Weather + Lice Analysis

In [1]:

```python
from pathlib import Path
import os
import sys
import warnings

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tools.sm_exceptions import ConvergenceWarning

warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=ConvergenceWarning)


def find_project_root() -> Path:
    cwd = Path.cwd().resolve()
    for candidate in [cwd, cwd.parent]:
        if (candidate / "CA2").exists() and (candidate / "data").exists():
            return candidate
    return cwd


PROJECT_ROOT = find_project_root()
DATA_DIR = PROJECT_ROOT / "data"
CA2_DIR = PROJECT_ROOT / "CA2"
OUTPUT_DIR = CA2_DIR / "outputs"
OUTPUT_DIR.mkdir(exist_ok=True)

USE_CASSANDRA = 1
TARGET_YEAR = 2022

print(f"PROJECT_ROOT: {PROJECT_ROOT}")
print(f"USE_CASSANDRA: {USE_CASSANDRA}")
print(f"TARGET_YEAR: {TARGET_YEAR}")
```

```
PROJECT_ROOT: /Users/endreasgard/NMBU/IND320/IND320
USE_CASSANDRA: 1
TARGET_YEAR: 2022
```

1) Load locality dataset and create pivot analyses

In [2]:

```python
def classify_region(lat: float) -> str:
    if pd.isna(lat):
        return "Unknown"
    if lat < 63.81:
        return "South"
    if lat <= 68.85:
        return "Middle"
    return "North"


def load_locations() -> pd.DataFrame:
    candidates = [
        DATA_DIR / "locations_df.csv",
        DATA_DIR / "all.csv",
        PROJECT_ROOT / "CA1" / "fishhealth.csv",
    ]
    for candidate in candidates:
        if candidate.exists():
            print(f"Loading locality data from: {candidate}")
            return pd.read_csv(candidate)
    raise FileNotFoundError("No locality dataset found in expected paths.")


locations = load_locations().copy()

rename_map = {
    "localityNo": "localityno",
    "avgAdultFemaleLice": "avgadultfemalelice",
    "hasPd": "haspd",
    "hasIla": "hasila",
}
locations = locations.rename(columns=rename_map)

required = ["week", "avgadultfemalelice", "lat", "haspd", "hasila"]
missing = [c for c in required if c not in locations.columns]
if missing:
    raise ValueError(f"Missing locality columns: {missing}")

if "region" not in locations.columns:
    locations["region"] = locations["lat"].apply(classify_region)
else:
    locations["region"] = locations["lat"].apply(classify_region)

if "year" in locations.columns:
    year_filtered = locations[locations["year"] == TARGET_YEAR]
    if not year_filtered.empty:
        locations = year_filtered

week_lice = (
    locations.groupby("week", dropna=True)["avgadultfemalelice"]
    .mean()
    .dropna()
    .sort_index()
)

pd_ila_lat = (
    locations.pivot_table(
        index=["haspd", "hasila"],
        values="lat",
        aggfunc="mean",
    )
    .reset_index()
    .sort_values(["haspd", "hasila"])
)

week_region_pivot = (
    locations.pivot_table(
        index="week",
        columns="region",
        values="avgadultfemalelice",
        aggfunc="mean",
    )
    .sort_index()
)

print(f"Locations rows used: {len(locations):,}")
print("PD/ILA pivot sample:")
print(pd_ila_lat.head(10))
week_region_pivot.head(10)
```

```
Loading locality data from: /Users/endreasgard/NMBU/IND320/IND320/data/location
s_df.csv
Locations rows used: 89,084
PD/ILA pivot sample:
   haspd  hasila        lat
0  False   False  64.088320
1  False    True  64.956952
2   True   False  61.868237
3   True    True  59.296028
```

Out[2]:

region	Middle	North	South
week			
1	0.140383	0.119362	0.178955
2	0.139032	0.115978	0.184441
3	0.147090	0.140115	0.188786
4	0.150056	0.147386	0.191480
5	0.166250	0.127595	0.200366
6	0.153314	0.136375	0.195725
7	0.163457	0.130303	0.191679
8	0.150616	0.158971	0.190577
9	0.148274	0.097931	0.190888
10	0.144562	0.145738	0.189809

In [3]:

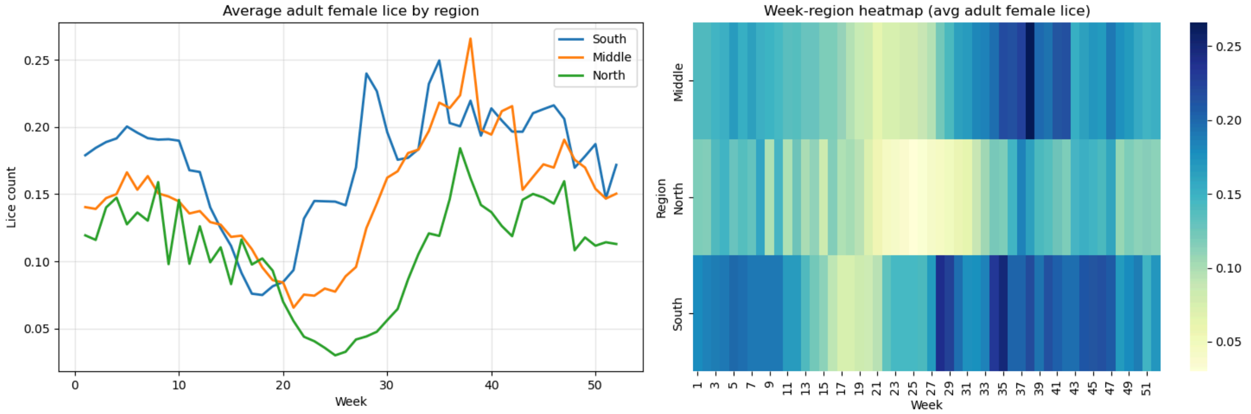
```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

for region in [c for c in ["South", "Middle", "North"] if c in week_region_piv
    axes[0].plot(week_region_pivot.index, week_region_pivot[region], label=reg

axes[0].set_title("Average adult female lice by region")
axes[0].set_xlabel("Week")
axes[0].set_ylabel("Lice count")
axes[0].grid(alpha=0.3)
axes[0].legend()

heat_df = week_region_pivot.copy()
if not heat_df.empty:
    sns.heatmap(heat_df.T, cmap="YlGnBu", ax=axes[1])
    axes[1].set_title("Week-region heatmap (avg adult female lice)")
    axes[1].set_xlabel("Week")
    axes[1].set_ylabel("Region")

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



2) Load synchronized weather + lice features (single locality)

In [4]:

```
def load_weather_lice_features() -> pd.DataFrame:
    preferred = DATA_DIR / "sar_weather_lice.csv"
    if preferred.exists():
        print(f"Loading merged feature data from: {preferred}")
        return pd.read_csv(preferred)

    lice_candidates = [DATA_DIR / "lice_df_2.csv", DATA_DIR / "lice.csv"]
    weather_candidates = [DATA_DIR / "weather_lice_lag.csv", DATA_DIR / "weath

    lice_df = None
    weather_df = None

    for candidate in lice_candidates:
        if candidate.exists():
            lice_df = pd.read_csv(candidate)
            break

    for candidate in weather_candidates:
        if candidate.exists():
            weather_df = pd.read_csv(candidate)
            break

    if lice_df is None or weather_df is None:
        raise FileNotFoundError("Could not build feature dataset from local fi

    lice_df = lice_df.rename(columns={
        "avgAdultFemaleLice": "avgadultfemalelice",
        "avgMobileLice": "avgmobilelice",
        "avgStationaryLice": "avgstationarylice",
        "seaTemperature": "seatemperature",
    })

    weather_df = weather_df.rename(columns={
        "mean(air_temperature P1D)": "mean_air_temperature",
        "mean(relative_humidity P1D)": "mean_relative_humidity",
        "mean(wind_speed P1D)": "mean_wind_speed",
        "sum(precipitation_amount P1D)": "sum_precipitation_amount",
    })

    merge_keys = [k for k in ["year", "week"] if k in lice_df.columns and k in
    if not merge_keys and "week" in lice_df.columns and "week" in weather_df.c
        merge_keys = ["week"]

    df = pd.merge(lice_df, weather_df, how="inner", on=merge_keys)

    if "smoothed_avgadultfemalelice" not in df.columns and "avgadultfemalelice
        df["smoothed_avgadultfemalelice"] = df["avgadultfemalelice"].rolling(w
    if "smoothed_avgmobilelice" not in df.columns and "avgmobilelice" in df.co
        df["smoothed_avgmobilelice"] = df["avgmobilelice"].rolling(window=2).m
    if "smoothed_avgstationarylice" not in df.columns and "avgstationarylice"
        df["smoothed_avgstationarylice"] = df["avgstationarylice"].rolling(wir

    for base_col in ["mean_air_temperature", "mean_relative_humidity", "mean_w
        lag_col = f"{base_col}_lag1"
        if base_col in df.columns and lag_col not in df.columns:
            df[lag_col] = df[base_col].shift(1)

    return df


feature_df = load_weather_lice_features().copy()
feature_df = feature_df.sort_values("week").reset_index(drop=True)

core_cols = [
    "week",
    "seatemperature",
    "avgadultfemalelice",
    "avgmobilelice",
    "avgstationarylice",
    "mean_air_temperature",
    "mean_relative_humidity",
    "mean_wind_speed",
    "sum_precipitation_amount",
    "mean_air_temperature_lag1",
    "mean_relative_humidity_lag1",
    "mean_wind_speed_lag1",
    "sum_precipitation_amount_lag1",
]

missing_core = [c for c in ["week", "seatemperature", "avgadultfemalelice", "a
if missing_core:
    raise ValueError(f"Missing required feature columns: {missing_core}")

for col in core_cols:
    if col in feature_df.columns:
        feature_df[col] = pd.to_numeric(feature_df[col], errors="coerce")

feature_df = feature_df.dropna(subset=["week", "seatemperature", "avgadultfema

print(f"Feature rows available: {len(feature_df)}")
feature_df.head(10)
```

```
Loading merged feature data from: /Users/endreasgard/NMBU/IND320/IND320/data/sa
r_weather_lice.csv
Feature rows available: 26
```

Out[4]:

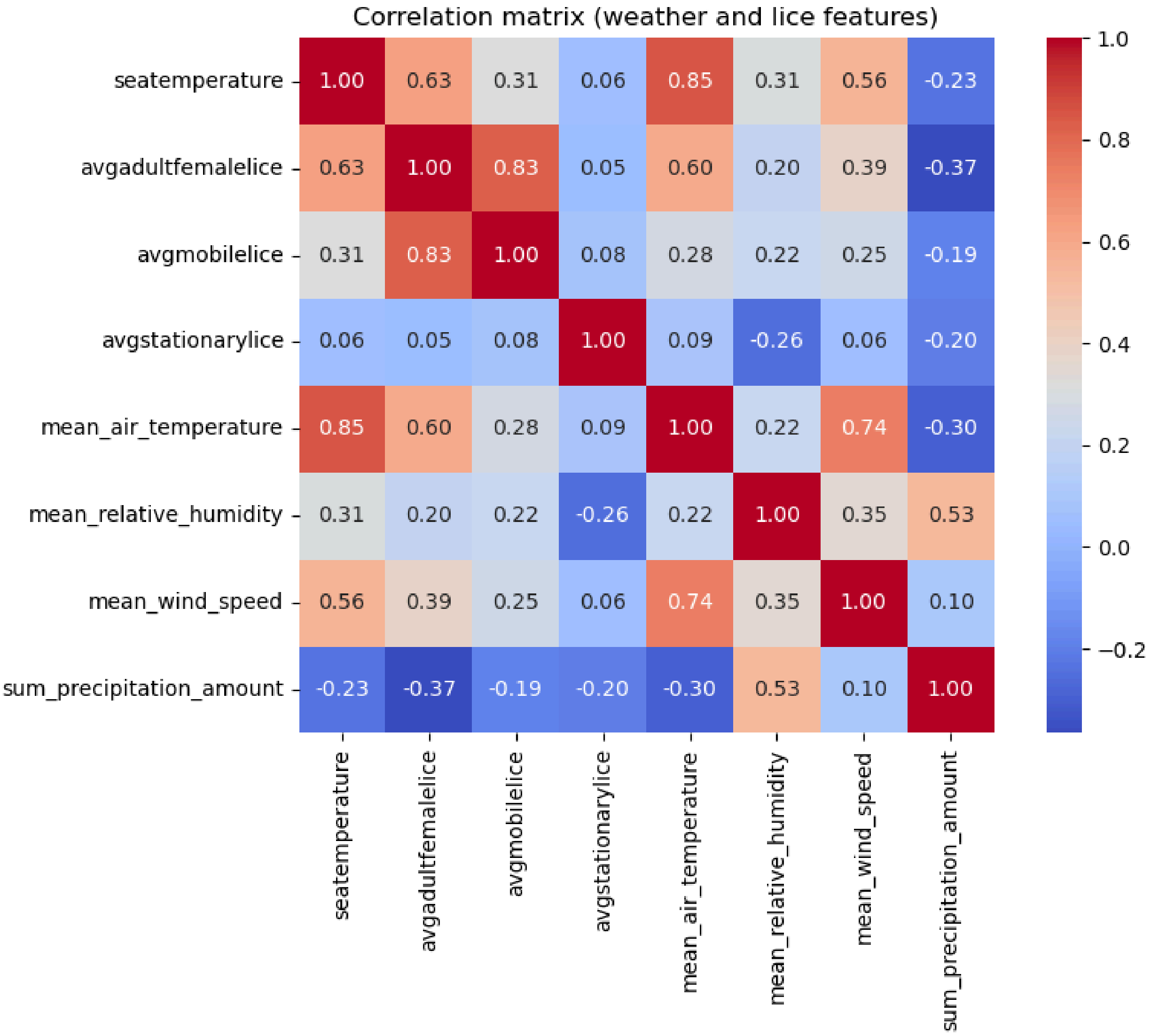
	week	avgadultfemalelice	avgmobilelice	avgstationarylice	mean_air_temper
0	3	0.04	0.29	0.0	3.45
1	4	0.05	0.27	0.0	1.98
2	5	0.07	0.27	0.0	-1.85
3	6	0.06	0.23	0.0	1.42
4	7	0.09	0.42	0.0	-1.71
5	9	0.12	0.55	0.0	3.41
6	11	0.07	0.38	0.0	6.52
7	13	0.19	0.47	0.0	0.52
8	15	0.12	0.58	0.0	5.45
9	16	0.16	0.69	0.0	8.80

In [5]:

```
corr_cols = [
    "seatemperature",
    "avgadultfemalelice",
    "avgmobilelice",
    "avgstationarylice",
    "mean_air_temperature",
    "mean_relative_humidity",
    "mean_wind_speed",
    "sum_precipitation_amount",
]

corr_cols = [c for c in corr_cols if c in feature_df.columns]
corr = feature_df[corr_cols].corr(numeric_only=True)

plt.figure(figsize=(9, 7))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", square=True)
plt.title("Correlation matrix (weather and lice features)")
plt.tight_layout()
plt.show()
```



3) Sliding Window Correlation (sea temperature vs lice)

```
In [6]: def sliding_window_correlation(temp: pd.Series, lice: pd.Series, window: int,
    """Positive lag means sea temperature leads lice by `lag` weeks."""
    temp = temp.reset_index(drop=True)
    lice = lice.reset_index(drop=True)

    if lag > 0:
        temp_aligned = temp.iloc[:-lag].reset_index(drop=True)
        lice_aligned = lice.iloc[lag:].reset_index(drop=True)
    elif lag < 0:
        temp_aligned = temp.iloc[-lag:].reset_index(drop=True)
        lice_aligned = lice.iloc[:lag].reset_index(drop=True)
    else:
        temp_aligned = temp
        lice_aligned = lice

    max_start = len(temp_aligned) - window
    if max_start < 0:
        return pd.Series(dtype=float)

    corr_values = []
    window_index = []

    for start in range(max_start + 1):
        temp_slice = temp_aligned.iloc[start : start + window]
        lice_slice = lice_aligned.iloc[start : start + window]
        corr_values.append(temp_slice.corr(lice_slice))
        window_index.append(start + 1)

    return pd.Series(corr_values, index=window_index)


def find_best_window_lag(temp: pd.Series, lice: pd.Series, windows, lags):
    best = None
    for window in windows:
        for lag in lags:
            corr_series = sliding_window_correlation(temp, lice, window=window
            if corr_series.empty:
                continue
            score = corr_series.abs().mean()
            if best is None or score > best["score"]:
                best = {
                    "window": window,
                    "lag": lag,
                    "score": float(score),
                    "series": corr_series,
                }
    return best


temp_series = feature_df["seatemperature"].astype(float)
lice_series_map = {
    "avgadultfemalelice": feature_df["avgadultfemalelice"].astype(float),
    "avgmobilelice": feature_df["avgmobilelice"].astype(float),
    "avgstationarylice": feature_df["avgstationarylice"].astype(float),
}

windows = [4, 6, 8, 10]
lags = [0, 1, 2, 3]

best_configs = {}
for key, lice_series in lice_series_map.items():
    best = find_best_window_lag(temp_series, lice_series, windows=windows, lag
    best_configs[key] = best

best_summary = pd.DataFrame(
    [
        {
            "series": key,
            "best_window": cfg["window"],
            "best_lag": cfg["lag"],
            "mean_abs_correlation": cfg["score"],
        }
        for key, cfg in best_configs.items()
        if cfg is not None
    ]
)

best_summary
```

```
/Users/endreasgard/miniconda3/envs/personlig311/lib/python3.11/site-packages/nu
mpy/lib/_function_base_impl.py:3065: RuntimeWarning: invalid value encountered
in divide
  c /= stddev[:, None]
/Users/endreasgard/miniconda3/envs/personlig311/lib/python3.11/site-packages/nu
mpy/lib/_function_base_impl.py:3066: RuntimeWarning: invalid value encountered
in divide
  c /= stddev[None, :]
```

Out[6]:

	series	best_window	best_lag	mean_abs_correlation
0	avgadultfemalelice	4	0	0.610241
1	avgmobilelice	4	0	0.606241
2	avgstationarylice	6	3	0.774091

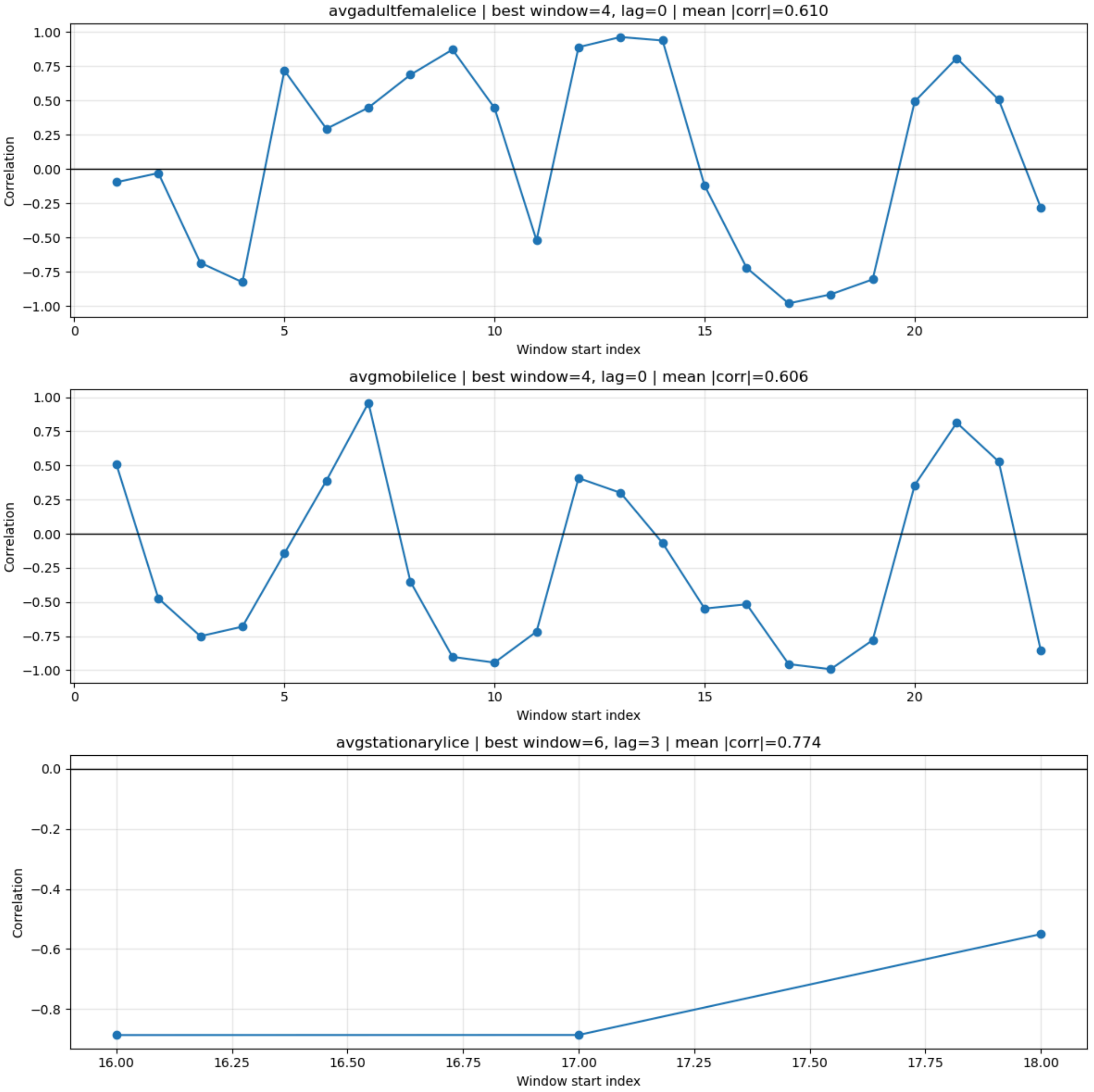
In [7]:

```
fig, axes = plt.subplots(3, 1, figsize=(12, 12), sharex=False)

for ax, (series_name, cfg) in zip(axes, best_configs.items()):
    if cfg is None:
        ax.set_title(f"{series_name}: no valid window/lag config")
        continue

    corr_series = cfg["series"]
    ax.plot(corr_series.index, corr_series.values, marker="o")
    ax.axhline(0, color="black", linewidth=1)
    ax.set_title(
        f"{series_name} | best window={cfg['window']}, lag={cfg['lag']} | "
        f"mean |corr|={cfg['score']:.3f}"
    )
    ax.set_xlabel("Window start index")
    ax.set_ylabel("Correlation")
    ax.grid(alpha=0.3)

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



4) ARIMAX forecasting for sea temperature

Model A: weather variables only. Model B: weather + lagged weather variables. Model C: Model B minus the least influential variable.

In [8]:

```
def fit_sarimax_with_holdout(df: pd.DataFrame, exog_cols, label: str):
    model_df = df[["week", "seatemperature"] + exog_cols].dropna().reset_index
    if len(model_df) < 14:
        raise ValueError(f"Not enough rows to train/test for {label}.")

    y = model_df["seatemperature"].astype(float)
    X = model_df[exog_cols].astype(float)

    split_idx = max(12, len(model_df) - 6)
    y_train, y_test = y.iloc[:split_idx], y.iloc[split_idx:]
    X_train, X_test = X.iloc[:split_idx], X.iloc[split_idx:]

    model = SARIMAX(
        y_train,
        exog=X_train,
        order=(1, 0, 0),
        trend="c",
        enforce_stationarity=False,
        enforce_invertibility=False,
    )
    result = model.fit(disp=False, maxiter=200)

    forecast = result.get_forecast(steps=len(y_test), exog=X_test).predicted_m
    rmse = float(np.sqrt(np.mean((y_test.values - forecast.values) ** 2)))

    output = {
        "label": label,
        "result": result,
        "aic": float(result.aic),
        "rmse": rmse,
        "weeks_test": model_df["week"].iloc[split_idx:].to_numpy(),
        "actual_test": y_test.to_numpy(),
        "pred_test": forecast.to_numpy(),
        "exog_cols": list(exog_cols),
    }
    return output


base_cols = [
    "mean_air_temperature",
    "mean_relative_humidity",
    "mean_wind_speed",
    "sum_precipitation_amount",
]
lag_cols = [
    "mean_air_temperature_lag1",
    "mean_relative_humidity_lag1",
    "mean_wind_speed_lag1",
    "sum_precipitation_amount_lag1",
]

base_cols = [c for c in base_cols if c in feature_df.columns]
lag_cols = [c for c in lag_cols if c in feature_df.columns]
```

```python
model_a = fit_sarimax_with_holdout(feature_df, base_cols, "A_base_weather")
model_b = fit_sarimax_with_holdout(feature_df, base_cols + lag_cols, "B_base_p

params_b = model_b["result"].params.drop(labels=["sigma2"], errors="ignore")
coef_b = params_b[[c for c in params_b.index if c in model_b["exog_cols"]]]
least_influential = coef_b.abs().sort_values().index[0]

reduced_cols = [c for c in model_b["exog_cols"] if c != least_influential]
model_c = fit_sarimax_with_holdout(feature_df, reduced_cols, "C_reduced")

metrics = pd.DataFrame([
    {"model": model_a["label"], "aic": model_a["aic"], "rmse_test": model_a["r
    {"model": model_b["label"], "aic": model_b["aic"], "rmse_test": model_b["r
    {"model": model_c["label"], "aic": model_c["aic"], "rmse_test": model_c["r
]).sort_values("rmse_test")

metrics
```

Out[8]:

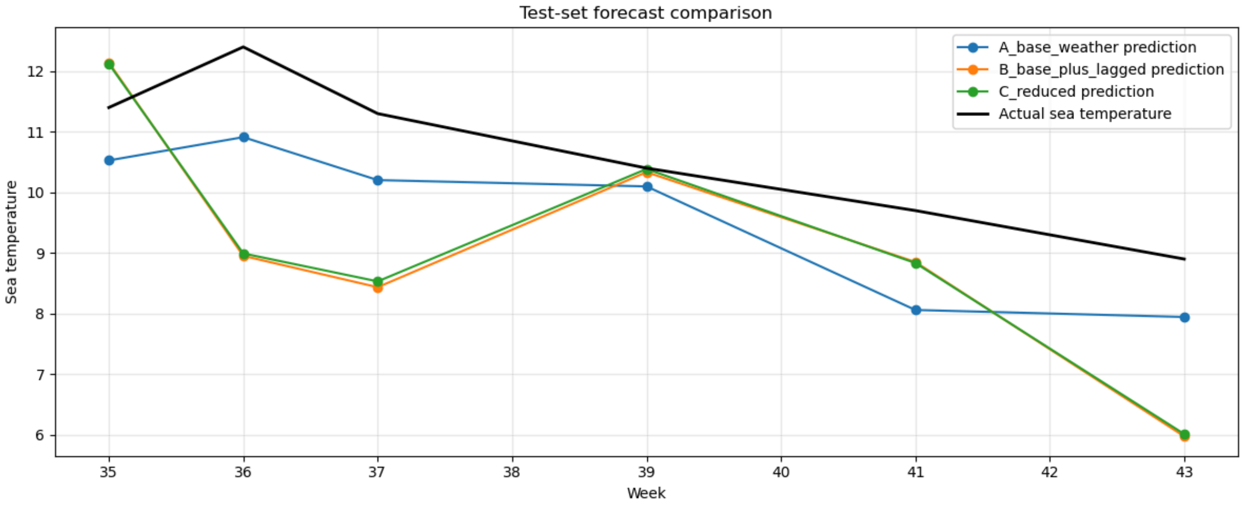
	model	aic	rmse_test	n_exog	removed_variable
0	A_base_weather	91.542499	1.145872	4	-
2	C_reduced	71.647437	2.194353	7	mean_relative_humidity_lag1
1	B_base_plus_lagged	73.636098	2.233651	8	-

In [9]:

```python
plt.figure(figsize=(12, 5))

for model in [model_a, model_b, model_c]:
    plt.plot(model["weeks_test"], model["pred_test"], marker="o", label=f"{moc

plt.plot(model_a["weeks_test"], model_a["actual_test"], color="black", linewic
plt.title("Test-set forecast comparison")
plt.xlabel("Week")
plt.ylabel("Sea temperature")
plt.grid(alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()
```



5) Cassandra export

In [10]:

```
if USE_CASSANDRA:
    try:
        from cassandra.cluster import Cluster

        cluster = Cluster(["localhost"], port=9042)
        session = cluster.connect()
        session.execute(
            """
            CREATE KEYSPACE IF NOT EXISTS fishhealth
            WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor
            """
        )
        session.set_keyspace("fishhealth")

        session.execute(
            """
            CREATE TABLE IF NOT EXISTS weather_lice_cv (
                week int,
                seatemperature double,
                avgadultfemalelice double,
                avgmobilelice double,
                avgstationarylice double,
                mean_air_temperature double,
                mean_relative_humidity double,
                mean_wind_speed double,
                sum_precipitation_amount double,
                PRIMARY KEY (week)
            )
            """
        )

        cols = [
            "week",
            "seatemperature",
            "avgadultfemalelice",
            "avgmobilelice",
            "avgstationarylice",
            "mean_air_temperature",
            "mean_relative_humidity",
            "mean_wind_speed",
            "sum_precipitation_amount",
        ]

        insert_stmt = session.prepare(
            """
            INSERT INTO weather_lice_cv
            (week, seatemperature, avgadultfemalelice, avgmobilelice, avgstati
             mean_air_temperature, mean_relative_humidity, mean_wind_speed, su
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
            """
        )

        inserted = 0
        for row in feature_df[cols].dropna(subset=["week"]).itertuples(index=F
            session.execute(
                insert_stmt,
                (
                    int(row.week),
                    float(row.seatemperature) if pd.notna(row.seatemperature)
                    float(row.avgadultfemalelice) if pd.notna(row.avgadultfema
                    float(row.avgmobilelice) if pd.notna(row.avgmobilelice) el
                    float(row.avgstationarylice) if pd.notna(row.avgstationary
                    float(row.mean_air_temperature) if pd.notna(row.mean_air_t
                    float(row.mean_relative_humidity) if pd.notna(row.mean_rel
                    float(row.mean_wind_speed) if pd.notna(row.mean_wind_speec
                    float(row.sum_precipitation_amount) if pd.notna(row.sum_pr
                ),
            )
            inserted += 1

        print(f"Inserted {inserted} rows into fishhealth.weather_lice_cv")

    except Exception as exc:
        print(f"Cassandra step skipped due to error: {exc}")
    finally:
        if "cluster" in locals():
            cluster.shutdown()
else:
    print("Skipping Cassandra step. Set BW_USE_CASSANDRA=1 to enable.")
```

```
Inserted 26 rows into fishhealth.weather_lice_cv
```

In [11]:

```
week_region_out = OUTPUT_DIR / "week_region_pivot.csv"
pd_ila_out = OUTPUT_DIR / "pd_ila_lat_pivot.csv"
features_out = OUTPUT_DIR / "weather_lice_features.csv"
metrics_out = OUTPUT_DIR / "forecast_metrics.csv"
sliding_out = OUTPUT_DIR / "sliding_window_best_configs.csv"
```

```
week_region_pivot.reset_index().to_csv(week_region_out, index=False)
pd_ila_lat.to_csv(pd_ila_out, index=False)
feature_df.to_csv(features_out, index=False)
metrics.to_csv(metrics_out, index=False)
best_summary.to_csv(sliding_out, index=False)

print(f"Saved: {week_region_out}")
print(f"Saved: {pd_ila_out}")
print(f"Saved: {features_out}")
print(f"Saved: {metrics_out}")
print(f"Saved: {sliding_out}")
```

```
Saved: /Users/endreasgard/NMBU/IND320/IND320/CA2/outputs/week_region_pivot.csv
Saved: /Users/endreasgard/NMBU/IND320/IND320/CA2/outputs/pd_ila_lat_pivot.csv
Saved: /Users/endreasgard/NMBU/IND320/IND320/CA2/outputs/weather_lice_feature
s.csv
Saved: /Users/endreasgard/NMBU/IND320/IND320/CA2/outputs/forecast_metrics.csv
Saved: /Users/endreasgard/NMBU/IND320/IND320/CA2/outputs/sliding_window_best_co
nfigs.csv
```