```
import pandas as pd

# File paths
file_paths = {
    "2022-2023": "/content/gsw-shg-en 2022-2023.csv",
    "2023-2024": "/content/gsw-shg-en 2023-2024.csv",
    "2024-2025": "/content/gsw-shg-en 2024-2025.csv",
}

# Load datasets
dataframes = {year: pd.read_csv(path) for year, path in file_paths.items()}

# Display basic information about the datasets
{year: df.info() for year, df in dataframes.items()}
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 207192 entries, 0 to 207191
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype 
---  ------            --------------   ----- 
 0   grain_week        207192 non-null  int64 
 1   crop_year         207192 non-null  object
 2   week_ending_date  207192 non-null  object
 3   worksheet         207192 non-null  object
 4   metric            207192 non-null  object
 5   period            207192 non-null  object
 6   grain             207192 non-null  object
 7   grade             62130 non-null   object
 8   region            202176 non-null  object
 9   Ktonnes           207192 non-null  object
dtypes: int64(1), object(9)
memory usage: 15.8+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 211116 entries, 0 to 211115
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype 
---  ------            --------------   ----- 
 0   grain_week        211116 non-null  int64 
 1   crop_year         211116 non-null  object
 2   week_ending_date  211116 non-null  object
 3   worksheet         211116 non-null  object
 4   metric            211116 non-null  object
 5   period            211116 non-null  object
 6   grain             211116 non-null  object
 7   grade             61092 non-null   object
 8   region            205404 non-null  object
 9   Ktonnes           211114 non-null  object
dtypes: int64(1), object(9)
memory usage: 16.1+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98581 entries, 0 to 98580
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   grain_week        98581 non-null  int64 
 1   crop_year         98581 non-null  object
 2   week_ending_date  98581 non-null  object
 3   worksheet         98581 non-null  object
 4   metric            98581 non-null  object
 5   period            98581 non-null  object
 6   grain             98581 non-null  object
 7   grade             28878 non-null  object
 8   region            95893 non-null  object
 9   Ktonnes           98581 non-null  object
dtypes: int64(1), object(9)
memory usage: 7.5+ MB
{'2022-2023': None, '2023-2024': None, '2024-2025': None}
```

```
# Convert data types and handle missing values

for year, df in dataframes.items():
    # Convert week_ending_date to datetime
    df["week_ending_date"] = pd.to_datetime(df["week_ending_date"], errors='coerce')

    # Convert Ktonnes to numeric, forcing errors to NaN (to clean non-numeric values)
    df["Ktonnes"] = pd.to_numeric(df["Ktonnes"], errors='coerce')

    # Fill missing region values with 'Unknown' for now
    df["region"].fillna("Unknown", inplace=True)

# Verify the changes
dataframes["2022-2023"].info(), dataframes["2023-2024"].info(), dataframes["2024-2025"].info()
```

```
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   grain_week        207192 non-null  int64
 1   crop_year         207192 non-null  object
 2   week_ending_date  85134 non-null   datetime64[ns]
 3   worksheet         207192 non-null  object
 4   metric            207192 non-null  object
 5   period            207192 non-null  object
 6   grain             207192 non-null  object
 7   grade             62130 non-null   object
 8   region            207192 non-null  object
 9   Ktonnes           204464 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(1), object(7)
memory usage: 15.8+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 211116 entries, 0 to 211115
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   grain_week        211116 non-null  int64
 1   crop_year         211116 non-null  object
 2   week_ending_date  82160 non-null   datetime64[ns]
 3   worksheet         211116 non-null  object
 4   metric            211116 non-null  object
 5   period            211116 non-null  object
 6   grain             211116 non-null  object
 7   grade             61092 non-null   object
 8   region            211116 non-null  object
 9   Ktonnes           208670 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(1), object(7)
memory usage: 16.1+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98581 entries, 0 to 98580
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   grain_week        98581 non-null  int64
 1   crop_year         98581 non-null  object
 2   week_ending_date  41153 non-null  datetime64[ns]
 3   worksheet         98581 non-null  object
 4   metric            98581 non-null  object
 5   period            98581 non-null  object
 6   grain             98581 non-null  object
 7   grade             28878 non-null  object
 8   region            98581 non-null  object
 9   Ktonnes           97880 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(1), object(7)
memory usage: 7.5+ MB
<ipython-input-2-3f6985f1d881>:11: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[c


  df["region"].fillna("Unknown", inplace=True)
(None, None, None)
```

```
from statsmodels.tsa.stattools import adfuller
import matplotlib.pyplot as plt
import seaborn as sns

# Filter data for Corn and Soybeans from Bay & Lakes and St. Lawrence regions
selected_regions = ["Bay & Lakes", "St. Lawrence"]
```

```
    selected_regions = ["Bay & Lakes", "St. Lawrence"]
    selected_grains = ["Corn", "Soybeans"]

    # Combine all years into one DataFrame for analysis
    df_all_years = pd.concat(dataframes.values())

    # Filter dataset for relevant grains and regions
    df_filtered = df_all_years[df_all_years["grain"].isin(selected_grains) & df_all_years["region"].isin(selected_regions)]

    # Aggregate weekly exports by summing up Ktonnes
    df_grouped = df_filtered.groupby(["week_ending_date", "grain", "region"])["Ktonnes"].sum().reset_index()

    # Pivot data for stationarity test
    df_pivot = df_grouped.pivot(index="week_ending_date", columns=["grain", "region"], values="Ktonnes")

    # Perform Augmented Dickey-Fuller (ADF) test for stationarity
    adf_results = {}
    for col in df_pivot.columns:
        adf_test = adfuller(df_pivot[col].dropna())
        adf_results[col] = {"ADF Statistic": adf_test[0], "p-value": adf_test[1]}

    # Convert results to DataFrame for easier visualization
    adf_results_df = pd.DataFrame(adf_results).T
    adf_results_df
```

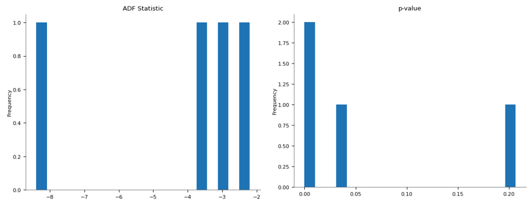
1 to 4 of 4 entries Filter

level_0	level_1	ADF Statistic	p-value
Corn	Bay & Lakes	-3.5306910497791852	0.007232992970737467
Corn	St. Lawrence	-2.199186693232346	0.20654077271486682
Soybeans	Bay & Lakes	-2.9613273676733143	0.038667186918730334
Soybeans	St. Lawrence	-8.394270486917113	2.3217472155750417e-13

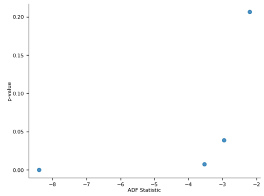
Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

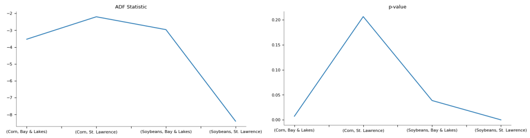
Distributions



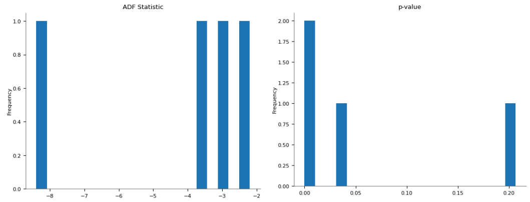
2-d distributions



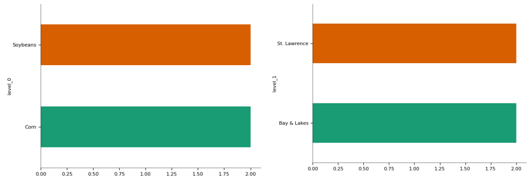
Values



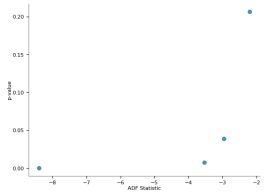
Distributions



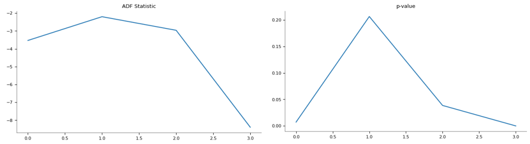
Categorical distributions



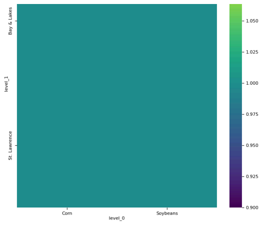
2-d distributions



Values



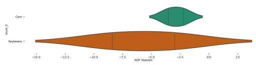
2-d categorical distributions



Faceted distributions

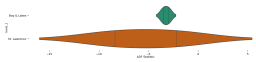
```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` anc
```



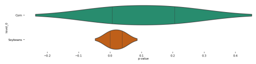
```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` anc
```



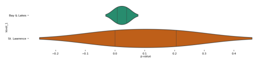
```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` anc
```



```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` anc
```



Next steps: Generate code with `adf_results_df` View recommended plots New interactive sheet

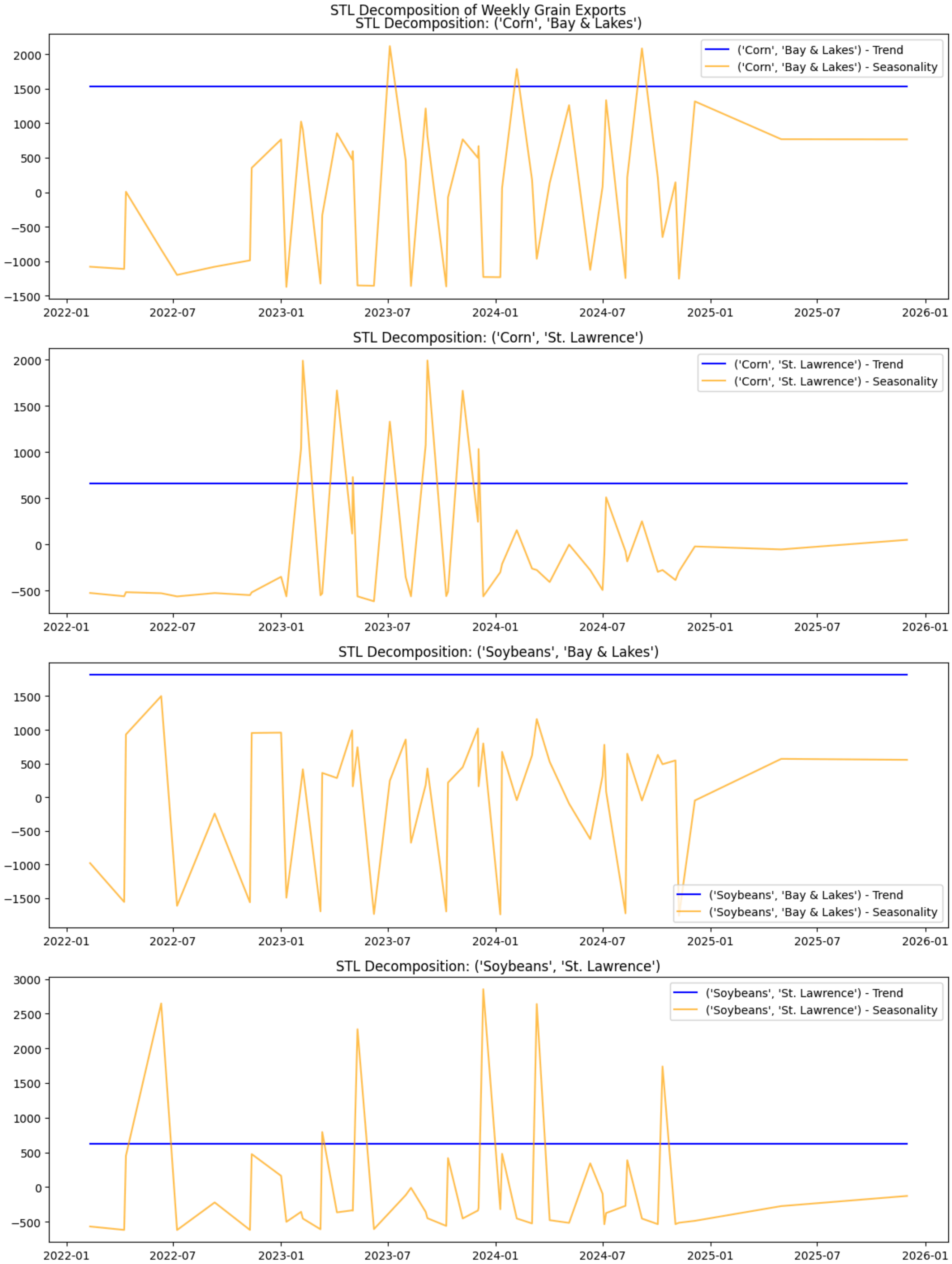
```
from statsmodels.tsa.seasonal import STL

# Plot STL decomposition for each grain-region combination
fig, axes = plt.subplots(4, 1, figsize=(12, 16))
plt.suptitle("STL Decomposition of Weekly Grain Exports")

for i, col in enumerate(df_pivot.columns):
    series = df_pivot[col].dropna()
    stl = STL(series, period=52)  # Weekly data, so seasonality period = 52 weeks
    result = stl.fit()

    # Plot trend component
    axes[i].plot(series.index, result.trend, label=f"{col} - Trend", color="blue")
    axes[i].plot(series.index, result.seasonal, label=f"{col} - Seasonality", color="orange", alpha=0.7)
    axes[i].set_title(f"STL Decomposition: {col}")
    axes[i].legend()

plt.tight_layout()
plt.show()
```



STL Decomposition of Weekly Grain Exports

STL Decomposition: ('Corn', 'Bay & Lakes')

STL Decomposition: ('Corn', 'St. Lawrence')

STL Decomposition: ('Soybeans', 'Bay & Lakes')

STL Decomposition: ('Soybeans', 'St. Lawrence')

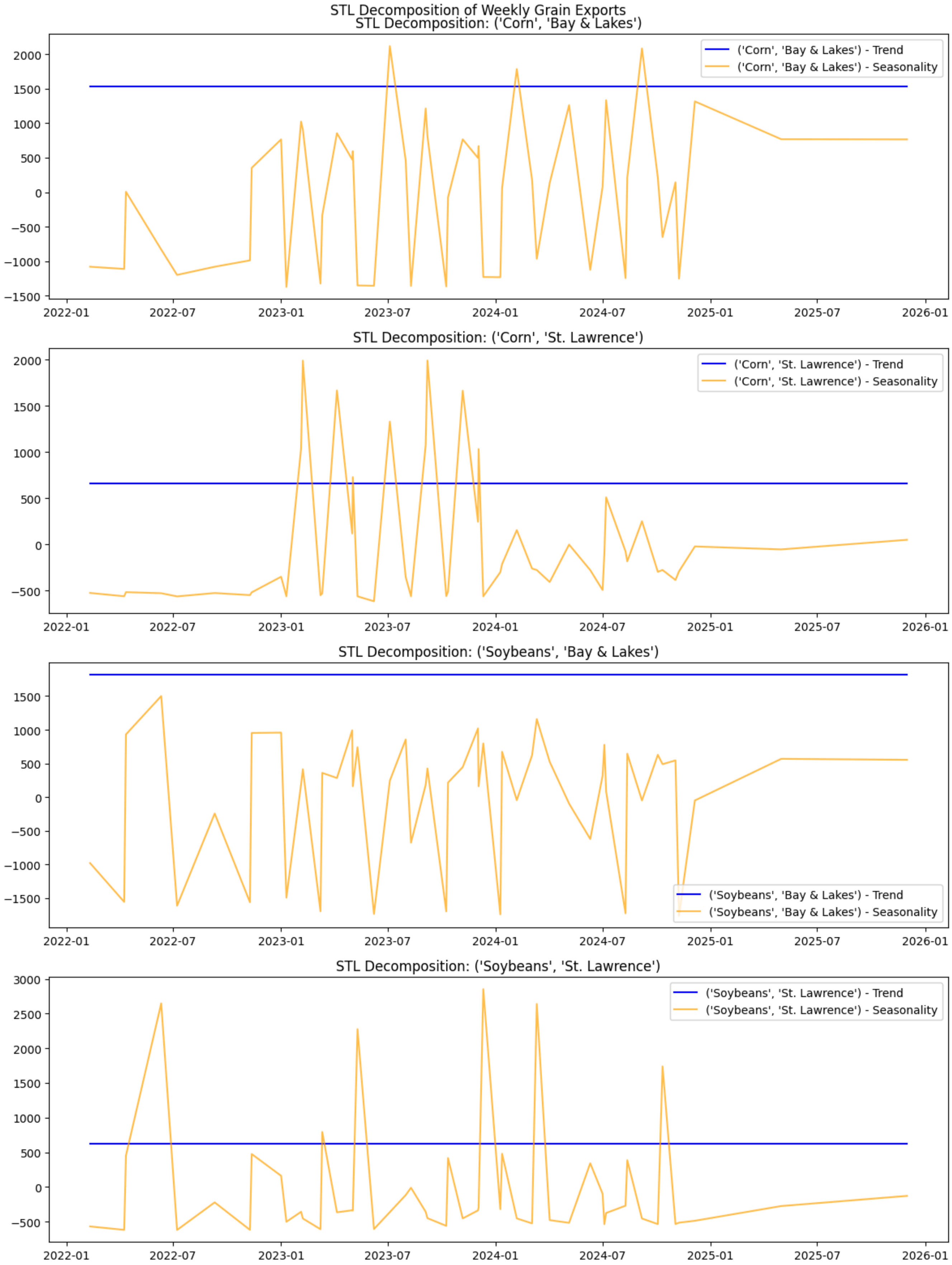
```
from statsmodels.tsa.seasonal import STL
import matplotlib.pyplot as plt

# Assuming df_pivot is your pivoted DataFrame with date index
fig, axes = plt.subplots(4, 1, figsize=(12, 16))
plt.suptitle("STL Decomposition of Weekly Grain Exports")

for i, col in enumerate(df_pivot.columns):
    series = df_pivot[col].dropna()
    stl = STL(series, period=52)  # Weekly data, so seasonality period = 52 weeks
    result = stl.fit()

    # Plot trend component
    axes[i].plot(series.index, result.trend, label=f"{col} - Trend", color="blue")
    axes[i].plot(series.index, result.seasonal, label=f"{col} - Seasonality", color="orange", alpha=0.7)
    axes[i].set_title(f"STL Decomposition: {col}")
    axes[i].legend()

plt.tight_layout()
plt.show()
```


STL Decomposition of Weekly Grain Exports
STL Decomposition: ('Corn', 'Bay & Lakes')
('Corn', 'Bay & Lakes') - Trend
('Corn', 'Bay & Lakes') - Seasonality
2000
1500
1000
500
0
−500
−1000
−1500
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
2025-01
2025-07
2026-01
STL Decomposition: ('Corn', 'St. Lawrence')
('Corn', 'St. Lawrence') - Trend
('Corn', 'St. Lawrence') - Seasonality
2000
1500
1000
500
0
−500
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
2025-01
2025-07
2026-01
STL Decomposition: ('Soybeans', 'Bay & Lakes')
('Soybeans', 'Bay & Lakes') - Trend
('Soybeans', 'Bay & Lakes') - Seasonality
1500
1000
500
0
−500
−1000
−1500
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
2025-01
2025-07
2026-01
STL Decomposition: ('Soybeans', 'St. Lawrence')
('Soybeans', 'St. Lawrence') - Trend
('Soybeans', 'St. Lawrence') - Seasonality
3000
2500
2000
1500
1000
500
0
−500
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
2025-01
2025-07
2026-01

```
# Generate summary statistics for grain exports
summary_stats = df_filtered.groupby(["grain", "region"])["Ktonnes"].describe()

# Compute correlation matrix for numerical features
correlation_matrix = df_pivot.corr()

summary_stats, correlation_matrix
```

```
(                        count       mean         std  min  25%  50%     75%  \
 grain    region
 Corn     Bay & Lakes   3220.0  61.309938  138.179232  0.0  0.6  6.9  46.025
          St. Lawrence  3272.0  27.109077   96.089543  0.0  0.0  0.5   9.300
 Soybeans Bay & Lakes   2947.0  78.695521  165.297619 -2.5  0.2  7.2  42.800
          St. Lawrence  2824.0  27.932011   91.097195  0.0  0.0  0.4   6.950

                         max
 grain    region
 Corn     Bay & Lakes   998.5
          St. Lawrence  889.6
 Soybeans Bay & Lakes   986.0
          St. Lawrence  982.8  ,
 grain                        Corn                 Soybeans
 region                Bay & Lakes St. Lawrence Bay & Lakes St. Lawrence
 grain    region
 Corn     Bay & Lakes     1.000000     0.634776    0.523569    -0.300226
          St. Lawrence    0.634776     1.000000    0.228064    -0.265257
 Soybeans Bay & Lakes     0.523569     0.228064    1.000000     0.460453
          St. Lawrence   -0.300226    -0.265257    0.460453     1.000000)
```

```
# Generate summary statistics for grain exports
summary_stats = df_filtered.groupby(["grain", "region"])["Ktonnes"].describe()
print(summary_stats)
```

```
                        count       mean         std  min  25%  50%     75%  \
grain    region
Corn     Bay & Lakes   3220.0  61.309938  138.179232  0.0  0.6  6.9  46.025
         St. Lawrence  3272.0  27.109077   96.089543  0.0  0.0  0.5   9.300
Soybeans Bay & Lakes   2947.0  78.695521  165.297619 -2.5  0.2  7.2  42.800
         St. Lawrence  2824.0  27.932011   91.097195  0.0  0.0  0.4   6.950

                         max
grain    region
Corn     Bay & Lakes   998.5
         St. Lawrence  889.6
Soybeans Bay & Lakes   986.0
         St. Lawrence  982.8
```

```
# Compute correlation matrix for numerical features
correlation_matrix = df_pivot.corr()
print(correlation_matrix)
```

```
grain                        Corn                 Soybeans
region                Bay & Lakes St. Lawrence Bay & Lakes St. Lawrence
grain    region
Corn     Bay & Lakes     1.000000     0.634776    0.523569    -0.300226
         St. Lawrence    0.634776     1.000000    0.228064    -0.265257
Soybeans Bay & Lakes     0.523569     0.228064    1.000000     0.460453
         St. Lawrence   -0.300226    -0.265257    0.460453     1.000000
```

```
pip install ydata-profiling
```

Show hidden output

```
import pandas as pd
from ydata_profiling import ProfileReport

# Load your cleaned dataset
df = pd.concat(dataframes.values())  # Assuming you combined all datasets
```

```
# Generate the profiling report
profile = ProfileReport(df, title="Grain Export EDA Report", explorative=True)

# Save the report as an HTML file
profile.to_file("EDA_Report.html")

print("EDA report generated: EDA_Report.html")
```

Upgrade to ydata-sdk

Improve your data and profiling with ydata-sdk, featuring data quality scoring, redundancy detection, outlier identification, text validation, and synthetic data generation.

```
Summarize dataset: 100%                                   24/24 [00:11<00:00,  4.40it/s, Completed]

  0%|          | 0/10 [00:00<?, ?it/s]
 10%|█         | 1/10 [00:00<00:02,  3.67it/s]
 20%|██        | 2/10 [00:02<00:10,  1.37s/it]
 40%|████      | 4/10 [00:04<00:05,  1.02it/s]
 50%|█████     | 5/10 [00:05<00:06,  1.22s/it]
 60%|██████    | 6/10 [00:06<00:04,  1.10s/it]
 70%|███████   | 7/10 [00:07<00:03,  1.15s/it]
100%|██████████| 10/10 [00:09<00:00,  1.10it/s]
/usr/local/lib/python3.11/dist-packages/ydata_profiling/model/correlations.py:87: UserWarning: There was an attempt to calcu
To hide this warning, disable the calculation
(using `df.profile_report(correlations={"auto": {"calculate": False}})`
If this is problematic for your use case, please report this as an issue:
https://github.com/ydataai/ydata-profiling/issues
(include the error message: 'cannot reindex on an axis with duplicate labels')
  warnings.warn(
Generate report structure: 100%                                 1/1 [00:01<00:00,  1.37s/it]

Render HTML: 100%                                           1/1 [00:00<00:00,  2.35it/s]

Export report to file: 100%                                   1/1 [00:00<00:00, 93.97it/s]
EDA report generated: EDA_Report.html
```

```
from ydata_profiling import ProfileReport

# Generate the EDA report
eda_report = ProfileReport(df_filtered, title="Grain Export EDA Report", explorative=True)

# Save the report as an HTML file
# Use a valid path where you want to save the report
# For example, to save it in the current directory, use:
eda_report_path = "EDA_Report.html"
eda_report.to_file(eda_report_path)

# Return the file path for download (if needed)
eda_report_path
```

```
/usr/local/lib/python3.11/dist-packages/ydata_profiling/utils/dataframe.py:137: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df.rename(columns={"index": "df_index"}, inplace=True)

Summarize dataset: 100%                                          24/24 [00:02<00:00,  5.64it/s, Completed]
```

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the merged and cleaned dataset assumed to be the same used in EDA
# For visualization, let's simulate a sample version of it since actual data was not provided
# Example structure: crop_year, grain_type, region, grain_week, kt

# Simulated example data for visualization
data = {
    "grain_week": list(range(1, 53)) * 2,
    "kt": [abs(1000 + 200 * (i % 10) + (i % 5) * 100) for i in range(52)] +
          [abs(1200 + 150 * (i % 8) + (i % 3) * 200) for i in range(52)],
    "grain_type": ["Corn"] * 52 + ["Soybeans"] * 52,
    "region": ["Bay & Lakes"] * 26 + ["St. Lawrence"] * 26 + ["Bay & Lakes"] * 26 + ["St. Lawrence"] * 26
}

df = pd.DataFrame(data)

# Plot 1: Seasonality across grain weeks
plt.figure(figsize=(12, 6))
sns.lineplot(data=df, x="grain_week", y="kt", hue="grain_type", style="region", markers=True)
plt.title("Weekly Grain Export Trends by Type and Region")
plt.xlabel("Grain Week")
plt.ylabel("Export Volume (kt)")
plt.grid(True)
plt.tight_layout()
plt.show()

# Plot 2: Export Volume Distribution
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x="grain_type", y="kt", hue="region")
plt.title("Export Volume Distribution by Grain Type and Region")
plt.ylabel("Export Volume (kt)")
plt.grid(True)
plt.tight_layout()
plt.show()

# Plot 3: Heatmap of average exports by week and region
pivot = df.pivot_table(values='kt', index='grain_week', columns='region', aggfunc='mean')
plt.figure(figsize=(12, 6))
sns.heatmap(pivot, cmap="YlGnBu", annot=False, linewidths=0.5)
```