DS_Proyecto_02_Synergy_Logistics

Generado por Doxygen 1.9.2

Capítulo 1

Indice de namespaces

1.1. Paquetes

Aquí van los paquetes con una breve descripción (si etá disponible):

Capítulo 2

Indice jerárquico

2.1. Jerarquía de la clase

Esta lista de herencias esta ordenada aproximadamente por orden alfabético:

Capítulo 3

Índice de clases

3.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

Capítulo 4

Indice de archivos

4.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

Capítulo 5

Documentación de namespaces

5.1. Referencia del Namespace ANALISIS_02_CASTANEDA_EDSON

Variables

- service = Service()
- list DIRECCIONES = [None, "Imports", "Exports"]
- list PERIODO_TIEMPO = [None, 2015, 2016, 2017, 2018, 2019, 2020]
- TRANSPORT_MODES = service.get_unique_values("transport_mode")
- ORIGIN_COUNTRIES = service.get_unique_values("origin")
- DESTINATION_COUNTRIES = service.get_unique_values("destination")
- ROUTES = service.get_routes_list()
- string OUTPUT_INITIAL_PATH = "exploration"
- list year_list = [ ]
- list value_pct_list = [ ]
- list frecuency_pct_list = [ ]
- string direction_str = "All"
- string output_folder = f"{OUTPUT_INITIAL_PATH}/opcion_1/{direction_str}"
- dictionary route_frecuency = {}
- dictionary route_value = {}
- dictionary route_frecuency_pct = {}
- dictionary route_value_pct = {}
- total_cases = service.get_total_elements(direction=direction, year=year)
- total_value = service.get_total_value(direction=direction, year=year)
- route
- direction
- year
- top_ten_frecuency = service.get_top_ten(route_frecuency)
- top_ten_value = service.get_top_ten(route_value)
- top_ten_frecuency_pct = service.get_top_ten(route_frecuency_pct)
- top_ten_value_pct = service.get_top_ten(route_value_pct)
- string year_str = "All"
- int i = 1
- string output_folder_year = f"{output_folder}/{year_str}"
- dictionary data
- index
- plot = Summary_Chart(output_folder_year)
- dictionary transport_frecuency = {}

- dictionary transport_value = {}
- dictionary transport_frecuency_pct = {}
- dictionary transport_value_pct = {}
- transport
- string output_import_folder = f"{OUTPUT_INITIAL_PATH}/opcion_3/Imports"
- string output_export_folder = f"{OUTPUT_INITIAL_PATH}/opcion_3/Exports"
- dictionary country_import_value = {}
- dictionary country_import_value_pct = {}
- dictionary country_export_value = {}
- dictionary country_export_value_pct = {}
- total_import = service.get_total_value(direction="Imports", year=year)
- total_export = service.get_total_value(direction="Exports", year=year)
- destination
- country
- origin
- dictionary country_total_value = {k: country_export_value.get(k, 0) + country_import_value.get(k, 0) for k in set(country_export_value) | set(country_import_value)}
- dictionary country_total_value_pct = {k: round((v/total_value)∗100,2) for k, v in country_total_value.items()}
- string output_import_folder_year = f"{output_import_folder}/{year_str}"
- string output_export_folder_year = f"{output_export_folder}/{year_str}"
- dictionary import_data
- df = pd.DataFrame({ key:pd.Series(value) for key, value in import_data.items()}).fillna(0)
- dictionary export_data
- dictionary total_data
- import_plot = Summary_Chart(output_import_folder_year)
- export_plot = Summary_Chart(output_export_folder_year)

5.1.1. Descripción detallada

```
ANALISIS SYNERGY LOGISTICS

En este script se exploran las 3 propuestas de enfoque de la empresa Synergy Logistics.
A partir de una revision total y por año se pretende determinar la mejor opción
que se puede considerar para los proximos años.
```

5.1.2. Documentación de las variables

5.1.2.1. country

```
ANALISIS_02_CASTANEDA_EDSON.country
```

Definición en la línea 205 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.2. country_export_value

```
ANALISIS_02_CASTANEDA_EDSON.country_export_value = {}
```

Definición en la línea 195 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.3. country_export_value_pct

```
ANALISIS_02_CASTANEDA_EDSON.country_export_value_pct = {}
```

Definición en la línea 196 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.4. country_import_value

```
ANALISIS_02_CASTANEDA_EDSON.country_import_value = {}
```

Definición en la línea 193 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.5. country_import_value_pct

```
ANALISIS_02_CASTANEDA_EDSON.country_import_value_pct = {}
```

Definición en la línea 194 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.6. country_total_value

```
ANALISIS_02_CASTANEDA_EDSON.country_total_value = {k: country_export_value.get(k, 0) + country↩
_import_value.get(k, 0) for k in set(country_export_value) | set(country_import_value)}
```

Definición en la línea 215 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.7. country_total_value_pct

```
ANALISIS_02_CASTANEDA_EDSON.country_total_value_pct = {k: round((v/total_value)*100,2) for k,
v in country_total_value.items()}
```

Definición en la línea 217 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.8. data

```
dictionary ANALISIS_02_CASTANEDA_EDSON.data
```

Valor inicial:

```
00001 =  {'route': list(route_frecuency.keys()), 'frecuency':list(route_frecuency.values()),
00002                'frecuency_pct':list(route_frecuency_pct.values()),
       'total_value':list(route_value.values()),
00003                'total_value_pct':list(route_value_pct.values())}
```

Definición en la línea 95 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.9. destination

```
ANALISIS_02_CASTANEDA_EDSON.destination
```

Definición en la línea 205 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.10. DESTINATION_COUNTRIES

```
ANALISIS_02_CASTANEDA_EDSON.DESTINATION_COUNTRIES = service.get_unique_values("destination")
```

Definición en la línea 24 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.11. df

```
ANALISIS_02_CASTANEDA_EDSON.df = pd.DataFrame({ key:pd.Series(value) for key, value in import↩
_data.items()}).fillna(0)
```

Definición en la línea 242 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.12. DIRECCIONES

```
list ANALISIS_02_CASTANEDA_EDSON.DIRECCIONES = [None, "Imports", "Exports"]
```

Definición en la línea 20 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.13. direction

```
ANALISIS_02_CASTANEDA_EDSON.direction
```

Definición en la línea 61 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.14. direction_str

```
string ANALISIS_02_CASTANEDA_EDSON.direction_str = "All"
```

Definición en la línea 42 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.15. export_data

```
dictionary ANALISIS_02_CASTANEDA_EDSON.export_data
```

Valor inicial:

```
00001 =  {'country': list(country_export_value.keys()), 'total_value':list(country_export_value.values()),
00002                   'total_value_pct':list(country_export_value_pct.values())}
```

Definición en la línea 246 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.16. export_plot

```
ANALISIS_02_CASTANEDA_EDSON.export_plot = Summary_Chart(output_export_folder_year)
```

Definición en la línea 263 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.17. frecuency_pct_list

```
list ANALISIS_02_CASTANEDA_EDSON.frecuency_pct_list = []
```

Definición en la línea 39 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.18. i

```
int ANALISIS_02_CASTANEDA_EDSON.i = 1
```

Definición en la línea 79 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.19. import_data

```
dictionary ANALISIS_02_CASTANEDA_EDSON.import_data
```

Valor inicial:

```
00001 =  {'country': list(country_import_value.keys()), 'total_value':list(country_import_value.values()),
00002                   'total_value_pct':list(country_import_value_pct.values())}
```

Definición en la línea 239 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.20. import_plot

```
ANALISIS_02_CASTANEDA_EDSON.import_plot = Summary_Chart(output_import_folder_year)
```

Definición en la línea 260 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.21. index

```
ANALISIS_02_CASTANEDA_EDSON.index
```

Definición en la línea 98 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.22. origin

```
ANALISIS_02_CASTANEDA_EDSON.origin
```

Definición en la línea 211 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.23. ORIGIN_COUNTRIES

```
ANALISIS_02_CASTANEDA_EDSON.ORIGIN_COUNTRIES = service.get_unique_values("origin")
```

Definición en la línea 23 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.24. output_export_folder

```
string ANALISIS_02_CASTANEDA_EDSON.output_export_folder = f"{OUTPUT_INITIAL_PATH}/opcion_↩
3/Exports"
```

Definición en la línea 190 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.25. output_export_folder_year

```
string ANALISIS_02_CASTANEDA_EDSON.output_export_folder_year = f"{output_export_folder}/{year_str}"
```

Definición en la línea 233 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.26. output_folder

```
string ANALISIS_02_CASTANEDA_EDSON.output_folder = f"{OUTPUT_INITIAL_PATH}/opcion_1/{direction_str}"
```

Definición en la línea 46 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.27. output_folder_year

```
string ANALISIS_02_CASTANEDA_EDSON.output_folder_year = f"{output_folder}/{year_str}"
```

Definición en la línea 90 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.28. output_import_folder

```
string ANALISIS_02_CASTANEDA_EDSON.output_import_folder = f"{OUTPUT_INITIAL_PATH}/opcion_↩
3/Imports"
```

Definición en la línea 189 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.29. output_import_folder_year

```
string ANALISIS_02_CASTANEDA_EDSON.output_import_folder_year = f"{output_import_folder}/{year_str}"
```

Definición en la línea 232 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.30. OUTPUT_INITIAL_PATH

```
string ANALISIS_02_CASTANEDA_EDSON.OUTPUT_INITIAL_PATH = "exploration"
```

Definición en la línea 28 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.31. PERIODO_TIEMPO

```
list ANALISIS_02_CASTANEDA_EDSON.PERIODO_TIEMPO = [None, 2015, 2016, 2017, 2018, 2019, 2020]
```

Definición en la línea 21 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.32. plot

```
ANALISIS_02_CASTANEDA_EDSON.plot = Summary_Chart(output_folder_year)
```

Definición en la línea 106 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.33. route

```
ANALISIS_02_CASTANEDA_EDSON.route
```

Definición en la línea 61 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.34. route_frecuency

```
dictionary ANALISIS_02_CASTANEDA_EDSON.route_frecuency = {}
```

Definición en la línea 50 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.35. route_frecuency_pct

```
dictionary ANALISIS_02_CASTANEDA_EDSON.route_frecuency_pct = {}
```

Definición en la línea 52 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.36. route_value

```
dictionary ANALISIS_02_CASTANEDA_EDSON.route_value = {}
```

Definición en la línea 51 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.37. route_value_pct

```
dictionary ANALISIS_02_CASTANEDA_EDSON.route_value_pct = {}
```

Definición en la línea 53 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.38. ROUTES

```
ANALISIS_02_CASTANEDA_EDSON.ROUTES = service.get_routes_list()
```

Definición en la línea 25 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.39. service

```
ANALISIS_02_CASTANEDA_EDSON.service = Service()
```

Definición en la línea 17 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.40. top_ten_frecuency

```
ANALISIS_02_CASTANEDA_EDSON.top_ten_frecuency = service.get_top_ten(route_frecuency)
```

Definición en la línea 67 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.41. top_ten_frecuency_pct

```
ANALISIS_02_CASTANEDA_EDSON.top_ten_frecuency_pct = service.get_top_ten(route_frecuency_pct)
```

Definición en la línea 69 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.42. top_ten_value

```
ANALISIS_02_CASTANEDA_EDSON.top_ten_value = service.get_top_ten(route_value)
```

Definición en la línea 68 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.43. top_ten_value_pct

```
ANALISIS_02_CASTANEDA_EDSON.top_ten_value_pct = service.get_top_ten(route_value_pct)
```

Definición en la línea 70 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.44. total_cases

```
ANALISIS_02_CASTANEDA_EDSON.total_cases = service.get_total_elements(direction=direction,
year=year)
```

Definición en la línea 56 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.45. total_data

```
dictionary ANALISIS_02_CASTANEDA_EDSON.total_data
```

Valor inicial:

```
00001 =  {'country': list(country_total_value.keys()), 'total_value':list(country_total_value.values()),
00002                  'total_value_pct':list(country_total_value_pct.values())}
```

Definición en la línea 253 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.46. total_export

```
ANALISIS_02_CASTANEDA_EDSON.total_export = service.get_total_value(direction="Exports", year=year)
```

Definición en la línea 200 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.47. total_import

```
ANALISIS_02_CASTANEDA_EDSON.total_import = service.get_total_value(direction="Imports", year=year)
```

Definición en la línea 199 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.48. total_value

```
ANALISIS_02_CASTANEDA_EDSON.total_value = service.get_total_value(direction=direction, year=year)
```

Definición en la línea 57 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.49. transport

```
ANALISIS_02_CASTANEDA_EDSON.transport
```

Definición en la línea 149 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.50. transport_frecuency

```
dictionary ANALISIS_02_CASTANEDA_EDSON.transport_frecuency = {}
```

Definición en la línea 138 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.51. transport_frecuency_pct

```
dictionary ANALISIS_02_CASTANEDA_EDSON.transport_frecuency_pct = {}
```

Definición en la línea 140 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.52. TRANSPORT_MODES

```
ANALISIS_02_CASTANEDA_EDSON.TRANSPORT_MODES = service.get_unique_values("transport_mode")
```

Definición en la línea 22 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.53. transport_value

```
dictionary ANALISIS_02_CASTANEDA_EDSON.transport_value = {}
```

Definición en la línea 139 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.54. transport_value_pct

```
dictionary ANALISIS_02_CASTANEDA_EDSON.transport_value_pct = {}
```

Definición en la línea 141 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.55. value_pct_list

```
list ANALISIS_02_CASTANEDA_EDSON.value_pct_list = []
```

Definición en la línea 38 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.56. year

```
ANALISIS_02_CASTANEDA_EDSON.year
```

Definición en la línea 61 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.57. year_list

```
list ANALISIS_02_CASTANEDA_EDSON.year_list = []
```

Definición en la línea 37 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.1.2.58. year_str

```
string ANALISIS_02_CASTANEDA_EDSON.year_str = "All"
```

Definición en la línea 73 del archivo ANALISIS_02_CASTANEDA_EDSON.py.

5.2. Referencia del Namespace graph_utils

Clases

- class Chart
- class Summary_Chart

5.3. Referencia del Namespace sl_filters

Clases

- class SynergyLogisticsFilters

Variables

- string DATA_FILE_PATH = "data\synergy_logistics_database.csv"

5.3.1. Documentación de las variables

5.3.1.1. DATA_FILE_PATH

```
string sl_filters.DATA_FILE_PATH = "data\synergy_logistics_database.csv"
```

Definición en la línea 7 del archivo sl_filters.py.

5.4. Referencia del Namespace synergy_services

Clases

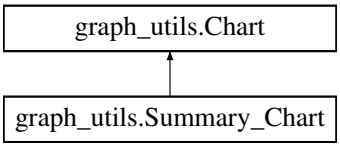
- class Service

Capítulo 6

Documentación de las clases

6.1. Referencia de la Clase graph_utils.Chart

Diagrama de herencias de graph_utils.Chart



Métodos públicos

- def save_as_image (self, FigureWidget fig, str file_name)
- def save_plot (self, FigureWidget fig, str file_name)

6.1.1. Descripción detallada

```
Funciones generales para la generación de gráficas.
```

Definición en la línea 9 del archivo graph_utils.py.

6.1.2. Documentación de las funciones miembro

6.1.2.1. save_as_image()

```
def graph_utils.Chart.save_as_image (
              self,
             FigureWidget fig,
             str file_name )

 Esta función guarda una gráfica como imagen.
Args:
    fig (plotly.graph_objects.Figure): Objeto de la gráfica creada.
    file_name (str): Ruta donde almacenar el archivo.
```

Definición en la línea 13 del archivo graph_utils.py.

6.1.2.2. save_plot()

```
def graph_utils.Chart.save_plot (
              self,
             FigureWidget fig,
             str file_name )
```

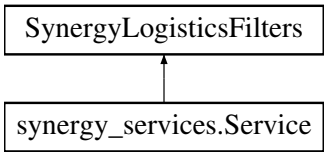
Definición en la línea 21 del archivo graph_utils.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- graph_utils.py

6.2. Referencia de la Clase synergy_services.Service

Diagrama de herencias de synergy_services.Service



Métodos públicos

- List get_routes_list (self, str or None direction=None)
- int get_total_elements (self, str or None direction=None, int or None year=None, str or None transport_↩ mode=None)
- int get_route_frecuency (self, str route, str or None direction=None, int or None year=None)
- int get_total_value (self, str or None direction=None, int or None year=None, str or None transport_↩ mode=None)
- int get_route_value (self, str route, str or None direction=None, int or None year=None)
- dict get_top_ten (self, dict all_cases)
- int get_transport_frecuency (self, str transport, str or None direction=None, int or None year=None)
- int get_transport_value (self, str transport, str or None direction=None, int or None year=None)
- int get_country_frecuency (self, str or None origin=None, str or None destination=None, str or None direction=None, int or None year=None)
- int get_country_value (self, str or None origin=None, str or None destination=None, str or None direction=None, int or None year=None)
- dict reorder_dict_max (self, dict data_dict)

6.2.1. Descripción detallada

```
Clase que contine servicios para el analisis de la tabla de Synergy Logistics.
```

Definición en la línea 5 del archivo synergy_services.py.

6.2.2. Documentación de las funciones miembro

6.2.2.1. get_country_frecuency()

```
int synergy_services.Service.get_country_frecuency (
              self,
             str or None  origin = None,
             str or None  destination = None,
             str or None  direction = None,
             int or None  year = None )

Cuenta las veces que un pais aparece en una tabla filtrada.
Se pueden filtrar resultados por dirección, año y/o medio de transporte.

Args:
    origin (str or None, optional): Pais de origen. Defaults to None.
    destination (str or None, optional): Pais de destino. Defaults to None.
    direction (str or None, optional): Dirección de transacción. Defaults to None.
    year (int or None, optional): Año de transacciones. Defaults to None.

Returns:
    int: Numero de apariciones de transporte en la tabla filtrada.
```

Definición en la línea 165 del archivo synergy_services.py.

6.2.2.2. get_country_value()

```
int synergy_services.Service.get_country_value (
              self,
             str or None  origin = None,
             str or None  destination = None,
             str or None  direction = None,
             int or None  year = None )

Suma el valor total para un pais especifico dentro de una tabla filtrada.
Se pueden filtrar resultados por dirección, año y/o medio de transporte.

Args:
    origin (str or None, optional): Pais de origen. Defaults to None.
    destination (str or None, optional): Pais de destino. Defaults to None.
    direction (str or None, optional): Dirección de transacción. Defaults to None.
    year (int or None, optional): Año de transacciones. Defaults to None.

Returns:
    int: suma de valor de elementos en tabla filtrada.
```

Definición en la línea 187 del archivo synergy_services.py.

6.2.2.3. get_route_frecuency()

```
int synergy_services.Service.get_route_frecuency (
              self,
             str route,
             str or None  direction = None,
             int or None  year = None )

Cuenta las veces que una ruta aparece en una tabla filtrada.
Se pueden filtrar resultados por dirección, año y/o medio de transporte.

Args:
    route (str): Rutas con formato origen-destino.
    direction (str or None, optional): Dirección de transacción. Defaults to None.
    year (int or None, optional): Año de transacciones. Defaults to None.

Returns:
    int: Numero de apariciones de ruta en la tabla filtrada.
```

Definición en la línea 49 del archivo synergy_services.py.

6.2.2.4. get_route_value()

```
int synergy_services.Service.get_route_value (
              self,
             str route,
             str or None  direction = None,
             int or None  year = None )

Suma el valor total para una ruta especifica dentro de una tabla filtrada.
Se pueden filtrar resultados por dirección, año y/o medio de transporte.

Args:
    route (str): Rutas con formato origen-destino.
    direction (str or None, optional): Dirección de transacción. Defaults to None.
    year (int or None, optional): Año de transacciones. Defaults to None.
    transport_mode (str or None, optional): Tipo de medio de transporte. Defaults to None.

Returns:
    int: suma de valor de elementos en tabla filtrada.
```

Definición en la línea 89 del archivo synergy_services.py.

6.2.2.5. get_routes_list()

```
List synergy_services.Service.get_routes_list (
              self,
             str or None  direction = None )

Genera una lista con todas las rutas diferentes de la tabla.

Args:
    direction (str or None, optional): Dirección de transacción. Defaults to None.

Returns:
    List: Lista con rutas con formato origen-destino.
```

Definición en la línea 9 del archivo synergy_services.py.

6.2.2.6. get_top_ten()

```
dict synergy_services.Service.get_top_ten (
              self,
             dict all_cases )

De un diccionario de elementos se obtienen los 10 casos con mejores resultados.

Args:
    all_cases (dict): Diccionario con todos los casos

Returns:
    List: Lista con los 10 casos con mejores resultados.
```

Definición en la línea 109 del archivo synergy_services.py.

6.2.2.7. get_total_elements()

```
int synergy_services.Service.get_total_elements (
              self,
             str or None  direction = None,
             int or None  year = None,
             str or None  transport_mode = None )

Cuenta el número de transacciones en una tabla filtrada.
Se pueden filtrar resultados por dirección, año y/o medio de transporte.

Args:
    direction (str or None, optional): Dirección de transacción. Defaults to None.
    year (int or None, optional): Año de transacciones. Defaults to None.
    transport_mode (str or None, optional): Tipo de medio de transporte. Defaults to None.

Returns:
    int: Total de casos en tabla filtrada.
```

Definición en la línea 29 del archivo synergy_services.py.

6.2.2.8. get_total_value()

```
int synergy_services.Service.get_total_value (
              self,
             str or None  direction = None,
             int or None  year = None,
             str or None  transport_mode = None )

Suma el valor total dentro de una tabla filtrada.
Se pueden filtrar resultados por dirección, año y/o medio de transporte.

Args:
    route (str): Rutas con formato origen-destino.
    direction (str or None, optional): Dirección de transacción. Defaults to None.
    year (int or None, optional): Año de transacciones. Defaults to None.
    transport_mode (str or None, optional): Tipo de medio de transporte. Defaults to None.

Returns:
    int: suma de valor de elementos en tabla filtrada.
```

Definición en la línea 71 del archivo synergy_services.py.

6.2.2.9. get_transport_frecuency()

```
int synergy_services.Service.get_transport_frecuency (
              self,
             str transport,
             str or None  direction = None,
             int or None  year = None )

Cuenta las veces que un transporte aparece en una tabla filtrada.
Se pueden filtrar resultados por dirección, año y/o medio de transporte.

Args:
    transport (str): Tipo de medio de transporte.
    direction (str or None, optional): Dirección de transacción. Defaults to None.
    year (int or None, optional): Año de transacciones. Defaults to None.

Returns:
    int: Numero de apariciones de transporte en la tabla filtrada.
```

Definición en la línea 125 del archivo synergy_services.py.

6.2.2.10. get_transport_value()

```
int synergy_services.Service.get_transport_value (
              self,
             str transport,
             str or None  direction = None,
             int or None  year = None )

Suma el valor total para un transporte especifico dentro de una tabla filtrada.
Se pueden filtrar resultados por dirección, año y/o medio de transporte.

Args:
    transport (str): Tipo de medio de transporte.
    direction (str or None, optional): Dirección de transacción. Defaults to None.
    year (int or None, optional): Año de transacciones. Defaults to None.

Returns:
    int: suma de valor de elementos en tabla filtrada.
```

Definición en la línea 146 del archivo synergy_services.py.

6.2.2.11. reorder_dict_max()

```
dict synergy_services.Service.reorder_dict_max (
              self,
             dict data_dict )

Ordena diccionario a partir de valores de mayor a menor.
Elimita los elementos del diccionario que tengan un valor de 0.

Args:
    data_dict (dict): Diccionario de datos desordenados.

Returns:
    dict: Diccionario de datos filtrados.
```

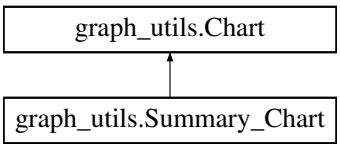
Definición en la línea 206 del archivo synergy_services.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- synergy_services.py

6.3. Referencia de la Clase graph_utils.Summary_Chart

Diagrama de herencias de graph_utils.Summary_Chart



Métodos públicos

- None __init__ (self, str file_path)
- None bar_summary (self, dict data, str plot_title, str x_axis_name, str y_axis_name, str file_name, str color="blue")
- None h_bar_summary (self, dict data, str plot_title, str x_axis_name, str y_axis_name, str file_name, str color="blue")
- def pie_summary (self, dict data, str plot_title, str file_name)

Atributos públicos

- file_path
- layout

6.3.1. Descripción detallada

```
Funciones para gráfica resultados de las consultas realizadas de las tablas de Lifestore.
```

Definición en la línea 32 del archivo graph_utils.py.

6.3.2. Documentación del constructor y destructor

6.3.2.1. __init__()

```
None graph_utils.Summary_Chart.__init__ (
              self,
             str file_path )
```

```
Establece parametros default para objetos de la clase.

Args:
    file_path (str): Ruta donde guardar gráficas.
```

Definición en la línea 36 del archivo graph_utils.py.

6.3.3. Documentación de las funciones miembro

6.3.3.1. bar_summary()

```
None graph_utils.Summary_Chart.bar_summary (
              self,
             dict data,
             str plot_title,
             str x_axis_name,
             str y_axis_name,
             str file_name,
             str  color = "blue" )
```

```
Gráfica de barras a partir de un diccionario.

Args:
    data (dict): Datos a graficar.
    plot_title (str): Titulo de la gráfica.
    x_axis_name (str): Nombre de eje x.
    y_axis_name (str): Nombre de eje y.
    file_name (str): Nombre del archivo.
    color (str, optional): Color del gráfico. Defaults to "blue".
```

Definición en la línea 59 del archivo graph_utils.py.

6.3.3.2. h_bar_summary()

```
None graph_utils.Summary_Chart.h_bar_summary (
              self,
             dict data,
             str plot_title,
             str x_axis_name,
             str y_axis_name,
             str file_name,
             str  color = "blue" )
```

```
Gráfica de barras a partir de un diccionario.

Args:
    data (dict): Datos a graficar.
    plot_title (str): Titulo de la gráfica.
    x_axis_name (str): Nombre de eje x.
    y_axis_name (str): Nombre de eje y.
    file_name (str): Nombre del archivo.
    color (str, optional): Color del gráfico. Defaults to "blue".
```

Definición en la línea 99 del archivo graph_utils.py.

6.3.3.3. pie_summary()

```
def graph_utils.Summary_Chart.pie_summary (
              self,
             dict data,
             str plot_title,
             str file_name )
```

Definición en la línea 142 del archivo graph_utils.py.

6.3.4. Documentación de los datos miembro

6.3.4.1. file_path

```
graph_utils.Summary_Chart.file_path
```

Definición en la línea 44 del archivo graph_utils.py.

6.3.4.2. layout

```
graph_utils.Summary_Chart.layout
```

Definición en la línea 46 del archivo graph_utils.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- graph_utils.py

6.4. Referencia de la Clase sl_filters.SynergyLogisticsFilters

Métodos públicos

- None __init__ (self)
- DataFrame filter_routes_df (self, str or None direction=None, str or None origin=None, str or None destination=None, int or None start_year=None, int or None end_year=None, str or None start_date=None, str or None end_date=None, str or None product=None, str or None transport_mode=None, str or None company↩_name=None, int or None min_value=None, int or None max_value=None)
- List get_unique_values (self, str category)

Atributos públicos

- SYNERGY_DB

6.4.1. Descripción detallada

Definición en la línea 9 del archivo sl_filters.py.

6.4.2. Documentación del constructor y destructor

6.4.2.1. __init__()

```
None sl_filters.SynergyLogisticsFilters.__init__ (
              self )
```

```
Lectura de la BD de Synergy Logistics.
```

Definición en la línea 10 del archivo sl_filters.py.

6.4.3. Documentación de las funciones miembro

6.4.3.1. filter_routes_df()

```
DataFrame sl_filters.SynergyLogisticsFilters.filter_routes_df (
              self,
             str or None  direction = None,
             str or None  origin = None,
             str or None  destination = None,
             int or None  start_year = None,
             int or None  end_year = None,
             str or None  start_date = None,
             str or None  end_date = None,
             str or None  product = None,
             str or None  transport_mode = None,
             str or None  company_name = None,
             int or None  min_value = None,
             int or None  max_value = None )
```

```
Filtra el dataframe de Synergy Logistics de acuerdo a valores en las columnas que tiene
la tabla generada. Si no hay filtro, se regresa un dataframe completo.

Args:
    direction (str or None, optional): Tipo de dirección (Import o Export). Defaults to None.
    origin (str or None, optional): Pais de origen. Defaults to None.
    destination (str or None, optional): Pais de destino. Defaults to None.
    start_year (int or None, optional): Año inicial de periodo. Defaults to None.
    end_year (int or None, optional): Año final de periodo. Defaults to None.
    start_date (str or None, optional): Fecha inicial de periodo. Defaults to None.
    end_date (str or None, optional): Fecha final de periodo. Defaults to None.
    product (str or None, optional): Tipo de producto. Defaults to None.
    transport_mode (str or None, optional): Tipo de medio de transporte. Defaults to None.
    company_name (str or None, optional): Nombre de compañia. Defaults to None.
    min_value (int or None, optional): Valor minimo. Defaults to None.
    max_value (int or None, optional): Valor maximo. Defaults to None.

Returns:
    DataFrame:  Dataframe con columnas de la tabla que cumplen con los filtros indicados.
```

Definición en la línea 17 del archivo sl_filters.py.

6.4.3.2. get_unique_values()

```
List sl_filters.SynergyLogisticsFilters.get_unique_values (
              self,
             str category )
```

```
Genera lista con valores unicos de columna de la base de datos.

Args:
    category (str): Nombre de la columna.

Returns:
    List: Valores unicos en columna de la tabla.
```

Definición en la línea 125 del archivo sl_filters.py.

6.4.4. Documentación de los datos miembro

6.4.4.1. SYNERGY_DB

```
sl_filters.SynergyLogisticsFilters.SYNERGY_DB
```

Definición en la línea 13 del archivo sl_filters.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- sl_filters.py

Capítulo 7

Documentación de archivos

7.1. Referencia del Archivo ANALISIS_02_CASTANEDA_EDSON.py

Namespaces

- namespace ANALISIS_02_CASTANEDA_EDSON

Variables

- ANALISIS_02_CASTANEDA_EDSON.service = Service()
- list ANALISIS_02_CASTANEDA_EDSON.DIRECCIONES = [None, "Imports", "Exports"]
- list ANALISIS_02_CASTANEDA_EDSON.PERIODO_TIEMPO = [None, 2015, 2016, 2017, 2018, 2019, 2020]
- ANALISIS_02_CASTANEDA_EDSON.TRANSPORT_MODES = service.get_unique_values("transport_↩ mode")
- ANALISIS_02_CASTANEDA_EDSON.ORIGIN_COUNTRIES = service.get_unique_values("origin")
- ANALISIS_02_CASTANEDA_EDSON.DESTINATION_COUNTRIES = service.get_unique_values("destination")
- ANALISIS_02_CASTANEDA_EDSON.ROUTES = service.get_routes_list()
- string ANALISIS_02_CASTANEDA_EDSON.OUTPUT_INITIAL_PATH = "exploration"
- list ANALISIS_02_CASTANEDA_EDSON.year_list = [ ]
- list ANALISIS_02_CASTANEDA_EDSON.value_pct_list = [ ]
- list ANALISIS_02_CASTANEDA_EDSON.frecuency_pct_list = [ ]
- string ANALISIS_02_CASTANEDA_EDSON.direction_str = "All"
- string ANALISIS_02_CASTANEDA_EDSON.output_folder = f"{OUTPUT_INITIAL_PATH}/opcion_↩ 1/{direction_str}"
- dictionary ANALISIS_02_CASTANEDA_EDSON.route_frecuency = {}
- dictionary ANALISIS_02_CASTANEDA_EDSON.route_value = {}
- dictionary ANALISIS_02_CASTANEDA_EDSON.route_frecuency_pct = {}
- dictionary ANALISIS_02_CASTANEDA_EDSON.route_value_pct = {}
- ANALISIS_02_CASTANEDA_EDSON.total_cases = service.get_total_elements(direction=direction, year=year)
- ANALISIS_02_CASTANEDA_EDSON.total_value = service.get_total_value(direction=direction, year=year)
- ANALISIS_02_CASTANEDA_EDSON.route
- ANALISIS_02_CASTANEDA_EDSON.direction
- ANALISIS_02_CASTANEDA_EDSON.year
- ANALISIS_02_CASTANEDA_EDSON.top_ten_frecuency = service.get_top_ten(route_frecuency)
- ANALISIS_02_CASTANEDA_EDSON.top_ten_value = service.get_top_ten(route_value)
- ANALISIS_02_CASTANEDA_EDSON.top_ten_frecuency_pct = service.get_top_ten(route_frecuency_pct)

- ANALISIS_02_CASTANEDA_EDSON.top_ten_value_pct = service.get_top_ten(route_value_pct)
- string ANALISIS_02_CASTANEDA_EDSON.year_str = "All"
- int ANALISIS_02_CASTANEDA_EDSON.i = 1
- string ANALISIS_02_CASTANEDA_EDSON.output_folder_year = f"{output_folder}/{year_str}"
- dictionary ANALISIS_02_CASTANEDA_EDSON.data
- ANALISIS_02_CASTANEDA_EDSON.index
- ANALISIS_02_CASTANEDA_EDSON.plot = Summary_Chart(output_folder_year)
- dictionary ANALISIS_02_CASTANEDA_EDSON.transport_frecuency = {}
- dictionary ANALISIS_02_CASTANEDA_EDSON.transport_value = {}
- dictionary ANALISIS_02_CASTANEDA_EDSON.transport_frecuency_pct = {}
- dictionary ANALISIS_02_CASTANEDA_EDSON.transport_value_pct = {}
- ANALISIS_02_CASTANEDA_EDSON.transport
- string ANALISIS_02_CASTANEDA_EDSON.output_import_folder = f"{OUTPUT_INITIAL_PATH}/opcion_↩ 3/Imports"
- string ANALISIS_02_CASTANEDA_EDSON.output_export_folder = f"{OUTPUT_INITIAL_PATH}/opcion_↩ 3/Exports"
- dictionary ANALISIS_02_CASTANEDA_EDSON.country_import_value = {}
- dictionary ANALISIS_02_CASTANEDA_EDSON.country_import_value_pct = {}
- dictionary ANALISIS_02_CASTANEDA_EDSON.country_export_value = {}
- dictionary ANALISIS_02_CASTANEDA_EDSON.country_export_value_pct = {}
- ANALISIS_02_CASTANEDA_EDSON.total_import = service.get_total_value(direction="Imports", year=year)
- ANALISIS_02_CASTANEDA_EDSON.total_export = service.get_total_value(direction="Exports", year=year)
- ANALISIS_02_CASTANEDA_EDSON.destination
- ANALISIS_02_CASTANEDA_EDSON.country
- ANALISIS_02_CASTANEDA_EDSON.origin
- dictionary ANALISIS_02_CASTANEDA_EDSON.country_total_value = {k: country_export_value.get(k, 0) + country_import_value.get(k, 0) for k in set(country_export_value) | set(country_import_value)}
- dictionary ANALISIS_02_CASTANEDA_EDSON.country_total_value_pct = {k: round((v/total_value)∗100,2) for k, v in country_total_value.items()}
- string ANALISIS_02_CASTANEDA_EDSON.output_import_folder_year = f"{output_import_folder}/{year_str}"
- string ANALISIS_02_CASTANEDA_EDSON.output_export_folder_year = f"{output_export_folder}/{year_str}"
- dictionary ANALISIS_02_CASTANEDA_EDSON.import_data
- ANALISIS_02_CASTANEDA_EDSON.df = pd.DataFrame({ key:pd.Series(value) for key, value in import_↩ data.items()}).fillna(0)
- dictionary ANALISIS_02_CASTANEDA_EDSON.export_data
- dictionary ANALISIS_02_CASTANEDA_EDSON.total_data
- ANALISIS_02_CASTANEDA_EDSON.import_plot = Summary_Chart(output_import_folder_year)
- ANALISIS_02_CASTANEDA_EDSON.export_plot = Summary_Chart(output_export_folder_year)

7.2. ANALISIS_02_CASTANEDA_EDSON.py

Ir a la documentación de este archivo.

```
00001 """
00002 ANALISIS SYNERGY LOGISTICS
00003
00004 En este script se exploran las 3 propuestas de enfoque de la empresa Synergy Logistics.
00005 A partir de una revision total y por año se pretende determinar la mejor opción
00006 que se puede considerar para los proximos años.
00007
00008 """
00009 import pandas as pd
00010 import os
00011
00012 from utils.graph_utils import Summary_Chart
00013 from services.synergy_services import Service
00014
00015
00016 # Crear objeto de la clase Services para las consultas
00017 service = Service()
```

```
00018
00019 # Definir constantes relacionadas a la DB y los analisis a realizar
00020 DIRECCIONES = [None, "Imports", "Exports"] # None se refiere a ambas dirrecciones juntas
00021 PERIODO_TIEMPO = [None, 2015, 2016, 2017, 2018, 2019, 2020] # Con None se analizan todos los años
00022 TRANSPORT_MODES = service.get_unique_values("transport_mode")  # Lista de medios de transporte
00023 ORIGIN_COUNTRIES = service.get_unique_values("origin")  # Lista de paises de origen
00024 DESTINATION_COUNTRIES = service.get_unique_values("destination")  # Lista de paises de destino
00025 ROUTES = service.get_routes_list()  # Lista de rutas
00026
00027 # Definir constantes relacionadas a la administración de archivos
00028 OUTPUT_INITIAL_PATH = "exploration"
00029
00030 # Opcion 1: 10 rutas más demandadas
00031 print(f"Hay {len(ROUTES)} rutas diferentes.")
00032 # Analizar demanda general y para direcciones especificas (import/export)
00033 # Se analizaran el no. de apiriciones de la ruta en la tabla y el valor de esas apariciones, los
00034 # factores se expresaran como no. entero y como porcentaje del total para el periodo y dirección
      analizados.
00035 for direction in DIRECCIONES:
00036     # Inicializar variables prinicipales de resumen
00037     year_list = []
00038     value_pct_list = []
00039     frecuency_pct_list = []
00040     # Manejo de casos none (crear str)
00041     if direction is None:
00042         direction_str = "All"
00043     else:
00044         direction_str = direction
00045     # Ruta para almacenar resultados
00046     output_folder = f"{OUTPUT_INITIAL_PATH}/opcion_1/{direction_str}"
00047     # Para cada caso, analizar el periodo completo (2015-2020) y por año
00048     for year in PERIODO_TIEMPO:
00049         # Inicializar variable del resumen de cada periodo
00050         route_frecuency = {}
00051         route_value = {}
00052         route_frecuency_pct = {}
00053         route_value_pct =  {}
00054         # Calcular totales para no. de apariciones y valor en direccion y año considerados
00055         # Este valor se utilizara para expresar frecuencia y valor tambien como porcentajes
00056         total_cases = service.get_total_elements(direction=direction, year=year)
00057         total_value = service.get_total_value(direction=direction, year=year)
00058         # Dentro del periodo indicado se analiza frecuencia y valor ruta por ruta
00059         for route in ROUTES:
00060             # Para cada ruta consultar frencuencia y valor total
00061             route_frecuency[route] = service.get_route_frecuency(route, direction=direction,
      year=year)
00062             route_value[route] = service.get_route_value(route, direction=direction, year=year)
00063             # Calcular porcentaje de resultados de ruta respecto a totales para el caso
00064             route_frecuency_pct[route] = round((route_frecuency[route]/total_cases)*100, 2)
00065             route_value_pct[route] = round((route_value[route]/total_value)*100, 2)
00066         # Obtener top ten en valor y frecuencia
00067         top_ten_frecuency = service.get_top_ten(route_frecuency)
00068         top_ten_value = service.get_top_ten(route_value)
00069         top_ten_frecuency_pct = service.get_top_ten(route_frecuency_pct)
00070         top_ten_value_pct = service.get_top_ten(route_value_pct)
00071         # Manejo de casos none (volver str)
00072         if year is None:
00073             year_str = "All"
00074         else:
00075             year_str = str(year)
00076         # Imprimir rutas con mejor valor y más uso
00077         print(f"Opción 1 - Direccion: {direction_str}, Año: {year_str} ")
00078         print("Rutas más utilizadas:")
00079         i=1
00080         for route in top_ten_frecuency.keys():
00081             print(f"{i}.- {route}: {top_ten_frecuency[route]}")
00082             i+=1
00083         print("Rutas mejor valoradas:")
00084         i=1
00085         for route in top_ten_value.keys():
00086             print(f"{i}.- {route}: {top_ten_value[route]}")
00087             i+=1
00088         print("")
00089          # Definir folder para almacenar resultados y crear si no existe
00090         output_folder_year = f"{output_folder}/{year_str}"
00091         if not os.path.exists(f"{output_folder_year}"):
00092             os.makedirs(f"{output_folder_year}")
00093         # Almacenar resultados como CSV
00094         # Todas las rutas
00095         data = {'route': list(route_frecuency.keys()), 'frecuency':list(route_frecuency.values()),
00096                 'frecuency_pct':list(route_frecuency_pct.values()),
      'total_value':list(route_value.values()),
00097                 'total_value_pct':list(route_value_pct.values())}
00098         pd.DataFrame(data).to_csv(output_folder_year+"/results.csv", index=False)
00099         # Top 10 en frecuency
00100         data = {'route': list(top_ten_frecuency.keys()), 'frecuency':list(top_ten_frecuency.values()),
      'frecuency_pct':list(top_ten_frecuency_pct.values())}
```

```
00101         pd.DataFrame(data).to_csv(output_folder_year+"/top10_frec.csv", index=False)
00102         # Top 10 en valor
00103         data = {'route': list(top_ten_value.keys()), 'total_value':list(top_ten_value.values()),
      'total_value_pct':list(top_ten_value_pct.values())}
00104         pd.DataFrame(data).to_csv(output_folder_year+"/top10_value.csv", index=False)
00105         # Graficar resultados
00106         plot = Summary_Chart(output_folder_year)
00107         plot.h_bar_summary(top_ten_frecuency, "Rutas con mayor demanda", "No. de Apariciones",
00108                            "Rutas", "ruta_frec")
00109         plot.h_bar_summary(top_ten_value, "Rutas con mayor valor", "Valor total", "Rutas",
00110                            "ruta_valor", "green")
00111         plot.h_bar_summary(top_ten_frecuency_pct, "Rutas con mayor demanda", "Apariciones (%)",
00112                            "Rutas", "ruta_frec_pct", "purple")
00113         plot.h_bar_summary(top_ten_value_pct, "Rutas con mayor valor", "Valor total (%)", "Rutas",
00114                            "ruta_valor_pct", "purple")
00115         # Resumen del periodo
00116         year_list.append(year_str)
00117         value_pct_list.append(sum(top_ten_value_pct.values()))
00118         frecuency_pct_list.append(sum(top_ten_frecuency_pct.values()))
00119     # Resumen multianual
00120     data = {'year': year_list, 'frecunecy_pct':frecuency_pct_list, 'total_value_pct':value_pct_list}
00121     pd.DataFrame(data).to_csv(output_folder+"/summary.csv", index=False)
00122
00123 # Opcion 2: Medios de transporte mas importantes
00124 # Se analizaran el no. de apiriciones del transporte en la tabla y el valor de esas apariciones, los
00125 # factores se expresaran como no. entero y como porcentaje del total para el periodo y dirección
      analizados.
00126 print("OPCION 2:\n")
00127 for direction in DIRECCIONES:
00128     # Manejo de casos none (crear str)
00129     if direction is None:
00130         direction_str = "All"
00131     else:
00132         direction_str = direction
00133     # Ruta para almacenar resultados
00134     output_folder = f"{OUTPUT_INITIAL_PATH}/opcion_2/{direction_str}"
00135     # Para cada caso, analizar el periodo completo (2015-2020) y por año
00136     for year in PERIODO_TIEMPO:
00137         # Inicializar variable del resumen de cada periodo
00138         transport_frecuency = {}
00139         transport_value = {}
00140         transport_frecuency_pct = {}
00141         transport_value_pct =  {}
00142         # Calcular totales para no. de apariciones y valor en direccion y año considerados
00143         # Este valor se utilizara para expresar frecuencia y valor tambien como porcentajes
00144         total_cases = service.get_total_elements(direction=direction, year=year)
00145         total_value = service.get_total_value(direction=direction, year=year)
00146         # Dentro del periodo indicado se analiza frecuencia y valor por medio de transporte
00147         for transport in TRANSPORT_MODES:
00148             # Para cada ruta consultar frencuencia y valor total
00149             transport_frecuency[transport] = service.get_transport_frecuency(transport,
      direction=direction, year=year)
00150             transport_value[transport] = service.get_transport_value(transport, direction=direction,
      year=year)
00151             # Calcular porcentaje de resultados de ruta respecto a totales para el caso
00152             transport_frecuency_pct[transport] =
      round((transport_frecuency[transport]/total_cases)*100, 2)
00153             transport_value_pct[transport] = round((transport_value[transport]/total_value)*100, 2)
00154         # Manejo de casos none (volver str)
00155         if year is None:
00156             year_str = "All"
00157         else:
00158             year_str = str(year)
00159         # Imprimir rutas con mejor valor y más uso
00160         print(f"\nOpción 2 - Direccion: {direction_str}, Año: {year_str} ")
00161         print("Transportes más utilizadas:")
00162         i=1
00163         for transport in transport_frecuency.keys():
00164             print(f"{i}.- {transport}. {transport_frecuency[transport]}. Valor:
      {transport_value[transport]}")
00165             i+=1
00166         print("")
00167          # Definir folder para almacenar resultados y crear si no existe
00168         output_folder_year = f"{output_folder}/{year_str}"
00169         if not os.path.exists(f"{output_folder_year}"):
00170             os.makedirs(f"{output_folder_year}")
00171         # Almacenar resultados como CSV
00172         # Todos los medios de transporte
00173         data = {'transport': list(transport_frecuency.keys()),
      'frecuency':list(transport_frecuency.values()),
00174                 'frecuency_pct':list(transport_frecuency_pct.values()),
      'total_value':list(transport_value.values()),
00175                 'total_value_pct':list(transport_value_pct.values())}
00176         pd.DataFrame(data).to_csv(output_folder_year+"/results.csv", index=False)
00177         # Graficar resultados
00178         plot = Summary_Chart(output_folder_year)
00179         plot.pie_summary(transport_frecuency, "Transporte con mayor demanda", "transporte_frec")
```

```
00180         plot.pie_summary(transport_value, "Transporte con mayor valor", "transporte_valor")
00181
00182 # Opcion 3: Paises que generen mayor valor
00183 # Paises que generan valor para importaciones: destino
00184 # Paises que generan valor para exportaciones: origen
00185 # Analisis para importaciones y exportaciones}
00186 print("\nOPCIÓN 3:")
00187 # Definir carpetas para guardar archivos
00188 output_folder = f"{OUTPUT_INITIAL_PATH}/opcion_3/All"
00189 output_import_folder = f"{OUTPUT_INITIAL_PATH}/opcion_3/Imports"
00190 output_export_folder = f"{OUTPUT_INITIAL_PATH}/opcion_3/Exports"
00191 for year in PERIODO_TIEMPO:
00192     # Inicializar variable del resumen para exportaciones e importaciones
00193     country_import_value = {}
00194     country_import_value_pct = {}
00195     country_export_value = {}
00196     country_export_value_pct = {}
00197     # Calcular valor en direccion y año considerados
00198     # Este valor se utilizara para expresar valor tambien como porcentajes
00199     total_import = service.get_total_value(direction="Imports", year=year)
00200     total_export = service.get_total_value(direction="Exports", year=year)
00201     total_value = total_import + total_export
00202     # Dentro del periodo indicado se analiza valor de cada pais de destino (Importaciones)
00203     for country in DESTINATION_COUNTRIES:
00204         # Para cada ruta consultar valor total
00205         country_import_value[country] = service.get_country_value(destination=country,
    direction="Imports", year=year)
00206         # Calcular porcentaje de resultados de ruta respecto a totales para el caso
00207         country_import_value_pct[country] = round((country_import_value[country]/total_import)*100, 2)
00208     # Dentro del periodo indicado se analiza valor de cada pais de origen (Exportaciones)
00209     for country in ORIGIN_COUNTRIES:
00210         # Para cada ruta consultar valor total
00211         country_export_value[country] = service.get_country_value(origin=country, direction="Exports",
    year=year)
00212         # Calcular porcentaje de resultados de ruta respecto a totales para el caso
00213         country_export_value_pct[country] = round((country_export_value[country]/total_export)*100, 2)
00214     # Sumar diccionarios de importaciones y exportaciones para tener total
00215     country_total_value = {k: country_export_value.get(k, 0) + country_import_value.get(k, 0) for k in
    set(country_export_value) | set(country_import_value)}
00216     # Obtener porcentaje
00217     country_total_value_pct = {k: round((v/total_value)*100,2) for k, v in
    country_total_value.items()}
00218     # Ordenar datos en diccionario de mayor a menor
00219     country_import_value = service.reorder_dict_max(country_import_value)
00220     country_import_value_pct = service.reorder_dict_max(country_import_value_pct)
00221     country_export_value = service.reorder_dict_max(country_export_value)
00222     country_export_value_pct = service.reorder_dict_max(country_export_value_pct)
00223     country_total_value = service.reorder_dict_max(country_total_value)
00224     country_total_value_pct = service.reorder_dict_max(country_total_value_pct)
00225     # Manejo de casos none (volver str)
00226     if year is None:
00227         year_str = "All"
00228     else:
00229         year_str = str(year)
00230     # Definir folder para almacenar resultados y crear si no existe
00231     output_folder_year = f"{output_folder}/{year_str}"
00232     output_import_folder_year = f"{output_import_folder}/{year_str}"
00233     output_export_folder_year = f"{output_export_folder}/{year_str}"
00234     for folder in [output_folder_year, output_import_folder_year, output_export_folder_year]:
00235         if not os.path.exists(folder):
00236             os.makedirs(folder)
00237     # Almacenar resultados como CSV
00238     # Importaciones
00239     import_data = {'country': list(country_import_value.keys()),
    'total_value':list(country_import_value.values()),
00240                    'total_value_pct':list(country_import_value_pct.values())}
00241     # Crear tabla. Si existen NaN, remplazarlos por 0s
00242     df = pd.DataFrame({ key:pd.Series(value) for key, value in import_data.items()}).fillna(0)
00243     # Guardar archivo
00244     df.to_csv(output_import_folder_year+"/results.csv", index=False)
00245     # Exportaciones
00246     export_data = {'country': list(country_export_value.keys()),
    'total_value':list(country_export_value.values()),
00247                    'total_value_pct':list(country_export_value_pct.values())}
00248     # Crear tabla. Si existen NaN, remplazarlos por 0s
00249     df = pd.DataFrame({ key:pd.Series(value) for key, value in export_data.items()}).fillna(0)
00250     # Guardar archivo
00251     df.to_csv(output_export_folder_year+"/results.csv", index=False)
00252     # Importaciones + Exportaciones
00253     total_data = {'country': list(country_total_value.keys()),
    'total_value':list(country_total_value.values()),
00254                   'total_value_pct':list(country_total_value_pct.values())}
00255     # Crear tabla. Si existen NaN, remplazarlos por 0s
00256     df = pd.DataFrame({ key:pd.Series(value) for key, value in total_data.items()}).fillna(0)
00257     # Guardar archivo
00258     df.to_csv(output_folder_year+"/results.csv", index=False)
00259     # Graficar resultados
```

```
00260     import_plot = Summary_Chart(output_import_folder_year)
00261     import_plot.h_bar_summary(country_import_value, "Paises con mayor valor", "Valor total (%)",
00262                               "Pais", "pais_valor")
00263     export_plot = Summary_Chart(output_export_folder_year)
00264     export_plot.h_bar_summary(country_export_value, "Paises con mayor valor", "Valor total (%)",
00265                               "Pais", "pais_valor")
00266     plot = Summary_Chart(output_folder_year)
00267     plot.h_bar_summary(country_total_value, "Paises con mayor valor", "Valor total (%)",
00268                        "Pais", "pais_valor")
```

7.3. Referencia del Archivo sl_filters.py

Clases

- class sl_filters.SynergyLogisticsFilters

Namespaces

- namespace sl_filters

Variables

- string sl_filters.DATA_FILE_PATH = "data\synergy_logistics_database.csv"

7.4. sl_filters.py

Ir a la documentación de este archivo.

```
00001 from typing import List
00002 import pandas as pd
00003 from pandas.core.frame import DataFrame
00004 from datetime import datetime
00005
00006 # Definir ubicación de archivo CSV
00007 DATA_FILE_PATH = "data\synergy_logistics_database.csv"
00008
00009 class SynergyLogisticsFilters():
00010     def __init__(self) -> None:
00011         """Lectura de la BD de Synergy Logistics.
00012         """
00013         self.SYNERGY_DBSYNERGY_DB = pd.read_csv(DATA_FILE_PATH, index_col="register_id")
00014         # Convert date column to datetime objects
00015         self.SYNERGY_DBSYNERGY_DB["date"] = pd.to_datetime(self.SYNERGY_DBSYNERGY_DB["date"])
00016
00017     def filter_routes_df(self, direction: str or None = None,
00018                          origin: str or None = None, destination: str or None = None,
00019                          start_year: int or None = None, end_year: int or None = None,
00020                          start_date: str or None = None, end_date: str or None = None,
00021                          product: str or None = None, transport_mode: str or None = None,
00022                          company_name: str or None = None, min_value: int or None = None,
00023                          max_value: int or None = None) -> DataFrame:
00024         """
00025         Filtra el dataframe de Synergy Logistics de acuerdo a valores en las columnas que tiene
00026         la tabla generada. Si no hay filtro, se regresa un dataframe completo.
00027
00028         Args:
00029             direction (str or None, optional): Tipo de dirección (Import o Export). Defaults to None.
00030             origin (str or None, optional): Pais de origen. Defaults to None.
00031             destination (str or None, optional): Pais de destino. Defaults to None.
00032             start_year (int or None, optional): Año inicial de periodo. Defaults to None.
00033             end_year (int or None, optional): Año final de periodo. Defaults to None.
00034             start_date (str or None, optional): Fecha inicial de periodo. Defaults to None.
00035             end_date (str or None, optional): Fecha final de periodo. Defaults to None.
00036             product (str or None, optional): Tipo de producto. Defaults to None.
00037             transport_mode (str or None, optional): Tipo de medio de transporte. Defaults to None.
00038             company_name (str or None, optional): Nombre de compañia. Defaults to None.
00039             min_value (int or None, optional): Valor minimo. Defaults to None.
```

```
00040             max_value (int or None, optional): Valor maximo. Defaults to None.
00041
00042         Returns:
00043             DataFrame:  Dataframe con columnas de la tabla que cumplen con los filtros indicados.
00044         """
00045         routes_table = self.SYNERGY_DBSYNERGY_DB
00046         # Get valid cases for string filters from unique values of each str column
00047         direction_cases = list(routes_table["direction"].unique())
00048         origin_countries = list(routes_table["origin"].unique())
00049         destination_countries = list(routes_table["destination"].unique())
00050         years = list(range(routes_table["year"].min(), datetime.now().year))
00051         product_types = list(routes_table["product"].unique())
00052         transport_modes = list(routes_table["transport_mode"].unique())
00053         companies = list(routes_table["company_name"].unique())
00054         date_format = ' %d/ %m/ %Y'
00055         # Add filter to column if valid input is given
00056         # Dirección
00057         if direction is not None:
00058             if direction in direction_cases:
00059                 routes_table=routes_table[routes_table["direction"] == direction]
00060             else:
00061                 print(f"El valor '{direction}' no es un filtro valido para la columna direction.")
00062         # Origen
00063         if origin is not None:
00064             if origin in origin_countries:
00065                 routes_table=routes_table[routes_table["origin"] == origin]
00066             else:
00067                 print(f"El valor '{origin}' no es un filtro valido para la columna origin.")
00068         # Destino
00069         if destination is not None:
00070             if destination in destination_countries:
00071                 routes_table=routes_table[routes_table["destination"] == destination]
00072             else:
00073                 print(f"El valor '{destination}' no es un filtro valido para la columna destination.")
00074         # Year
00075         if start_year is not None:
00076             if start_year in years:
00077                 routes_table=routes_table[routes_table["year"] >= start_year]
00078             else:
00079                 print(f"El valor '{start_year}' no es un año valido para la columna year.")
00080         if end_year is not None:
00081             if end_year in years:
00082                 routes_table=routes_table[routes_table["year"] <= end_year]
00083             else:
00084                 print(f"El valor '{end_year}' no es un año valido para la columna year.")
00085         # Date
00086         if start_date is not None:
00087             try:
00088                 datetime.strptime(start_date, date_format)
00089                 routes_table=routes_table[routes_table["date"] >= start_date]
00090             except ValueError:
00091                 print("Fecha invalida. Debe usarse el formato DD/MM/YYYY")
00092         if end_date is not None:
00093             try:
00094                 datetime.strptime(end_date, date_format)
00095                 routes_table=routes_table[routes_table["date"] <= end_date]
00096             except ValueError:
00097                 print("Fecha invalida. Debe usarse el formato DD/MM/YYYY")
00098         # Producto
00099         if product is not None:
00100             if product in product_types:
00101                 routes_table=routes_table[routes_table["product"] == product]
00102             else:
00103                 print(f"El valor '{product}' no es un filtro valido para la columna product.")
00104         # Modo de transporte
00105         if transport_mode is not None:
00106             if transport_mode in transport_modes:
00107                 routes_table=routes_table[routes_table["transport_mode"] == transport_mode]
00108             else:
00109                 print(f"El valor '{transport_mode}' no es un filtro valido para la columna product.")
00110         # Nombre de Compañia
00111         if company_name is not None:
00112             if company_name in companies:
00113                 routes_table=routes_table[routes_table["company_name"] == company_name]
00114             else:
00115                 print(f"El valor '{company_name}' no es un filtro valido para la columna
      company_name.")
00116         # Valor total
00117         if min_value is not None:
00118             routes_table=routes_table[routes_table["total_value"] >= min_value]
00119         if max_value is not None:
00120             routes_table=routes_table[routes_table["total_value"] <= max_value]
00121
00122         return routes_table
00123
00124
00125     def get_unique_values(self, category:str) -> List:
```

```
00126         """Genera lista con valores unicos de columna de la base de datos.
00127
00128         Args:
00129             category (str): Nombre de la columna.
00130
00131         Returns:
00132             List: Valores unicos en columna de la tabla.
00133         """
00134         # Si elemento es columna de la tabla, obtiene valores distintos.
00135         if category in self.SYNERGY_DBSYNERGY_DB.columns.values.tolist():
00136             unique_column_values = list(self.SYNERGY_DBSYNERGY_DB[category].unique())
00137         else:
00138             print("La categoría indicada no existe dentro de la Base de Datos")
00139             unique_column_values = []
00140         return unique_column_values
```

7.5. Referencia del Archivo synergy_services.py

Clases

- class synergy_services.Service

Namespaces

- namespace synergy_services

7.6. synergy_services.py

Ir a la documentación de este archivo.

```
00001 from typing import List
00002
00003 from processing.sl_filters import SynergyLogisticsFilters
00004
00005 class Service(SynergyLogisticsFilters):
00006     """
00007     Clase que contine servicios para el analisis de la tabla de Synergy Logistics.
00008     """
00009     def get_routes_list(self,  direction:str or None = None) -> List:
00010         """Genera una lista con todas las rutas diferentes de la tabla.
00011
00012         Args:
00013             direction (str or None, optional): Dirección de transacción. Defaults to None.
00014
00015         Returns:
00016             List: Lista con rutas con formato origen-destino.
00017         """
00018         routes_list = []
00019         # Filter tables by direction
00020         filtered_table = self.filter_routes_df(direction=direction)
00021         # Check row by row table
00022         for index, row in filtered_table.iterrows():
00023             # route=origin-destination
00024             route = (row['origin']+ "-" + row['destination'])
00025             if not route in routes_list:
00026                 routes_list.append(route)
00027         return routes_list
00028
00029     def get_total_elements(self, direction:str or None = None, year:int or None = None,
      transport_mode:str or None = None) -> int:
00030         """
00031         Cuenta el número de transacciones en una tabla filtrada.
00032         Se pueden filtrar resultados por dirección, año y/o medio de transporte.
00033
00034         Args:
00035             direction (str or None, optional): Dirección de transacción. Defaults to None.
00036             year (int or None, optional): Año de transacciones. Defaults to None.
00037             transport_mode (str or None, optional): Tipo de medio de transporte. Defaults to None.
00038
00039         Returns:
00040             int: Total de casos en tabla filtrada.
```

```
00041         """
00042         # Tabla filtrada
00043         filtered_table = self.filter_routes_df(direction=direction, start_year=year,
00044                                                end_year=year, transport_mode=transport_mode)
00045         # Contar filas en la tabla
00046         elements_count= len(filtered_table)
00047         return elements_count
00048 
00049     def get_route_frecuency(self, route:str, direction:str or None = None, year:int or None = None)->
      int:
00050         """
00051         Cuenta las veces que una ruta aparece en una tabla filtrada.
00052         Se pueden filtrar resultados por dirección, año y/o medio de transporte.
00053 
00054         Args:
00055             route (str): Rutas con formato origen-destino.
00056             direction (str or None, optional): Dirección de transacción. Defaults to None.
00057             year (int or None, optional): Año de transacciones. Defaults to None.
00058 
00059         Returns:
00060             int: Numero de apariciones de ruta en la tabla filtrada.
00061         """
00062         # Obtener origen y destino para filtros
00063         origin, destination = route.split("-")
00064         # Tabla filtrada
00065         filtered_table = self.filter_routes_df(origin=origin, destination=destination,
      direction=direction,
00066                                                start_year=year, end_year=year)
00067         # Contar filas en la tabla
00068         route_frecuency = len(filtered_table)
00069         return route_frecuency
00070 
00071     def get_total_value(self, direction:str or None = None, year:int or None = None, transport_mode:
      str or None = None) -> int:
00072         """
00073         Suma el valor total dentro de una tabla filtrada.
00074         Se pueden filtrar resultados por dirección, año y/o medio de transporte.
00075 
00076         Args:
00077             route (str): Rutas con formato origen-destino.
00078             direction (str or None, optional): Dirección de transacción. Defaults to None.
00079             year (int or None, optional): Año de transacciones. Defaults to None.
00080             transport_mode (str or None, optional): Tipo de medio de transporte. Defaults to None.
00081 
00082         Returns:
00083             int: suma de valor de elementos en tabla filtrada.
00084         """
00085         filtered_table = self.filter_routes_df(direction=direction, start_year=year, end_year=year,
      transport_mode=transport_mode)
00086         total_value = filtered_table["total_value"].sum()
00087         return total_value
00088 
00089     def get_route_value(self, route:str, direction:str or None = None, year:int or None = None) ->
      int:
00090         """
00091         Suma el valor total para una ruta especifica dentro de una tabla filtrada.
00092         Se pueden filtrar resultados por dirección, año y/o medio de transporte.
00093 
00094         Args:
00095             route (str): Rutas con formato origen-destino.
00096             direction (str or None, optional): Dirección de transacción. Defaults to None.
00097             year (int or None, optional): Año de transacciones. Defaults to None.
00098             transport_mode (str or None, optional): Tipo de medio de transporte. Defaults to None.
00099 
00100         Returns:
00101             int: suma de valor de elementos en tabla filtrada.
00102         """
00103         origin, destination = route.split("-")
00104         filtered_table = self.filter_routes_df(origin=origin, destination=destination,
      direction=direction,
00105                                                start_year=year, end_year=year)
00106         route_value = filtered_table["total_value"].sum()
00107         return route_value
00108 
00109     def get_top_ten(self, all_cases: dict) -> dict:
00110         """De un diccionario de elementos se obtienen los 10 casos con mejores resultados.
00111 
00112         Args:
00113             all_cases (dict): Diccionario con todos los casos
00114 
00115         Returns:
00116             List: Lista con los 10 casos con mejores resultados.
00117         """
00118         top_ten_cases = sorted(all_cases, key=all_cases.get, reverse=True)[:10]
00119         top_ten_dict = {}
00120         for case in top_ten_cases:
00121             top_ten_dict[case] = all_cases[case]
```

```
00122
00123         return top_ten_dict
00124
00125     def get_transport_frecuency(self, transport:str, direction:str or None = None, year:int or None =
      None)-> int:
00126         """
00127         Cuenta las veces que un transporte aparece en una tabla filtrada.
00128         Se pueden filtrar resultados por dirección, año y/o medio de transporte.
00129
00130         Args:
00131             transport (str): Tipo de medio de transporte.
00132             direction (str or None, optional): Dirección de transacción. Defaults to None.
00133             year (int or None, optional): Año de transacciones. Defaults to None.
00134
00135         Returns:
00136             int: Numero de apariciones de transporte en la tabla filtrada.
00137         """
00138         # Tabla filtrada
00139         filtered_table = self.filter_routes_df(transport_mode=transport, direction=direction,
00140                                                start_year=year, end_year=year)
00141         # Contar filas en la tabla
00142         transport_frecuency = len(filtered_table)
00143         return transport_frecuency
00144
00145
00146     def get_transport_value(self, transport:str, direction:str or None = None, year:int or None =
      None) -> int:
00147         """
00148         Suma el valor total para un transporte especifico dentro de una tabla filtrada.
00149         Se pueden filtrar resultados por dirección, año y/o medio de transporte.
00150
00151         Args:
00152             transport (str): Tipo de medio de transporte.
00153             direction (str or None, optional): Dirección de transacción. Defaults to None.
00154             year (int or None, optional): Año de transacciones. Defaults to None.
00155
00156         Returns:
00157             int: suma de valor de elementos en tabla filtrada.
00158         """
00159         filtered_table = self.filter_routes_df(transport_mode=transport, direction=direction,
00160                                                start_year=year, end_year=year)
00161         transport_value = filtered_table["total_value"].sum()
00162         return transport_value
00163
00164
00165     def get_country_frecuency(self, origin:str or None = None, destination:str or None = None,
      direction:str or None = None, year:int or None = None)-> int:
00166         """
00167         Cuenta las veces que un pais aparece en una tabla filtrada.
00168         Se pueden filtrar resultados por dirección, año y/o medio de transporte.
00169
00170         Args:
00171             origin (str or None, optional): Pais de origen. Defaults to None.
00172             destination (str or None, optional): Pais de destino. Defaults to None.
00173             direction (str or None, optional): Dirección de transacción. Defaults to None.
00174             year (int or None, optional): Año de transacciones. Defaults to None.
00175
00176         Returns:
00177             int: Numero de apariciones de transporte en la tabla filtrada.
00178         """
00179         # Tabla filtrada
00180         filtered_table = self.filter_routes_df(origin=origin, destination=destination,
      direction=direction,
00181                                                start_year=year, end_year=year)
00182         # Contar filas en la tabla
00183         transport_frecuency = len(filtered_table)
00184         return transport_frecuency
00185
00186
00187     def get_country_value(self, origin:str or None = None, destination:str or None = None,
      direction:str or None = None, year:int or None = None) -> int:
00188         """
00189         Suma el valor total para un pais especifico dentro de una tabla filtrada.
00190         Se pueden filtrar resultados por dirección, año y/o medio de transporte.
00191
00192         Args:
00193             origin (str or None, optional): Pais de origen. Defaults to None.
00194             destination (str or None, optional): Pais de destino. Defaults to None.
00195             direction (str or None, optional): Dirección de transacción. Defaults to None.
00196             year (int or None, optional): Año de transacciones. Defaults to None.
00197
00198         Returns:
00199             int: suma de valor de elementos en tabla filtrada.
00200         """
00201         filtered_table = self.filter_routes_df(origin=origin, destination=destination,
      direction=direction,
00202                                                start_year=year, end_year=year)
```

```
00203         transport_value = filtered_table["total_value"].sum()
00204         return transport_value
00205
00206     def reorder_dict_max(self, data_dict: dict) -> dict:
00207         """
00208         Ordena diccionario a partir de valores de mayor a menor.
00209         Elimita los elementos del diccionario que tengan un valor de 0.
00210
00211         Args:
00212             data_dict (dict): Diccionario de datos desordenados.
00213
00214         Returns:
00215             dict: Diccionario de datos filtrados.
00216         """
00217         # Crear nuevo diccionario para almacenar datos ordenados
00218         ordered_data_dict = {}
00219         ordered_keys = sorted(data_dict, key=data_dict.get, reverse=True)
00220         for key in ordered_keys:
00221             # if value is 0, skip
00222             if data_dict[key] == 0:
00223                 continue
00224             # if value > 0
00225             else:
00226                 ordered_data_dict[key]=data_dict[key]
00227         return ordered_data_dict
```

7.7. Referencia del Archivo graph_utils.py

Clases

- class graph_utils.Chart
- class graph_utils.Summary_Chart

Namespaces

- namespace graph_utils

7.8. graph_utils.py

Ir a la documentación de este archivo.

```
00001 import os
00002
00003 import plotly.graph_objects as go
00004 from plotly.missing_ipywidgets import FigureWidget
00005
00006 # Comando necesario para usar librerias de imagenes fijas
00007 os.environ["PATH"] = os.environ["PATH"] +
       f";{os.path.abspath('venv/lib/site-packages/kaleido/executable/')}"
00008
00009 class Chart:
00010     """
00011     Funciones generales para la generación de gráficas.
00012     """
00013     def save_as_image(self, fig:FigureWidget, file_name: str):
00014         """ Esta función guarda una gráfica como imagen.
00015         Args:
00016             fig (plotly.graph_objects.Figure): Objeto de la gráfica creada.
00017             file_name (str): Ruta donde almacenar el archivo.
00018         """
00019         fig.write_image(file_name, width=1350, height=730)
00020
00021     def save_plot(self, fig:FigureWidget, file_name:str):
00022         # Validar si folder existe, o si es necesario crearlo
00023         if not os.path.exists(f"{self.file_path}"):
00024             os.makedirs(f"{self.file_path}")
00025         # Guardar grafico interactivo como html
00026         fig.write_html(f"{self.file_path}/{file_name}.html")
00027         # Guardar imagen statica en
00028         self.save_as_imagesave_as_image(fig, f"{self.file_path}/{file_name}.png")
```

```
00029         # Indicar a usuario en consola
00030         print(f"Resultado gráficado en {self.file_path}/{file_name}.png")
00031
00032 class Summary_Chart(Chart):
00033     """
00034     Funciones para gráfica resultados de las consultas realizadas de las tablas de Lifestore.
00035     """
00036     def __init__(self, file_path: str) -> None:
00037         """Establece parametros default para objetos de la clase.
00038
00039         Args:
00040             file_path (str): Ruta donde guardar gráficas.
00041         """
00042         super().__init__()
00043         # Definir folder donde se ubicaran las gráficas
00044         self.file_pathfile_path = file_path
00045         # Agregar atributos de formato de gráfica (visualización)
00046         self.layoutlayout = go.Layout(
00047             title=dict(y=0.99, x=0.5, xanchor='center', yanchor='top'),
00048             xaxis=dict(showgrid=True, showline=True, linewidth=1,
00049                        linecolor='black', mirror=True, gridwidth=0.4,
00050                        gridcolor='rgb(204,209,208)', tickfont=dict(size=20),
00051                        type='category'),
00052             yaxis=dict(zeroline=False, showline=True, linewidth=1,
00053                        linecolor='black', mirror=True, gridwidth=0.4,
00054                        gridcolor='rgb(204, 209, 208)', tickfont=dict(size=20)),
00055             margin=dict(r=20, t=35),
00056             plot_bgcolor='rgba(0,0,0,0)', width=1080, height=566,
00057             font=dict(family='Arial, monospace', size=18))
00058
00059     def bar_summary(self, data: dict, plot_title: str, x_axis_name: str, y_axis_name: str,
00060                     file_name: str, color:str = "blue") -> None:
00061         """Gráfica de barras a partir de un diccionario.
00062
00063         Args:
00064             data (dict): Datos a graficar.
00065             plot_title (str): Titulo de la gráfica.
00066             x_axis_name (str): Nombre de eje x.
00067             y_axis_name (str): Nombre de eje y.
00068             file_name (str): Nombre del archivo.
00069             color (str, optional): Color del gráfico. Defaults to "blue".
00070         """
00071         # Se da de alta diccionario con colores disponibles
00072         color_dict = {"blue": "rgb(15, 78, 171)", "green": "rgb(15, 171, 72)",
00073                       "red": "rgb(232, 4, 0)", "purple": "rgb(109, 15, 171)",
00074                       "yellow": "rgb(199, 199, 18)", "orange": "rgb(232, 155, 0)"}
00075         # Seleccionar color del gráfico
00076         plot_color = color_dict[color]
00077
00078         # Separar diccionarios en dos listas, una para cada eje
00079         x_data = list(data.keys())
00080         y_data = list(data.values())
00081
00082         # Generar objeto de gráfica
00083         plot_bar = go.Bar(x=x_data, y=y_data)
00084         mydata = [plot_bar]
00085
00086         # Formato de la grafica de barras
00087         fig = go.Figure(data=mydata, layout=self.layoutlayout)
00088
00089         # Actualizar color del grafico
00090         fig.update_traces(marker_color=plot_color, marker_line_color=plot_color)
00091         # Actualizar gráfico con datos de entrada
00092         fig.update_layout(
00093             title_text=plot_title,
00094             xaxis_title=x_axis_name,
00095             yaxis_title=y_axis_name)
00096         # Salvar resultados
00097         self.save_plotsave_plot(fig, file_name)
00098
00099     def h_bar_summary(self, data: dict, plot_title: str, x_axis_name: str, y_axis_name: str,
00100                       file_name: str, color:str = "blue") -> None:
00101         """Gráfica de barras a partir de un diccionario.
00102
00103         Args:
00104             data (dict): Datos a graficar.
00105             plot_title (str): Titulo de la gráfica.
00106             x_axis_name (str): Nombre de eje x.
00107             y_axis_name (str): Nombre de eje y.
00108             file_name (str): Nombre del archivo.
00109             color (str, optional): Color del gráfico. Defaults to "blue".
00110         """
00111         # Se da de alta diccionario con colores disponibles
00112         color_dict = {"blue": "rgb(15, 78, 171)", "green": "rgb(15, 171, 72)",
00113                       "red": "rgb(232, 4, 0)", "purple": "rgb(109, 15, 171)",
00114                       "yellow": "rgb(199, 199, 18)", "orange": "rgb(232, 155, 0)"}
00115         # Seleccionar color del gráfico
```

```
00116         plot_color = color_dict[color]
00117
00118         # Separar diccionarios en dos listas, una para cada eje
00119         y_data = list(data.keys())
00120         y_data.reverse()
00121         x_data = list(data.values())
00122         x_data.reverse()
00123
00124         # Generar objeto de gráfica
00125         plot_bar = go.Bar(x=x_data, y=y_data, orientation="h")
00126         mydata = [plot_bar]
00127
00128         # Formato de la grafica de barras
00129         fig = go.Figure(data=mydata, layout=self.layoutlayout)
00130
00131         # Actualizar color del grafico
00132         fig.update_traces(marker_color=plot_color, marker_line_color=plot_color)
00133         # Actualizar gráfico con datos de entrada
00134         fig.update_layout(
00135             title_text=plot_title,
00136             xaxis_title=x_axis_name,
00137             yaxis_title=y_axis_name,
00138             xaxis_type = "linear")
00139         # Salvar resultados
00140         self.save_plotsave_plot(fig, file_name)
00141
00142     def pie_summary(self, data: dict, plot_title: str, file_name: str):
00143         # Separar diccionarios en dos listas, una para cada eje
00144         x_data = list(data.keys())
00145         y_data = list(data.values())
00146         # Definir colores
00147         colors =  ['rgb(68,138,216)', 'rgb(184,204,208)', 'rgb(161,0,0)', 'rgb(169,161,82)']
00148         # Generar objeto de gráfica
00149         layout = go.Layout(
00150             title=dict(y=0.96, x=0.5, xanchor='center', yanchor='top',
00151                        font=dict(size=30)),
00152             margin=dict(t=80),
00153             plot_bgcolor='rgba(0,0,0,0)', width=1080, height=566,
00154             font=dict(family='Arial, monospace', size=18),
00155             legend = dict(font = dict(size=30)))
00156         plot_bar = go.Pie(labels=x_data, values=y_data, marker_colors=colors)
00157         mydata = [plot_bar]
00158         # Formato de la grafica de barras
00159         fig = go.Figure(data=mydata, layout=layout)
00160         fig.update_traces(hoverinfo='label+percent', textfont_size=30)
00161         fig.update_layout(
00162             title_text=plot_title,
00163             uniformtext_minsize=30, uniformtext_mode='hide')
00164         self.save_plotsave_plot(fig, file_name)
```

Índice alfabético