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
```

In [2]:
```
df_export_ANALYSIS = pd.read_pickle("data/cleaned/total_export_FirstAnalysis
display(df_export_ANALYSIS)
```

class	Agriculture	Rice	Rubber	Durian	Other_Fruits	Horticultural_products,_n.i.e.	Anim
1995-01-01	584.19	196.61	229.04	0.54	1.97	116.57	
1995-02-01	550.96	186.2	216.29	0.58	2.6	104.63	
1995-03-01	605.24	175.64	238.01	3.29	4.13	132.28	
1995-04-01	453.51	107.99	181.01	11.73	6.4	103.94	
1995-05-01	587.16	191.15	216.77	15.06	12.38	107.54	
...	...	...	...	...	...	...	
2023-11-01	1552.72	637.49	361.63	48.69	191.81	159.33	
2023-12-01	1394.08	532.27	306.66	77.54	210.95	129.23	
2024-01-01	1536.73	602.49	326.87	48.63	240.71	163.27	
2024-02-01	1446.91	523.91	419.1	50.46	127.29	169.21	
2024-03-01	1576.87	484.76	444.62	64.81	159.33	258.22	

351 rows × 40 columns

In [3]:
```
mapper = pd.read_pickle("data/cleaned/MAP_exports.pkl")
mapper
```

Out[3]:

	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
13	NaN	NaN	NaN	NaN
14	NaN	NaN	NaN	NaN
15	NaN	NaN	NaN	NaN
16	NaN	NaN	NaN	NaN

In [4]:
```
# mapper["Total_Exports_(Customs_basis)"] = mapper.columns[:-1].tolist() + [
# mapper.to_pickle("data/cleaned/MAP_exports_FirstAnalysis.pkl")
# mapper.to_csv("data/cleaned/MAP_exports_FirstAnalysis.csv")

mapper = pd.read_pickle("data/cleaned/MAP_exports_FirstAnalysis.pkl")
```

In [5]:
```
mapper
```

Out[5]:

	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
13	NaN	NaN	NaN	NaN
14	NaN	NaN	NaN	NaN
15	NaN	NaN	NaN	NaN
16	NaN	NaN	NaN	NaN

Total Exports Weightage

On rebuilding back the inverse CLR & reseasonalising, sharp spike due to mathematical conversion results in hige spike for larger than normal forecast, creating exaggerated extrapolation error. This is shown below. A percentile cap is used to regulate jumps in the future which shows and increase of R^2 significantly for the regression model.

Will require X-13 PATH to run

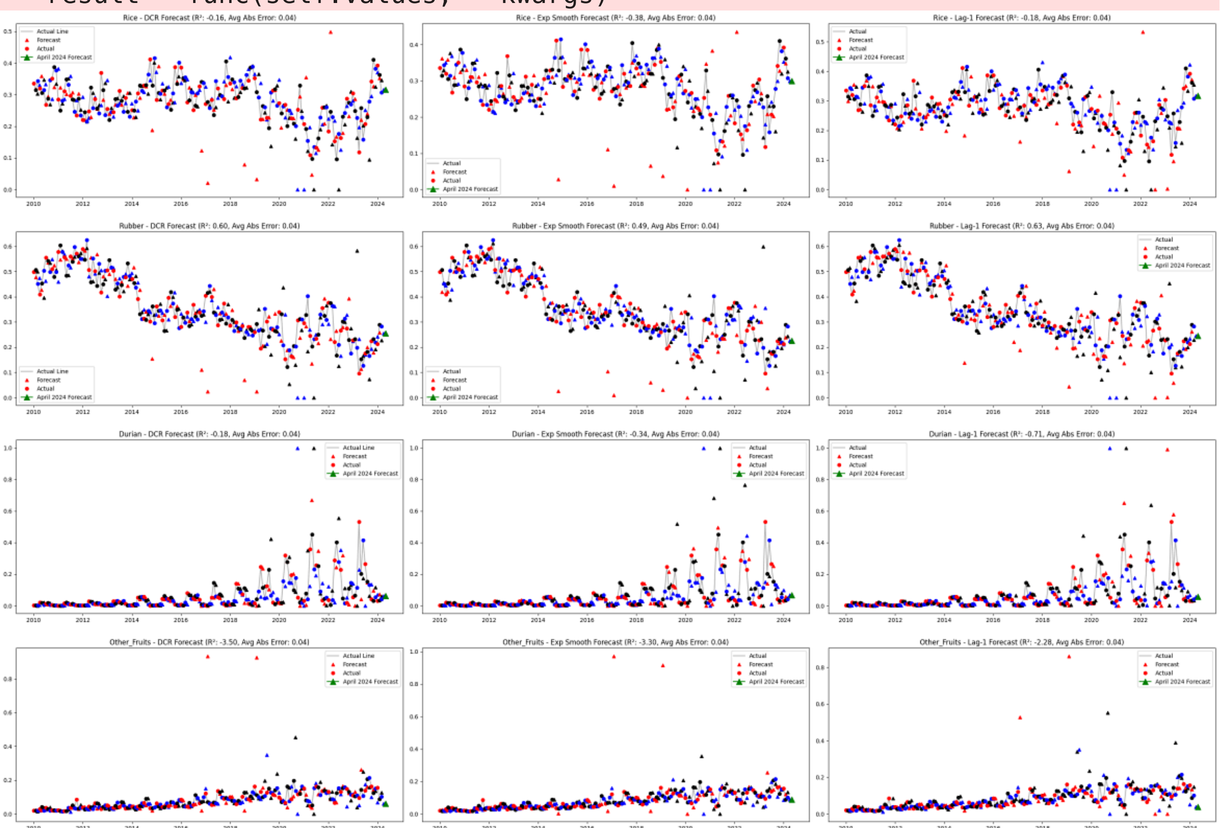
In [7]:
```
# dict_R2 = {}
# dict_R2_OLD_VERSION_NO_CAP_PLOT = {}

# # Suppress specific X13 warnings
# with pd.option_context('display.max_rows', 10, 'display.max_columns', None
#     warnings.filterwarnings("ignore", category=X13Warning)

#     with open("data/cleaned/COMBINED_deseasonalised_value_dict.pkl", "rb")
#         dict_deseasonalized_value = pickle.load(f)

#     for name in tqdm(mapper.columns, desc="Walk-Forward Forecasting", dyna
#         if name != "Mining":
#             temp_mapper = mapper[name].dropna().tolist()
#             if temp_mapper:
#                 df = df_export_ANALYSIS[temp_mapper]

#                 # Ensure data is numeric
#                 df = df.apply(pd.to_numeric, errors='coerce')

#                 # Normalize the data to weights
#                 constituent_total = df.sum(axis=1)
#                 df_actual = df.divide(constituent_total, axis=0)

#                 # CLR transform
#                 df = clr_transformation(df_actual.copy(deep=True))

#                 # X13 deseasonalize
#                 deseasonalized_df = pd.DataFrame()
#                 seasonal_factors_df = pd.DataFrame()

#                 for col in df.columns:
#                     df[col] = df[col].replace([np.inf, -np.inf], np.nan).r
#                     seasadj, trend, seasonal, irregular, seasonal_factors
#                     deseasonalized_df[col] = seasadj
#                     seasonal_factors_df[col] = seasonal_factors

#                 old_deseasonalized_df = deseasonalized_df.copy(deep=True)

#                 # Walk-forward parameters
#                 window_size = 180  # 15 years of monthly data
#                 total_observations = len(deseasonalized_df)  # Total data
#                 steps = total_observations - window_size  # Remaining poin

#                 # Generate periods DataFrame
#                 periods_df = generate_periods_df(deseasonalized_df, window

#                 # Perform walk-forward forecasting
#                 rolling_forecasts_dcr = walk_forward_forecasting_dcr(desea
#                 forecast_period = rolling_forecasts_dcr.index

#                 # Additional Forecasting Models (SImple Benchmarks)
```

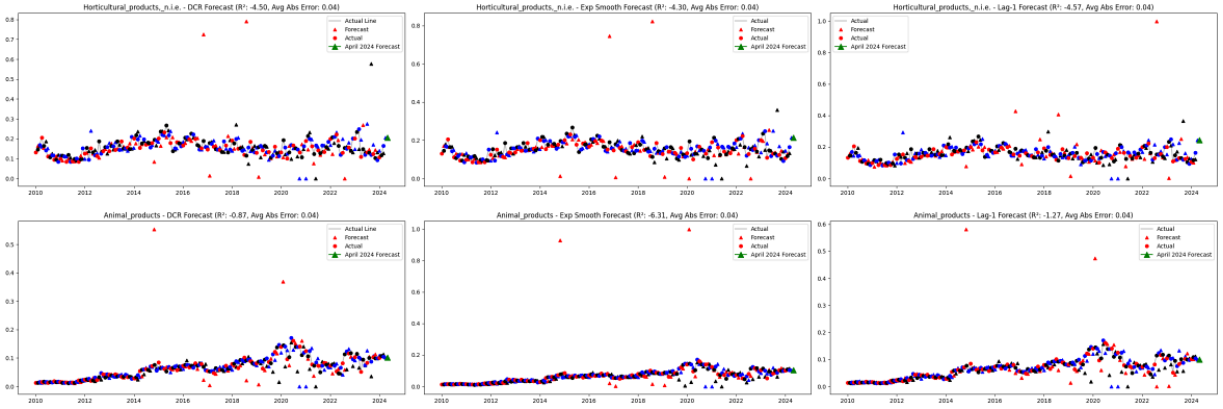
```
#                 deseasonalized_df = pd.concat([deseasonalized_df, pd.DataF
#                 rolling_forecasts_exp_smooth = deseasonalized_df.apply(exp
#                 rolling_forecasts_exp_smooth = rolling_forecasts_exp_smoot
#                 rolling_forecasts_lag_1 = deseasonalized_df.apply(lag_1_mo
#                 rolling_forecasts_lag_1 = rolling_forecasts_lag_1[rolling_

#                 # Reseasonalize the forecasts
#                 extended_seasonal_factors = seasonal_factors_df[seasonal_f
#                 reseasonalized_forecast_dcr = rolling_forecasts_dcr.mul(ex
#                 reseasonalized_forecast_exp_smooth = rolling_forecasts_exp
#                 reseasonalized_forecast_lag_1 = rolling_forecasts_lag_1.mu

#                 # # Cap extreme values before applying inverse CLR transfo
#                 # reseasonalized_forecast_dcr = cap_extreme_values(reseaso
#                 # reseasonalized_forecast_exp_smooth = cap_extreme_values(
#                 # reseasonalized_forecast_lag_1 = cap_extreme_values(resea

#                 # Apply inverse CLR transformation
#                 forecast_df_original_dcr = inverse_clr_transformation(rese
#                 forecast_df_original_exp_smooth = inverse_clr_transformati
#                 forecast_df_original_lag_1 = inverse_clr_transformation(re

#                 # # Cap extreme values before applying inverse CLR transfo
#                 # forecast_df_original_dcr = cap_extreme_values(forecast_d
#                 # forecast_df_original_exp_smooth = cap_extreme_values(for
#                 # forecast_df_original_lag_1 = cap_extreme_values(forecast
#                 forecast_df_original_dcr = forecast_df_original_dcr.ffill(
#                 forecast_df_original_exp_smooth = forecast_df_original_exp
#                 forecast_df_original_lag_1 = forecast_df_original_lag_1.ff
#                 # #save weights as pkl
#                 # forecast_df_original_dcr.to_pickle(f"data/cleaned/foreca
#                 # forecast_df_original_exp_smooth.to_pickle(f"data/cleaned
#                 # forecast_df_original_lag_1.to_pickle(f"data/cleaned/fore


#                 # Evaluate forecasts
#                 actual_df = df_actual[old_deseasonalized_df.index.isin(for

#                 r2_scores_dcr = evaluate_forecasts(actual_df, forecast_df_
#                 r2_scores_exp_smooth = evaluate_forecasts(actual_df, forec
#                 r2_scores_lag_1 = evaluate_forecasts(actual_df, forecast_d

#                 # Calculate average absolute error
#                 avg_abs_error_dcr = mean_absolute_error(actual_df, forecas
#                 avg_abs_error_exp_smooth = mean_absolute_error(actual_df,
#                 avg_abs_error_lag_1 = mean_absolute_error(actual_df, forec

#                 # Store R² scores in DataFrame
#                 dftemp = pd.DataFrame([r2_scores_dcr, r2_scores_exp_smooth
#                 dict_R2[name] = dftemp

#                 # Visualization
#                 colors = ['red', 'black', 'blue'] # alternate points

#                 for col in actual_df.columns:
#                     fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(30,
```

```
#                     # Plot DCR forecast with alternating colors for marker
#                     ax[0].plot(actual_df[col], label="Actual Line", color=
#                     for i in range(len(actual_df[col])):
#                         ax[0].plot(forecast_df_original_dcr.index[i], fore
#                                    marker='^', alpha=1, color=colors[i % l
#                         ax[0].plot(actual_df.index[i], actual_df[col].iloc
#                                    marker='o', alpha=1, color=colors[i % l
#                     # Highlight the last point in green with alpha=1
#                     ax[0].plot(forecast_df_original_dcr[col].index[-1], fo
#                                marker='^', alpha=1, color='green', markers
#                     ax[0].legend()
#                     ax[0].set_title(f"{col} - DCR Forecast (R²: {r2_scores

#                     # Plot Exponential Smoothing forecast with alternating
#                     ax[1].plot(actual_df[col], label="Actual", color='blac
#                     for i in range(len(actual_df[col])):
#                         ax[1].plot(forecast_df_original_exp_smooth.index[i
#                                    marker='^', alpha=1, color=colors[i % l
#                         ax[1].plot(actual_df.index[i], actual_df[col].iloc
#                                    marker='o', alpha=1, color=colors[i % l
#                     # Highlight the last point in green with alpha=1
#                     ax[1].plot(forecast_df_original_exp_smooth[col].index[
#                                marker='^', alpha=1, color='green', markers
#                     ax[1].legend()
#                     ax[1].set_title(f"{col} - Exp Smooth Forecast (R²: {r2

#                     # Plot Lag-1 forecast with alternating colors for mark
#                     ax[2].plot(actual_df[col], label="Actual", color='blac
#                     for i in range(len(actual_df[col])):
#                         ax[2].plot(forecast_df_original_lag_1.index[i], fo
#                                    marker='^', alpha=1, color=colors[i % l
#                         ax[2].plot(actual_df.index[i], actual_df[col].iloc
#                                    marker='o', alpha=1, color=colors[i % l
#                     # Highlight the last point in green with alpha=1
#                     ax[2].plot(forecast_df_original_lag_1[col].index[-1],
#                                marker='^', alpha=1, color='green', markers
#                     ax[2].legend()
#                     ax[2].set_title(f"{col} - Lag-1 Forecast (R²: {r2_scor

#                     plt.tight_layout()

#                     # Store the figure in the dictionary
#                     dict_R2_OLD_VERSION_NO_CAP_PLOT[col] = fig

#                     plt.show()



# dict_R2_OLD_VERSION_NO_CAP= dict_R2.copy()
# with open("data/cleaned/forecasted_weights/dict_R2_OLD_VERSION_NO_CAP.pkl"
#     pickle.dump(dict_R2_OLD_VERSION_NO_CAP, f)

with open("data/cleaned/forecasted_weights/dict_R2_OLD_VERSION_NO_CAP.pkl",
    dict_R2_OLD_VERSION_NO_CAP = pickle.load(f)
```

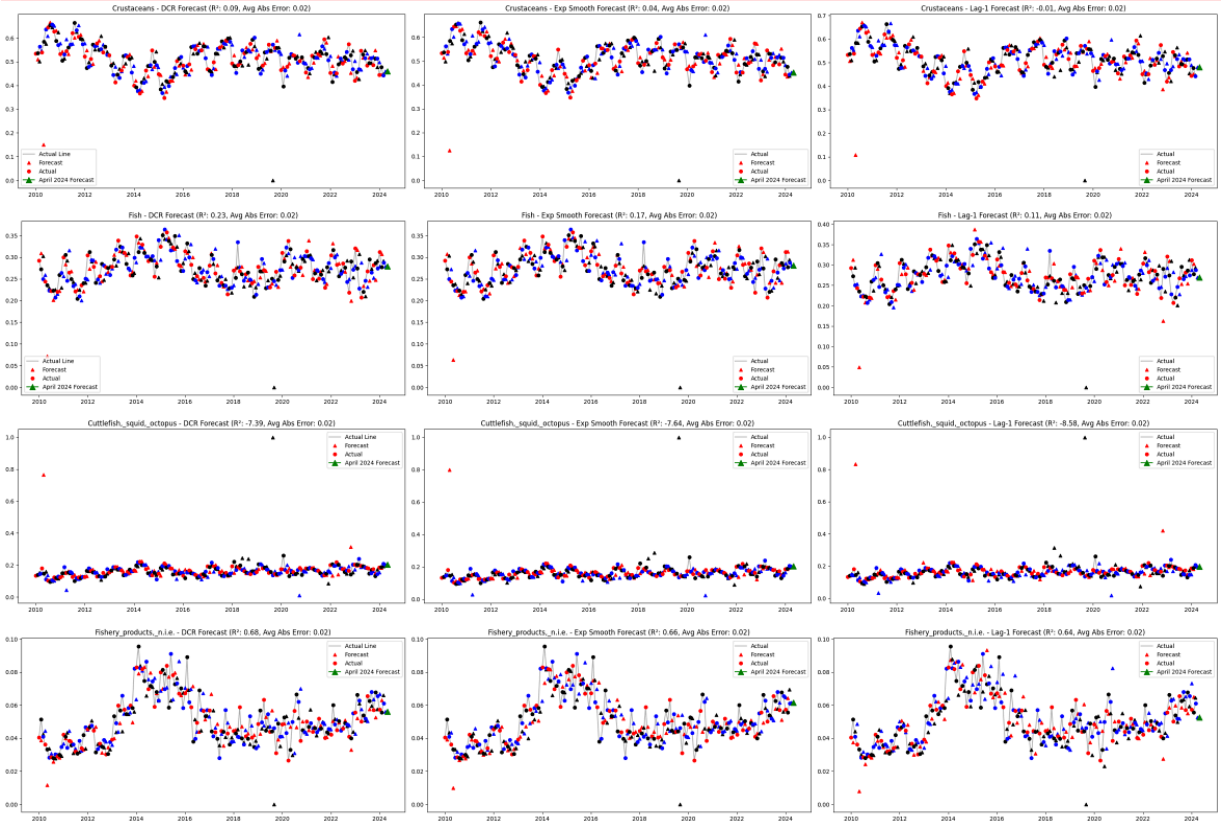
```python
# with open("data/cleaned/dict_R2_OLD_VERSION_NO_CAP_PLOT.pkl", "wb") as f:
#     pickle.dump(dict_R2_OLD_VERSION_NO_CAP_PLOT, f)

with open("data/cleaned/dict_R2_OLD_VERSION_NO_CAP_PLOT.pkl", "rb") as f:
    dict_R2_OLD_VERSION_NO_CAP_PLOT = pickle.load(f)

#open pre-saved plot from dict
for col, fig in dict_R2_OLD_VERSION_NO_CAP_PLOT.items():
    plt.figure(fig.number)
    plt.show()
```

```
Walk-Forward Forecasting (DCR): 100%|██████████| 172/172 [00:02<00:00, 70.46 it/s]
/home/wheelfredie/.local/share/virtualenvs/BoT_Exports-7Oxff3EB/lib/python3.10/site-packages/pandas/core/internals/blocks.py:393: RuntimeWarning: overflow encountered in exp
  result = func(self.values, **kwargs)
/home/wheelfredie/.local/share/virtualenvs/BoT_Exports-7Oxff3EB/lib/python3.10/site-packages/pandas/core/internals/blocks.py:393: RuntimeWarning: overflow encountered in exp
  result = func(self.values, **kwargs)
/home/wheelfredie/.local/share/virtualenvs/BoT_Exports-7Oxff3EB/lib/python3.10/site-packages/pandas/core/internals/blocks.py:393: RuntimeWarning: overflow encountered in exp
  result = func(self.values, **kwargs)
```

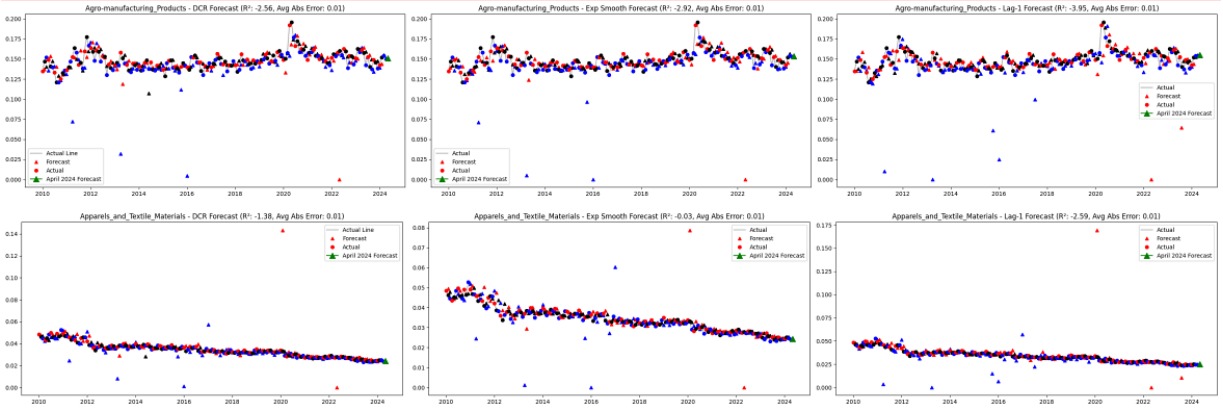


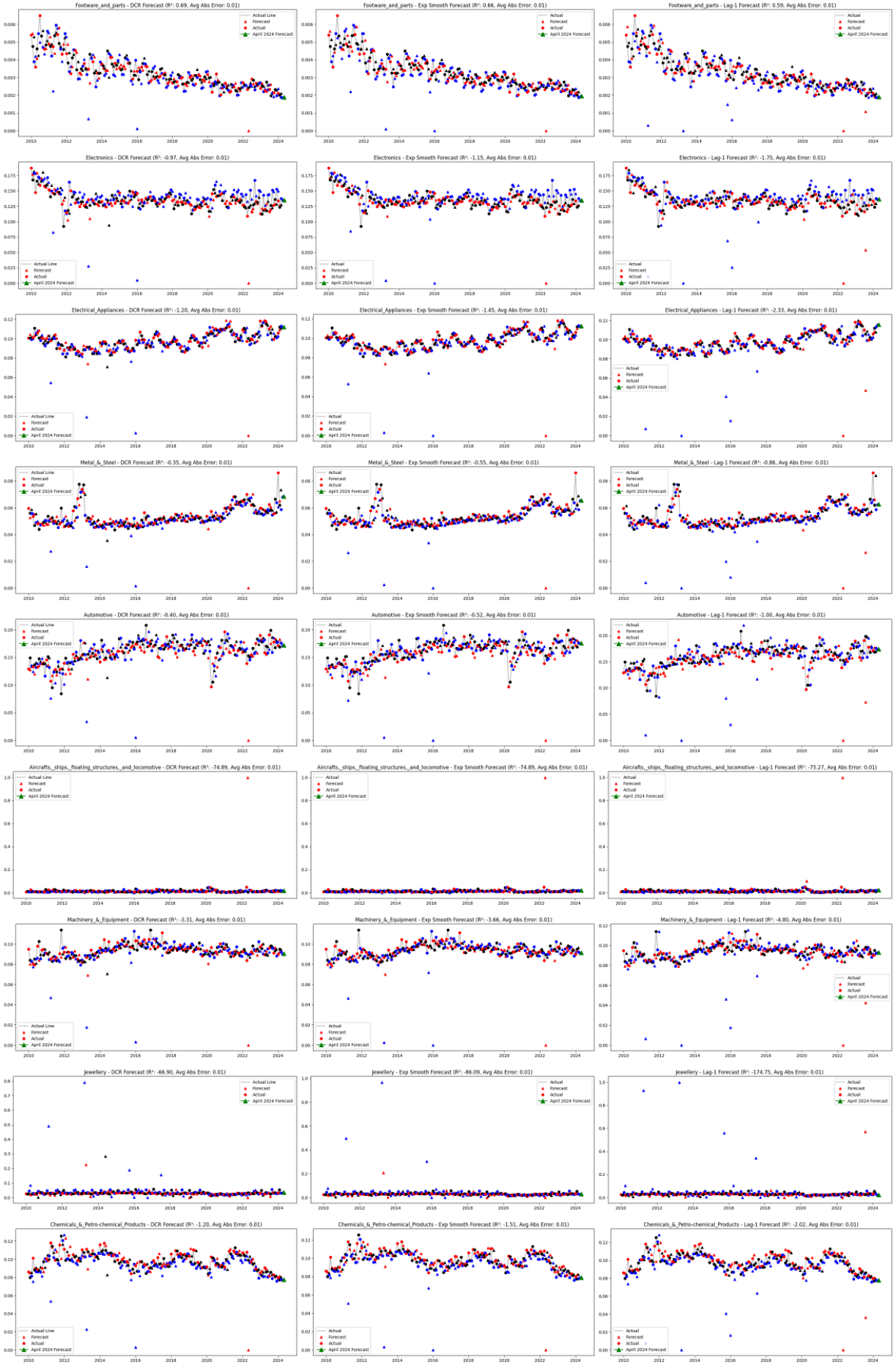


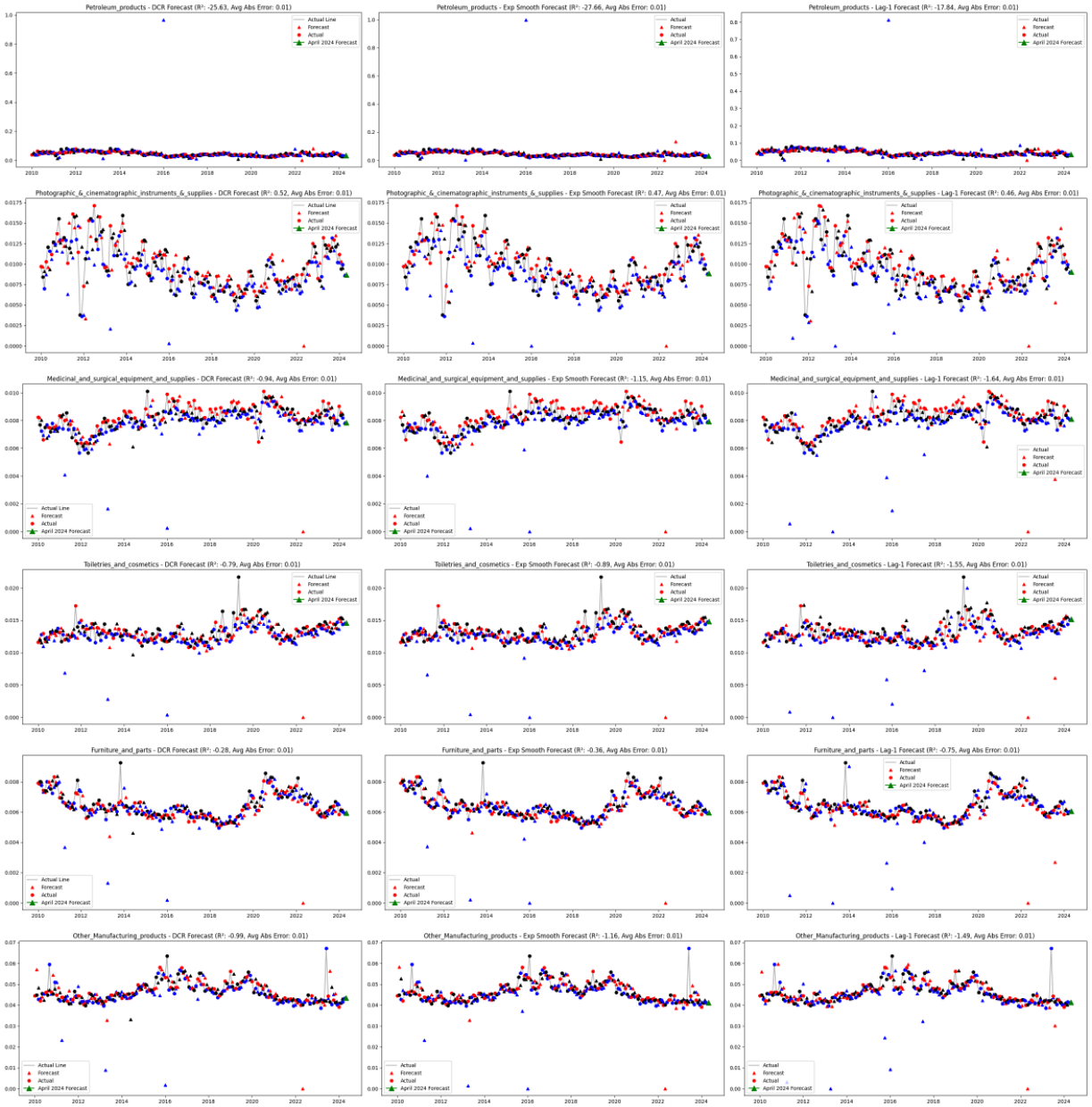

```
Walk-Forward Forecasting (DCR): 100%|██████████| 172/172 [00:01<00:00, 108.95it/s]
```



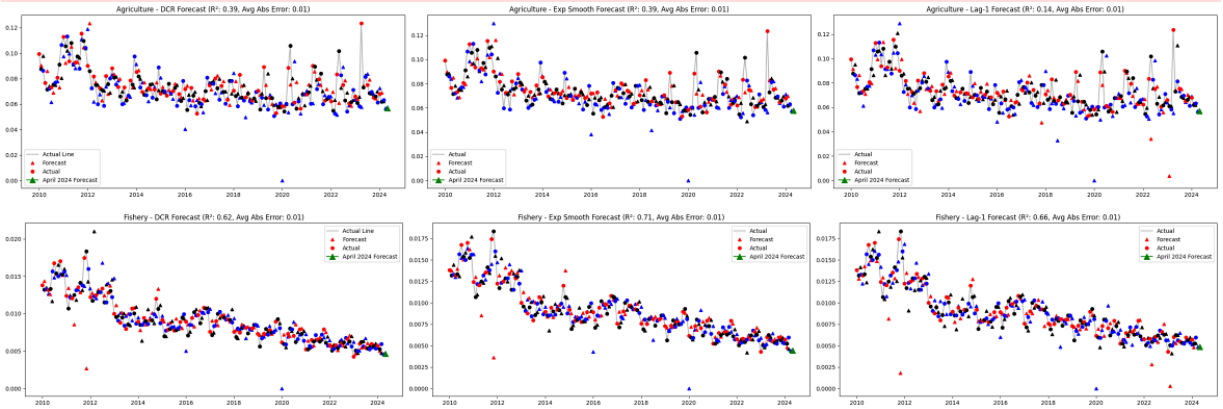

```
Walk-Forward Forecasting (DCR): 100%|██████████| 172/172 [00:06<00:00, 25.83it/s]
```

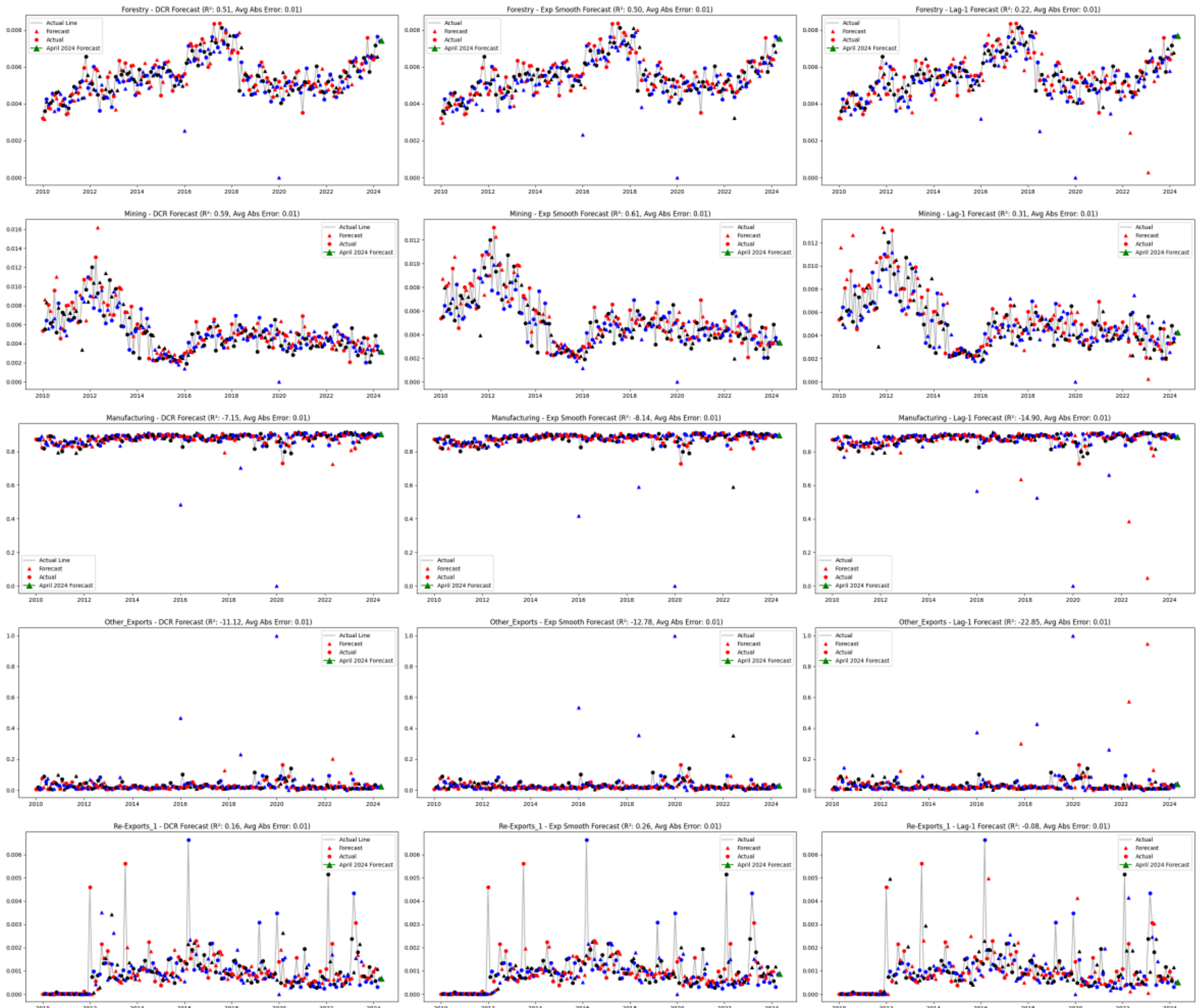



Footware_and_parts - DCR Forecast (R²: 0.69, Avg Abs Error: 0.01)
Footware_and_parts - Exp Smooth Forecast (R²: 0.66, Avg Abs Error: 0.01)
Footware_and_parts - Lag-1 Forecast (R²: 0.59, Avg Abs Error: 0.01)
Electronics - DCR Forecast (R²: -0.97, Avg Abs Error: 0.01)
Electronics - Exp Smooth Forecast (R²: -1.15, Avg Abs Error: 0.01)
Electronics - Lag-1 Forecast (R²: -1.75, Avg Abs Error: 0.01)
Electrical_Appliances - DCR Forecast (R²: -1.20, Avg Abs Error: 0.01)
Electrical_Appliances - Exp Smooth Forecast (R²: -1.45, Avg Abs Error: 0.01)
Electrical_Appliances - Lag-1 Forecast (R²: -2.33, Avg Abs Error: 0.01)
Metal_&_Steel - DCR Forecast (R²: -0.35, Avg Abs Error: 0.01)
Metal_&_Steel - Exp Smooth Forecast (R²: -0.55, Avg Abs Error: 0.01)
Metal_&_Steel - Lag-1 Forecast (R²: -0.86, Avg Abs Error: 0.01)
Automotive - DCR Forecast (R²: -0.40, Avg Abs Error: 0.01)
Automotive - Exp Smooth Forecast (R²: -0.52, Avg Abs Error: 0.01)
Automotive - Lag-1 Forecast (R²: -1.00, Avg Abs Error: 0.01)
Aircrafts,_ships,_floating_structures,_and_locomotive - DCR Forecast (R²: -74.89, Avg Abs Error: 0.01)
Aircrafts,_ships,_floating_structures,_and_locomotive - Exp Smooth Forecast (R²: -74.89, Avg Abs Error: 0.01)
Aircrafts,_ships,_floating_structures,_and_locomotive - Lag-1 Forecast (R²: -75.27, Avg Abs Error: 0.01)
Machinery_&_Equipment - DCR Forecast (R²: -3.31, Avg Abs Error: 0.01)
Machinery_&_Equipment - Exp Smooth Forecast (R²: -3.66, Avg Abs Error: 0.01)
Machinery_&_Equipment - Lag-1 Forecast (R²: -4.80, Avg Abs Error: 0.01)
Jewellery - DCR Forecast (R²: -66.90, Avg Abs Error: 0.01)
Jewellery - Exp Smooth Forecast (R²: -86.09, Avg Abs Error: 0.01)
Jewellery - Lag-1 Forecast (R²: -174.75, Avg Abs Error: 0.01)
Chemicals_&_Petro-chemical_Products - DCR Forecast (R²: -1.20, Avg Abs Error: 0.01)
Chemicals_&_Petro-chemical_Products - Exp Smooth Forecast (R²: -1.51, Avg Abs Error: 0.01)
Chemicals_&_Petro-chemical_Products - Lag-1 Forecast (R²: -2.02, Avg Abs Error: 0.01)
Actual Line
Forecast
Actual
April 2024 Forecast




```
Walk-Forward Forecasting:  62%|██████    | 5/8 [02:27<01:32, 30.85s/it]/hom
e/wheelfredie/.local/share/virtualenvs/BoT_Exports-7Oxff3EB/lib/python3.10/s
ite-packages/pandas/core/internals/blocks.py:393: RuntimeWarning: divide by
zero encountered in log
  result = func(self.values, **kwargs)
Walk-Forward Forecasting (DCR): 100%|██████████| 172/172 [00:02<00:00, 63.77
it/s]
```






```
Walk-Forward Forecasting: 100%|██████████| 8/8 [02:54<00:00, 21.78s/it]
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
```

```
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
```

Will require X-13 PATH to run

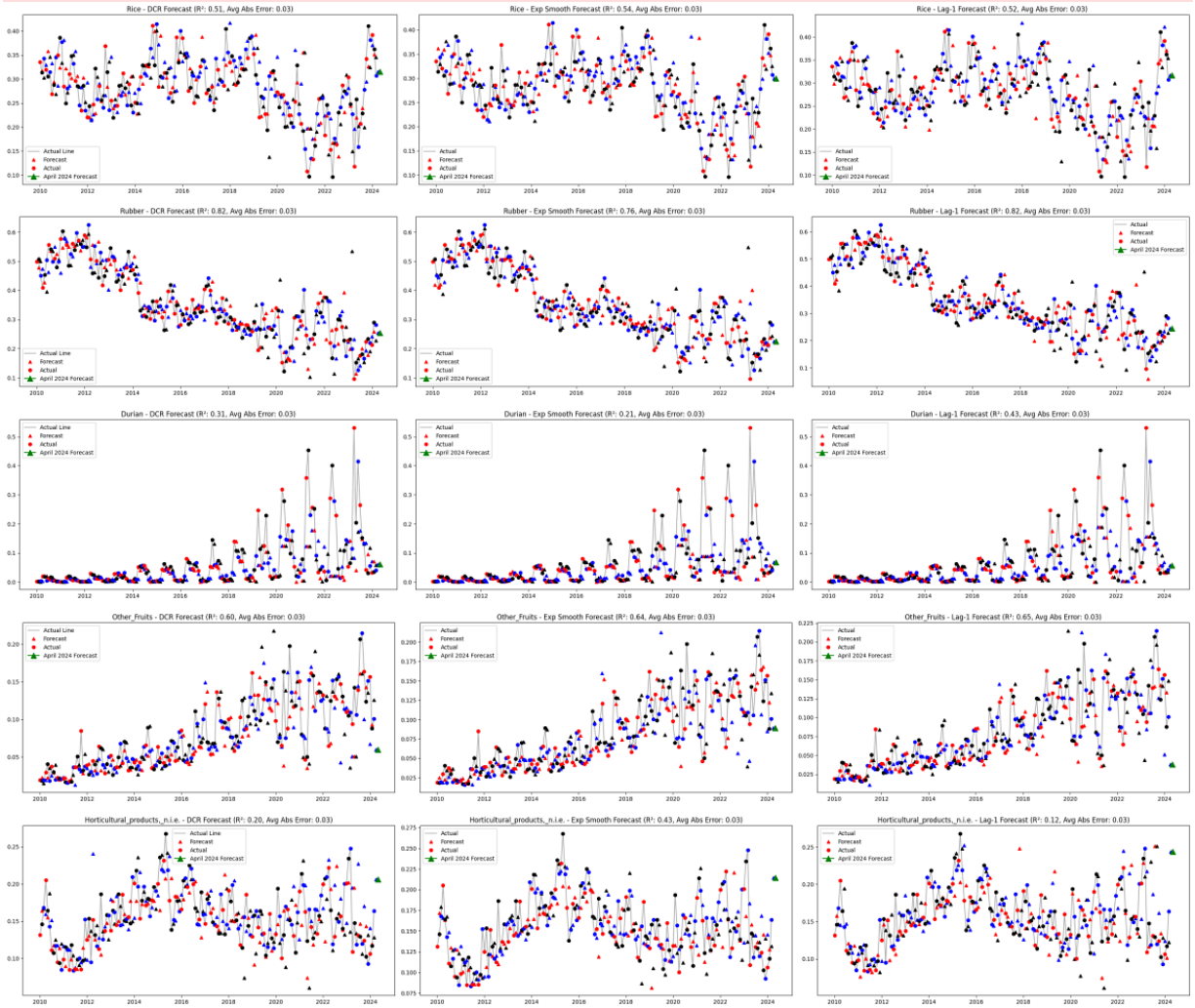
In [8]:
```
# dict_R2 = {}
# figures_dict_CAP_PLOT = {}

# # Suppress specific X13 warnings
# with pd.option_context('display.max_rows', None, 'display.max_columns', No
#     warnings.filterwarnings("ignore", category=X13Warning)

#     with open("data/cleaned/COMBINED_deseasonalised_value_dict.pkl", "rb")
#         dict_deseasonalized_value = pickle.load(f)

#     for name in tqdm(mapper.columns, desc="Walk-Forward Forecasting", dyna
#         if name != "Mining":
#             temp_mapper = mapper[name].dropna().tolist()
#             if temp_mapper:
#                 df = df_export_ANALYSIS[temp_mapper]

#                 # Ensure data is numeric
#                 df = df.apply(pd.to_numeric, errors='coerce')

#                 # Normalize the data to weights
#                 constituent_total = df.sum(axis=1)
#                 df_actual = df.divide(constituent_total, axis=0)

#                 # CLR transform
#                 df = clr_transformation(df_actual.copy(deep=True))

#                 # X13 deseasonalize
#                 deseasonalized_df = pd.DataFrame()
#                 seasonal_factors_df = pd.DataFrame()

#                 for col in df.columns:
#                     df[col] = df[col].replace([np.inf, -np.inf], np.nan).r
#                     seasadj, trend, seasonal, irregular, seasonal_factors
#                     deseasonalized_df[col] = seasadj
#                     seasonal_factors_df[col] = seasonal_factors

#                 old_deseasonalized_df = deseasonalized_df.copy(deep=True)

#                 # Walk-forward parameters
#                 window_size = 180  # 15 years of monthly data
#                 total_observations = len(deseasonalized_df)  # Total data
```

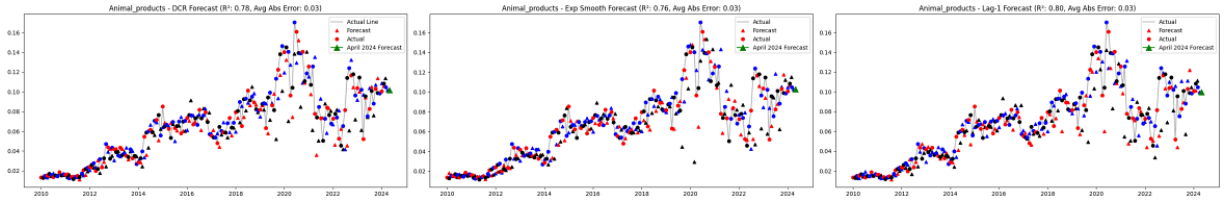
```
#                 steps = total_observations - window_size  # Remaining poir

#                 # Generate periods DataFrame
#                 periods_df = generate_periods_df(deseasonalized_df, windo

#                 # Perform walk-forward forecasting
#                 rolling_forecasts_dcr = walk_forward_forecasting_dcr(desea
#                 forecast_period = rolling_forecasts_dcr.index

#                 # Additional Forecasting Models (SImple Benchmarks)
#                 deseasonalized_df = pd.concat([deseasonalized_df, pd.DataF
#                 rolling_forecasts_exp_smooth = deseasonalized_df.apply(exp
#                 rolling_forecasts_exp_smooth = rolling_forecasts_exp_smoot
#                 rolling_forecasts_lag_1 = deseasonalized_df.apply(lag_1_mo
#                 rolling_forecasts_lag_1 = rolling_forecasts_lag_1[rolling_

#                 # Reseasonalize the forecasts
#                 extended_seasonal_factors = seasonal_factors_df[seasonal_f
#                 reseasonalized_forecast_dcr = rolling_forecasts_dcr.mul(ex
#                 reseasonalized_forecast_exp_smooth = rolling_forecasts_exp
#                 reseasonalized_forecast_lag_1 = rolling_forecasts_lag_1.mu

#                 # Cap extreme values before applying inverse CLR transform
#                 reseasonalized_forecast_dcr = cap_extreme_values(reseasona
#                 reseasonalized_forecast_exp_smooth = cap_extreme_values(re
#                 reseasonalized_forecast_lag_1 = cap_extreme_values(reseaso

#                 # Apply inverse CLR transformation
#                 forecast_df_original_dcr = inverse_clr_transformation(rese
#                 forecast_df_original_exp_smooth = inverse_clr_transformati
#                 forecast_df_original_lag_1 = inverse_clr_transformation(re

#                 # Cap extreme values before applying inverse CLR transform
#                 forecast_df_original_dcr = cap_extreme_values(forecast_df_
#                 forecast_df_original_exp_smooth = cap_extreme_values(forec
#                 forecast_df_original_lag_1 = cap_extreme_values(forecast_d

#                 #save weights as pkl
#                 forecast_df_original_dcr.to_pickle(f"data/cleaned/forecast
#                 forecast_df_original_exp_smooth.to_pickle(f"data/cleaned/f
#                 forecast_df_original_lag_1.to_pickle(f"data/cleaned/foreca


#                 # Evaluate forecasts
#                 actual_df = df_actual[old_deseasonalized_df.index.isin(for

#                 r2_scores_dcr = evaluate_forecasts(actual_df, forecast_df_
#                 r2_scores_exp_smooth = evaluate_forecasts(actual_df, forec
#                 r2_scores_lag_1 = evaluate_forecasts(actual_df, forecast_d

#                 # Calculate average absolute error
#                 avg_abs_error_dcr = mean_absolute_error(actual_df, forecas
#                 avg_abs_error_exp_smooth = mean_absolute_error(actual_df,
#                 avg_abs_error_lag_1 = mean_absolute_error(actual_df, forec

#                 # Store R² scores in DataFrame
```

```
#                 dftemp = pd.DataFrame([r2_scores_dcr, r2_scores_exp_smooth
#                 dict_R2[name] = dftemp

#                 # Visualization
#                 colors = ['red', 'black', 'blue'] # alternate points

#                 for col in actual_df.columns:
#                     fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(30,

#                     # Plot DCR forecast with alternating colors for marker
#                     ax[0].plot(actual_df[col], label="Actual Line", color=
#                     for i in range(len(actual_df[col])):
#                         ax[0].plot(forecast_df_original_dcr.index[i], fore
#                                    marker='^', alpha=1, color=colors[i % l
#                         ax[0].plot(actual_df.index[i], actual_df[col].iloc
#                                    marker='o', alpha=1, color=colors[i % l
#                     # Highlight the last point in green with alpha=1
#                     ax[0].plot(forecast_df_original_dcr[col].index[-1], fo
#                                marker='^', alpha=1, color='green', markers
#                     ax[0].legend()
#                     ax[0].set_title(f"{col} - DCR Forecast (R²: {r2_scores

#                     # Plot Exponential Smoothing forecast with alternating
#                     ax[1].plot(actual_df[col], label="Actual", color='blac
#                     for i in range(len(actual_df[col])):
#                         ax[1].plot(forecast_df_original_exp_smooth.index[i
#                                    marker='^', alpha=1, color=colors[i % l
#                         ax[1].plot(actual_df.index[i], actual_df[col].iloc
#                                    marker='o', alpha=1, color=colors[i % l
#                     # Highlight the last point in green with alpha=1
#                     ax[1].plot(forecast_df_original_exp_smooth[col].index[
#                                marker='^', alpha=1, color='green', markers
#                     ax[1].legend()
#                     ax[1].set_title(f"{col} - Exp Smooth Forecast (R²: {r2

#                     # Plot Lag-1 forecast with alternating colors for mark
#                     ax[2].plot(actual_df[col], label="Actual", color='blac
#                     for i in range(len(actual_df[col])):
#                         ax[2].plot(forecast_df_original_lag_1.index[i], fo
#                                    marker='^', alpha=1, color=colors[i % l
#                         ax[2].plot(actual_df.index[i], actual_df[col].iloc
#                                    marker='o', alpha=1, color=colors[i % l
#                     # Highlight the last point in green with alpha=1
#                     ax[2].plot(forecast_df_original_lag_1[col].index[-1],
#                                marker='^', alpha=1, color='green', markers
#                     ax[2].legend()
#                     ax[2].set_title(f"{col} - Lag-1 Forecast (R²: {r2_scor

#                     plt.tight_layout()

#                     # Store the figure in the dictionary
#                     figures_dict_CAP_PLOT[col] = fig

#                     plt.show()
```

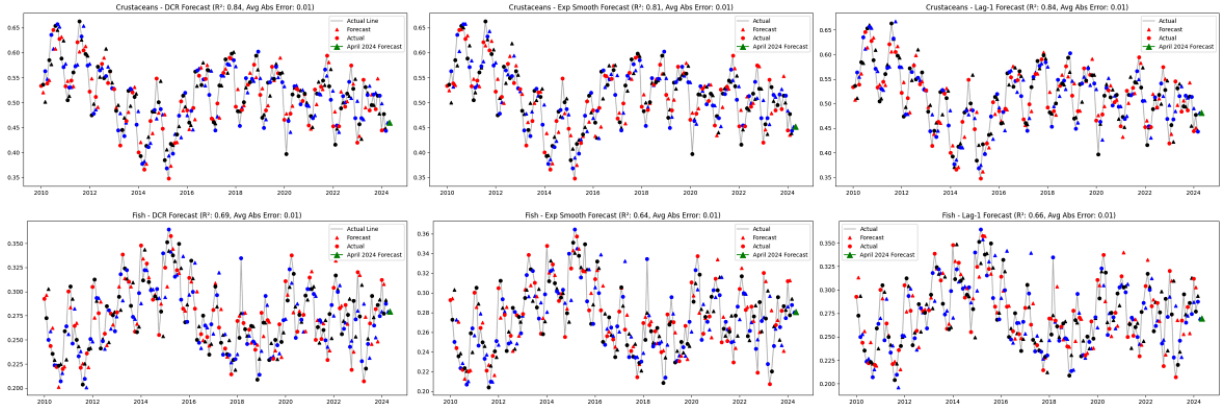
```
# dict_R2_cap_extreme_values = dict_R2.copy()
# with open("data/cleaned/forecasted_weights/dict_R2_cap_extreme_values.pkl"
#     pickle.dump(dict_R2_cap_extreme_values, f)

with open("data/cleaned/forecasted_weights/dict_R2_cap_extreme_values.pkl",
    dict_R2_cap_extreme_values = pickle.load(f)



# with open("data/cleaned/figures_dict_CAP_PLOT.pkl", "wb") as f:
#     pickle.dump(figures_dict_CAP_PLOT, f)

with open("data/figures_dict_CAP_PLOT.pkl", "rb") as f:
    figures_dict_CAP_PLOT = pickle.load(f)

#open pre-saved plot from dict
for col, fig in figures_dict_CAP_PLOT.items():
    plt.figure(fig.number)
    plt.show()
```

```
Walk-Forward Forecasting (DCR): 100%|██████████| 172/172 [00:02<00:00, 73.79
it/s]
```

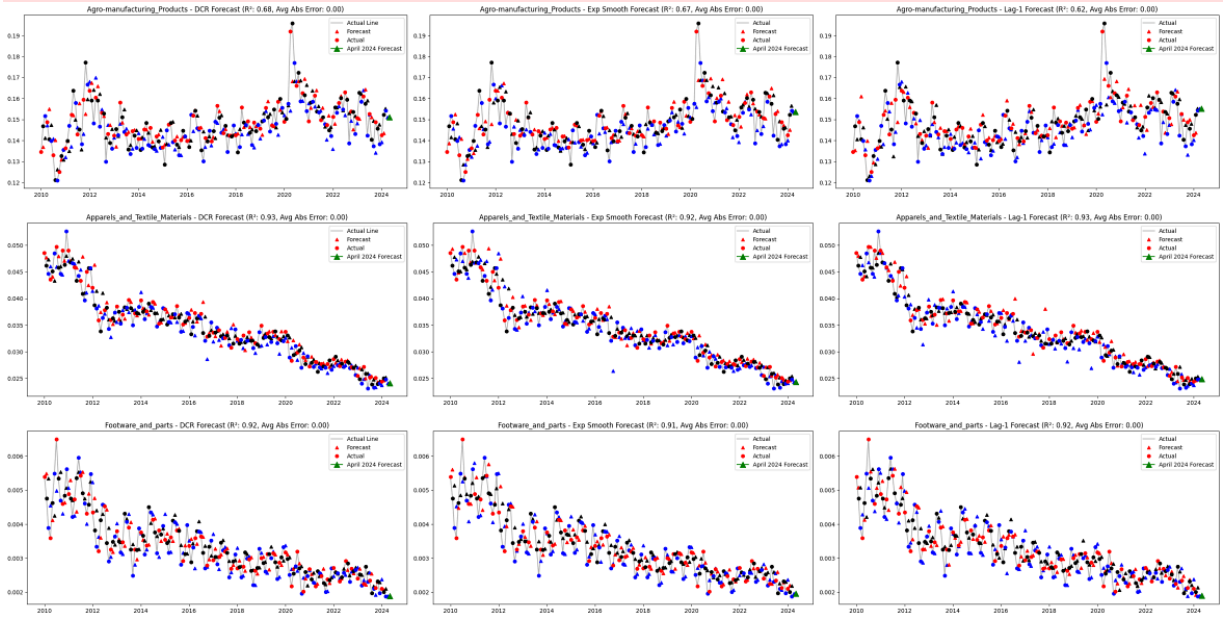


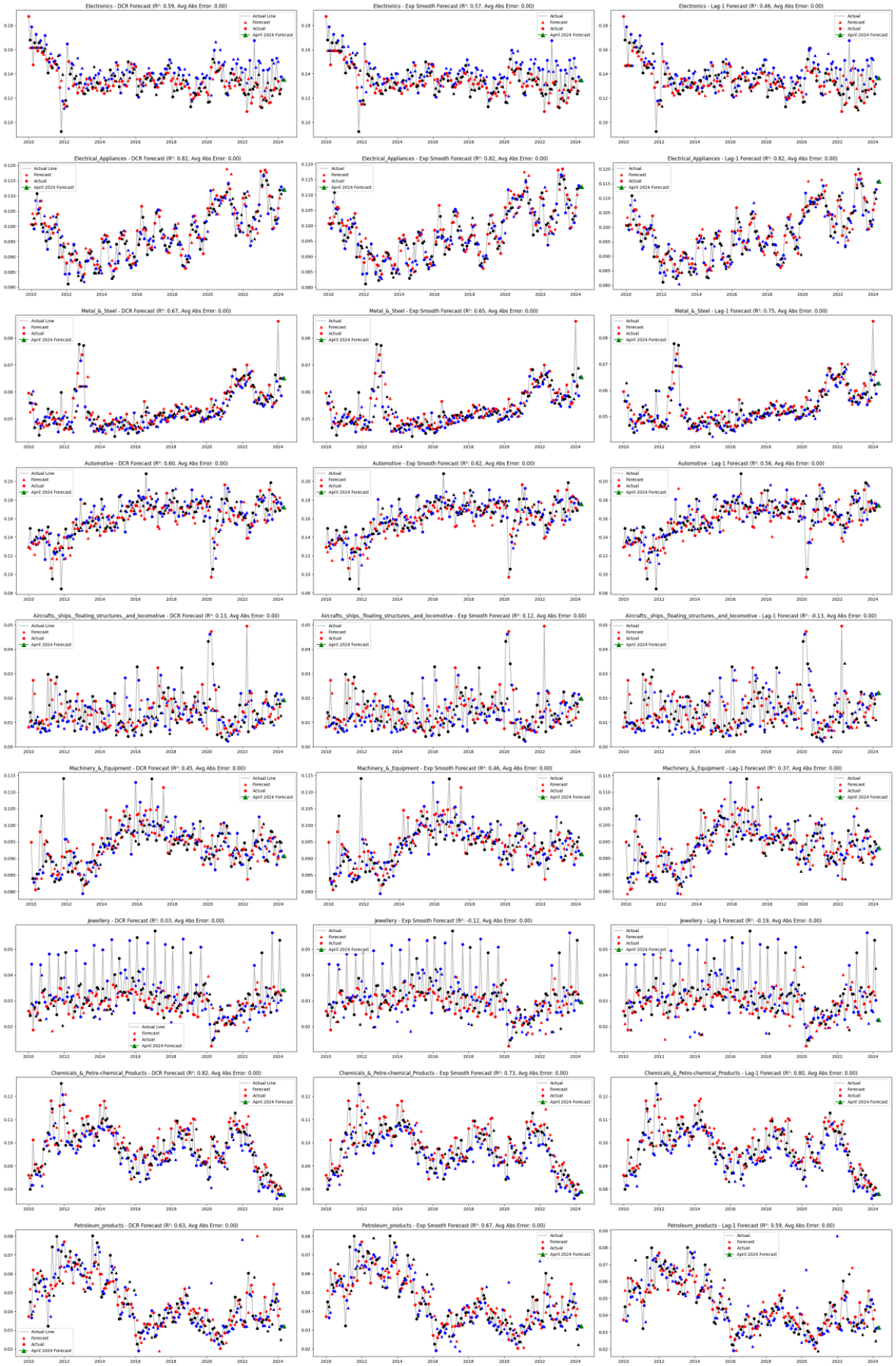


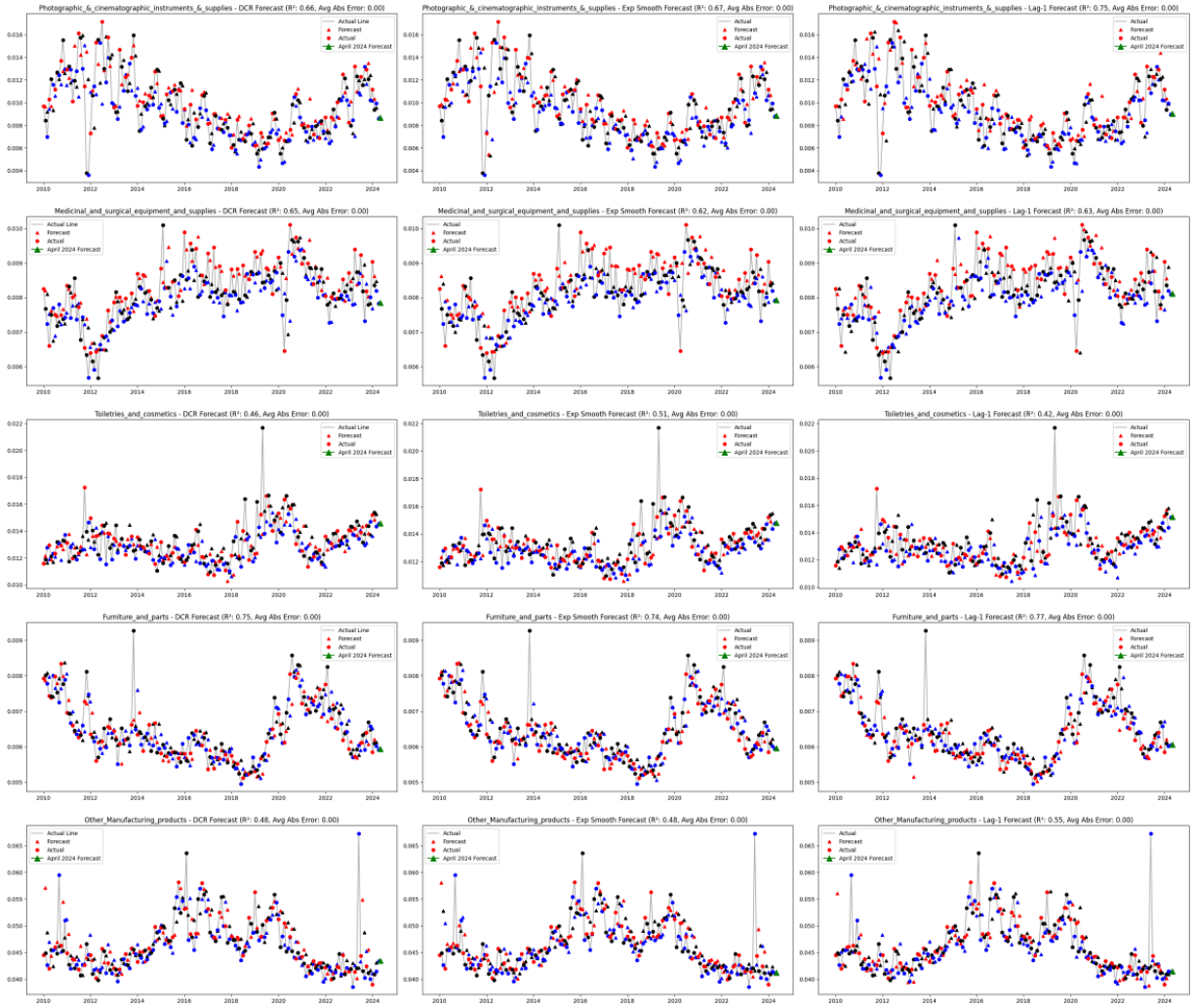

```
Walk-Forward Forecasting (DCR): 100%|██████████| 172/172 [00:01<00:00, 108.12it/s]
```



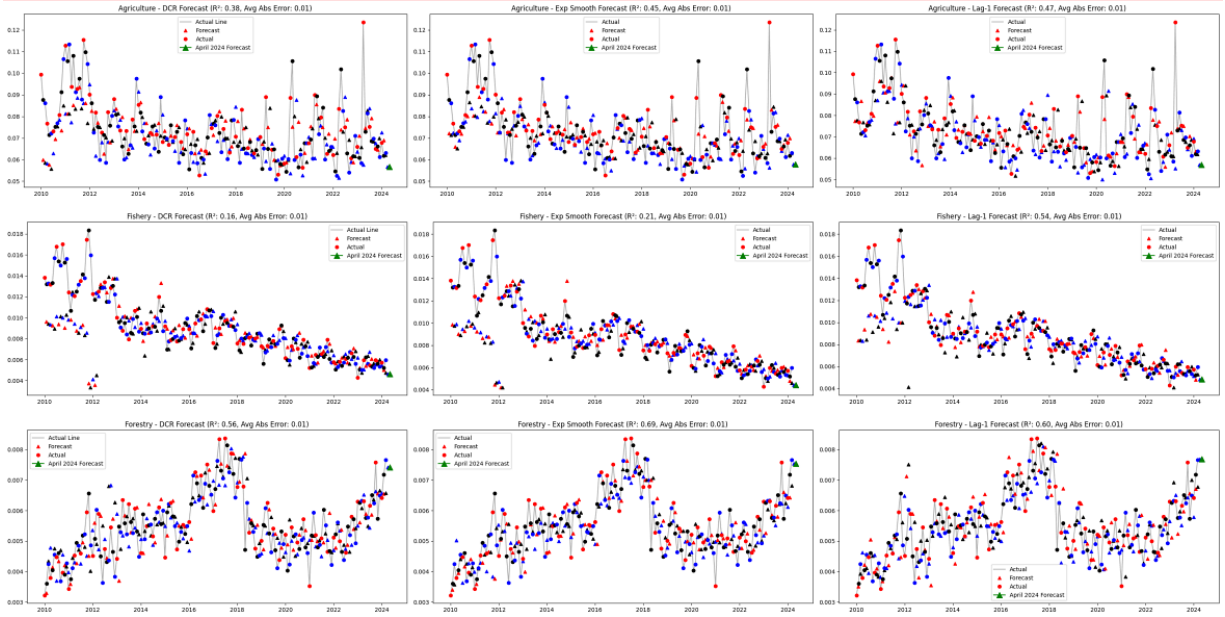

```
Walk-Forward Forecasting (DCR): 100%|██████████| 172/172 [00:06<00:00, 25.64it/s]
```

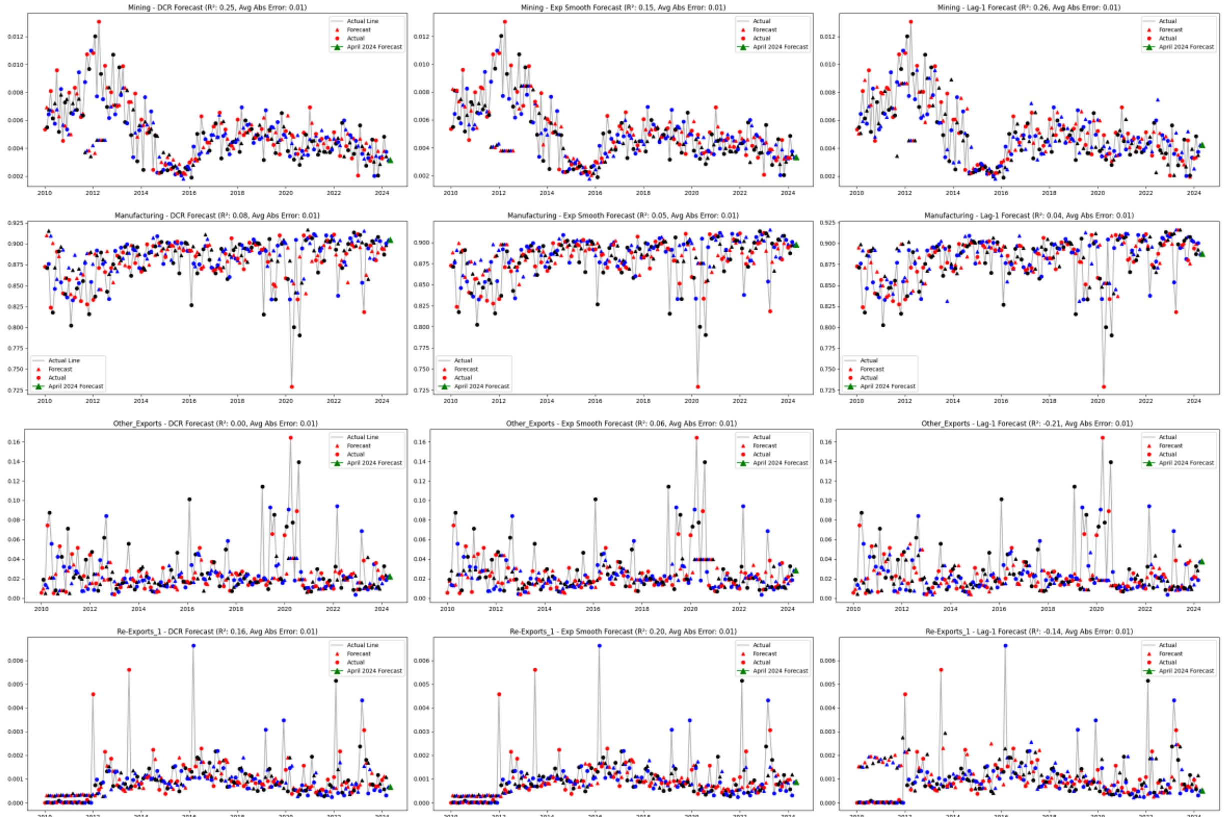



Electronics - DCR Forecast (R²: 0.59, Avg Abs Error: 0.00)
Electronics - Exp Smooth Forecast (R²: 0.57, Avg Abs Error: 0.00)
Electronics - Lag-1 Forecast (R²: 0.46, Avg Abs Error: 0.00)
Electrical_Appliances - DCR Forecast (R²: 0.82, Avg Abs Error: 0.00)
Electrical_Appliances - Exp Smooth Forecast (R²: 0.82, Avg Abs Error: 0.00)
Electrical_Appliances - Lag-1 Forecast (R²: 0.82, Avg Abs Error: 0.00)
Metal_&_Steel - DCR Forecast (R²: 0.67, Avg Abs Error: 0.00)
Metal_&_Steel - Exp Smooth Forecast (R²: 0.65, Avg Abs Error: 0.00)
Metal_&_Steel - Lag-1 Forecast (R²: 0.75, Avg Abs Error: 0.00)
Automotive - DCR Forecast (R²: 0.60, Avg Abs Error: 0.00)
Automotive - Exp Smooth Forecast (R²: 0.62, Avg Abs Error: 0.00)
Automotive - Lag-1 Forecast (R²: 0.56, Avg Abs Error: 0.00)
Aircrafts,_ships,_floating_structures,_and_locomotive - DCR Forecast (R²: 0.13, Avg Abs Error: 0.00)
Aircrafts,_ships,_floating_structures,_and_locomotive - Exp Smooth Forecast (R²: 0.12, Avg Abs Error: 0.00)
Aircrafts,_ships,_floating_structures,_and_locomotive - Lag-1 Forecast (R²: -0.13, Avg Abs Error: 0.00)
Machinery_&_Equipment - DCR Forecast (R²: 0.45, Avg Abs Error: 0.00)
Machinery_&_Equipment - Exp Smooth Forecast (R²: 0.46, Avg Abs Error: 0.00)
Machinery_&_Equipment - Lag-1 Forecast (R²: 0.37, Avg Abs Error: 0.00)
Jewellery - DCR Forecast (R²: 0.03, Avg Abs Error: 0.00)
Jewellery - Exp Smooth Forecast (R²: -0.12, Avg Abs Error: 0.00)
Jewellery - Lag-1 Forecast (R²: -0.19, Avg Abs Error: 0.00)
Chemicals_&_Petro-chemical_Products - DCR Forecast (R²: 0.82, Avg Abs Error: 0.00)
Chemicals_&_Petro-chemical_Products - Exp Smooth Forecast (R²: 0.73, Avg Abs Error: 0.00)
Chemicals_&_Petro-chemical_Products - Lag-1 Forecast (R²: 0.80, Avg Abs Error: 0.00)
Petroleum_products - DCR Forecast (R²: 0.63, Avg Abs Error: 0.00)
Petroleum_products - Exp Smooth Forecast (R²: 0.67, Avg Abs Error: 0.00)
Petroleum_products - Lag-1 Forecast (R²: 0.59, Avg Abs Error: 0.00)
Actual Line
Forecast
Actual
April 2024 Forecast




```
Walk-Forward Forecasting:  62%|██████    | 5/8 [02:29<01:33, 31.07s/it]/hom
e/wheelfredie/.local/share/virtualenvs/BoT_Exports-7Oxff3EB/lib/python3.10/s
ite-packages/pandas/core/internals/blocks.py:393: RuntimeWarning: divide by
zero encountered in log
  result = func(self.values, **kwargs)
Walk-Forward Forecasting (DCR): 100%|██████████| 172/172 [00:02<00:00, 62.32
it/s]
```






```
Walk-Forward Forecasting: 100%|██████████| 8/8 [02:59<00:00, 22.47s/it]
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
```

```
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
```

In [9]:
```
for name in dict_R2_cap_extreme_values.keys():
    display(name)
    display("NO CAP VERSION")
    display(dict_R2_OLD_VERSION_NO_CAP[name])
    display("CAP VERSION")
    display(dict_R2_cap_extreme_values[name])
    display("-"*300)
```

```
'Agriculture'
'NO CAP VERSION'
```

	Rice	Rubber	Durian	Other_Fruits	Horticultural_products,_n.i.e.	Animal_
DCR	-0.162706	0.599180	-0.178544	-3.499456	-4.498937	-(
Exp Smooth	-0.375026	0.488174	-0.337574	-3.297442	-4.299431	-(
Lag-1	-0.182621	0.626980	-0.712069	-2.275874	-4.571859	-1

```
'CAP VERSION'
```

	Rice	Rubber	Durian	Other_Fruits	Horticultural_products,_n.i.e.	Animal_pr
DCR	0.506269	0.823796	0.313762	0.595394	0.199577	0.
Exp Smooth	0.539955	0.760789	0.208335	0.643292	0.429406	0.
Lag-1	0.516214	0.821173	0.429784	0.645472	0.120082	0.

```
'------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------'
'Fishery'
'NO CAP VERSION'
```

	Crustaceans	Fish	Cuttlefish,_squid,_octopus	Fishery_products,_n.i.e.
DCR	0.086075	0.230391	-7.390352	0.683307
Exp Smooth	0.042196	0.165188	-7.635124	0.664516
Lag-1	-0.009864	0.112286	-8.575727	0.638547

```
'CAP VERSION'
```

	Crustaceans	Fish	Cuttlefish,_squid,_octopus	Fishery_products,_n.i.e.
DCR	0.841380	0.693257	0.568901	0.760821
Exp Smooth	0.814723	0.639270	0.529092	0.730930
Lag-1	0.839820	0.660677	0.437075	0.706157

```
'-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------'
'Manufacturing'
'NO CAP VERSION'
```

	Agro-manufacturing_Products	Apparels_and_Textile_Materials	Footware_and_parts	Elect
DCR	-2.559946	-1.380467	0.687504	-0.9
Exp Smooth	-2.919082	-0.026935	0.663625	-1.1
Lag-1	-3.954058	-2.586119	0.594735	-1.7

```
'CAP VERSION'
```

	Agro-manufacturing_Products	Apparels_and_Textile_Materials	Footware_and_parts	Elect
DCR	0.676848	0.930115	0.916426	0.5
Exp Smooth	0.667611	0.917084	0.909511	0.5
Lag-1	0.624439	0.929957	0.919231	0.4

```
'-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------'
'Total_Exports_(Customs_basis)'
'NO CAP VERSION'
```

	Agriculture	Fishery	Forestry	Mining	Manufacturing	Other_Exports	R Exports_
DCR	0.385523	0.623717	0.512300	0.593015	-7.148801	-11.117505	0.15506
Exp Smooth	0.390328	0.712690	0.497783	0.612632	-8.143212	-12.777226	0.26404
Lag-1	0.135675	0.657060	0.219790	0.306419	-14.900882	-22.853327	-0.07882

```
'CAP VERSION'
```

	Agriculture	Fishery	Forestry	Mining	Manufacturing	Other_Exports	R Exports_
DCR	0.375374	0.163061	0.563713	0.252655	0.083092	0.004026	0.16053
Exp Smooth	0.454942	0.206607	0.692899	0.152601	0.052322	0.058823	0.20484
Lag-1	0.472380	0.539492	0.595167	0.261575	0.035167	-0.208714	-0.14272

```
'-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------'
```

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