίζει όλα τα γραφήματα και τα εξαγάγει σε csv μορφή και στη sqlite database.

Οι έλεγχοι για το csv label καλούνται μία φορά στο load και στο download.

Υπάρχει μεταβλητή `config.save_to_sql` η οποία αν τεθεί σε False δεν γίνεται εξαγωγή σε βάση sqlite για λόγους συμβασιμότητας.

III: Υπόμνημα

Παρατίθεται ο πηγαίος κώδικας:

III.a SQLite Schema

```sql
CREATE TABLE "top5_commodities_by_country" (
"Country" TEXT,
"Commodity" TEXT,
"Value (Dollars)" INTEGER
);

CREATE TABLE "top_weekday_for_commodity" (
"Commodity" TEXT,
"Weekday" TEXT,
"Value (Dollars)" INTEGER
);

CREATE TABLE "total_value_by_commodity" (
"Commodity" TEXT,
  "Value (Dollars)" INTEGER,
  "Value (Tonnes)" REAL
);

CREATE TABLE "total_value_by_country" (
"Country" TEXT,
  "Value (Dollars)" INTEGER,
  "Value (Tonnes)" REAL
);

CREATE TABLE "total_value_by_month" (
"Month" TEXT,
  "Value (Dollars)" INTEGER,
  "Value (Tonnes)" INTEGER
);

CREATE TABLE "total_value_by_month_top5" (
"Month" TEXT,
  "Value (Dollars)" REAL,
  "Value (Tonnes)" REAL
);

CREATE TABLE "total_value_by_transport_mode" (
"Transport_Mode" TEXT,
  "Value (Dollars)" REAL,
  "Value (Tonnes)" REAL
);

```

```
CREATE TABLE "total_value_by_weekday" (
"Weekday" TEXT,
  "Value (Dollars)" INTEGER,
  "Value (Tonnes)" INTEGER
);
```

III.b main.py

```python
import config
import fun

def main():
    config.init()
    #config.save_to_sql=False #uncomment if you dont want to save to sqlite3 db.
    root = fun.gui_and_sql_init()
    root.mainloop()

if __name__ == "__main__":
    main()
```

III.c config.py

```python
def init():
    global sqlconn
    global df
    global graph_month
    global graph_transport
    global graph_country
    global graph_weekday
    global graph_commodity
    global graph_top5_month
    global graph_country_top5_commodity
    global graph_commodity_top_weekday
    global save_to_sql

    sqlconn = None
    df = None
    graph_month = None
    graph_transport = None
    graph_country = None
    graph_weekday = None
    graph_commodity = None
    graph_top5_month = None
    graph_country_top5_commodity = None
    graph_commodity_top_weekday = None
    save_to_sql = True
```

III.d fun.py

```python
import calendar
import os
import sqlite3

import matplotlib
from matplotlib import pyplot as plt
from matplotlib.backends.backend_tkagg import (
    FigureCanvasTkAgg,
    NavigationToolbar2Tk
)
from matplotlib.figure import Figure
import numpy as np
import pandas as pd
import tkinter as tk
from tkinter import font as tkFont

import config

#Erwthma 1
def graph_total_value_by_month(graph_df: pd.DataFrame,
    ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
    """Show graph for total value by month.

    Args:
    -----
        graph_df: ``DataFrame`` calculated by ``calculate_total_value_by_month``
        ax1: ``Axes`` object for dollars.
        ax1: ``Axes`` object for tonnes.
        figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
    """
    #Calculate series objects if they haven't been calculated already
    if config.graph_month is None:
        config.graph_month = calculate_total_value_by_month(config.df)
        graph_df = config.graph_month

    x_indices = np.arange(len(graph_df.index))  #calculate spacing for x labels.
    ax1.clear()
    ax2.clear()

    ax1.set_title('Total value by month')
    ax1.set_xlabel('Month')
    ax1.set_xticks(x_indices, graph_df.index)  #manually set labels to fit 2 bars in
    each x value
    color = 'tab:green'
    ax1.bar(x_indices-0.2, graph_df['Value (Dollars)'], 0.3, color=color)
    ax1.set_ylabel('Value (Dollars)', color=color)
    ax1.tick_params(axis='y', labelcolor=color)

    color = 'tab:orange'
    ax2.bar(x_indices+0.2, graph_df['Value (Tonnes)'], 0.3, color=color)
    ax2.set_ylabel('Value (Tonnes)', color=color)  # we already handled the x-label with
    ax1
    ax2.tick_params(axis='y', labelcolor=color)

    #Move spines and info back to the right
    ax2.spines['right'].set_position(('outward', 0))  # Move the spine to the right
    ax2.yaxis.set_label_position('right')
    ax2.yaxis.set_ticks_position('right')

    figure_canvas.draw()

def calculate_total_value_by_month(df: pd.DataFrame):
    """Calculate the DataFrame variable required by ``graph_total_value_by_month`` and
    `calculate_total_value_by_month_topn`.

    Args:
    -----
        df: ``DataFrame`` with Date column in datetime format.

    Returns:
    --------
        graph_by_month: ``DataFrame`` series object ready to be used with
`calculate_total_value_by_month_topn`
        or in order to be reused with `graph_total_value_by_month`.
    """
    #Dollars
    df_by_month_dollar_group = df.query('Country=="All" & Commodity=="All" &
    Transport_Mode=="All" & Measure=="$"')\
    .groupby(df.Date.dt.month)
    df_by_month_dollar_group = df_by_month_dollar_group['Value'].sum()
    df_by_month_dollar_group = df_by_month_dollar_group.rename('Value (Dollars)')
    df_by_month_dollar_group = df_by_month_dollar_group.rename(lambda x:
    calendar.month_abbr[x])  #match number to month
    df_by_month_dollar_group = df_by_month_dollar_group.rename_axis("Month")

    #Tonnes
    df_by_month_tonne_group = df.query('Country=="All" & Transport_Mode=="All"&
    Measure=="Tonnes"')\
    .groupby(df.Date.dt.month)
    df_by_month_tonne_group = df_by_month_tonne_group['Value'].sum()
    df_by_month_tonne_group = df_by_month_tonne_group.rename('Value (Tonnes)')
    df_by_month_tonne_group = df_by_month_tonne_group.rename(lambda x:
    calendar.month_abbr[x])  #match number to month
    df_by_month_tonne_group = df_by_month_tonne_group.rename_axis("Month")

    graph_by_month=pd.concat([df_by_month_dollar_group, df_by_month_tonne_group],
    axis=1) #  Merge Series objects

    graph_by_month.to_csv('1_total_value_by_month.csv')  #save to csv
    if config.save_to_sql:
        graph_by_month.to_sql('total_value_by_month', config.sqlconn, 'replace') #save
        to sql

    return graph_by_month

#Erwthma 2
def graph_total_value_by_country(graph_df: pd.DataFrame,
    ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
    """Show graph for total value by country.

    Args:
    -----
        graph_df: ``DataFrame`` calculated by ``calculate_total_value_by_country``
        ax1: ``Axes`` object for dollars.
        ax1: ``Axes`` object for tonnes.
        figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
    """
    #Calculate series objects if they haven't been calculated already.
    if config.graph_country is None:
        config.graph_country = calculate_total_value_by_country(config.df)
        graph_df = config.graph_country

    x_indices = np.arange(len(graph_df.index))  #calculate spacing for x labels.
    ax1.clear()
    ax2.clear()

    ax1.set_title('Total value by country')
    ax1.set_xlabel('Country')
    ax1.set_xticks(x_indices, graph_df.index, rotation=25, ha='right')  #manually set
    labels to fit 2 bars in each x value

    color = 'tab:green'
    ax1.bar(x_indices-0.2, graph_df['Value (Dollars)'], 0.3, color=color)
    ax1.set_ylabel('Value (Dollars)', color=color)
    ax1.tick_params(axis='y', labelcolor=color)

    color = 'tab:orange'
    ax2.bar(x_indices+0.2, graph_df['Value (Tonnes)'], 0.3, color=color)
    ax2.set_ylabel('Value (Tonnes)', color=color)  # we already handled the x-label with
    ax1
    ax2.tick_params(axis='y', labelcolor=color)

    #Move spines and info back to the right
    ax2.spines['right'].set_position(('outward', 0))  # Move the spine to the right
    ax2.yaxis.set_label_position('right')
    ax2.yaxis.set_ticks_position('right')

    figure_canvas.draw()

def calculate_total_value_by_country(df: pd.DataFrame):
    """Calculate the DataFrame variable required by ``graph_total_value_by_country``

    Args:
    -----
        df: ``DataFrame`` with Date column in datetime format.

    Returns:
    --------
        graph_by_country: ``Dataframe`` series object ready to be reused with
``graph_total_value_by_country``
    -------
    """

    #Dollars
    df_by_country_dollar_group = df.query('Country != "All" & Commodity=="All" &
    Transport_Mode=="All" & Measure=="$"')\
    .groupby('Country')
    df_by_country_dollar_group = df_by_country_dollar_group['Value'].sum()
    df_by_country_dollar_group = df_by_country_dollar_group.rename('Value (Dollars)')

    #Tonnes
    df_by_country_tonne_group = df.query('Country != "All" & Transport_Mode=="All" &
    Measure=="Tonnes"')\
    .groupby('Country')
    df_by_country_tonne_group = df_by_country_tonne_group['Value'].sum()
    df_by_country_tonne_group = df_by_country_tonne_group.rename('Value (Tonnes)')

    graph_by_country=pd.concat([df_by_country_dollar_group, df_by_country_tonne_group],
    axis=1) #  Merge Series objects

    graph_by_country.to_csv('2_total_value_by_country.csv')  #save to csv
    if config.save_to_sql:
        graph_by_country.to_sql('total_value_by_country', config.sqlconn, 'replace')
        #save to sql
    return graph_by_country

#Erwthma 3
```

```python
def graph_total_value_by_transport(graph_df: pd.DataFrame,
    ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
    """Show graph for total value by transport mode.

    Args:
    -----
        graph_df: ``DataFrame`` calculated by ``calculate_total_value_by_transport``.
        ax1: ``Axes`` object for dollars.
        ax1: ``Axes`` object for tonnes.
        figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
    """
    #Calculate series objects if they haven't been calculated already
    if config.graph_transport is None:
        config.graph_transport = calculate_total_value_by_transport(config.df)
        graph_df = config.graph_transport

    x_indices = np.arange(len(graph_df.index))  #calculate spacing for x labels.
    ax1.clear()
    ax2.clear()

    ax1.set_title('Total value by transport mode')
    ax1.set_xlabel('Transport Mode')
    ax1.set_xticks(x_indices, graph_df.index)  #manually set labels to fit 2 bars in
    each x value

    color = 'tab:green'
    ax1.bar(x_indices-0.2, graph_df['Value (Dollars)'], 0.3, color=color)
    ax1.set_ylabel('Value (Dollars)', color=color)
    ax1.tick_params(axis='y', labelcolor=color)

    color = 'tab:orange'
    ax2.bar(x_indices+0.2, graph_df['Value (Tonnes)'], 0.3, color=color)
    ax2.set_ylabel('Value (Tonnes)', color=color)  # we already handled the x-label with
    ax1
    ax2.tick_params(axis='y', labelcolor=color)

    #Move spines and info back to the right
    ax2.spines['right'].set_position(('outward', 0))  # Move the spine to the right
    ax2.yaxis.set_label_position('right')
    ax2.yaxis.set_ticks_position('right')

    figure_canvas.draw()

def calculate_total_value_by_transport(df: pd.DataFrame):
    """Calculate the DataFrame variable required by ``graph_total_value_by_transport``.

    Args:
```

```python
    -----
        df: ``DataFrame`` with Date column in datetime format.

    Returns:
    --------
        graph_by_transport: ``DataFrame`` series object ready to be reused with
 ``graph_total_value_by_transport``
    -------
    """

    #Dollars
    df_by_transport_dollar_group = df.query('Country=="All" & Commodity=="All" &
    Transport_Mode != "All" & Measure=="$"')\
    .groupby('Transport_Mode')
    df_by_transport_dollar_group = df_by_transport_dollar_group['Value'].sum()
    df_by_transport_dollar_group = df_by_transport_dollar_group.rename('Value
    (Dollars)')

    #Tonnes
    df_by_transport_tonne_group = df.query('Country=="All" & Measure=="Tonnes"')\
    .groupby('Transport_Mode')  #! Transport modes don't exist for tonnes
    df_by_transport_tonne_group = df_by_transport_tonne_group['Value'].sum()
    df_by_transport_tonne_group = df_by_transport_tonne_group.rename('Value (Tonnes)')

    graph_by_transport=pd.concat([df_by_transport_dollar_group,
    df_by_transport_tonne_group], axis=1) #  Merge Series objects

    graph_by_transport.to_csv('3_total_value_by_transport_mode.csv')  #save to csv
    if config.save_to_sql:
        graph_by_transport.to_sql('total_value_by_transport_mode', config.sqlconn,
        'replace')
    return graph_by_transport

#Erwthma 4
def graph_total_value_by_weekday(graph_df: pd.DataFrame,
    ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
    """Show graph for total value by weekday.

    Args:
    -----
        graph_df: ``DataFrame`` calculated by ``calculate_total_value_by_weekday``
        ax1: ``Axes`` object for dollars.
        ax1: ``Axes`` object for tonnes.
        figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
    """
    #Calculate series objects if they haven't been calculated already
    if config.graph_weekday is None:
        config.graph_weekday = calculate_total_value_by_weekday(config.df)
        graph_df = config.graph_weekday

    x_indices = np.arange(len(graph_df.index))  #calculate spacing for x labels.
    ax1.clear()
    ax2.clear()

    ax1.set_title('Total value by weekday')
    ax1.set_xlabel('Weekday')
    ax1.set_xticks(x_indices, graph_df.index)  #manually set labels to fit 2 bars in each x value

    color = 'tab:green'
    ax1.bar(x_indices-0.2, graph_df['Value (Dollars)'], 0.3, color=color)
    ax1.set_ylabel('Value (Dollars)', color=color)
    ax1.tick_params(axis='y', labelcolor=color)

    color = 'tab:orange'
    ax2.bar(x_indices+0.2, graph_df['Value (Tonnes)'], 0.3, color=color)
    ax2.set_ylabel('Value (Tonnes)', color=color)  # we already handled the x-label with ax1
    ax2.tick_params(axis='y', labelcolor=color)

    #Move spines and info back to the right
    ax2.spines['right'].set_position(('outward', 0))  # Move the spine to the right
    ax2.yaxis.set_label_position('right')
    ax2.yaxis.set_ticks_position('right')

    figure_canvas.draw()

def calculate_total_value_by_weekday(df: pd.DataFrame):
    """Calculate the DataFrame required by ``graph_total_value_by_weekday``.

    Args:
    -----
        df: ``DataFrame`` with Date column in datetime format.

    Returns:
    --------
        graph_by_weekday: ``DataFrame`` series object ready to be reused with ``graph_total_value_by_weekday``.
    -------
    """

    #Dollars
    df_by_weekday_dollar_group = df.query('Country=="All" & Commodity=="All" & Transport_Mode=="All" & Measure=="$"')\
    .groupby('Weekday')
    df_by_weekday_dollar_group = df_by_weekday_dollar_group['Value'].sum()
    df_by_weekday_dollar_group =
    df_by_weekday_dollar_group.reindex(list(calendar.day_name))  #sort days
    df_by_weekday_dollar_group = df_by_weekday_dollar_group.rename('Value (Dollars)')

    #Tonnes
    df_by_weekday_tonne_group = df.query('Country=="All" & Transport_Mode=="All" &
    Measure=="Tonnes"')\
    .groupby('Weekday')
    df_by_weekday_tonne_group = df_by_weekday_tonne_group['Value'].sum()
    df_by_weekday_tonne_group =
    df_by_weekday_tonne_group.reindex(list(calendar.day_name))  #sort days
    df_by_weekday_tonne_group = df_by_weekday_tonne_group.rename('Value (Tonnes)')

    graph_by_weekday=pd.concat([df_by_weekday_dollar_group, df_by_weekday_tonne_group],
    axis=1) #  Merge Series objects

    graph_by_weekday.to_csv('4_total_value_by_weekday.csv')  #save to csv
    if config.save_to_sql:
        graph_by_weekday.to_sql('total_value_by_weekday', config.sqlconn, 'replace')
    return graph_by_weekday

#Erwthma 5
def graph_total_value_by_commodity(graph_df: pd.DataFrame,
    ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
    """Show graph for total value by commodity.

    Args:
    -----
        graph_df: ``DataFrame`` calculated by ``calculate_total_value_by_commodity``
        ax1:  ``Axes`` object for dollars.
        ax1:  ``Axes`` object for tonnes.
        figure_canvas:  ``FigureCanvasTkAgg`` object for drawing on.
    """
    #Calculate series objects if they haven't been calculated already
    if config.graph_commodity is None:
        config.graph_commodity = calculate_total_value_by_commodity(config.df)
        graph_df = config.graph_commodity

    x_indices = np.arange(len(graph_df.index))  #calculate spacing for x labels.
    ax1.clear()
    ax2.clear()

    ax1.set_title('Total value by commodity')
    ax1.set_xlabel('Commodity')
    ax1.set_xticks(x_indices, graph_df.index, rotation=25, ha='right')  #manually set
    labels to fit 2 bars in each x value

    color = 'tab:green'
    ax1.bar(x_indices-0.2, graph_df['Value (Dollars)'], 0.3, color=color)
    ax1.set_ylabel('Value (Dollars)', color=color)
    ax1.tick_params(axis='y', labelcolor=color)

    color = 'tab:orange'
    ax2.bar(x_indices+0.2, graph_df['Value (Tonnes)'], 0.3, color=color)
    ax2.set_ylabel('Value (Tonnes)', color=color)  # we already handled the x-label with
    ax1
    ax2.tick_params(axis='y', labelcolor=color)

    #Move spines and info back to the right
    ax2.spines['right'].set_position(('outward', 0))  # Move the spine to the right
    ax2.yaxis.set_label_position('right')
    ax2.yaxis.set_ticks_position('right')

    figure_canvas.draw()

def calculate_total_value_by_commodity(df: pd.DataFrame):
    """Calculate the DataFrame variable required by ``graph_total_value_by_commodity``.

    Args:
    -----
        df: ``DataFrame`` with Date column in datetime format.

    Returns:
    --------
        graph_by_commodity: ``DataFrame`` series object ready to be used with
``graph_total_value_by_commodity``.
    -------
    """

    #Dollars
    df_by_commodity_dollar_group = df.query('Country=="All" & Commodity != "All" &
    Transport_Mode=="All" & Measure=="$"')\
    .groupby('Commodity')
    df_by_commodity_dollar_group = df_by_commodity_dollar_group['Value'].sum()
    df_by_commodity_dollar_group = df_by_commodity_dollar_group.rename('Value
    (Dollars)')

    #Tonnes
    df_by_commodity_tonne_group = df.query('Country=="All" & Transport_Mode=="All" &
    Measure=="Tonnes"')\
    .groupby('Commodity')
```

```python
    df_by_commodity_tonne_group = df_by_commodity_tonne_group['Value'].sum()
    df_by_commodity_tonne_group = df_by_commodity_tonne_group.rename('Value (Tonnes)')

    graph_by_commodity=pd.concat([df_by_commodity_dollar_group,
    df_by_commodity_tonne_group], axis=1) #  Merge Series objects

    graph_by_commodity.to_csv('5_total_value_by_commodity.csv')  #save to csv
    if config.save_to_sql:
        graph_by_commodity.to_sql('total_value_by_commodity', config.sqlconn, 'replace')
    return graph_by_commodity

#Erwthma 6
def graph_total_value_by_month_topn(graph_df: pd.DataFrame, n: int,
    ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
    """Show graph for total value by month, top n values.

    Args:
    -----
        graph_df: ``DataFrame`` calculated by ``calculate_total_value_by_month``.
        n: ``int`` how many of the top values to show.
        ax1: ``Axes`` object for dollars.
        ax1: ``Axes`` object for tonnes.
        figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
    """
    #Calculate series objects if they haven't been calculated already
    if config.graph_top5_month is None:
        config.graph_top5_month = calculate_total_value_by_month_topn(config.df, 5)
        graph_df = config.graph_top5_month

    x_indices = np.arange(len(graph_df.index))  #calculate spacing for x labels.
    ax1.clear()
    ax2.clear()

    ax1.set_title('Top 5 months by value')
    ax1.set_xlabel('Month')
    ax1.set_xticks(x_indices, graph_df.index)  #manually set labels to fit 2 bars in
    each x value

    color = 'tab:green'
    ax1.bar(x_indices-0.2, graph_df['Value (Dollars)'], 0.3, color=color)
    ax1.set_ylabel('Value (Dollars)', color=color)
    ax1.tick_params(axis='y', labelcolor=color)

    color = 'tab:orange'
    ax2.bar(x_indices+0.2, graph_df['Value (Tonnes)'], 0.3, color=color)
    ax2.set_ylabel('Value (Tonnes)', color=color)  # we already handled the x-label with
    ax1
    ax2.tick_params(axis='y', labelcolor=color)

    #Move spines and info back to the right
    ax2.spines['right'].set_position(('outward', 0))  # Move the spine to the right
    ax2.yaxis.set_label_position('right')
    ax2.yaxis.set_ticks_position('right')

    figure_canvas.draw()

def calculate_total_value_by_month_topn(graph_df: pd.DataFrame, n: int):
    """Calculate the DataFrame variable required by ``graph_total_value_by_month_topn``.

    Args:
    -----
        graph_df: ``DataFrame`` calculated from `calculate_total_value_by_month`.
        n: ``int`` how many of the top values to calculate.

    Returns:
    --------
        graph_by_month_topn: ``DataFrame`` series object ready to be used with
`graph_total_value_by_month_topn`.
    """
    if config.graph_month is None:
        config.graph_month = calculate_total_value_by_month(config.df)
        graph_df = config.graph_month

    df_top5_dollars=graph_df.nlargest(5, 'Value (Dollars)')['Value (Dollars)']
    df_top5_tonnes=graph_df.nlargest(5, 'Value (Tonnes)')['Value (Tonnes)']
    graph_by_month_topn = pd.concat([df_top5_dollars, df_top5_tonnes], axis=1)  #concat
    into one DataFrame

    graph_by_month_topn.to_csv('6_total_value_by_month_top5.csv')
    if config.save_to_sql:
        graph_by_month_topn.to_sql('total_value_by_month_top5', config.sqlconn,
        'replace')

    return graph_by_month_topn

#Erwthma 7
def graph_total_value_by_country_commodity_top5(dollar_df: pd.DataFrame,
    ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
    """Show graph for top 5 commodities of each country.

    Args:
    -----
        dollar_df: ``DataFrame`` calculated by
``calculate_total_value_by_ccountry_commodity``
```

```python
        ax1: ``Axes`` object for dollars.
        ax1: ``Axes`` object for tonnes.
        figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
    """
    #Calculate DataFrame objects if they haven't been calculated already
    if config.graph_country_top5_commodity is None:
        config.graph_country_top5_commodity =
        calculate_total_value_by_country_commodity_top5(config.df)
        dollar_df = config.graph_country_top5_commodity

    ax1.clear()
    ax2.clear()

    ax1 = dollar_df.plot(kind="bar", ax=ax1)
    ax1.set_xticklabels(ax1.get_xticklabels(), rotation=25, ha='right')
    ax1.set_title("Top 5 commodities for each country")

    figure_canvas.draw()

def calculate_total_value_by_country_commodity_top5(df: pd.DataFrame):
    """Calculate the series variable required by
    ``graph_total_value_by_country_commodity``.

    Args:
    -----
        df: ``DataFrame`` with Date column in datetime format.

    Returns:
    --------
        df_by_country_commodity_dollar: ``DataFrame`` object ready to be used with
 ``graph_total_value_by_country_commodity``.
    -------
    """

    #Dollars
    df_by_country_commodity_dollar_group = df.query('Country!="All" & Measure=="$"')\
    .groupby(['Country',
    'Commodity']).agg({'Value':sum}).sort_values('Value').groupby('Country').head(5)
    #regroup by country to get top 5 commodities of each country
    df_by_country_commodity_dollar_group =
    df_by_country_commodity_dollar_group.rename(columns={'Value':'Value (Dollars)'})

    df_by_country_commodity_dollar_group.to_csv('7_top5_commodities_by_country.csv')
    #save to csv
    if config.save_to_sql:
        df_by_country_commodity_dollar_group.to_sql('top5_commodities_by_country',
        config.sqlconn, 'replace')

    df_by_country_commodity_dollar_group=
    pd.pivot_table(df_by_country_commodity_dollar_group,
        values='Value (Dollars)', index='Country', columns='Commodity')

    return df_by_country_commodity_dollar_group

#Erwthma 8
def graph_top_value_by_weekday_for_commodity(dollar_df: pd.DataFrame,
    ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
    """Show graph for top weekday for each commodity.

    Args:
    -----
        dollar_df: ``DataFrame`` calculated by
``calculate_top_value_by_weekday_for_commodity``
        ax1: ``Axes`` object for dollars.
        figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
    """
    #Calculate DataFrame objects if they haven't been calculated already
    if config.graph_commodity_top_weekday is None:
        config.graph_commodity_top_weekday =
        calculate_top_value_by_weekday_for_commodity(config.df)
        dollar_df = config.graph_commodity_top_weekday

    ax1.clear()
    ax2.clear()

    ax1 = dollar_df.plot(kind="bar", ax=ax1)
    ax1.set_xticklabels(ax1.get_xticklabels(), rotation=25, ha='right')
    ax1.set_title("Top weekday for each commodity")

    figure_canvas.draw()

def calculate_top_value_by_weekday_for_commodity(df: pd.DataFrame):
    """Calculate the series variable required by
    ``graph_top_value_by_weekday_for_commodity``.

    Args:
    -----
        df: ``DataFrame`` with Date column in datetime format.

    Returns:
    --------
        df_by_country_commodity_dollar: ``DataFrame`` object ready to be used with
``graph_top_value_by_weekday_for_commodity``.
    -------
    """

    #Dollars
    df_by_country_commodity_dollar_group = df.query('Commodity!="All" & Measure=="$"')\
    .groupby(['Commodity',
    'Weekday']).agg({'Value':sum}).sort_values('Value').groupby('Commodity').head(1)
    #sort by value, regroup by country to get top weekday for each commodity
    df_by_country_commodity_dollar_group =
    df_by_country_commodity_dollar_group.rename(columns={'Value':'Value (Dollars)'})

    df_by_country_commodity_dollar_group.to_csv('8_top_weekday_for_commodity.csv')
    #save to csv
    if config.save_to_sql:
        df_by_country_commodity_dollar_group.to_sql('top_weekday_for_commodity',
        config.sqlconn, 'replace')

    df_by_country_commodity_dollar_group=
    pd.pivot_table(df_by_country_commodity_dollar_group,
        values='Value (Dollars)', index='Commodity', columns='Weekday')


    return df_by_country_commodity_dollar_group

def calculate_all_data_and_export(csv: tk.Label):
    if config.graph_month is None:
        config.graph_month = calculate_total_value_by_month(config.df)

    if config.graph_country is None:
        config.graph_country = calculate_total_value_by_country(config.df)

    if config.graph_transport is None:
        config.graph_transport = calculate_total_value_by_transport(config.df)

    if config.graph_weekday is None:
        config.graph_weekday = calculate_total_value_by_weekday(config.df)

    if config.graph_commodity is None:
        config.graph_commodity = calculate_total_value_by_commodity(config.df)

    if config.graph_top5_month is None:
        config.graph_top5_month =
        calculate_total_value_by_month_topn(config.graph_month, 5)

    if config.graph_country_top5_commodity is None:
        config.graph_country_top5_commodity =
        calculate_total_value_by_country_commodity_top5(config.df)

    if config.graph_commodity_top_weekday is None:
        config.graph_commodity_top_weekday =
        calculate_top_value_by_weekday_for_commodity(config.df)

    csv["fg"] = "#7CAC06"
    csv["text"] = "Data exported!"

def check_for_csv(csv: tk.Label):
    fileName = 'covid19_effects_trades.csv'
    path = os.getcwd() + os.sep + fileName
    if os.path.exists(path):
        csv["fg"] = "#7CAC06"
        csv["text"] = "CSV OK!"
        df = pd.read_csv(path)
        df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y')
        config.df = df
    else:
        csv["fg"] = "#DC143C"
        csv["text"] = "CSV not found!"

def download_csv(csv: tk.Label):
    url =
    'https://www.stats.govt.nz/assets/Uploads/Effects-of-COVID-19-on-trade/Effects-of-COVID-19-on-trac
December-2021-provisional/Download-data/effects-of-covid-19-on-trade-at-15-december-2021-provisional.
    fileName = 'covid19_effects_trades'

    # download csv and save to fileName locally
    df = pd.read_csv(url)
    df.to_csv(fileName+'.csv', index=False)
    df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y')
    config.df = df

    csv["fg"] = "#7CAC06"
    csv["text"] = "CSV Downloaded!"

def db_create_connection():
    '''
    Returns connection object for sqlite3 database.
    '''
    conn = None
    try:
        conn = sqlite3.connect('new_zealand_covid_effects.db')
        return conn
    except sqlite3.Error as e:
        print(e)


```

```python
def gui_and_sql_init():
    """Initialize GUI and sql.

    Returns
    -----
        `root`.

    """
    config.sqlconn = db_create_connection()

    matplotlib.use('TkAgg')
    #Create root window
    root = tk.Tk()
    root.title("New Zealand - Effects of COVID-19")
    width=800
    height=600
    screenwidth = root.winfo_screenwidth()
    screenheight = root.winfo_screenheight()
    #align to middle of screen
    alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight
    - height) / 2)
    root.geometry(alignstr)
    root.resizable(width=False, height=False)

    #Create labels
    ft = tkFont.Font(family='Roboto',size=13)
    label_graphs=tk.Label(root)
    label_graphs["font"] = ft
    label_graphs["fg"] = "#333333"
    label_graphs["justify"] = "center"
    label_graphs["text"] = "Graphs"
    label_graphs.place(x=605,y=70,width=190,height=30)

    label_csv_status=tk.Label(root)
    label_csv_status["font"] = ft
    label_csv_status["fg"] = "#333333"
    label_csv_status["justify"] = "center"
    label_csv_status["text"] = "CSV status?"
    label_csv_status.place(x=605,y=450,width=190,height=30)
    check_for_csv(label_csv_status) #update label and color

    #Create buttons
    ft = tkFont.Font(family='Roboto',size=10)

    button_download_csv=tk.Button(root)
    button_download_csv["bg"] = "#e9e9ed"
    button_download_csv["font"] = ft
    button_download_csv["fg"] = "#000000"
    button_download_csv["justify"] = "center"
    button_download_csv["text"] = "Download csv"
    button_download_csv.place(x=605,y=500,width=190,height=30)
    button_download_csv["command"] = lambda: download_csv(label_csv_status)

    button_by_month=tk.Button(root)
    button_by_month["bg"] = "#e9e9ed"
    button_by_month["font"] = ft
    button_by_month["fg"] = "#000000"
    button_by_month["justify"] = "center"
    button_by_month["text"] = "Value by month"
    button_by_month.place(x=605,y=100,width=190,height=30)
    button_by_month["command"] = lambda: graph_total_value_by_month(config.graph_month,
        ax1, ax2, figure_canvas)

    button_by_country=tk.Button(root)
    button_by_country["bg"] = "#e9e9ed"
    button_by_country["font"] = ft
    button_by_country["fg"] = "#000000"
    button_by_country["justify"] = "center"
    button_by_country["text"] = "Value by country"
    button_by_country.place(x=605,y=130,width=190,height=30)
    button_by_country["command"] = lambda:
    graph_total_value_by_country(config.graph_country,
        ax1, ax2, figure_canvas)

    button_by_transport=tk.Button(root)
    button_by_transport["bg"] = "#e9e9ed"
    button_by_transport["font"] = ft
    button_by_transport["fg"] = "#000000"
    button_by_transport["justify"] = "center"
    button_by_transport["text"] = "Value by transport"
    button_by_transport.place(x=605,y=160,width=190,height=30)
    button_by_transport["command"] = lambda:
    graph_total_value_by_transport(config.graph_transport,
        ax1, ax2, figure_canvas)

    button_by_weekday=tk.Button(root)
    button_by_weekday["bg"] = "#e9e9ed"
    button_by_weekday["font"] = ft
    button_by_weekday["fg"] = "#000000"
    button_by_weekday["justify"] = "center"
    button_by_weekday["text"] = "Value by weekday"
    button_by_weekday.place(x=605,y=190,width=190,height=30)
    button_by_weekday["command"] = lambda:
    graph_total_value_by_weekday(config.graph_weekday,
        ax1, ax2, figure_canvas)

    button_by_commodity=tk.Button(root)
    button_by_commodity["bg"] = "#e9e9ed"
    button_by_commodity["font"] = ft
    button_by_commodity["fg"] = "#000000"
    button_by_commodity["justify"] = "center"
    button_by_commodity["text"] = "Value by commodity"
    button_by_commodity.place(x=605,y=220,width=190,height=30)
    button_by_commodity["command"] = lambda:
    graph_total_value_by_commodity(config.graph_commodity,
        ax1, ax2, figure_canvas)

    button_top5_months=tk.Button(root)
    button_top5_months["bg"] = "#e9e9ed"
    button_top5_months["font"] = ft
    button_top5_months["fg"] = "#000000"
    button_top5_months["justify"] = "center"
    button_top5_months["text"] = "Top 5 months"
    button_top5_months.place(x=605,y=250,width=190,height=30)
    button_top5_months["command"] = lambda:
    graph_total_value_by_month_topn(config.graph_top5_month,
        5, ax1, ax2, figure_canvas)

    button_top5_commodities_by_country=tk.Button(root)
    button_top5_commodities_by_country["bg"] = "#e9e9ed"
    button_top5_commodities_by_country["font"] = ft
    button_top5_commodities_by_country["fg"] = "#000000"
    button_top5_commodities_by_country["justify"] = "center"
    button_top5_commodities_by_country["text"] = "Top 5 commodities by country"
    button_top5_commodities_by_country.place(x=605,y=280,width=190,height=30)
    button_top5_commodities_by_country["command"] = lambda:
    graph_total_value_by_country_commodity_top5(config.graph_country_top5_commodity,
        ax1, ax2, figure_canvas)

    button_top_weekday_by_com=tk.Button(root)
    button_top_weekday_by_com["bg"] = "#e9e9ed"
    button_top_weekday_by_com["font"] = ft
    button_top_weekday_by_com["fg"] = "#000000"
    button_top_weekday_by_com["justify"] = "center"
    button_top_weekday_by_com["text"] = "Top day by commodity"
    button_top_weekday_by_com.place(x=605,y=310,width=190,height=30)
    button_top_weekday_by_com["command"] = lambda:
    graph_top_value_by_weekday_for_commodity(config.graph_commodity_top_weekday,
        ax1, ax2, figure_canvas)

    button_top_weekday_by_com=tk.Button(root)
```

```
    button_top_weekday_by_com["bg"] = "#e9e9ed"
    button_top_weekday_by_com["font"] = ft
    button_top_weekday_by_com["fg"] = "#000000"
    button_top_weekday_by_com["justify"] = "center"
    button_top_weekday_by_com["text"] = "Calculate & export csv and sql"
    button_top_weekday_by_com.place(x=605,y=340,width=190,height=30)
    button_top_weekday_by_com["command"] = lambda:
    calculate_all_data_and_export(label_csv_status)

    #Create figures and toolbox
    figure = Figure(figsize=(12, 6), dpi=100, tight_layout=True)
    figure_canvas = FigureCanvasTkAgg(figure, root)
    toolbar = NavigationToolbar2Tk(figure_canvas, root)
    toolbar.pack(side=tk.TOP)
    ax1 = figure.add_subplot()
    figure_canvas.get_tk_widget().place(x=0,y=40,width=600,height=560)

    ax1.clear()
    ax2=ax1.twinx()

    return root
```