Environments Guide

Development OS/Kernel Distro: `Ubuntu 24.04 Desktop (AMD Ryzen, 12 Threads, 128GB RAM)` & `Ubuntu 22.04 Server (HP DL360p Gen 8, 32 Threads, 256GB RAM)`

(Code has been test run on `Windows 10` and `Mac OS (M1/2)` with no environment issues)

Python: `3.10.8` across all systems using pyenv `source`

Packages & Libraries managed by `pipenv` with easy `pip install pipenv` on `python 3.10.8` followed by `pipenv shell` (Standard `Conda Mini/Anaconda` for `python 3.10.8` supported)

.ipynb files are where the main logic of my code is at, and custom defined functions used regulary across notebook are store in `BoT_Exports/helper.py` , Certain code is ran on `.py` files for parallelization due to the walk forward simulations being computationally expensive (However logic is written & explained in the notebook and viewing/running those files would not be needed but are located at `BoT_Exports/data/archive` if interested)

Each .ipynb notebook files runtime are kept within < ~5mins, data that takes long to load have been pre-run and saved in binary pickle files in `BoT_Exports/data/` which will be called automatically. Notebook should not be compute/memory intesive outside the usual normal conditions.

Thank you for your time & insight. It was interesting and this project has taught me alot. I would love to hear your feedback over email(wilfred.edward.tan@gmail.com), or coffee/tea(telegram @wheelfredie or +65 93841810). Have a great week ahead!

Notebook `0_data_ETL.ipynb` : Pre-processing & Data Integrity Checks

Upcoming Notebook `1_Composote_export_value.ipynb` : check statistical

properties of timeseries & Out of Sample forecasting of export value for individual components

Upcoming Notebook `2_composite_weight_analysis.ipynb` : check if weights change drastically across months and Out of Sample forecast of next month(t) using last release(t-1,...,t-ar_lag) for individual components

Upcoming Notebook `3_stitching_series.ipynb` : Restitching the Components into the Total Export Series(BoP) with Error Optimisation

Import Relevant Packages and Custom Helper Functions

- Stored in `BoT_Exports/helper.py` to make code cleaner across .ipynb notebooks and .py scripts

In [1]:
```
from helper import *

# Ignore FutureWarning
warnings.simplefilter(action='ignore', category=FutureWarning)
pd.set_option('mode.chained_assignment', None)  # to avoid SettingWithCopyWa
```

Focus on Monthly Dataset, clean into one

In [2]:
```
df_total_export_raw_M1 = pd.read_excel("data/given/EC_XT_009_S3_ENG_ALL.XLSX
                                       sheet_name="Monthly")
df_total_export_raw_M2 = pd.read_excel("data/given/EC_XT_009_S3_ENG_ALL.XLSX
                                          sheet_name="Monthly (2)")
display(df_total_export_raw_M1.head(15))
display(df_total_export_raw_M1.tail(15))
display(df_total_export_raw_M2.head(15))
display(df_total_export_raw_M2.tail(15))
```

	Bank of Thailand	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Un
0	EC_XT_009_S3 : Total Value and Quantity of Exp...	NaN	NaN	NaN	NaN	NaN	NaN	
1	(Unit : Millions of US Dollars)	NaN	NaN	NaN	NaN	NaN	NaN	
2	Last Updated :	NaN	NaN	NaN	NaN	NaN	NaN	
3	Retrieved date :	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	JAN 1995	FEB 1995	MAR 1995	APR 1995	MAY 1995	JU
5	1	Agriculture	584.19	550.96	605.24	453.51	587.16	
6	2	Rice	196.61	186.2	175.64	107.99	191.15	
7	3	: Quantity (Metric Ton)	724863.53	680984.39	593857.64	354495.85	655052.39	42
8	4	Rubber	229.04	216.29	238.01	181.01	216.77	
9	5	: Quantity (Metric Ton)	178792.79	146901.62	148892.29	109139.54	131094.16	1
10	6	Durian	0.54	0.58	3.29	11.73	15.06	
11	7	: Quantity (Metric Ton)	285.22	240.99	2029.83	15009.54	18769.56	
12	8	Other Fruits	1.97	2.6	4.13	6.4	12.38	
13	9	: Quantity (Metric Ton)	3957.65	5446.82	8129.21	10454.96	15811.13	
14	10	Horticultural products, n.i.e.	116.57	104.63	132.28	103.94	107.54	

15 rows × 256 columns

	Bank of Thailand	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unn
92	88	Other products	44.56	31.22	57.73	38.24	57.97	
93	89	Re-Exports 1/	11.44	4.8	6.24	7.11	6.44	
94	90	Total Exports (Customs basis)	3995.51	3995.11	5188.91	4065.8	4880.42	5(
95	91	Adjustment for balance of payments 2/	0	0	0	0	0	
96	92	Coverage Adjustment	-67.51	-46.11	-105.91	-60.8	-91.42	
97	93	Classification adjustment	0	0	0	0	0	
98	94	Valuation Adjustment	0	0	0	0	0	
99	95	Timing Adjustment	0	0	0	0	0	
100	96	Exports, f.o.b. (BOP basis)	3928	3949	5083	4005	4789	
101	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
102	Source:	NaN	NaN	NaN	NaN	NaN	NaN	
103	Customs Department (Compiled by The Bank of Th...	NaN	NaN	NaN	NaN	NaN	NaN	
104	Remark:	NaN	NaN	NaN	NaN	NaN	NaN	
105	1/ Re-exports are exports of foreign goods whi...	NaN	NaN	NaN	NaN	NaN	NaN	
106	2/ Please refer to the metadata of this table ...	NaN	NaN	NaN	NaN	NaN	NaN	

15 rows × 256 columns

	Bank of Thailand	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Un
0	EC_XT_009_S3 : Total Value and Quantity of Exp...	NaN	NaN	NaN	NaN	NaN	NaN	
1	(Unit : Millions of US Dollars)	NaN	NaN	NaN	NaN	NaN	NaN	
2	Last Updated :	NaN	NaN	NaN	NaN	NaN	NaN	
3	Retrieved date :	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	MAR 2016	APR 2016	MAY 2016	JUN 2016	JUL 2016	Al
5	1	Agriculture	1210.32	1138.96	1179.68	1059.21	899.56	
6	2	Rice	428.88	313.81	333.33	326	225.59	
7	3	: Quantity (Metric Ton)	991380.71	704952.25	728895.72	718879.6	445102.3	6
8	4	Rubber	359.93	384.41	336.07	329.9	331.19	
9	5	: Quantity (Metric Ton)	335325.36	300349.17	256108.57	248608.35	256759.84	28
10	6	Durian	24.5	92.14	80.77	47.02	41.2	
11	7	: Quantity (Metric Ton)	22552.77	77027.09	61738.9	34138.98	31030.08	5
12	8	Other Fruits	53.08	48.62	77.33	51.62	48.21	
13	9	: Quantity (Metric Ton)	67952.1	58199.7	85476.31	50601.81	55158.16	10
14	10	Horticultural products, n.i.e.	251.93	223.6	265.76	234.22	179.66	

15 rows × 99 columns

	Bank of Thailand	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unn
92	88	Other products	33.48	27.99	38.9	40.76	32.07	
93	89	Re-Exports 1/	127.26	23.99	33.82	18.98	39.06	
94	90	Total Exports (Customs basis)	19170.19	15609.27	17697.18	18152.04	17064.08	18
95	91	Adjustment for balance of payments 2/	0	0	0	0	0	
96	92	Coverage Adjustment	-470.42	12.56	18.68	-38.97	-62.62	
97	93	Classification adjustment	0	0	0	0	0	
98	94	Valuation Adjustment	0	0	0	0	0	
99	95	Timing Adjustment	0	0	131.19	251.23	289.05	-
100	96	Exports, f.o.b. (BOP basis)	18699.77	15621.83	17847.05	18364.3	17290.51	18
101	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
102	Source:	NaN	NaN	NaN	NaN	NaN	NaN	
103	Customs Department (Compiled by The Bank of Th...	NaN	NaN	NaN	NaN	NaN	NaN	
104	Remark:	NaN	NaN	NaN	NaN	NaN	NaN	
105	1/ Re-exports are exports of foreign goods whi...	NaN	NaN	NaN	NaN	NaN	NaN	
106	2/ Please refer to the metadata of this table ...	NaN	NaN	NaN	NaN	NaN	NaN	

15 rows × 99 columns

In [3]:
```
#get headers
COLNAME_df_total_export_raw_M1 = df_total_export_raw_M1.iloc[4].tolist()
COLNAME_df_total_export_raw_M1[1] = "class"
COLNAME_df_total_export_raw_M1 = COLNAME_df_total_export_raw_M1[1:]
#remove unwanted rows
df_total_export_raw_M1 = df_total_export_raw_M1.iloc[5:101,1:]
df_total_export_raw_M1.columns = COLNAME_df_total_export_raw_M1
df_total_export_raw_M1.reset_index(drop=True, inplace=True)
display(df_total_export_raw_M1)

#same for M(2)
COLNAME_df_total_export_raw_M2 = df_total_export_raw_M2.iloc[4].tolist()
COLNAME_df_total_export_raw_M2[1] = "class"
COLNAME_df_total_export_raw_M2 = COLNAME_df_total_export_raw_M2[1:]
#remove unwanted rows
df_total_export_raw_M2 = df_total_export_raw_M2.iloc[5:101,1:]
df_total_export_raw_M2.columns = COLNAME_df_total_export_raw_M2
df_total_export_raw_M2.reset_index(drop=True, inplace=True)
display(df_total_export_raw_M2)

#check that class order are the same
display((df_total_export_raw_M1["class"] == df_total_export_raw_M2["class"])
#drop class from M2
df_total_export_raw_M2.drop(columns=["class"], inplace=True)

#side concat
df_total_export = pd.concat([df_total_export_raw_M1, df_total_export_raw_M2]
df_total_export["class"] = [col.strip().replace(" ", "_") if isinstance(col,
display(df_total_export)

#remove "/" from class names
df_total_export["class"] = df_total_export["class"].str.replace("/", "")
df_total_export.to_csv("data/cleaned/total_export.csv", index=False)
df_total_export.to_pickle("data/cleaned/total_export.pkl")
```

	class	JAN 1995	FEB 1995	MAR 1995	APR 1995	MAY 1995	JUN 1995	JUL 1
0	Agriculture	584.19	550.96	605.24	453.51	587.16	516.47	51
1	Rice	196.61	186.2	175.64	107.99	191.15	130.92	14
2	: Quantity (Metric Ton)	724863.53	680984.39	593857.64	354495.85	655052.39	428995.16	44671
3	Rubber	229.04	216.29	238.01	181.01	216.77	250.77	
4	: Quantity (Metric Ton)	178792.79	146901.62	148892.29	109139.54	131094.16	160773.2	16550
...	...	...	...	...	...	...	...	
91	Coverage Adjustment	-67.51	-46.11	-105.91	-60.8	-91.42	-82.37	-9
92	Classification adjustment	0	0	0	0	0	0	
93	Valuation Adjustment	0	0	0	0	0	0	
94	Timing Adjustment	0	0	0	0	0	0	
95	Exports, f.o.b. (BOP basis)	3928	3949	5083	4005	4789	4943	4

96 rows × 255 columns

	class	MAR 2016	APR 2016	MAY 2016	JUN 2016	JUL 2016	AUG 2016	SEP 2
0	Agriculture	1210.32	1138.96	1179.68	1059.21	899.56	1141.34	116
1	Rice	428.88	313.81	333.33	326	225.59	302.3	35
2	: Quantity (Metric Ton)	991380.71	704952.25	728895.72	718879.6	445102.3	629600.2	78720
3	Rubber	359.93	384.41	336.07	329.9	331.19	367.78	36
4	: Quantity (Metric Ton)	335325.36	300349.17	256108.57	248608.35	256759.84	282243.17	27961
...	...	...	...	...	...	...	...	
91	Coverage Adjustment	-470.42	12.56	18.68	-38.97	-62.62	10.48	-20
92	Classification adjustment	0	0	0	0	0	0	
93	Valuation Adjustment	0	0	0	0	0	0	
94	Timing Adjustment	0	0	131.19	251.23	289.05	-677.84	
95	Exports, f.o.b. (BOP basis)	18699.77	15621.83	17847.05	18364.3	17290.51	18077.42	1922

96 rows × 98 columns

True

	class	JAN 1995	FEB 1995	MAR 1995	APR 1995	MAY 1995	JUN
0	Agriculture	584.19	550.96	605.24	453.51	587.16	5
1	Rice	196.61	186.2	175.64	107.99	191.15	1
2	:_Quantity_(Metric_Ton)	724863.53	680984.39	593857.64	354495.85	655052.39	4289
3	Rubber	229.04	216.29	238.01	181.01	216.77	2
4	:_Quantity_(Metric_Ton)	178792.79	146901.62	148892.29	109139.54	131094.16	160
...	...	...	...	...	...	...	
91	Coverage_Adjustment	-67.51	-46.11	-105.91	-60.8	-91.42	-
92	Classification_adjustment	0	0	0	0	0	
93	Valuation_Adjustment	0	0	0	0	0	
94	Timing_Adjustment	0	0	0	0	0	
95	Exports,_f.o.b._(BOP_basis)	3928	3949	5083	4005	4789	

96 rows × 352 columns

Check that the constituent classes sums up to the total exports, if so, we can use weighted matrix after decomposition to build back

In [4]:
```
class_map = pd.read_excel("data/given/MAP_exports.xlsx",
                          sheet_name="export_value")
#standardise names
class_map.columns = [col.strip().replace(" ", "_") if isinstance(col, str) a
class_map = class_map.map(lambda x: str(x).strip().replace(" ", "_") if isir

#also for values
#manufacturing has alot of subsets, however we cannot ignore as it has a hug
# and seasonality might differ between components
manufacturing_map = pd.read_excel("data/given/MAP_exports.xlsx",
                                  sheet_name="manufacturing export_value")
#standardise names
manufacturing_map.columns = [col.strip().replace(" ", "_") if isinstance(col
manufacturing_map = manufacturing_map.map(lambda x: str(x).strip().replace("


display(class_map)
display(manufacturing_map)
```

	Agriculture	Fishery	Forestry	Mining
0	Rice	Crustaceans	NaN	Crude_oil
1	Rubber	Fish	NaN	Mineral_products,_n.i.e.
2	Durian	Cuttlefish,_squid,_octopus	NaN	NaN
3	Other_Fruits	Fishery_products,_n.i.e.	NaN	NaN
4	Horticultural_products,_n.i.e.	NaN	NaN	NaN
5	Animal_products	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN P
13	NaN	NaN	NaN	NaN
14	NaN	NaN	NaN	NaN
15	NaN	NaN	NaN	NaN
16	NaN	NaN	NaN	NaN

	Agro-manufacturing_Products	Apparels_and_Textile_Materials	Footwar
0	Sugar	NaN	
1	Fish,_canned,_prepared,_or_preserved	NaN	
2	Crustaceans_canned,_prepared,_or_preserved	NaN	
3	Meat_of_poultry,_canned,_prepared,_or_preserved	NaN	
4	Tapioca_&_cassava_starch	NaN	
5	Other_fruits_&_vegetables,_canned,_prepared,_o...	NaN	
6	Preparation_of_cereals_flour_or_starch	NaN	
7	Food_products,_n.i.e.	NaN	
8	Beverages	NaN	
9	Rubber_products	NaN	
10	Paper_products	NaN	
11	Wood_&_wood_products	NaN	
12	Leather_&_leather_products	NaN	
13	Animal_food	NaN	
14	Other_agro-manufacturing_products	NaN	

In [5]:
```
df_total_export
```

Out[5]:

	class	JAN 1995	FEB 1995	MAR 1995	APR 1995	MAY 1995	JU
0	Agriculture	584.19	550.96	605.24	453.51	587.16	
1	Rice	196.61	186.2	175.64	107.99	191.15	
2	:_Quantity_(Metric_Ton)	724863.53	680984.39	593857.64	354495.85	655052.39	428
3	Rubber	229.04	216.29	238.01	181.01	216.77	
4	:_Quantity_(Metric_Ton)	178792.79	146901.62	148892.29	109139.54	131094.16	16
...	...	...	...	...	...	...	
91	Coverage_Adjustment	-67.51	-46.11	-105.91	-60.8	-91.42	
92	Classification_adjustment	0	0	0	0	0	
93	Valuation_Adjustment	0	0	0	0	0	
94	Timing_Adjustment	0	0	0	0	0	
95	Exports,_f.o.b._(BOP_basis)	3928	3949	5083	4005	4789	

96 rows × 352 columns

In [6]:
```
df_map_overall = pd.DataFrame({"Total_Exports_(Customs_basis)": class_map.co
df_map_overall
```

Out[6]:

	Total_Exports_(Customs_basis)
0	Agriculture
1	Fishery
2	Forestry
3	Mining
4	Manufacturing
5	Other_Exports
6	Re-Exports_1

In [7]:
```
# For export map
def is_equal_ConstituentSum(df: pd.DataFrame,
                            df_export: pd.DataFrame) -> pd.DataFrame:
    cols = df.columns

    lst_col = []
    lst_is_equal = []
    for col in cols:
        constituents = df[col].dropna()
        if len(constituents) == 0:
            lst_col.append(col)
            lst_is_equal.append(True)
            continue

        #get col timeseries
        data_col = df_export[df_export["class"] == col].iloc[0]

        #get timeseries for constituents & sum rows
        data_constituents = df_export[df_export["class"].isin(constituents)]

        #Ensure that the sum of the constituents is equal(with tolerance) to
        tolerance = 1e-2 #per constituent
        is_equal = (data_col[1:] - data_constituents[1:]).sum() <= (toleranc

        lst_col.append(col)
        lst_is_equal.append(is_equal)

    df_is_equal = pd.DataFrame({"class": lst_col, "is_equal": lst_is_equal})
    return df_is_equal

df_is_equal_class_map = is_equal_ConstituentSum(class_map, df_total_export)
df_is_equal_manufacturing_map = is_equal_ConstituentSum(manufacturing_map, d
df_is_equal_map_overall = is_equal_ConstituentSum(df_map_overall, df_total_e

display(df_is_equal_class_map)
display(df_is_equal_manufacturing_map)
display(df_is_equal_map_overall)
```

	class	is_equal
0	Agriculture	True
1	Fishery	True
2	Forestry	True
3	Mining	True
4	Manufacturing	True
5	Other_Exports	False
6	Re-Exports_1	True
7	Total_Exports_(Customs_basis)	True

	class	is_equal
0	Agro-manufacturing_Products	True
1	Apparels_and_Textile_Materials	True
2	Footware_and_parts	True
3	Electronics	True
4	Electrical_Appliances	True
5	Metal_&_Steel	True
6	Automotive	True
7	Aircrafts,_ships,_floating_structures,_and_loc...	True
8	Machinery_&_Equipment	True
9	Jewellery	True
10	Chemicals_&_Petro-chemical_Products	True
11	Petroleum_products	True
12	Photographic_&_cinematographic_instruments_&_s...	True
13	Medicinal_and_surgical_equipment_and_supplies	True
14	Toiletries_and_cosmetics	True
15	Furniture_and_parts	True
16	Other_Manufacturing_products	True

	class	is_equal
0	Total_Exports_(Customs_basis)	True

In [8]:
```
#investigate the ones that are not equal: Forestry, Other Exports, Total Exp
col = "Other_Exports"
cols = class_map[col].dropna()
constituents = class_map[col].dropna()
df_export = df_total_export.copy(deep=True)
```

```
#get col timeseries
data_col = df_export[df_export["class"] == col].iloc[0]
#get timeseries for constituents & sum rows
data_constituents = df_export[df_export["class"].isin(constituents)].sum(axi
diff = data_col[1:] - data_constituents[1:]
total_diff = diff.sum()

#composite & class weight
composite_weight = df_export[df_export["class"].isin(constituents)]

composite_weight.set_index("class",
                           inplace=True,
                           drop=True)

composite_weight = composite_weight.T

df_compare = pd.DataFrame({"class_total": data_col[1:],
                           "constituents_total": data_constituents[1:],
                           "diff": diff})
df_compare = pd.concat([df_compare, composite_weight], axis=1)
for col in constituents:
    df_compare[col] = df_compare[col] / df_compare["constituents_total"]


#enfore float
df_compare = df_compare.astype(float)
df_compare = df_compare.round(2)

# for diff get the outliers via quantiles
q1 = df_compare["diff"].quantile(0.25)
q3 = df_compare["diff"].quantile(0.75)
iqr = q3 - q1
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr
df_compare_outliers = df_compare[(df_compare["diff"] < lower_bound) | (df_cc

display(total_diff)
display(df_compare)
display(df_compare_outliers)
```

```
4199.369999999999
```

	class_total	constituents_total	diff	Non-monetary_gold_(excl._articles_of_goldsmiths)	Ot
JAN 1995	50.89	50.72	0.17	0.12	
FEB 1995	40.87	40.74	0.13	0.23	
MAR 1995	67.21	66.93	0.28	0.14	
APR 1995	43.43	43.21	0.22	0.12	
MAY 1995	62.26	62.00	0.26	0.06	
...	...	...	...	...	
NOV 2023	266.27	263.32	2.95	0.96	
DEC 2023	454.12	452.65	1.47	0.93	
JAN 2024	493.70	491.55	2.15	0.96	
FEB 2024	771.18	769.90	1.28	0.97	
MAR 2024	463.03	428.78	34.25	0.93	

351 rows × 5 columns

	class_total	constituents_total	diff	Non-monetary_gold_(excl._articles_of_goldsmiths)
DEC 1995	92.70	87.75	4.95	0.03
JUL 1997	98.65	78.50	20.15	0.07
FEB 2001	175.33	172.08	3.25	0.01
APR 2002	166.18	162.97	3.21	0.04
MAR 2003	296.74	286.78	9.96	0.03
APR 2004	271.70	236.90	34.80	0.05
JUN 2004	215.91	193.61	22.30	0.19
FEB 2005	139.44	129.98	9.46	0.05
MAR 2005	243.17	228.54	14.63	0.04
OCT 2005	162.73	157.91	4.82	0.12
APR 2006	304.53	295.59	8.94	0.61
AUG 2006	206.58	196.82	9.76	0.08
FEB 2007	292.15	287.52	4.63	0.63
FEB 2008	219.20	214.13	5.07	0.71
APR 2008	108.78	97.87	10.91	0.35
MAY 2008	226.46	215.19	11.27	0.75
AUG 2008	115.11	80.32	34.79	0.27
SEP 2008	486.26	476.96	9.30	0.81
MAR 2010	219.67	211.99	7.68	0.62
SEP 2010	757.06	753.75	3.31	0.93
MAY 2011	401.20	397.57	3.63	0.84

	class_total	constituents_total	diff	Non-monetary_gold_(excl._articles_of_goldsmiths)
MAR 2013	130.39	126.54	3.85	0.56
JUN 2014	406.74	403.42	3.32	0.83
MAR 2015	222.62	217.95	4.67	0.79
JUN 2015	221.73	218.16	3.57	0.78
JUL 2015	199.23	196.15	3.08	0.79
MAY 2016	792.30	789.09	3.21	0.95
OCT 2016	264.42	246.13	18.29	0.91
DEC 2016	454.31	450.04	4.27	0.94
MAR 2017	555.33	529.80	25.53	0.85
APR 2017	392.86	384.51	8.35	0.93
FEB 2018	296.48	291.36	5.12	0.86
MAY 2018	450.58	447.00	3.58	0.92
DEC 2018	441.30	437.30	4.00	0.93
FEB 2019	2463.81	542.84	1920.97	0.94
NOV 2019	392.65	376.63	16.02	0.88
MAR 2020	2035.08	1339.77	695.31	0.96
APR 2020	3116.36	2547.02	569.34	0.98
AUG 2021	434.03	424.91	9.12	0.94
MAR 2022	2715.39	2704.75	10.64	0.99
APR 2022	599.66	488.59	111.07	0.95
JUN 2022	598.49	592.89	5.60	0.96

	class_total	constituents_total	diff	Non-monetary_gold_(excl._articles_of_goldsmiths)
MAR 2023 r	1922.47	1596.09	326.38	0.99
AUG 2023	260.65	256.19	4.46	0.91
MAR 2024	463.03	428.78	34.25	0.93

In [9]:
```
#checking if the outliers are consistent across all months
len(df_compare_outliers.index.map(lambda x : x[:3]).unique())
```

Out[9]: 11

weight change and total error is not small, would be better to not decompose the 'Other Exports' class

In [10]:
```
#remove the constituents from the class map for 'Other Exports'
class_map["Other_Exports"] = np.nan
display(class_map)
class_map.to_csv("data/cleaned/MAP_exports.csv", index=False)
class_map.to_pickle("data/cleaned/MAP_exports.pkl")
```

	Agriculture	Fishery	Forestry	Mining
0	Rice	Crustaceans	NaN	Crude_oil
1	Rubber	Fish	NaN	Mineral_products,_n.i.e.
2	Durian	Cuttlefish,_squid,_octopus	NaN	NaN
3	Other_Fruits	Fishery_products,_n.i.e.	NaN	NaN
4	Horticultural_products,_n.i.e.	NaN	NaN	NaN
5	Animal_products	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN P
13	NaN	NaN	NaN	NaN
14	NaN	NaN	NaN	NaN
15	NaN	NaN	NaN	NaN
16	NaN	NaN	NaN	NaN

In [11]:
```
#manufacturing no changes
manufacturing_map.to_csv("data/cleaned/manufacturing_MAP_exports.csv", index
manufacturing_map.to_pickle("data/cleaned/manufacturing_MAP_exports.pkl")
```

In [12]:
```
df_total_export
```

Out[12]:

	class	JAN 1995	FEB 1995	MAR 1995	APR 1995	MAY 1995	JU
0	Agriculture	584.19	550.96	605.24	453.51	587.16	
1	Rice	196.61	186.2	175.64	107.99	191.15	
2	:_Quantity_(Metric_Ton)	724863.53	680984.39	593857.64	354495.85	655052.39	428
3	Rubber	229.04	216.29	238.01	181.01	216.77	
4	:_Quantity_(Metric_Ton)	178792.79	146901.62	148892.29	109139.54	131094.16	16
...	...	...	...	...	...	...	
91	Coverage_Adjustment	-67.51	-46.11	-105.91	-60.8	-91.42	
92	Classification_adjustment	0	0	0	0	0	
93	Valuation_Adjustment	0	0	0	0	0	
94	Timing_Adjustment	0	0	0	0	0	
95	Exports,_f.o.b._(BOP_basis)	3928	3949	5083	4005	4789	

96 rows × 352 columns

Convert and Standardise Month-Year to Datetime

In [13]:
```
#change months-year to datetime
display(df_total_export.columns)
df_total_export.set_index("class", inplace=True)
display(df_total_export.columns)
# change index to datetime
dates = df_total_export.columns

wrong_dates = []
for date in dates:
    try:
        pd.to_datetime(date, format='%b %Y ')
    except ValueError as e:
        wrong_dates.append(date)

display(wrong_dates)
```

```
Index(['class', 'JAN 1995 ', 'FEB 1995 ', 'MAR 1995 ', 'APR 1995 ',
       'MAY 1995 ', 'JUN 1995 ', 'JUL 1995 ', 'AUG 1995 ', 'SEP 1995 ',
       ...
       'JUN 2023 r', 'JUL 2023 ', 'AUG 2023 ', 'SEP 2023 ', 'OCT 2023 ',
       'NOV 2023 ', 'DEC 2023 ', 'JAN 2024 ', 'FEB 2024 ', 'MAR 2024 '],
      dtype='object', length=352)
```

```
Index(['JAN 1995 ', 'FEB 1995 ', 'MAR 1995 ', 'APR 1995 ', 'MAY 1995 ',
       'JUN 1995 ', 'JUL 1995 ', 'AUG 1995 ', 'SEP 1995 ', 'OCT 1995 ',
       ...
       'JUN 2023 r', 'JUL 2023 ', 'AUG 2023 ', 'SEP 2023 ', 'OCT 2023 ',
       'NOV 2023 ', 'DEC 2023 ', 'JAN 2024 ', 'FEB 2024 ', 'MAR 2024 '],
      dtype='object', length=351)
['JAN 2023 r',
 'FEB 2023 r',
 'MAR 2023 r',
 'APR 2023 r',
 'MAY 2023 r',
 'JUN 2023 r']
```

In [14]:
```
#fix the dates
fixed_dates = []
wrong_dates = []
for date in dates:
    try:
        pd.to_datetime(date, format='%b %Y ')
        fixed_dates.append(date)
    except ValueError as e:
        date = date.replace('r', '')
        fixed_dates.append(date)

for date in fixed_dates:
    try:
        pd.to_datetime(date, format='%b %Y ')
    except ValueError as e:
        wrong_dates.append(date)

display(wrong_dates)
```

```
[]
```

In [15]:
```
df_total_export.columns = pd.to_datetime(fixed_dates, format='%b %Y ')
display(df_total_export.columns)
display(df_total_export)
```

```
DatetimeIndex(['1995-01-01', '1995-02-01', '1995-03-01', '1995-04-01',
               '1995-05-01', '1995-06-01', '1995-07-01', '1995-08-01',
               '1995-09-01', '1995-10-01',
               ...
               '2023-06-01', '2023-07-01', '2023-08-01', '2023-09-01',
               '2023-10-01', '2023-11-01', '2023-12-01', '2024-01-01',
               '2024-02-01', '2024-03-01'],
              dtype='datetime64[ns]', length=351, freq=None)
```

class	1995-01-01	1995-02-01	1995-03-01	1995-04-01	1995-05-01	1995-0
Agriculture	584.19	550.96	605.24	453.51	587.16	516.4
Rice	196.61	186.2	175.64	107.99	191.15	130.9
:_Quantity_(Metric_Ton)	724863.53	680984.39	593857.64	354495.85	655052.39	428995.1
Rubber	229.04	216.29	238.01	181.01	216.77	250.7
:_Quantity_(Metric_Ton)	178792.79	146901.62	148892.29	109139.54	131094.16	160773
...	...	...	...	...	...	
Coverage_Adjustment	-67.51	-46.11	-105.91	-60.8	-91.42	-82.3
Classification_adjustment	0	0	0	0	0	
Valuation_Adjustment	0	0	0	0	0	
Timing_Adjustment	0	0	0	0	0	
Exports,_f.o.b._(BOP_basis)	3928	3949	5083	4005	4789	494

96 rows × 351 columns

In [16]:
```
#check that for every year 1995-2023, there are 12 months; and 2024 has up t
df_total_export.columns.map(lambda x : x.year).value_counts().sort_index() #
```

Out[16]:
```
1995    12
1996    12
1997    12
1998    12
1999    12
2000    12
2001    12
2002    12
2003    12
2004    12
2005    12
2006    12
2007    12
2008    12
2009    12
2010    12
2011    12
2012    12
2013    12
2014    12
2015    12
2016    12
2017    12
2018    12
2019    12
2020    12
2021    12
2022    12
2023    12
2024     3
Name: count, dtype: int64
```

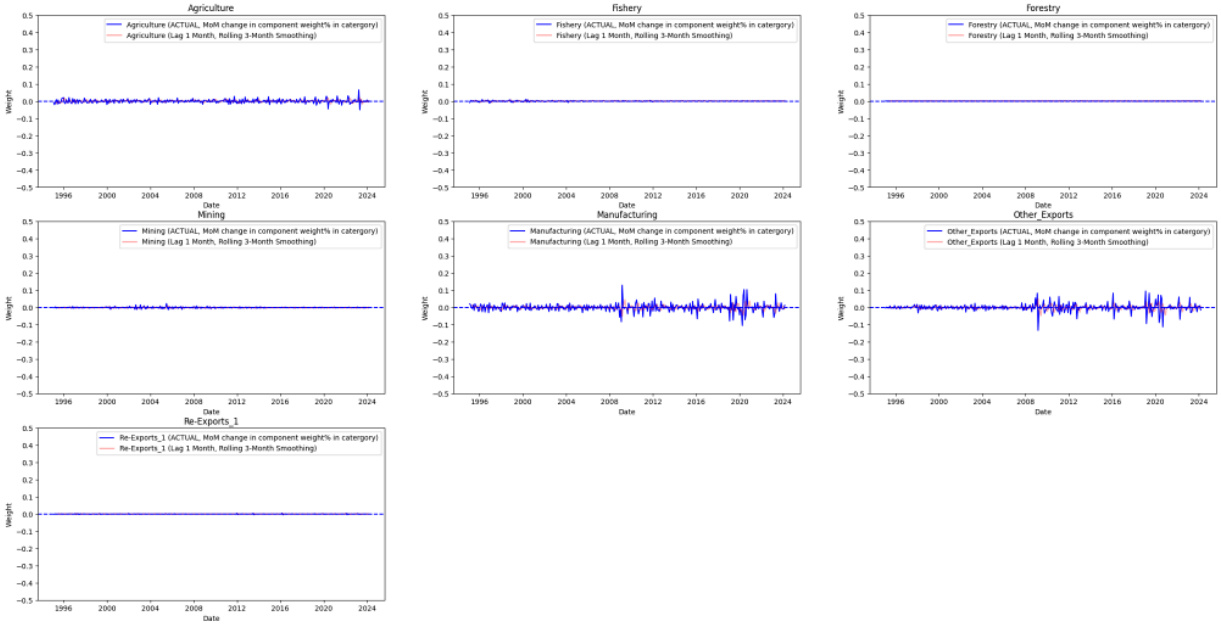
In [17]:
```
#save
df_total_export.to_csv("data/cleaned/total_export.csv")
df_total_export.to_pickle("data/cleaned/total_export.pkl")
```

Analyse weights month on month change

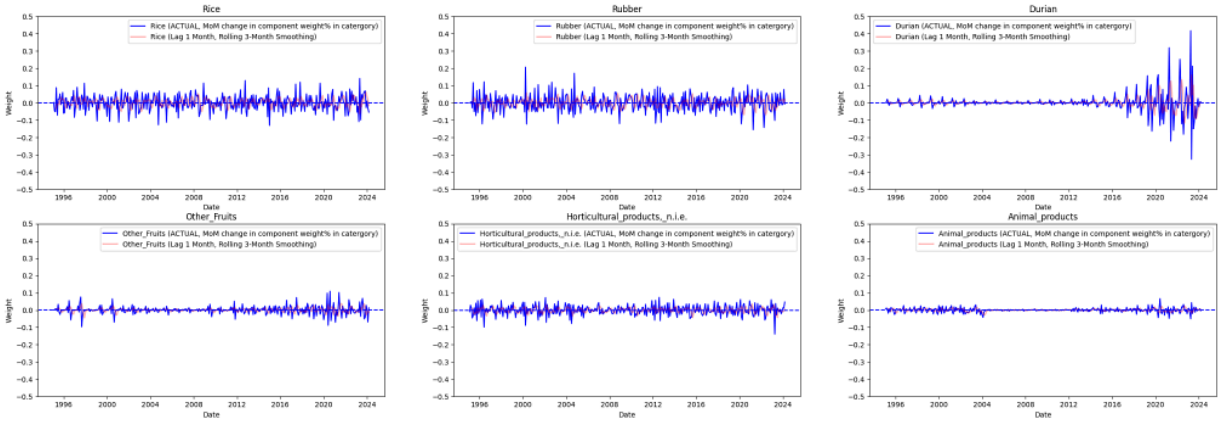
In [18]:
```
# For export map
def analyseComponentsWeight(df: pd.DataFrame,
                            df_export: pd.DataFrame) -> pd.DataFrame:
    cols = df.columns

    for col in cols:
        constituents = df[col].dropna()
        if len(constituents) == 0:
            # data_col = df_export[df_export.index == col].iloc[0]
            # data_col = data_col.T.pct_change(axis=0)
            continue

        #get col timeseries
        data_col = df_export[df_export.index == col].iloc[0, 1:]
```

```
        #get timeseries for constituents & sum rows
        data_constituents = df_export[df_export.index.isin(constituents)]
        data_constituents_sum = df_export[df_export.index.isin(constituents)

        #data_constituents weight
        data_constituents_weight = data_constituents.div(data_constituents_s
        data_constituents_weight = data_constituents_weight.T

        display(col)
        # display(data_constituents_weight)
        # Plot data
        ncols = 3
        nrows = len(data_constituents_weight.columns) // ncols + 1
        nplots = len(data_constituents_weight.columns)

        fig, ax = plt.subplots(figsize=(30, nrows * 5),
                               nrows=nrows, ncols=ncols)
        ax = ax.flatten()

        for i, col in enumerate(data_constituents_weight.columns):
            ax[i].plot(data_constituents_weight[col].diff(),
                       label=f"{col} (ACTUAL, MoM change in component weight

            ax[i].plot(data_constituents_weight[col].shift(1).rolling(windo
                       label=f"{col} (Lag 1 Month, Rolling 3-Month Smoothing
                       color='r')
            # ax[i].plot(data_constituents_weight[col].shift(1).ewm(span=2).
            #            label=f"{col} (Lag 1 Month, Rolling 3-Month Smoothi
            #            color='r')
            ax[i].set_xlabel("Date")
            ax[i].set_ylabel("Weight")
            #draw mean line
            ax[i].axhline(data_constituents_weight[col].diff().mean(), color
            ax[i].legend()
            ax[i].set_title(col)

            # Set y-axis limits
            ax[i].set_ylim(-0.5, 0.5)
            ax[i].set_yticks(list(np.arange(-0.5, 0.6, 0.1)))

        #hide remaining plots
        for i in range(nplots, nrows * ncols):
            ax[i].axis('off')

        plt.show()
analyseComponentsWeight(df_map_overall, df_total_export)
analyseComponentsWeight(class_map, df_total_export)
analyseComponentsWeight(manufacturing_map, df_total_export)
```

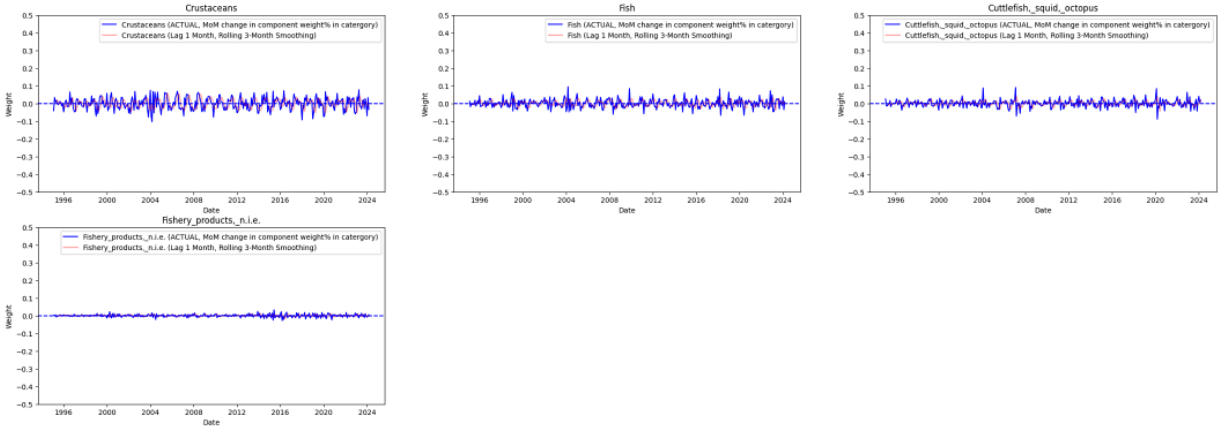
```
'Total_Exports_(Customs_basis)'
```



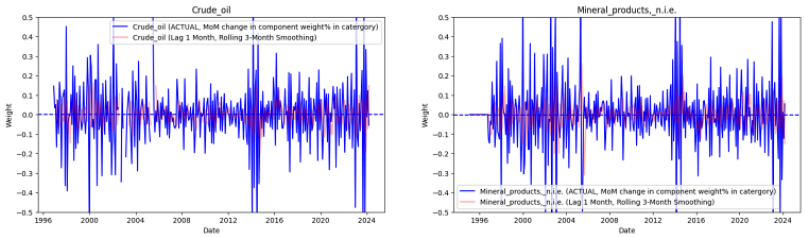
'Agriculture'



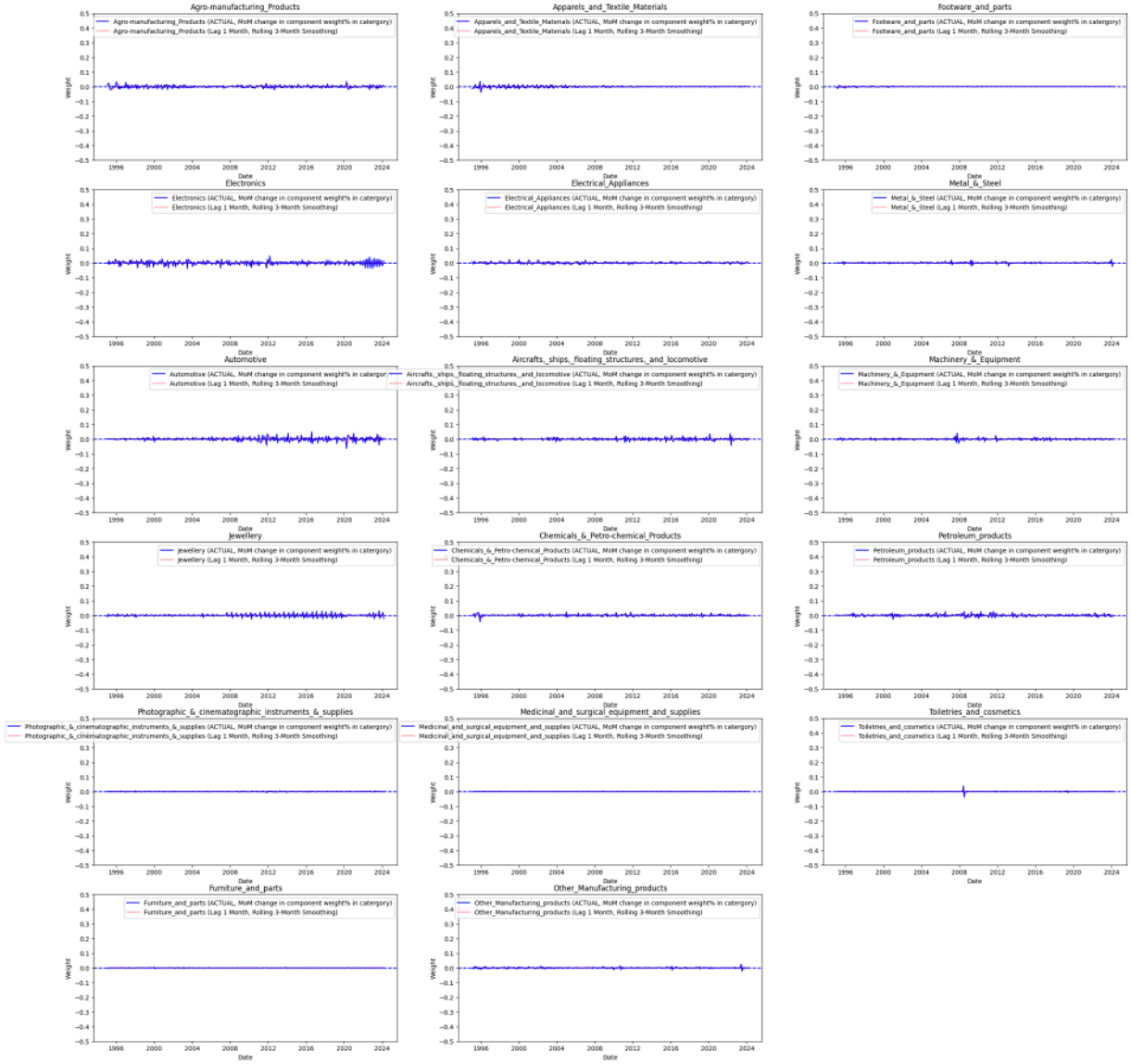
'Fishery'



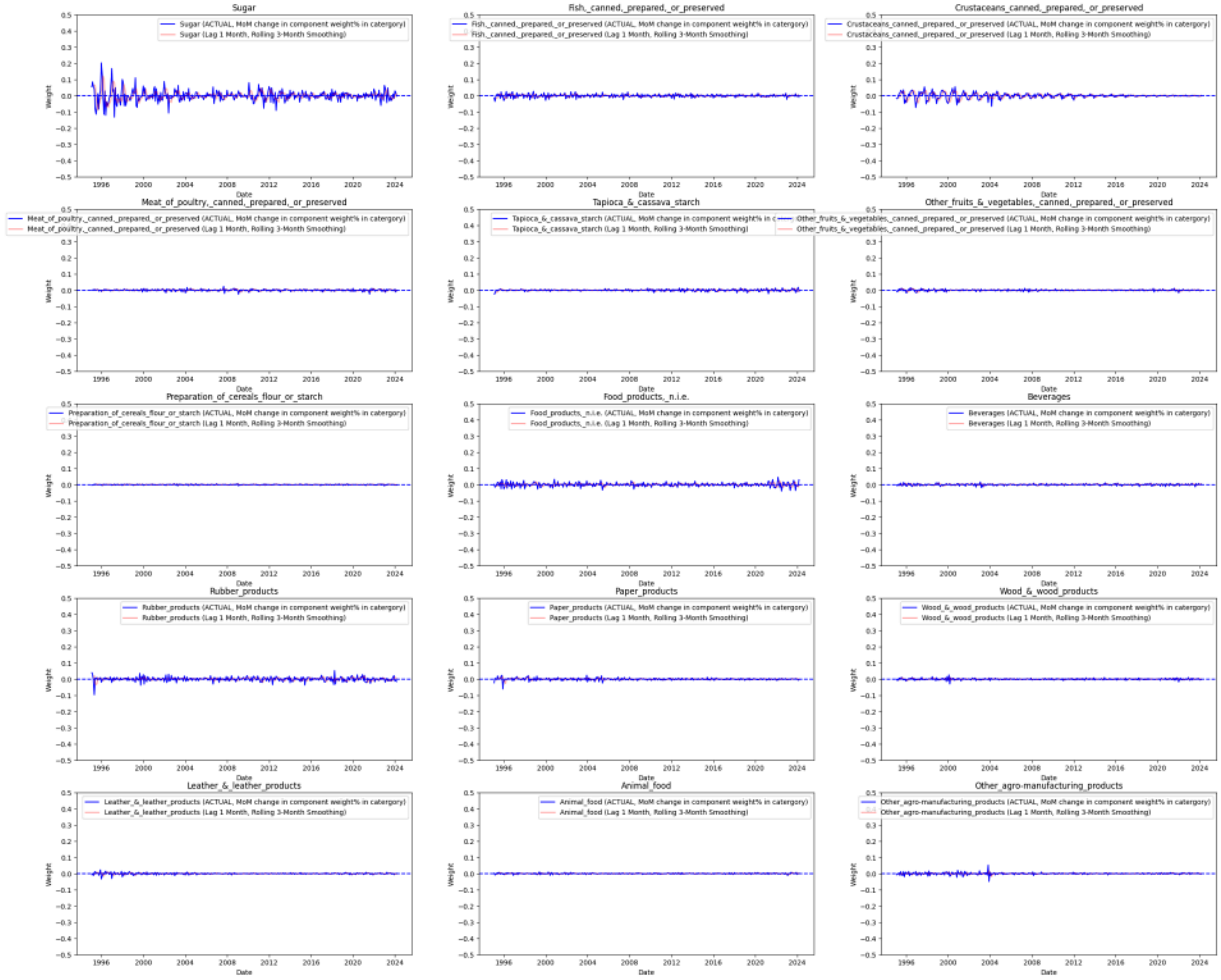
'Mining'



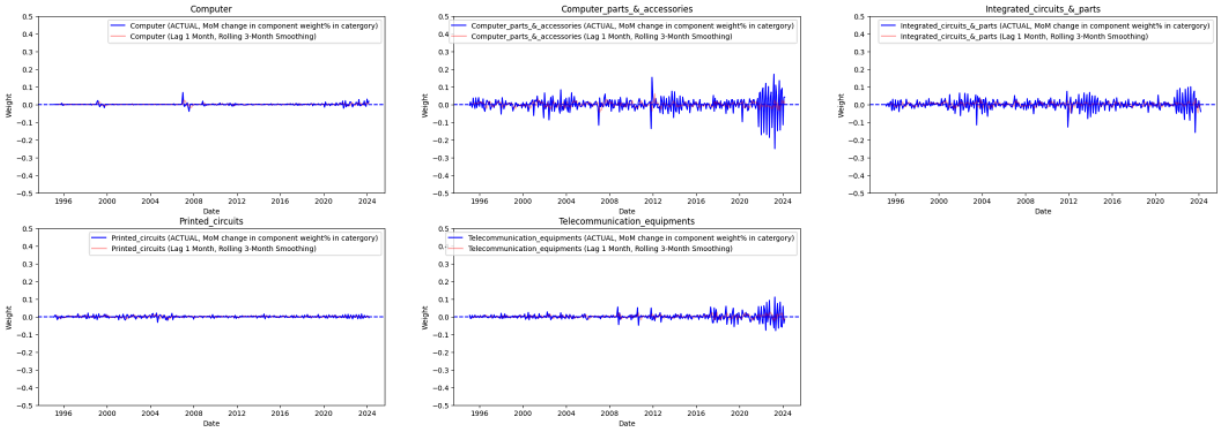
'Manufacturing'



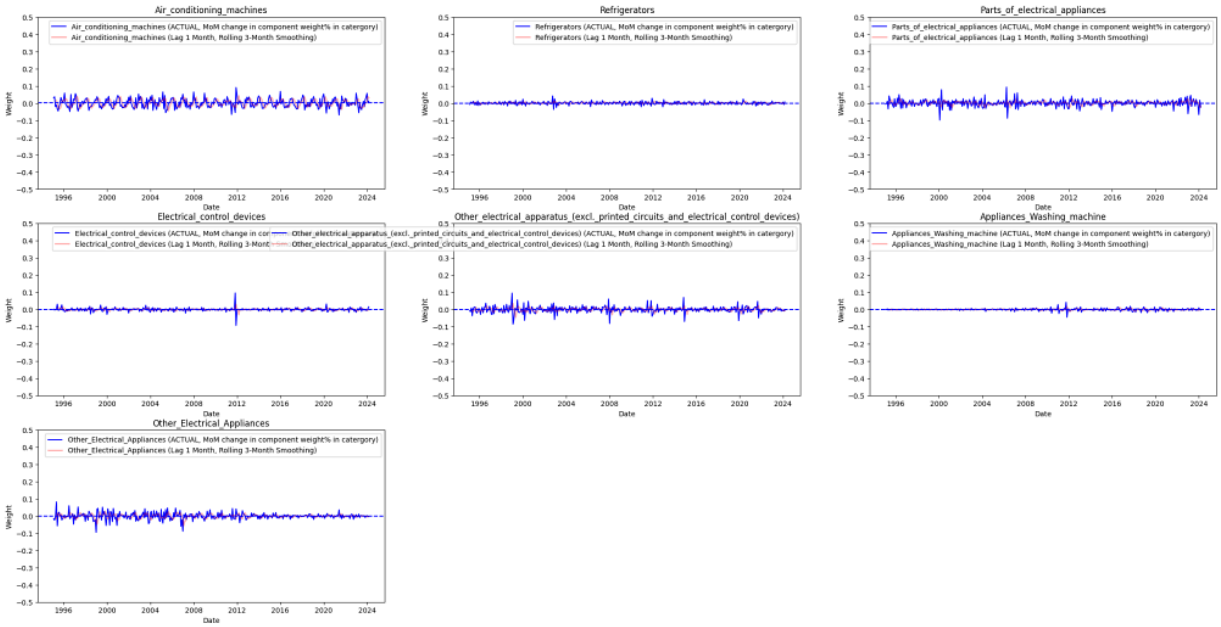
'Agro-manufacturing_Products'



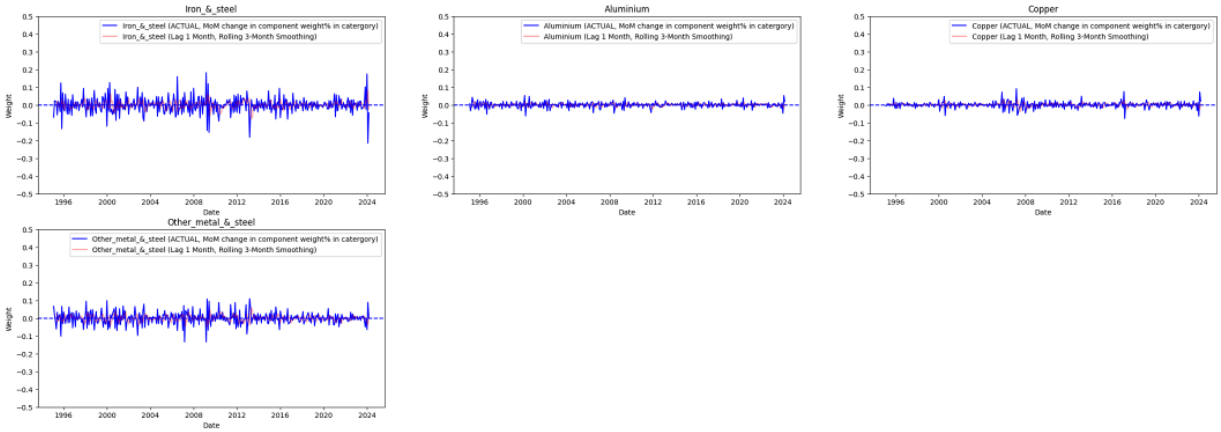
'Electronics'



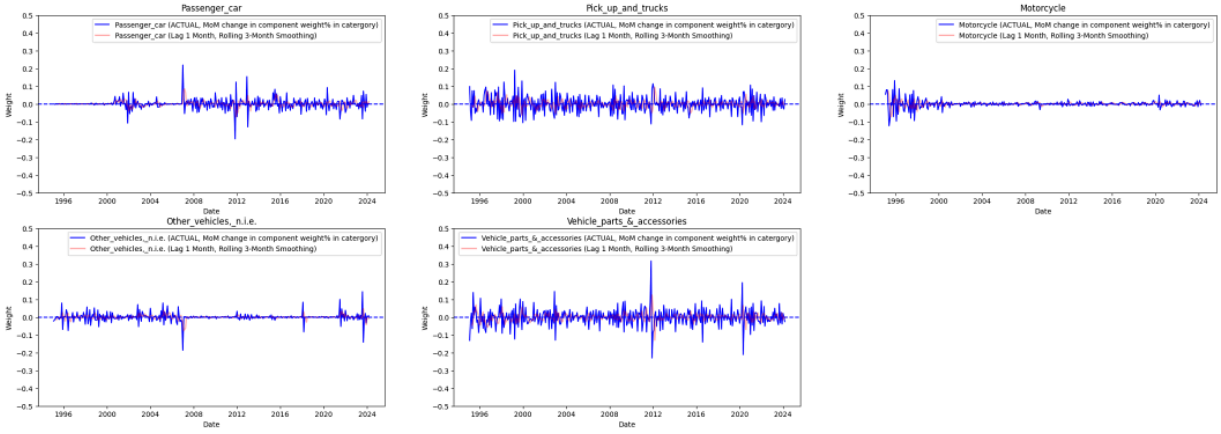
'Electrical_Appliances'



'Metal_&_Steel'



'Automotive'



Generally, the MoM weight change is not drastic and can be modelled using smoothening, however some components do have fluctuation which might benefit from further timeseries transformation like deseasonality

simplify df into the timeseries needed for analysis

In [19]:
```
class_map = pd.read_pickle("data/cleaned/MAP_exports.pkl")
manufacturing_map = pd.read_pickle("data/cleaned/manufacturing_MAP_exports.p
df_total_export = pd.read_pickle("data/cleaned/total_export.pkl")
display(class_map)
display(manufacturing_map)
display(df_total_export)
```

	Agriculture	Fishery	Forestry	Mining	
0	Rice	Crustaceans	NaN	Crude_oil	
1	Rubber	Fish	NaN	Mineral_products,_n.i.e.	
2	Durian	Cuttlefish,_squid,_octopus	NaN	NaN	
3	Other_Fruits	Fishery_products,_n.i.e.	NaN	NaN	
4	Horticultural_products,_n.i.e.	NaN	NaN	NaN	
5	Animal_products	NaN	NaN	NaN	
6	NaN	NaN	NaN	NaN	
7	NaN	NaN	NaN	NaN	
8	NaN	NaN	NaN	NaN	
9	NaN	NaN	NaN	NaN	
10	NaN	NaN	NaN	NaN	
11	NaN	NaN	NaN	NaN	
12	NaN	NaN	NaN	NaN	P
13	NaN	NaN	NaN	NaN	
14	NaN	NaN	NaN	NaN	
15	NaN	NaN	NaN	NaN	
16	NaN	NaN	NaN	NaN	

	Agro-manufacturing_Products	Apparels_and_Textile_Materials	Footwar
0	Sugar	NaN	
1	Fish,_canned,_prepared,_or_preserved	NaN	
2	Crustaceans_canned,_prepared,_or_preserved	NaN	
3	Meat_of_poultry,_canned,_prepared,_or_preserved	NaN	
4	Tapioca_&_cassava_starch	NaN	
5	Other_fruits_&_vegetables,_canned,_prepared,_o...	NaN	
6	Preparation_of_cereals_flour_or_starch	NaN	
7	Food_products,_n.i.e.	NaN	
8	Beverages	NaN	
9	Rubber_products	NaN	
10	Paper_products	NaN	
11	Wood_&_wood_products	NaN	
12	Leather_&_leather_products	NaN	
13	Animal_food	NaN	
14	Other_agro-manufacturing_products	NaN	

class	1995-01-01	1995-02-01	1995-03-01	1995-04-01	1995-05-01	1995-0
Agriculture	584.19	550.96	605.24	453.51	587.16	516.4
Rice	196.61	186.2	175.64	107.99	191.15	130.9
:_Quantity_(Metric_Ton)	724863.53	680984.39	593857.64	354495.85	655052.39	428995.1
Rubber	229.04	216.29	238.01	181.01	216.77	250.7
:_Quantity_(Metric_Ton)	178792.79	146901.62	148892.29	109139.54	131094.16	160773
...	...	...	...	...	...	
Coverage_Adjustment	-67.51	-46.11	-105.91	-60.8	-91.42	-82.3
Classification_adjustment	0	0	0	0	0	
Valuation_Adjustment	0	0	0	0	0	
Timing_Adjustment	0	0	0	0	0	
Exports,_f.o.b._(BOP_basis)	3928	3949	5083	4005	4789	494

96 rows × 351 columns

In [20]:
```
#identify important mandatory adjustments
for col in ["Adjustment_for_balance_of_payments_2/", "Coverage_Adjustment",
            "Classification_adjustment", "Valuation_Adjustment", "Timing_Adj
    total = df_total_export[df_total_export.index == col].sum(axis=1)
    display(total)
```

```
Series([], dtype: object)
class
Coverage_Adjustment   -56515.14
dtype: object
class
Classification_adjustment    0
dtype: object
class
Valuation_Adjustment    0
dtype: object
class
Timing_Adjustment    269.01
dtype: object
```

Coverage_Adjustment & Timing_Adjustment must be considered for adjustedTotal

In [21]:
```
df_map_ANALYSIS = pd.concat([class_map, manufacturing_map], axis=1)
df_map_ANALYSIS.drop(columns=["Manufacturing"], inplace=True)
df_map_ANALYSIS.to_csv("data/cleaned/MAP_exports_ANALYSIS.csv", index=False)
df_map_ANALYSIS.to_pickle("data/cleaned/MAP_exports_ANALYSIS.pkl")

cols = list(df_map_ANALYSIS.columns)

df = []
for col in cols:
    constituents = df_map_ANALYSIS[col].dropna()
    if len(constituents) == 0:
        #if its a single class with no constituents, thats for analysis
        #get col timeseries
        df.append(df_total_export[df_total_export.index == col])
        continue
    #otherwise all constituents are for analysis
    #get timeseries for constituents & sum rows
    df.append(df_total_export[df_total_export.index.isin(constituents)])

#mandatory additional classes
for col in [ "Coverage_Adjustment", "Timing_Adjustment",
            "Total_Exports_(Customs_basis)", "Exports,_f.o.b._(BOP_basis)"]:
    df.append(df_total_export[df_total_export.index == col])

df = pd.concat(df, axis=0)
df


#this df is the one to be used for analysis
df_export_ANALYSIS = df.copy(deep=True).T
#drop dupliated columns
df_export_ANALYSIS = df_export_ANALYSIS.loc[:,~df_export_ANALYSIS.columns.du
```

```
df_export_ANALYSIS.to_csv("data/cleaned/total_export_ANALYSIS.csv")
df_export_ANALYSIS.to_pickle("data/cleaned/total_export_ANALYSIS.pkl")
```

check the rough trend across the components Export Value

In [22]:
```
df_export_ANALYSIS = pd.read_pickle("data/cleaned/total_export_ANALYSIS.pkl"
df_map_ANALYSIS = pd.read_pickle("data/cleaned/MAP_exports_ANALYSIS.pkl")
display(df_export_ANALYSIS)
display(df_map_ANALYSIS)
```

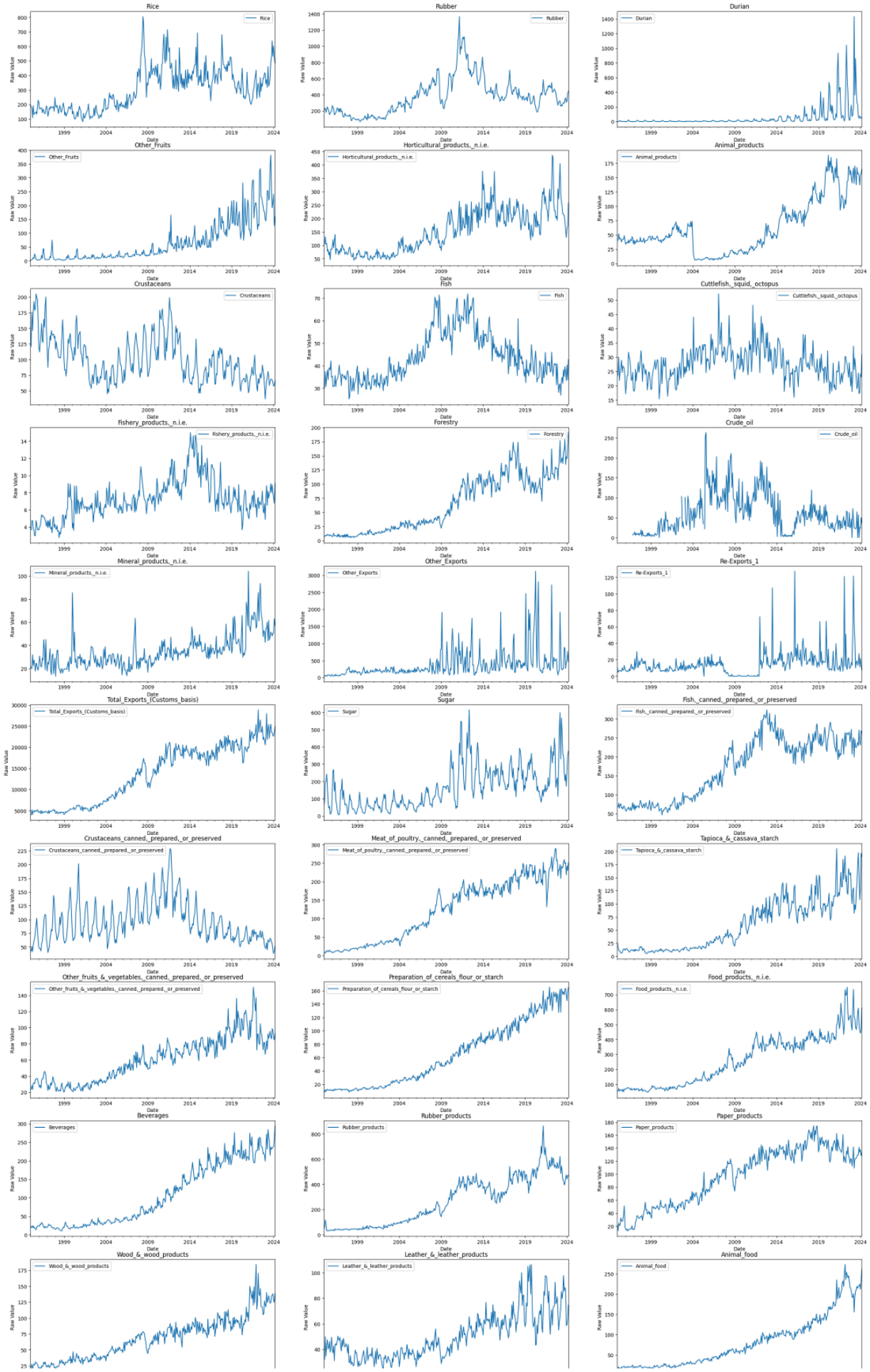
class	Rice	Rubber	Durian	Other_Fruits	Horticultural_products,_n.i.e.	Animal_products
1995-01-01	196.61	229.04	0.54	1.97	116.57	39.46
1995-02-01	186.2	216.29	0.58	2.6	104.63	40.65
1995-03-01	175.64	238.01	3.29	4.13	132.28	51.9
1995-04-01	107.99	181.01	11.73	6.4	103.94	42.44
1995-05-01	191.15	216.77	15.06	12.38	107.54	44.26
...	...	...	...	...	...	...
2023-11-01	637.49	361.63	48.69	191.81	159.33	153.77
2023-12-01	532.27	306.66	77.54	210.95	129.23	137.43
2024-01-01	602.49	326.87	48.63	240.71	163.27	154.76
2024-02-01	523.91	419.1	50.46	127.29	169.21	156.94
2024-03-01	484.76	444.62	64.81	159.33	258.22	165.14

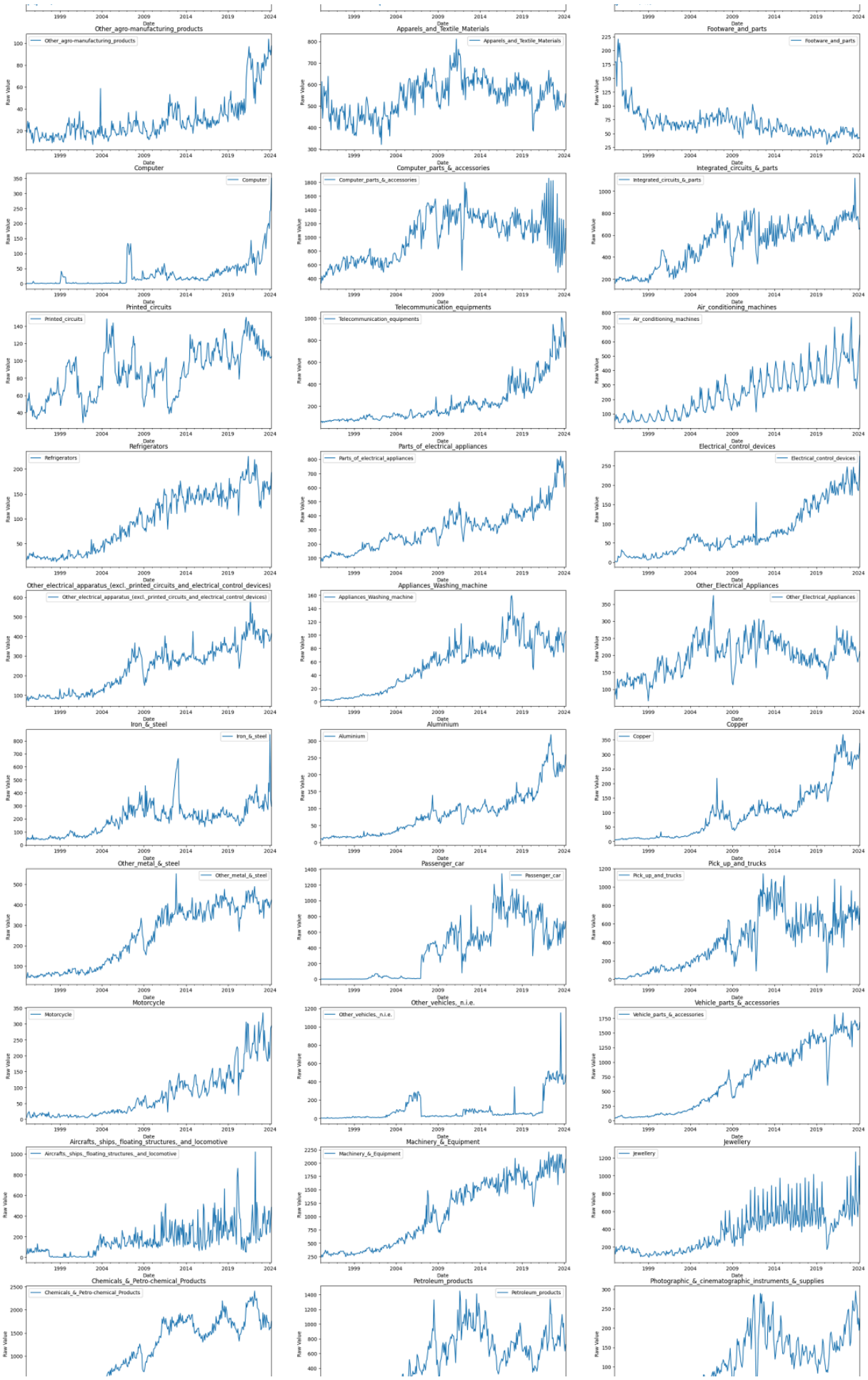
351 rows × 67 columns

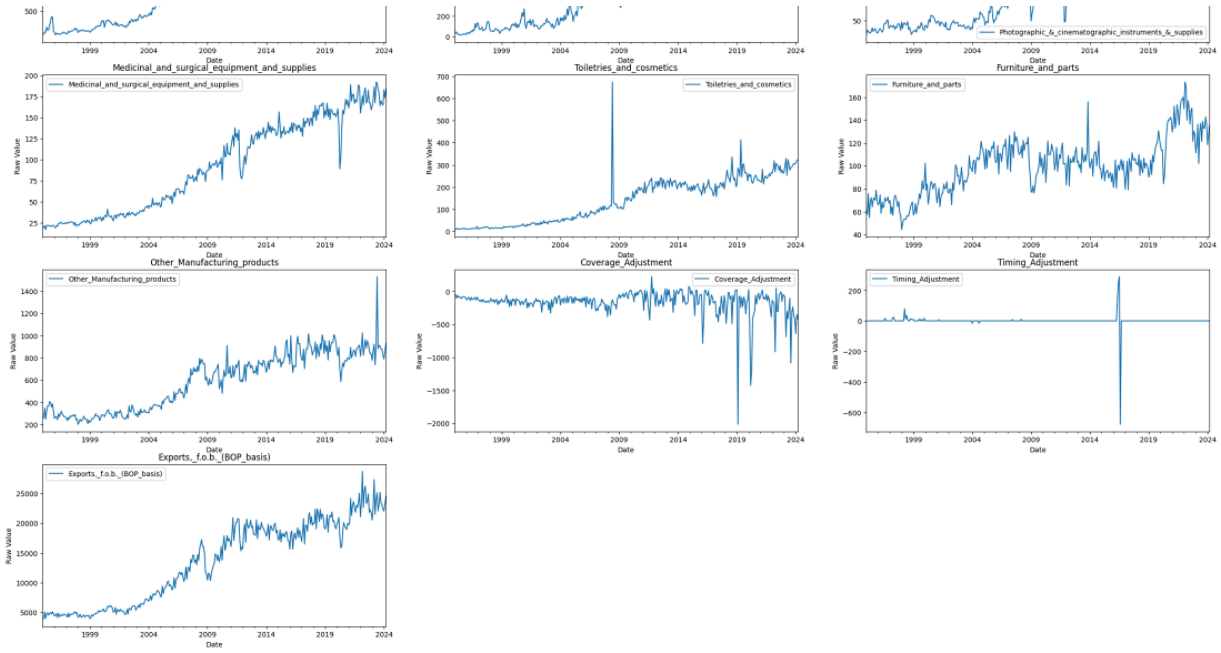
	Agriculture	Fishery	Forestry	Mining	O
0	Rice	Crustaceans	NaN	Crude_oil	
1	Rubber	Fish	NaN	Mineral_products,_n.i.e.	
2	Durian	Cuttlefish,_squid,_octopus	NaN	NaN	
3	Other_Fruits	Fishery_products,_n.i.e.	NaN	NaN	
4	Horticultural_products,_n.i.e.	NaN	NaN	NaN	
5	Animal_products	NaN	NaN	NaN	
6	NaN	NaN	NaN	NaN	
7	NaN	NaN	NaN	NaN	
8	NaN	NaN	NaN	NaN	
9	NaN	NaN	NaN	NaN	
10	NaN	NaN	NaN	NaN	
11	NaN	NaN	NaN	NaN	
12	NaN	NaN	NaN	NaN	
13	NaN	NaN	NaN	NaN	
14	NaN	NaN	NaN	NaN	
15	NaN	NaN	NaN	NaN	
16	NaN	NaN	NaN	NaN	

17 rows × 24 columns

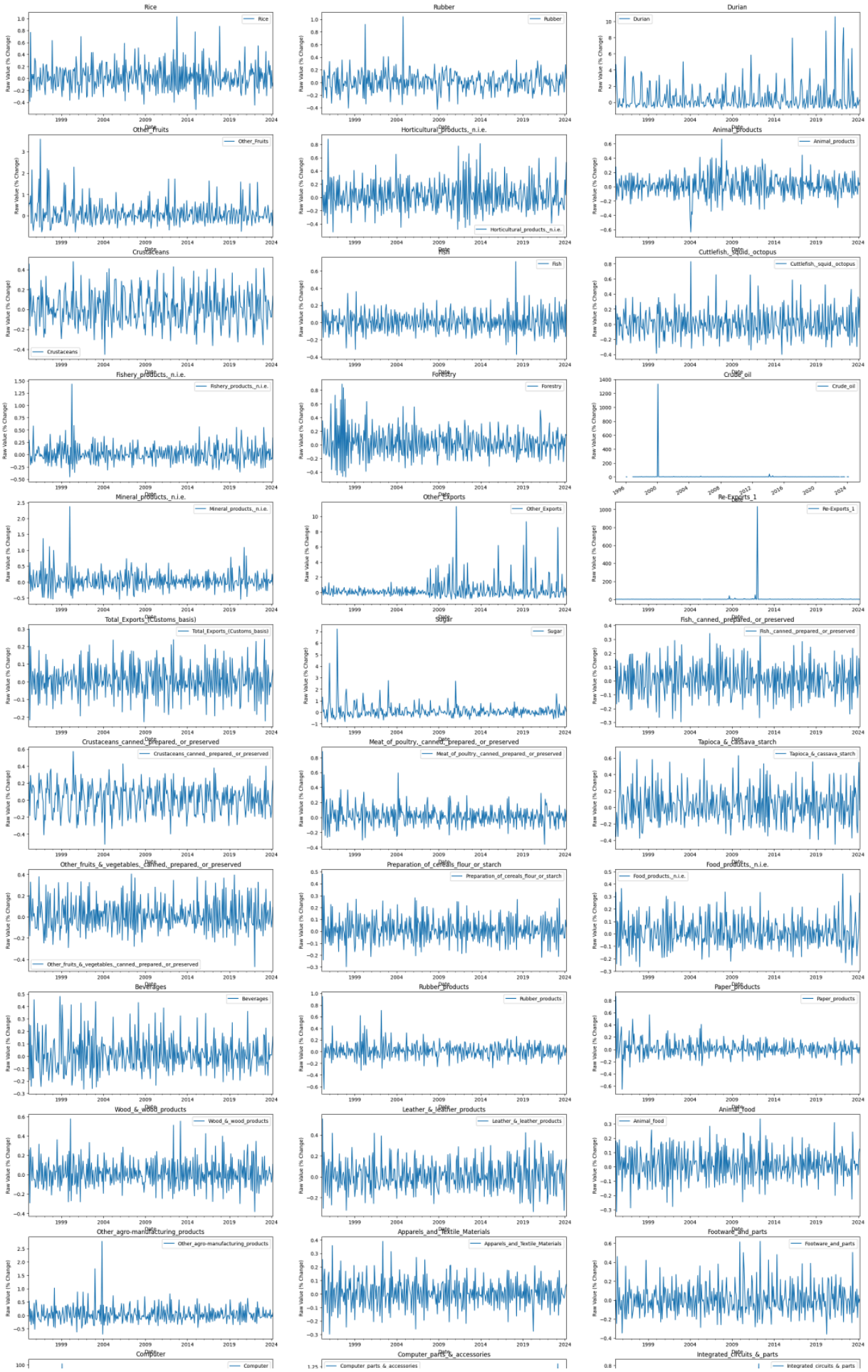
In [23]:
```
#plot all components timeseries trend (Raw Value)
num_columns = 3
num_rows = (len(df_export_ANALYSIS.columns) + num_columns - 1) // num_columr
fig, axes = plt.subplots(nrows=num_rows, ncols=num_columns, figsize=(30, 5*r
axes = axes.flatten()

for i, column in enumerate(df_export_ANALYSIS.columns):
    df_export_ANALYSIS[column].plot(ax=axes[i], title=column, legend=True)
    axes[i].set_xlabel('Date')
    axes[i].set_ylabel('Raw Value')

# Hide empty subplots
for ax in axes[len(df_export_ANALYSIS.columns):]:
    ax.axis('off')


plt.show()
```

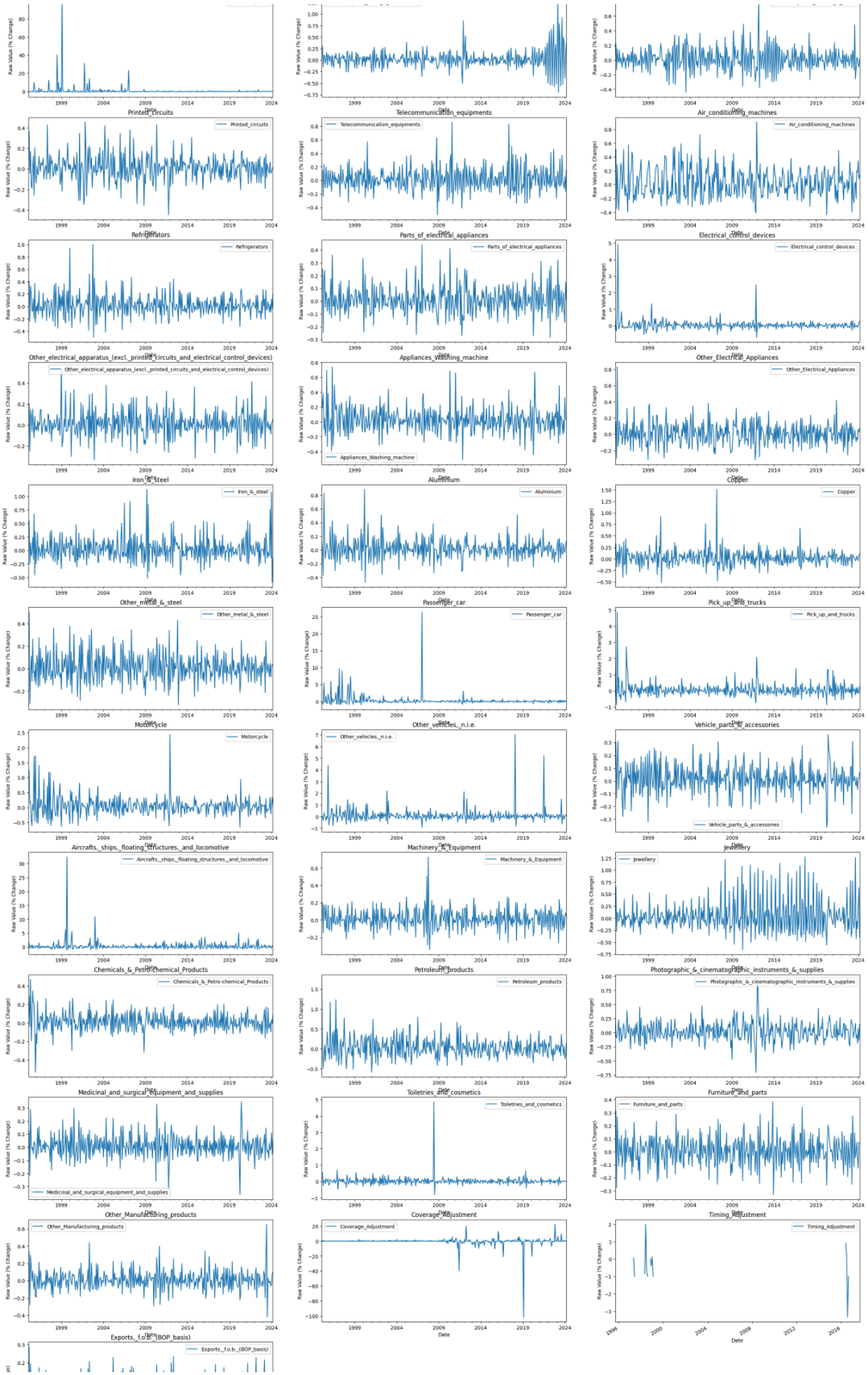

Rice
Rubber
Durian
Other_Fruits
Horticultural_products,_n.i.e.
Animal_products
Crustaceans
Fish
Cuttlefish,_squid,_octopus
Fishery_products,_n.i.e.
Forestry
Crude_oil
Mineral_products,_n.i.e.
Other_Exports
Re-Exports_1
Total_Exports_(Customs_basis)
Sugar
Fish,_canned,_prepared,_or_preserved
Crustaceans,_canned,_prepared,_or_preserved
Meat_of_poultry,_canned,_prepared,_or_preserved
Tapioca_&_cassava_starch
Other_fruits_&_vegetables,_canned,_prepared,_or_preserved
Preparation_of_cereals_flour_or_starch
Food_products,_n.i.e.
Beverages
Rubber_products
Paper_products
Wood_&_wood_products
Leather_&_leather_products
Animal_food
Raw Value
Date

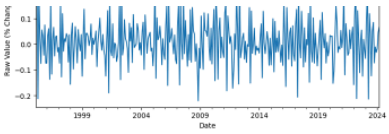

Other_agro-manufacturing_products
Apparels_and_Textile_Materials
Footware_and_parts
Computer
Computer_parts_&_accessories
Integrated_circuits_&_parts
Printed_circuits
Telecommunication_equipments
Air_conditioning_machines
Refrigerators
Parts_of_electrical_appliances
Electrical_control_devices
Other_electrical_apparatus_(excl._printed_circuits_and_electrical_control_devices)
Appliances_Washing_machine
Other_Electrical_Appliances
Iron_&_steel
Aluminium
Copper
Other_metal_&_steel
Passenger_car
Pick_up_and_trucks
Motorcycle
Other_vehicles,_n.i.e.
Vehicle_parts_&_accessories
Aircrafts,_ships,_floating_structures,_and_locomotive
Machinery_&_Equipment
Jewellery
Chemicals_&_Petro-chemical_Products
Petroleum_products
Photographic_&_cinematographic_instruments_&_supplies
Raw Value
Date



In [24]:
```
#plot all components timeseries trend (Raw Value % Change)
num_columns = 3
num_rows = (len(df_export_ANALYSIS.columns) + num_columns - 1) // num_columr
fig, axes = plt.subplots(nrows=num_rows, ncols=num_columns, figsize=(30, 5*r
axes = axes.flatten()

for i, column in enumerate(df_export_ANALYSIS.columns):
    df_export_ANALYSIS[column].pct_change().dropna().plot(ax=axes[i], title=
    axes[i].set_xlabel('Date')
    axes[i].set_ylabel('Raw Value (% Change)')

# Hide empty subplots
for ax in axes[len(df_export_ANALYSIS.columns):]:
    ax.axis('off')


plt.show()
```


Rice
Rubber
Durian
Other_Fruits
Horticultural_products,_n.i.e.
Animal_products
Crustaceans
Fish
Cuttlefish,_squid,_octopus
Fishery_products,_n.i.e.
Forestry
Crude_oil
Mineral_products,_n.i.e.
Other_Exports
Re-Exports_1
Total_Exports_(Customs_basis)
Sugar
Fish,_canned,_prepared,_or_preserved
Crustaceans,_canned,_prepared,_or_preserved
Meat_of_poultry,_canned,_prepared,_or_preserved
Tapioca_&_cassava_starch
Other_fruits_&_vegetables,_canned,_prepared,_or_preserved
Preparation_of_cereals_flour_or_starch
Food_products,_n.i.e.
Beverages
Rubber_products
Paper_products
Wood_&_wood_products
Leather_&_leather_products
Animal_food
Other_agro-manufacturing_products
Apparels_and_Textile_Materials
Footware_and_parts
Computer
Computer_parts_&_accessories
Integrated_circuits_&_parts
Raw Value (% Change)
Date


Printed_circuits
Telecommunication_equipments
Air_conditioning_machines
Refrigerators
Parts_of_electrical_appliances
Electrical_control_devices
Other_electrical_apparatus_(excl._printed_circuits_and_electrical_control_devices)
Appliances_Washing_machine
Other_Electrical_Appliances
Iron_&_steel
Aluminium
Copper
Other_metal_&_steel
Passenger_car
Pick_up_and_trucks
Motorcycle
Other_vehicles,_n.i.e.
Vehicle_parts_&_accessories
Aircrafts,_ships,_floating_structures,_and_locomotive
Machinery_&_Equipment
Jewellery
Chemicals_&_Petro-chemical_Products
Petroleum_products
Photographic_&_cinematographic_instruments_&_supplies
Medicinal_and_surgical_equipment_and_supplies
Toiletries_and_cosmetics
Furniture_and_parts
Other_Manufacturing_products
Coverage_Adjustment
Timing_Adjustment
Exports,_f.o.b._(BOP_basis)
Raw Value (% Change)
Date



Upcoming Notebook `1_Composote_export_value.ipynb`: check statistical properties of timeseries & Out of Sample forecasting of export value for individual components

Upcoming Notebook `2_composite_weight_analysis.ipynb`: check if weights change drastically across months and Out of Sample forecast of next month(t) using last release(t-1,...,t-ar_lag)

Upcoming Notebook `3_stitching_series.ipynb`: Restitching the Components into the Total Export Series(BoP) with Error Optimisation