

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 ^ο |

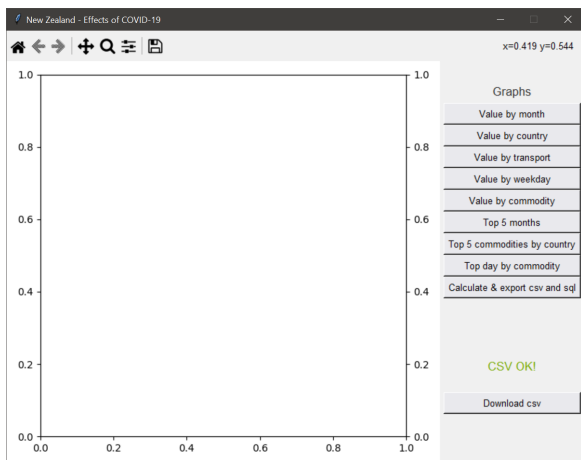
Περιεχόμενα

| | |
|------------------------------------|----------|
| I: Παραδείγματα λειτουργίας | 3 |
| I.a Βασικά σημεία του UI | 3 |
| I.b Γραφήματα | 4 |
| I.c SQLite Schema | 6 |
| II: Σχόλια-Παραδοχές | 7 |
| II.a Γενικά | 7 |
| II.b Οργάνωση κώδικα | 7 |
| III:Υπόμνημα | 8 |
| III.a SQLite Schema | 8 |
| III.b main.py | 9 |
| III.c config.py | 9 |
| III.d fun.py | 10 |

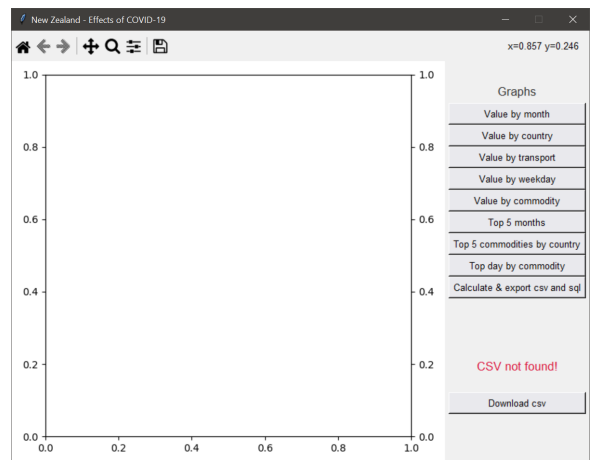


I: Παραδείγματα λειτουργίας

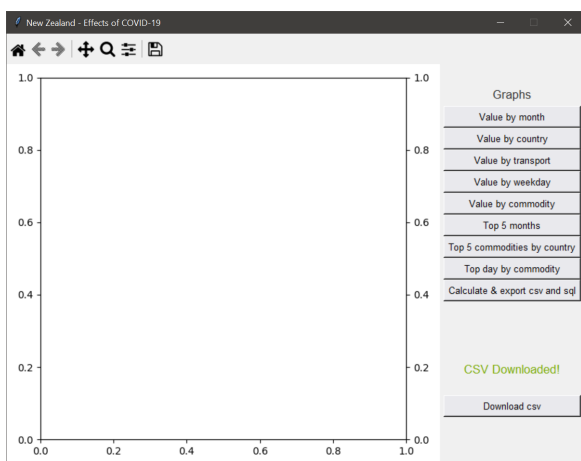
I.a Βασικά σημεία του UI



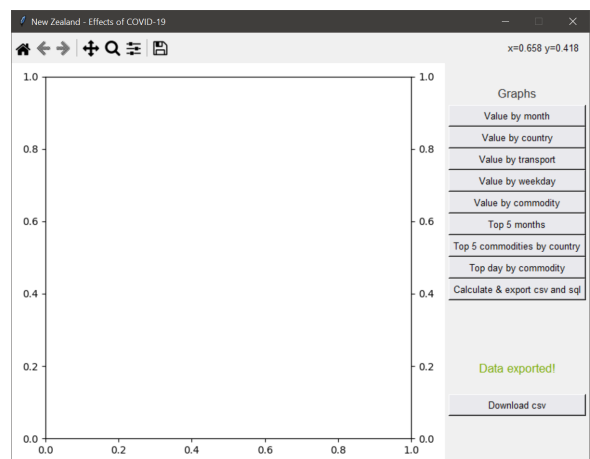
Βασική οθόνη, csv αρχείο βρέθηκε.



Το αρχείο csv δεν βρέθηκε.



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



Μήνυμα μετά από την εξαγωγή των δεδομένων.



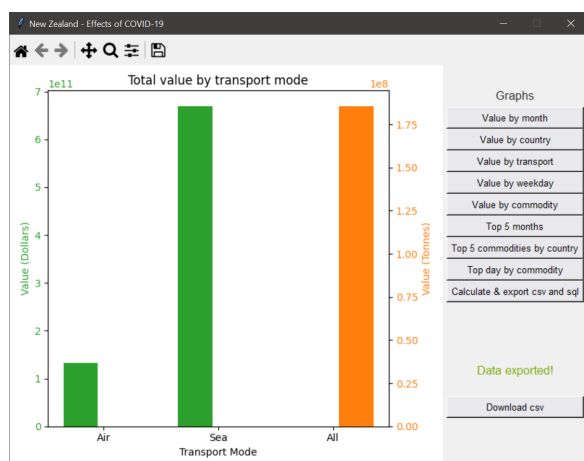
I.b Γραφήματα



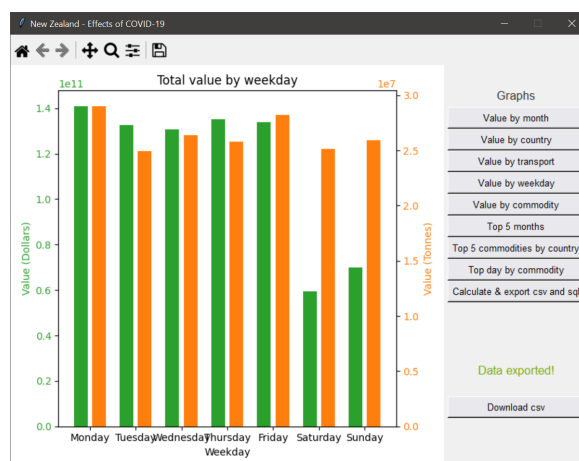
Σύνολο τζίρου ανά μήνα.



Σύνολο τζίρου ανά χώρα.



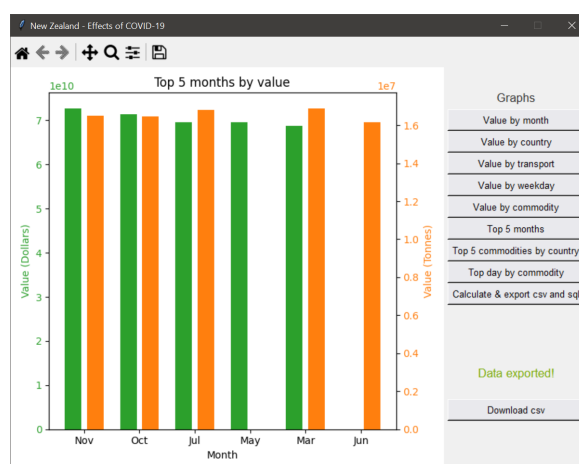
Σύνολο τζίρου ανά μέσο μεταφοράς.



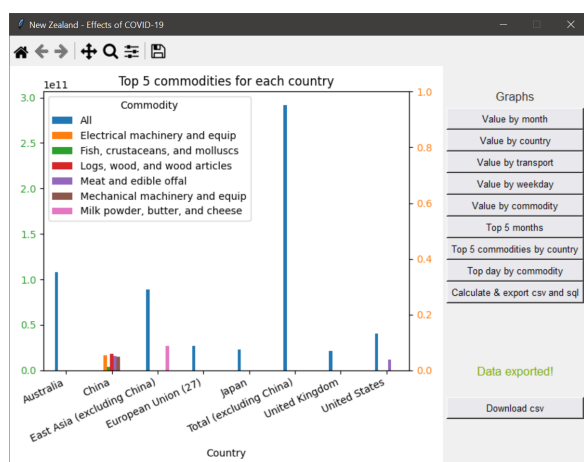
Σύνολο τζίρου ανά μέρα της εβδομάδας.



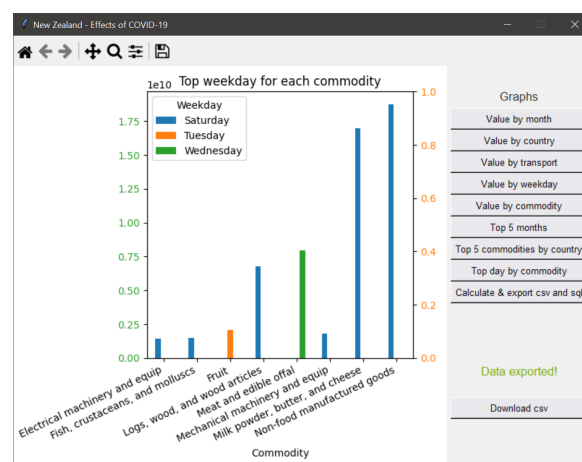
Σύνολο τζίρου ανά κατηγορίας εμπορεύματος.



5 μήνες με τον μεγαλύτερο τζίρο.



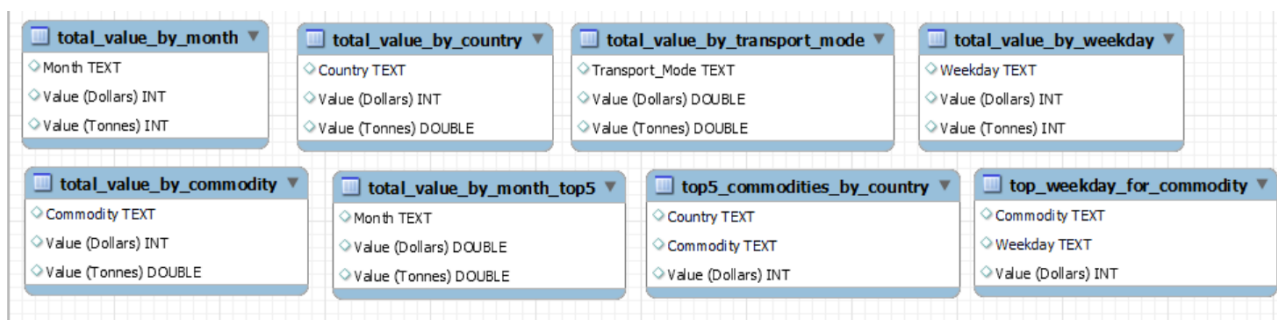
5 κατηγορίες εμπορευμάτων με τον μεγαλύτερο τζίρο για κάθε χώρα.



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 από 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

```
1 CREATE TABLE "top5_commodities_by_country" (  
2   "Country" TEXT,  
3   "Commodity" TEXT,  
4   "Value (Dollars)" INTEGER  
5 );  
6  
7 CREATE TABLE "top_weekday_for_commodity" (  
8   "Commodity" TEXT,  
9   "Weekday" TEXT,  
10  "Value (Dollars)" INTEGER  
11 );  
12  
13 CREATE TABLE "total_value_by_commodity" (  
14   "Commodity" TEXT,  
15   "Value (Dollars)" INTEGER,  
16   "Value (Tonnes)" REAL  
17 );  
18  
19 CREATE TABLE "total_value_by_country" (  
20   "Country" TEXT,  
21   "Value (Dollars)" INTEGER,  
22   "Value (Tonnes)" REAL  
23 );  
24  
25 CREATE TABLE "total_value_by_month" (  
26   "Month" TEXT,  
27   "Value (Dollars)" INTEGER,  
28   "Value (Tonnes)" INTEGER  
29 );  
30  
31 CREATE TABLE "total_value_by_month_top5" (  
32   "Month" TEXT,  
33   "Value (Dollars)" REAL,  
34   "Value (Tonnes)" REAL  
35 );  
36  
37 CREATE TABLE "total_value_by_transport_mode" (  
38   "Transport_Mode" TEXT,  
39   "Value (Dollars)" REAL,  
40   "Value (Tonnes)" REAL  
41 );  
42
```




```
43 CREATE TABLE "total_value_by_weekday" (  
44     "Weekday" TEXT,  
45     "Value (Dollars)" INTEGER,  
46     "Value (Tonnes)" INTEGER  
47 );
```

III.b main.py

```
1 import config  
2 import fun  
3  
4 def main():  
5     config.init()  
6     #config.save_to_sql=False #uncomment if you dont want to save to sqlite3 db.  
7     root = fun.gui_and_sql_init()  
8     root.mainloop()  
9  
10 if __name__ == "__main__":  
11     main()
```

III.c config.py

```
1 def init():  
2     global sqlconn  
3     global df  
4     global graph_month  
5     global graph_transport  
6     global graph_country  
7     global graph_weekday  
8     global graph_commodity  
9     global graph_top5_month  
10    global graph_country_top5_commodity  
11    global graph_commodity_top_weekday  
12    global save_to_sql  
13  
14    sqlconn = None  
15    df = None  
16    graph_month = None  
17    graph_transport = None  
18    graph_country = None  
19    graph_weekday = None  
20    graph_commodity = None  
21    graph_top5_month = None  
22    graph_country_top5_commodity = None  
23    graph_commodity_top_weekday = None  
24    save_to_sql = True
```



III.d fun.py

```
1 import calendar
2 import os
3 import sqlite3
4
5 import matplotlib
6 from matplotlib import pyplot as plt
7 from matplotlib.backends.backend_tkagg import (
8     FigureCanvasTkAgg,
9     NavigationToolbar2Tk
10 )
11 from matplotlib.figure import Figure
12 import numpy as np
13 import pandas as pd
14 import tkinter as tk
15 from tkinter import font as tkFont
16
17 import config
18
19 #Erwthma 1
20 def graph_total_value_by_month(graph_df: pd.DataFrame,
21     ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
22     """Show graph for total value by month.
23
24     Args:
25     -----
26
27     graph_df: ``DataFrame`` calculated by ``calculate_total_value_by_month``
28     ax1: ``Axes`` object for dollars.
29     ax1: ``Axes`` object for tonnes.
30     figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
31     """
32
33     #Calculate series objects if they haven't been calculated already
34     if config.graph_month is None:
35         config.graph_month = calculate_total_value_by_month(config.df)
36         graph_df = config.graph_month
37
38     x_indices = np.arange(len(graph_df.index)) #calculate spacing for x labels.
39     ax1.clear()
40     ax2.clear()
41
42     ax1.set_title('Total value by month')
43     ax1.set_xlabel('Month')
44     ax1.set_xticks(x_indices, graph_df.index) #manually set labels to fit 2 bars in
45     each x value
46     color = 'tab:green'
47     ax1.bar(x_indices-0.2, graph_df['Value (Dollars)'], 0.3, color=color)
48     ax1.set_ylabel('Value (Dollars)', color=color)
```



```
46 ax1.tick_params(axis='y', labelcolor=color)
47
48 color = 'tab:orange'
49 ax2.bar(x_indices+0.2, graph_df['Value (Tonnes)'], 0.3, color=color)
50 ax2.set_ylabel('Value (Tonnes)', color=color) # we already handled the x-label with
51 ax1
52 ax2.tick_params(axis='y', labelcolor=color)
53
54 #Move spines and info back to the right
55 ax2.spines['right'].set_position(('outward', 0)) # Move the spine to the right
56 ax2.yaxis.set_label_position('right')
57 ax2.yaxis.set_ticks_position('right')
58
59 figure_canvas.draw()
60
61 def calculate_total_value_by_month(df: pd.DataFrame):
62     """Calculate the DataFrame variable required by ``graph_total_value_by_month`` and
63     `calculate_total_value_by_month_topn`.
64
65     Args:
66     -----
67
68         df: ``DataFrame`` with Date column in datetime format.
69
70     Returns:
71     -----
72
73         graph_by_month: ``DataFrame`` series object ready to be used with
74         `calculate_total_value_by_month_topn`
75         or in order to be reused with `graph_total_value_by_month`.
76
77     """
78
79     #Dollars
80     df_by_month_dollar_group = df.query('Country=="All" & Commodity=="All" &
81     Transport_Mode=="All" & Measure=="$")\
82     .groupby(df.Date.dt.month)
83     df_by_month_dollar_group = df_by_month_dollar_group['Value'].sum()
84     df_by_month_dollar_group = df_by_month_dollar_group.rename('Value (Dollars)')
85     df_by_month_dollar_group = df_by_month_dollar_group.rename(lambda x:
86     calendar.month_abbr[x]) #match number to month
87     df_by_month_dollar_group = df_by_month_dollar_group.rename_axis("Month")
88
89     #Tonnes
90     df_by_month_tonne_group = df.query('Country=="All" & Transport_Mode=="All"&
91     Measure=="Tonnes")\
92     .groupby(df.Date.dt.month)
93     df_by_month_tonne_group = df_by_month_tonne_group['Value'].sum()
94     df_by_month_tonne_group = df_by_month_tonne_group.rename('Value (Tonnes)')
95     df_by_month_tonne_group = df_by_month_tonne_group.rename(lambda x:
96     calendar.month_abbr[x]) #match number to month
```



```
86 df_by_month_tonne_group = df_by_month_tonne_group.rename_axis("Month")
87
88 graph_by_month=pd.concat([df_by_month_dollar_group, df_by_month_tonne_group],
89 axis=1) # Merge Series objects
90
91 graph_by_month.to_csv('1_total_value_by_month.csv') #save to csv
92 if config.save_to_sql:
93     graph_by_month.to_sql('total_value_by_month', config.sqlconn, 'replace') #save
94     to sql
95
96 return graph_by_month
97
98 #Erwthma 2
99 def graph_total_value_by_country(graph_df: pd.DataFrame,
100 ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
101     """Show graph for total value by country.
102
103     Args:
104     -----
105     graph_df: ``DataFrame`` calculated by ``calculate_total_value_by_country``
106     ax1: ``Axes`` object for dollars.
107     ax1: ``Axes`` object for tonnes.
108     figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
109     """
110     #Calculate series objects if they haven't been calculated already.
111     if config.graph_country is None:
112         config.graph_country = calculate_total_value_by_country(config.df)
113         graph_df = config.graph_country
114
115     x_indices = np.arange(len(graph_df.index)) #calculate spacing for x labels.
116     ax1.clear()
117     ax2.clear()
118
119     ax1.set_title('Total value by country')
120     ax1.set_xlabel('Country')
121     ax1.set_xticks(x_indices, graph_df.index, rotation=25, ha='right') #manually set
122     labels to fit 2 bars in each x value
123
124     color = 'tab:green'
125     ax1.bar(x_indices-0.2, graph_df['Value (Dollars)'], 0.3, color=color)
126     ax1.set_ylabel('Value (Dollars)', color=color)
127     ax1.tick_params(axis='y', labelcolor=color)
128
129     color = 'tab:orange'
130     ax2.bar(x_indices+0.2, graph_df['Value (Tonnes)'], 0.3, color=color)
131     ax2.set_ylabel('Value (Tonnes)', color=color) # we already handled the x-label with
132     ax1
```



```
129 ax2.tick_params(axis='y', labelcolor=color)
130
131 #Move spines and info back to the right
132 ax2.spines['right'].set_position(('outward', 0)) # Move the spine to the right
133 ax2.yaxis.set_label_position('right')
134 ax2.yaxis.set_ticks_position('right')
135
136 figure_canvas.draw()
137
138 def calculate_total_value_by_country(df: pd.DataFrame):
139     """Calculate the DataFrame variable required by ``graph_total_value_by_country``
140
141     Args:
142     -----
143         df: ``DataFrame`` with Date column in datetime format.
144
145     Returns:
146     -----
147         graph_by_country: ``Dataframe`` series object ready to be reused with
148         ``graph_total_value_by_country``
149     """
150
151     #Dollars
152     df_by_country_dollar_group = df.query('Country != "All" & Commodity=="All" &
153     Transport_Mode=="All" & Measure=="$")\
154     .groupby('Country')
155     df_by_country_dollar_group = df_by_country_dollar_group['Value'].sum()
156     df_by_country_dollar_group = df_by_country_dollar_group.rename('Value (Dollars)')
157
158     #Tonnes
159     df_by_country_tonne_group = df.query('Country != "All" & Transport_Mode=="All" &
160     Measure=="Tonnes")\
161     .groupby('Country')
162     df_by_country_tonne_group = df_by_country_tonne_group['Value'].sum()
163     df_by_country_tonne_group = df_by_country_tonne_group.rename('Value (Tonnes)')
164
165     graph_by_country=pd.concat([df_by_country_dollar_group, df_by_country_tonne_group],
166     axis=1) # Merge Series objects
167
168     graph_by_country.to_csv('2_total_value_by_country.csv') #save to csv
169     if config.save_to_sql:
170         graph_by_country.to_sql('total_value_by_country', config.sqlconn, 'replace')
171         #save to sql
172     return graph_by_country
173
174 #Erwthma 3
```



```
171 def graph_total_value_by_transport(graph_df: pd.DataFrame,
172 ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
173     """Show graph for total value by transport mode.
174
175     Args:
176     -----
177         graph_df: ``DataFrame`` calculated by ``calculate_total_value_by_transport``.
178         ax1: ``Axes`` object for dollars.
179         ax1: ``Axes`` object for tonnes.
180         figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
181     """
182     #Calculate series objects if they haven't been calculated already
183     if config.graph_transport is None:
184         config.graph_transport = calculate_total_value_by_transport(config.df)
185         graph_df = config.graph_transport
186
187     x_indices = np.arange(len(graph_df.index)) #calculate spacing for x labels.
188     ax1.clear()
189     ax2.clear()
190
191     ax1.set_title('Total value by transport mode')
192     ax1.set_xlabel('Transport Mode')
193     ax1.set_xticks(x_indices, graph_df.index) #manually set labels to fit 2 bars in
194     each x value
195
196     color = 'tab:green'
197     ax1.bar(x_indices-0.2, graph_df['Value (Dollars)'], 0.3, color=color)
198     ax1.set_ylabel('Value (Dollars)', color=color)
199     ax1.tick_params(axis='y', labelcolor=color)
200
201     color = 'tab:orange'
202     ax2.bar(x_indices+0.2, graph_df['Value (Tonnes)'], 0.3, color=color)
203     ax2.set_ylabel('Value (Tonnes)', color=color) # we already handled the x-label with
204     ax1
205     ax2.tick_params(axis='y', labelcolor=color)
206
207     #Move spines and info back to the right
208     ax2.spines['right'].set_position(('outward', 0)) # Move the spine to the right
209     ax2.yaxis.set_label_position('right')
210     ax2.yaxis.set_ticks_position('right')
211
212     figure_canvas.draw()
213
214 def calculate_total_value_by_transport(df: pd.DataFrame):
215     """Calculate the DataFrame variable required by ``graph_total_value_by_transport``.
216
217     Args:
```



```
216     -----
217     df: ``DataFrame`` with Date column in datetime format.
218
219     Returns:
220     -----
221     graph_by_transport: ``DataFrame`` series object ready to be reused with
222     ``graph_total_value_by_transport``
223     -----
224     """
225
226     #Dollars
227     df_by_transport_dollar_group = df.query('Country=="All" & Commodity=="All" &
228     Transport_Mode != "All" & Measure=="$")\
229     .groupby('Transport_Mode')
230     df_by_transport_dollar_group = df_by_transport_dollar_group['Value'].sum()
231     df_by_transport_dollar_group = df_by_transport_dollar_group.rename('Value
232     (Dollars)')
233
234     #Tonnes
235     df_by_transport_tonne_group = df.query('Country=="All" & Measure=="Tonnes")\
236     .groupby('Transport_Mode') #! Transport modes don't exist for tonnes
237     df_by_transport_tonne_group = df_by_transport_tonne_group['Value'].sum()
238     df_by_transport_tonne_group = df_by_transport_tonne_group.rename('Value (Tonnes)')
239
240     graph_by_transport=pd.concat([df_by_transport_dollar_group,
241     df_by_transport_tonne_group], axis=1) # Merge Series objects
242
243
244     graph_by_transport.to_csv('3_total_value_by_transport_mode.csv') #save to csv
245     if config.save_to_sql:
246         graph_by_transport.to_sql('total_value_by_transport_mode', config.sqlconn,
247         'replace')
248     return graph_by_transport
249
250
251 #Erwthma 4
252 def graph_total_value_by_weekday(graph_df: pd.DataFrame,
253 ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
254     """Show graph for total value by weekday.
255
256     Args:
257     -----
258
259     graph_df: ``DataFrame`` calculated by ``calculate_total_value_by_weekday``
260     ax1: ``Axes`` object for dollars.
261     ax1: ``Axes`` object for tonnes.
262     figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
263     """
264
265     #Calculate series objects if they haven't been calculated already
266     if config.graph_weekday is None:
```



```
258     config.graph_weekday = calculate_total_value_by_weekday(config.df)
259     graph_df = config.graph_weekday
260
261     x_indices = np.arange(len(graph_df.index)) #calculate spacing for x labels.
262     ax1.clear()
263     ax2.clear()
264
265     ax1.set_title('Total value by weekday')
266     ax1.set_xlabel('Weekday')
267     ax1.set_xticks(x_indices, graph_df.index) #manually set labels to fit 2 bars in
each x value
268
269     color = 'tab:green'
270     ax1.bar(x_indices-0.2, graph_df['Value (Dollars)'], 0.3, color=color)
271     ax1.set_ylabel('Value (Dollars)', color=color)
272     ax1.tick_params(axis='y', labelcolor=color)
273
274     color = 'tab:orange'
275     ax2.bar(x_indices+0.2, graph_df['Value (Tonnes)'], 0.3, color=color)
276     ax2.set_ylabel('Value (Tonnes)', color=color) # we already handled the x-label with
ax1
277     ax2.tick_params(axis='y', labelcolor=color)
278
279     #Move spines and info back to the right
280     ax2.spines['right'].set_position(('outward', 0)) # Move the spine to the right
281     ax2.yaxis.set_label_position('right')
282     ax2.yaxis.set_ticks_position('right')
283
284     figure_canvas.draw()
285
286 def calculate_total_value_by_weekday(df: pd.DataFrame):
287     """Calculate the DataFrame required by ``graph_total_value_by_weekday``.
288
289     Args:
290     -----
291     df: ``DataFrame`` with Date column in datetime format.
292
293     Returns:
294     -----
295     graph_by_weekday: ``DataFrame`` series object ready to be reused with
``graph_total_value_by_weekday``.
296     -----
297     """
298
299     #Dollars
300     df_by_weekday_dollar_group = df.query('Country=="All" & Commodity=="All" &
        Transport_Mode=="All" & Measure=="$")\'
```




```
301     .groupby('Weekday')
302 df_by_weekday_dollar_group = df_by_weekday_dollar_group['Value'].sum()
303 df_by_weekday_dollar_group =
304 df_by_weekday_dollar_group.reindex(list(calendar.day_name)) #sort days
305 df_by_weekday_dollar_group = df_by_weekday_dollar_group.rename('Value (Dollars)')
306
307 #Tonnes
308 df_by_weekday_tonne_group = df.query('Country=="All" & Transport_Mode=="All" &
309 Measure=="Tonnes"')\
310 .groupby('Weekday')
311 df_by_weekday_tonne_group = df_by_weekday_tonne_group['Value'].sum()
312 df_by_weekday_tonne_group =
313 df_by_weekday_tonne_group.reindex(list(calendar.day_name)) #sort days
314 df_by_weekday_tonne_group = df_by_weekday_tonne_group.rename('Value (Tonnes)')
315
316 graph_by_weekday=pd.concat([df_by_weekday_dollar_group, df_by_weekday_tonne_group],
317 axis=1) # Merge Series objects
318
319 graph_by_weekday.to_csv('4_total_value_by_weekday.csv') #save to csv
320 if config.save_to_sql:
321     graph_by_weekday.to_sql('total_value_by_weekday', config.sqlconn, 'replace')
322 return graph_by_weekday
323
324 #Erwthma 5
325 def graph_total_value_by_commodity(graph_df: pd.DataFrame,
326 ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
327     """Show graph for total value by commodity.
328
329     Args:
330     -----
331         graph_df: ``DataFrame`` calculated by ``calculate_total_value_by_commodity``
332         ax1: ``Axes`` object for dollars.
333         ax2: ``Axes`` object for tonnes.
334         figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
335     """
336     #Calculate series objects if they haven't been calculated already
337     if config.graph_commodity is None:
338         config.graph_commodity = calculate_total_value_by_commodity(config.df)
339         graph_df = config.graph_commodity
340
341     x_indices = np.arange(len(graph_df.index)) #calculate spacing for x labels.
342     ax1.clear()
343     ax2.clear()
344
345     ax1.set_title('Total value by commodity')
346     ax1.set_xlabel('Commodity')
```



```
343 ax1.set_xticks(x_indices, graph_df.index, rotation=25, ha='right') #manually set  
344 labels to fit 2 bars in each x value  
345  
346 color = 'tab:green'  
347 ax1.bar(x_indices-0.2, graph_df['Value (Dollars)'], 0.3, color=color)  
348 ax1.set_ylabel('Value (Dollars)', color=color)  
349 ax1.tick_params(axis='y', labelcolor=color)  
350  
351 color = 'tab:orange'  
352 ax2.bar(x_indices+0.2, graph_df['Value (Tonnes)'], 0.3, color=color)  
353 ax2.set_ylabel('Value (Tonnes)', color=color) # we already handled the x-label with  
354 ax1  
355 ax2.tick_params(axis='y', labelcolor=color)  
356  
357 #Move spines and info back to the right  
358 ax2.spines['right'].set_position(('outward', 0)) # Move the spine to the right  
359 ax2.yaxis.set_label_position('right')  
360 ax2.yaxis.set_ticks_position('right')  
361  
362 figure_canvas.draw()  
363  
364 def calculate_total_value_by_commodity(df: pd.DataFrame):  
365     """Calculate the DataFrame variable required by ``graph_total_value_by_commodity``.  
366  
367     Args:  
368     -----  
369     df: ``DataFrame`` with Date column in datetime format.  
370  
371     Returns:  
372     -----  
373     graph_by_commodity: ``DataFrame`` series object ready to be used with  
374     ``graph_total_value_by_commodity``.  
375     -----  
376     """  
377  
378     #Dollars  
379     df_by_commodity_dollar_group = df.query('Country=="All" & Commodity != "All" &  
380     Transport_Mode=="All" & Measure=="$")\  
381     .groupby('Commodity')  
382     df_by_commodity_dollar_group = df_by_commodity_dollar_group['Value'].sum()  
383     df_by_commodity_dollar_group = df_by_commodity_dollar_group.rename('Value  
384     (Dollars)')  
385  
386     #Tonnes  
387     df_by_commodity_tonne_group = df.query('Country=="All" & Transport_Mode=="All" &  
388     Measure=="Tonnes")\  
389     .groupby('Commodity')
```



```
384 df_by_commodity_tonne_group = df_by_commodity_tonne_group['Value'].sum()
385 df_by_commodity_tonne_group = df_by_commodity_tonne_group.rename('Value (Tonnes)')
386
387 graph_by_commodity=pd.concat([df_by_commodity_dollar_group,
388 df_by_commodity_tonne_group], axis=1) # Merge Series objects
389
389 graph_by_commodity.to_csv('5_total_value_by_commodity.csv') #save to csv
390 if config.save_to_sql:
391     graph_by_commodity.to_sql('total_value_by_commodity', config.sqlconn, 'replace')
392 return graph_by_commodity
393
394 #Erwthma 6
395 def graph_total_value_by_month_topn(graph_df: pd.DataFrame, n: int,
396 ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
397     """Show graph for total value by month, top n values.
398
399     Args:
400     -----
401     graph_df: ``DataFrame`` calculated by ``calculate_total_value_by_month``.
402     n: ``int`` how many of the top values to show.
403     ax1: ``Axes`` object for dollars.
404     ax1: ``Axes`` object for tonnes.
405     figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
406     """
407     #Calculate series objects if they haven't been calculated already
408     if config.graph_top5_month is None:
409         config.graph_top5_month = calculate_total_value_by_month_topn(config.df, 5)
410         graph_df = config.graph_top5_month
411
412     x_indices = np.arange(len(graph_df.index)) #calculate spacing for x labels.
413     ax1.clear()
414     ax2.clear()
415
416     ax1.set_title('Top 5 months by value')
417     ax1.set_xlabel('Month')
418     ax1.set_xticks(x_indices, graph_df.index) #manually set labels to fit 2 bars in
419     each x value
420
421     color = 'tab:green'
422     ax1.bar(x_indices-0.2, graph_df['Value (Dollars)'], 0.3, color=color)
423     ax1.set_ylabel('Value (Dollars)', color=color)
424     ax1.tick_params(axis='y', labelcolor=color)
425
426     color = 'tab:orange'
427     ax2.bar(x_indices+0.2, graph_df['Value (Tonnes)'], 0.3, color=color)
428     ax2.set_ylabel('Value (Tonnes)', color=color) # we already handled the x-label with
429     ax1
```



```
428     ax2.tick_params(axis='y', labelcolor=color)
429
430     #Move spines and info back to the right
431     ax2.spines['right'].set_position(('outward', 0)) # Move the spine to the right
432     ax2.yaxis.set_label_position('right')
433     ax2.yaxis.set_ticks_position('right')
434
435     figure_canvas.draw()
436
437 def calculate_total_value_by_month_topn(graph_df: pd.DataFrame, n: int):
438     """Calculate the DataFrame variable required by ``graph_total_value_by_month_topn``.
439
440     Args:
441     -----
442
443     graph_df: ``DataFrame`` calculated from ``calculate_total_value_by_month``.
444     n: ``int`` how many of the top values to calculate.
445
446     Returns:
447     -----
448
449     graph_by_month_topn: ``DataFrame`` series object ready to be used with
450     ``graph_total_value_by_month_topn``.
451     """
452
453     if config.graph_month is None:
454         config.graph_month = calculate_total_value_by_month(config.df)
455         graph_df = config.graph_month
456
457     df_top5_dollars=graph_df.nlargest(5, 'Value (Dollars)')['Value (Dollars)']
458     df_top5_tonnes=graph_df.nlargest(5, 'Value (Tonnes)')['Value (Tonnes)']
459     graph_by_month_topn = pd.concat([df_top5_dollars, df_top5_tonnes], axis=1) #concat
460     into one DataFrame
461
462     graph_by_month_topn.to_csv('6_total_value_by_month_top5.csv')
463     if config.save_to_sql:
464         graph_by_month_topn.to_sql('total_value_by_month_top5', config.sqlconn,
465                                     'replace')
466
467     return graph_by_month_topn
468
469 #Erwthma 7
470 def graph_total_value_by_country_commodity_top5(dollar_df: pd.DataFrame,
471 ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
472     """Show graph for top 5 commodities of each country.
473
474     Args:
475     -----
476
477     dollar_df: ``DataFrame`` calculated by
478     ``calculate_total_value_by_ccountry_commodity``
```



```
471     ax1: ``Axes`` object for dollars.
472     ax1: ``Axes`` object for tonnes.
473     figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
474     """
475     #Calculate DataFrame objects if they haven't been calculated already
476     if config.graph_country_top5_commodity is None:
477         config.graph_country_top5_commodity =
478         calculate_total_value_by_country_commodity_top5(config.df)
479         dollar_df = config.graph_country_top5_commodity
480
481     ax1.clear()
482     ax2.clear()
483
484     ax1 = dollar_df.plot(kind="bar", ax=ax1)
485     ax1.set_xticklabels(ax1.get_xticklabels(), rotation=25, ha='right')
486     ax1.set_title("Top 5 commodities for each country")
487
488     figure_canvas.draw()
489
490 def calculate_total_value_by_country_commodity_top5(df: pd.DataFrame):
491     """Calculate the series variable required by
492     ``graph_total_value_by_country_commodity``.
493
494     Args:
495     -----
496     df: ``DataFrame`` with Date column in datetime format.
497
498     Returns:
499     -----
500     df_by_country_commodity_dollar: ``DataFrame`` object ready to be used with
501     ``graph_total_value_by_country_commodity``.
502     """
503
504     #Dollars
505     df_by_country_commodity_dollar_group = df.query('Country!="All" & Measure=="$")\
506     .groupby(['Country',
507     'Commodity']).agg({'Value':sum}).sort_values('Value').groupby('Country').head(5)
508     #regroup by country to get top 5 commodities of each country
509     df_by_country_commodity_dollar_group =
510     df_by_country_commodity_dollar_group.rename(columns={'Value':'Value (Dollars)'})
511
512     df_by_country_commodity_dollar_group.to_csv('7_top5_commodities_by_country.csv')
513     #save to csv
514     if config.save_to_sql:
515         df_by_country_commodity_dollar_group.to_sql('top5_commodities_by_country',
516         config.sqlconn, 'replace')
```



```
510
511 df_by_country_commodity_dollar_group=
512     pd.pivot_table(df_by_country_commodity_dollar_group,
513                     values='Value (Dollars)', index='Country', columns='Commodity')
514
515     return df_by_country_commodity_dollar_group
516
517 #Erwthma 8
518 def graph_top_value_by_weekday_for_commodity(dollar_df: pd.DataFrame,
519 ax1: plt.Axes , ax2: plt.Axes, figure_canvas: FigureCanvasTkAgg):
520     """Show graph for top weekday for each commodity.
521
522     Args:
523     -----
524
525     dollar_df: ``DataFrame`` calculated by
526     ``calculate_top_value_by_weekday_for_commodity``
527     ax1: ``Axes`` object for dollars.
528     figure_canvas: ``FigureCanvasTkAgg`` object for drawing on.
529     """
530
531     #Calculate DataFrame objects if they haven't been calculated already
532     if config.graph_commodity_top_weekday is None:
533         config.graph_commodity_top_weekday =
534             calculate_top_value_by_weekday_for_commodity(config.df)
535         dollar_df = config.graph_commodity_top_weekday
536
537     ax1.clear()
538     ax2.clear()
539
540     ax1 = dollar_df.plot(kind="bar", ax=ax1)
541     ax1.set_xticklabels(ax1.get_xticklabels(), rotation=25, ha='right')
542     ax1.set_title("Top weekday for each commodity")
543
544     figure_canvas.draw()
545
546 def calculate_top_value_by_weekday_for_commodity(df: pd.DataFrame):
547     """Calculate the series variable required by
548     ``graph_top_value_by_weekday_for_commodity``.
549
550     Args:
551     -----
552
553     df: ``DataFrame`` with Date column in datetime format.
554
555     Returns:
556     -----
557
558     df_by_country_commodity_dollar: ``DataFrame`` object ready to be used with
559     ``graph_top_value_by_weekday_for_commodity``.
560     -----
```



```
552 """
553
554 #Dollars
555 df_by_country_commodity_dollar_group = df.query('Commodity!="All" & Measure=="$")\
556 .groupby(['Commodity',
557 'Weekday']).agg({'Value':sum}).sort_values('Value').groupby('Commodity').head(1)
558 #sort by value, regroup by country to get top weekday for each commodity
559 df_by_country_commodity_dollar_group =
560 df_by_country_commodity_dollar_group.rename(columns={'Value':'Value (Dollars)'})
561
562 df_by_country_commodity_dollar_group.to_csv('8_top_weekday_for_commodity.csv')
563 #save to csv
564 if config.save_to_sql:
565     df_by_country_commodity_dollar_group.to_sql('top_weekday_for_commodity',
566         config.sqlconn, 'replace')
567
568 df_by_country_commodity_dollar_group=
569 pd.pivot_table(df_by_country_commodity_dollar_group,
570     values='Value (Dollars)', index='Commodity', columns='Weekday')
571
572 return df_by_country_commodity_dollar_group
573
574 def calculate_all_data_and_export(csv: tk.Label):
575     if config.graph_month is None:
576         config.graph_month = calculate_total_value_by_month(config.df)
577
578     if config.graph_country is None:
579         config.graph_country = calculate_total_value_by_country(config.df)
580
581     if config.graph_transport is None:
582         config.graph_transport = calculate_total_value_by_transport(config.df)
583
584     if config.graph_weekday is None:
585         config.graph_weekday = calculate_total_value_by_weekday(config.df)
586
587     if config.graph_commodity is None:
588         config.graph_commodity = calculate_total_value_by_commodity(config.df)
589
590     if config.graph_top5_month is None:
591         config.graph_top5_month =
592             calculate_total_value_by_month_topn(config.graph_month, 5)
593
594     if config.graph_country_top5_commodity is None:
595         config.graph_country_top5_commodity =
596             calculate_total_value_by_country_commodity_top5(config.df)
```



```
591     if config.graph_commodity_top_weekday is None:
592         config.graph_commodity_top_weekday =
593             calculate_top_value_by_weekday_for_commodity(config.df)
594
595     csv["fg"] = "#7CAC06"
596     csv["text"] = "Data exported!"
597
598 def check_for_csv(csv: tk.Label):
599     fileName = 'covid19_effects_trades.csv'
600     path = os.getcwd() + os.sep + fileName
601     if os.path.exists(path):
602         csv["fg"] = "#7CAC06"
603         csv["text"] = "CSV OK!"
604         df = pd.read_csv(path)
605         df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y')
606         config.df = df
607     else:
608         csv["fg"] = "#DC143C"
609         csv["text"] = "CSV not found!"
610
611 def download_csv(csv: tk.Label):
612     url =
613         'https://www.stats.govt.nz/assets/Uploads/Effects-of-COVID-19-on-trade/Effects-of-COVID-19-on-trade-at-15-december-2021-provisional/Download-data/effects-of-covid-19-on-trade-at-15-december-2021-provisional.csv'
614     fileName = 'covid19_effects_trades'
615
616     # download csv and save to fileName locally
617     df = pd.read_csv(url)
618     df.to_csv(fileName+'.csv', index=False)
619     df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y')
620     config.df = df
621
622     csv["fg"] = "#7CAC06"
623     csv["text"] = "CSV Downloaded!"
624
625 def db_create_connection():
626     '''
627     Returns connection object for sqlite3 database.
628     '''
629     conn = None
630     try:
631         conn = sqlite3.connect('new_zealand_covid_effects.db')
632         return conn
633     except sqlite3.Error as e:
634         print(e)
635
```




```
636 def gui_and_sql_init():
637     """Initialize GUI and sql.
638
639     Returns
640     -----
641     `root`.
642
643     """
644     config.sqlconn = db_create_connection()
645
646     matplotlib.use('TkAgg')
647     #Create root window
648     root = tk.Tk()
649     root.title("New Zealand - Effects of COVID-19")
650     width=800
651     height=600
652     screenwidth = root.winfo_screenwidth()
653     screenheight = root.winfo_screenheight()
654     #align to middle of screen
655     alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight
        - height) / 2)
656     root.geometry(alignstr)
657     root.resizable(width=False, height=False)
658
659     #Create labels
660     ft = tkFont.Font(family='Roboto',size=13)
661     label_graphs=tk.Label(root)
662     label_graphs["font"] = ft
663     label_graphs["fg"] = "#333333"
664     label_graphs["justify"] = "center"
665     label_graphs["text"] = "Graphs"
666     label_graphs.place(x=605,y=70,width=190,height=30)
667
668     label_csv_status=tk.Label(root)
669     label_csv_status["font"] = ft
670     label_csv_status["fg"] = "#333333"
671     label_csv_status["justify"] = "center"
672     label_csv_status["text"] = "CSV status?"
673     label_csv_status.place(x=605,y=450,width=190,height=30)
674     check_for_csv(label_csv_status) #update label and color
675
676     #Create buttons
677     ft = tkFont.Font(family='Roboto',size=10)
678
679     button_download_csv=tk.Button(root)
680     button_download_csv["bg"] = "#e9e9ed"
681     button_download_csv["font"] = ft
```



```
682 button_download_csv["fg"] = "#000000"
683 button_download_csv["justify"] = "center"
684 button_download_csv["text"] = "Download csv"
685 button_download_csv.place(x=605,y=500,width=190,height=30)
686 button_download_csv["command"] = lambda: download_csv(label_csv_status)
687
688 button_by_month=tk.Button(root)
689 button_by_month["bg"] = "#e9e9ed"
690 button_by_month["font"] = ft
691 button_by_month["fg"] = "#000000"
692 button_by_month["justify"] = "center"
693 button_by_month["text"] = "Value by month"
694 button_by_month.place(x=605,y=100,width=190,height=30)
695 button_by_month["command"] = lambda: graph_total_value_by_month(config.graph_month,
696     ax1, ax2, figure_canvas)
697
698 button_by_country=tk.Button(root)
699 button_by_country["bg"] = "#e9e9ed"
700 button_by_country["font"] = ft
701 button_by_country["fg"] = "#000000"
702 button_by_country["justify"] = "center"
703 button_by_country["text"] = "Value by country"
704 button_by_country.place(x=605,y=130,width=190,height=30)
705 button_by_country["command"] = lambda:
706     graph_total_value_by_country(config.graph_country,
707     ax1, ax2, figure_canvas)
708
709 button_by_transport=tk.Button(root)
710 button_by_transport["bg"] = "#e9e9ed"
711 button_by_transport["font"] = ft
712 button_by_transport["fg"] = "#000000"
713 button_by_transport["justify"] = "center"
714 button_by_transport["text"] = "Value by transport"
715 button_by_transport.place(x=605,y=160,width=190,height=30)
716 button_by_transport["command"] = lambda:
717     graph_total_value_by_transport(config.graph_transport,
718     ax1, ax2, figure_canvas)
719
720 button_by_weekday=tk.Button(root)
721 button_by_weekday["bg"] = "#e9e9ed"
722 button_by_weekday["font"] = ft
723 button_by_weekday["fg"] = "#000000"
724 button_by_weekday["justify"] = "center"
725 button_by_weekday["text"] = "Value by weekday"
726 button_by_weekday.place(x=605,y=190,width=190,height=30)
727 button_by_weekday["command"] = lambda:
728     graph_total_value_by_weekday(config.graph_weekday,
```



```
726         ax1, ax2, figure_canvas)
727
728     button_by_commodity=tk.Button(root)
729     button_by_commodity["bg"] = "#e9e9ed"
730     button_by_commodity["font"] = ft
731     button_by_commodity["fg"] = "#000000"
732     button_by_commodity["justify"] = "center"
733     button_by_commodity["text"] = "Value by commodity"
734     button_by_commodity.place(x=605,y=220,width=190,height=30)
735     button_by_commodity["command"] = lambda:
736         graph_total_value_by_commodity(config.graph_commodity,
737             ax1, ax2, figure_canvas)
738
739     button_top5_months=tk.Button(root)
740     button_top5_months["bg"] = "#e9e9ed"
741     button_top5_months["font"] = ft
742     button_top5_months["fg"] = "#000000"
743     button_top5_months["justify"] = "center"
744     button_top5_months["text"] = "Top 5 months"
745     button_top5_months.place(x=605,y=250,width=190,height=30)
746     button_top5_months["command"] = lambda:
747         graph_total_value_by_month_topn(config.graph_top5_month,
748             5, ax1, ax2, figure_canvas)
749
750     button_top5_commodities_by_country=tk.Button(root)
751     button_top5_commodities_by_country["bg"] = "#e9e9ed"
752     button_top5_commodities_by_country["font"] = ft
753     button_top5_commodities_by_country["fg"] = "#000000"
754     button_top5_commodities_by_country["justify"] = "center"
755     button_top5_commodities_by_country["text"] = "Top 5 commodities by country"
756     button_top5_commodities_by_country.place(x=605,y=280,width=190,height=30)
757     button_top5_commodities_by_country["command"] = lambda:
758         graph_total_value_by_country_commodity_top5(config.graph_country_top5_commodity,
759             ax1, ax2, figure_canvas)
760
761     button_top_weekday_by_com=tk.Button(root)
762     button_top_weekday_by_com["bg"] = "#e9e9ed"
763     button_top_weekday_by_com["font"] = ft
764     button_top_weekday_by_com["fg"] = "#000000"
765     button_top_weekday_by_com["justify"] = "center"
766     button_top_weekday_by_com["text"] = "Top day by commodity"
767     button_top_weekday_by_com.place(x=605,y=310,width=190,height=30)
768     button_top_weekday_by_com["command"] = lambda:
769         graph_top_value_by_weekday_for_commodity(config.graph_commodity_top_weekday,
770             ax1, ax2, figure_canvas)
```



```
769 button_top_weekday_by_com["bg"] = "#e9e9ed"
770 button_top_weekday_by_com["font"] = ft
771 button_top_weekday_by_com["fg"] = "#000000"
772 button_top_weekday_by_com["justify"] = "center"
773 button_top_weekday_by_com["text"] = "Calculate & export csv and sql"
774 button_top_weekday_by_com.place(x=605,y=340,width=190,height=30)
775 button_top_weekday_by_com["command"] = lambda:
    calculate_all_data_and_export(label_csv_status)
776
777 #Create figures and toolbox
778 figure = Figure(figsize=(12, 6), dpi=100, tight_layout=True)
779 figure_canvas = FigureCanvasTkAgg(figure, root)
780 toolbar = NavigationToolbar2Tk(figure_canvas, root)
781 toolbar.pack(side=tk.TOP)
782 ax1 = figure.add_subplot()
783 figure_canvas.get_tk_widget().place(x=0,y=40,width=600,height=560)
784
785 ax1.clear()
786 ax2=ax1.twinx()
787
788 return root
```