Fish Health Pipeline

In [41]:
```python
from pathlib import Path
import os
import sys
import warnings

import matplotlib.pyplot as plt
import pandas as pd

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


def find_project_root() -> Path:
    cwd = Path.cwd().resolve()
    candidates = [cwd, cwd.parent]
    for candidate in candidates:
        if (candidate / "CA1").exists() and (candidate / "data").exists():
            return candidate
    return cwd


PROJECT_ROOT = find_project_root()
CA1_DIR = PROJECT_ROOT / "CA1"
DATA_DIR = PROJECT_ROOT / "data"
OUTPUT_DIR = CA1_DIR / "outputs"
OUTPUT_DIR.mkdir(exist_ok=True)

if str(CA1_DIR) not in sys.path:
    sys.path.insert(0, str(CA1_DIR))

TARGET_YEAR = 2022
TARGET_LOCALITY = 18755
USE_API = 1
USE_CASSANDRA = 1

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

```
PROJECT_ROOT: /Users/endreasgard/NMBU/IND320/IND320
TARGET_YEAR: 2022
TARGET_LOCALITY: 18755
USE_API: 1
USE_CASSANDRA: 1
```

1) Load locality-level fish health data

The notebook tries API first only when `BW_USE_API=1`; otherwise it loads local files.

```
In [42]: def normalize_localities(df: pd.DataFrame) -> pd.DataFrame:
             rename_map = {
                 "localityNo": "localityno",
                 "localityWeekId": "localityweekid",
                 "avgAdultFemaleLice": "avgadultfemalelice",
                 "hasCleanerfishDeployed": "hascleanerfishdeployed",
                 "hasMechanicalRemoval": "hasmechanicalremoval",
                 "hasSubstanceTreatments": "hassubstancetreatments",
                 "hasReportedLice": "hasreportedlice",
                 "hasSalmonoids": "hassalmonoids",
                 "isFallow": "isfallow",
                 "isOnLand": "isonland",
                 "isSlaughterHoldingCage": "isslaughterholdingcage",
                 "inFilteredSelection": "infilteredselection",
                 "municipalityNo": "municipalityno",
             }
             out = df.rename(columns=rename_map).copy()

             # API payloads may contain both camelCase and snake_case keys that collide
             duplicate_cols = out.columns[out.columns.duplicated()].unique()
             for col in duplicate_cols:
                 dup_df = out.loc[:, out.columns == col]
                 merged = dup_df.iloc[:, 0].copy()
                 for idx in range(1, dup_df.shape[1]):
                     merged = merged.combine_first(dup_df.iloc[:, idx])
                 out = out.drop(columns=col)
                 out[col] = merged

             for col in ["localityno", "week", "year"]:
                 if col in out.columns:
                     out[col] = pd.to_numeric(out[col], errors="coerce")

             for col in ["localityno", "week", "year"]:
                 if col in out.columns:
                     out[col] = out[col].astype("Int64")

             return out


         def load_localities_from_csv() -> pd.DataFrame:
             candidates = [
                 DATA_DIR / "locations_df.csv",
                 DATA_DIR / "all.csv",
                 CA1_DIR / "fishhealth.csv",
             ]
             for candidate in candidates:
```

```python
        if candidate.exists():
            print(f"Loading localities from: {candidate}")
            return pd.read_csv(candidate)
    raise FileNotFoundError("Could not find local fallback data for localities


def fetch_localities_from_api(year: int) -> pd.DataFrame:
    from tokenizer import Tokenizer
    from weeksummary import WeekSummary

    token = Tokenizer().get_token()
    client = WeekSummary(token)

    frames = []
    for week in range(1, 53):
        payload = client.get_summary(year, week)
        rows = payload.get("localities", [])
        if not rows:
            continue
        frame = pd.DataFrame(rows)
        frame["week"] = week
        frame["year"] = year
        frames.append(frame)

    if not frames:
        raise RuntimeError("API returned no locality rows.")

    return pd.concat(frames, ignore_index=True)


if USE_API:
    try:
        localities_raw = fetch_localities_from_api(TARGET_YEAR)
        localities_source = "BarentsWatch API"
    except Exception as exc:
        print(f"API fetch failed, using local fallback. Reason: {exc}")
        localities_raw = load_localities_from_csv()
        localities_source = "Local CSV fallback"
else:
    localities_raw = load_localities_from_csv()
    localities_source = "Local CSV fallback"

localities_df = normalize_localities(localities_raw)

if "year" in localities_df.columns:
    filtered = localities_df[localities_df["year"] == TARGET_YEAR]
    if not filtered.empty:
        localities_df = filtered

print(f"Loaded {len(localities_df):,} rows from {localities_source}.")
localities_df.head(5)
```

```
Token request successful
Loaded 89,070 rows from BarentsWatch API.
```

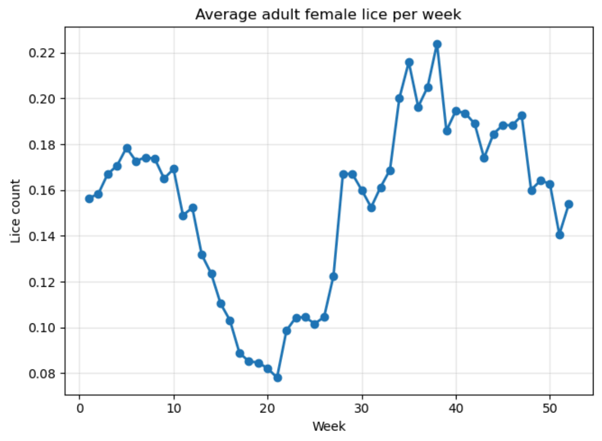
Out[42]:

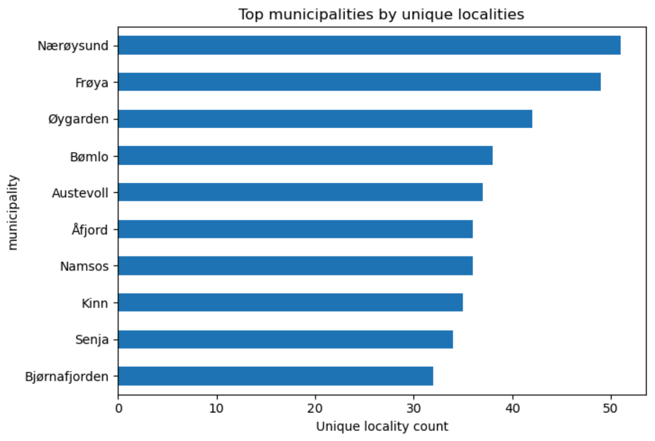
	localityno	localityweekid	name	hasreportedlice	isfallow	avgadultfem
0	14746	1344789	Aarsand	False	True	
1	31937	1345158	Abelsnes	False	True	
2	10665	1344175	Adamselv	False	True	
3	29196	1345110	Adjetjohka	False	True	
4	13823	1344733	Ådlandsvatn	False	True	

5 rows × 21 columns

In [43]:

```
required_columns = ["week", "avgadultfemalelice", "municipality", "localityno"
missing = [c for c in required_columns if c not in localities_df.columns]
if missing:
    raise ValueError(f"Missing required locality columns: {missing}")

weekly_lice = (
    localities_df.groupby("week", dropna=True)["avgadultfemalelice"]
    .mean()
    .dropna()
    .sort_index()
)

municipality_top = (
    localities_df.groupby("municipality", dropna=True)["localityno"]
    .nunique()
    .sort_values(ascending=False)
    .head(10)
)

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

axes[0].plot(weekly_lice.index, weekly_lice.values, marker="o", linewidth=2)
axes[0].set_title("Average adult female lice per week")
axes[0].set_xlabel("Week")
axes[0].set_ylabel("Lice count")
axes[0].grid(alpha=0.3)

municipality_top.sort_values().plot(kind="barh", ax=axes[1])
axes[1].set_title("Top municipalities by unique localities")
axes[1].set_xlabel("Unique locality count")

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





2) Load lice and sea temperature for one locality

Default fallback uses `data/lice_df_2.csv` or `data/lice.csv`.

In [44]:
```
def normalize_lice(df: pd.DataFrame) -> pd.DataFrame:
    rename_map = {
        "localityNo": "localityno",
        "avgAdultFemaleLice": "avgadultfemalelice",
        "avgMobileLice": "avgmobilelice",
        "avgStationaryLice": "avgstationarylice",
        "hasReportedLice": "hasreportedlice",
        "seaTemperature": "seatemperature",
    }
    out = df.rename(columns=rename_map).copy()

    # Handle duplicate labels (e.g. both localityNo and localityno in API reco
    duplicate_cols = out.columns[out.columns.duplicated()].unique()
    for col in duplicate_cols:
        dup_df = out.loc[:, out.columns == col]
        merged = dup_df.iloc[:, 0].copy()
        for idx in range(1, dup_df.shape[1]):
            merged = merged.combine_first(dup_df.iloc[:, idx])
        out = out.drop(columns=col)
        out[col] = merged

    for col in ["localityno", "week", "year"]:
        if col in out.columns:
            out[col] = pd.to_numeric(out[col], errors="coerce")
            out[col] = out[col].astype("Int64")

    return out


def load_lice_from_csv() -> pd.DataFrame:
    candidates = [
        DATA_DIR / "lice_df_2.csv",
```

```python
        DATA_DIR / "lice.csv",
    ]
    for candidate in candidates:
        if candidate.exists():
            print(f"Loading lice data from: {candidate}")
            return pd.read_csv(candidate)
    raise FileNotFoundError("Could not find local fallback data for lice.")


def fetch_lice_from_api(year: int, localityno: int) -> pd.DataFrame:
    from tokenizer import Tokenizer
    from weeksummary import WeekSummary

    token = Tokenizer().get_token()
    client = WeekSummary(token)

    rows = []
    for week in range(1, 53):
        payload = client.get_summary_lice(year, week, localityno)

        record = payload.get("localityWeek") or payload.get("liceCountPrevious
        if not isinstance(record, dict):
            continue

        record = dict(record)
        record["week"] = week
        record["year"] = year
        record["localityno"] = localityno
        rows.append(record)

    if not rows:
        raise RuntimeError("API returned no lice rows.")

    return pd.DataFrame(rows)


if USE_API:
    try:
        lice_raw = fetch_lice_from_api(TARGET_YEAR, TARGET_LOCALITY)
        lice_source = "BarentsWatch API"
    except Exception as exc:
        print(f"API lice fetch failed, using local fallback. Reason: {exc}")
        lice_raw = load_lice_from_csv()
        lice_source = "Local CSV fallback"
else:
    lice_raw = load_lice_from_csv()
    lice_source = "Local CSV fallback"

lice_df = normalize_lice(lice_raw)

if "localityno" in lice_df.columns:
    candidate = lice_df[lice_df["localityno"] == TARGET_LOCALITY]
    if not candidate.empty:
        lice_df = candidate

if "year" in lice_df.columns:
    candidate = lice_df[lice_df["year"] == TARGET_YEAR]
    if not candidate.empty:
        lice_df = candidate

lice_df = lice_df.sort_values("week") if "week" in lice_df.columns else lice_c

print(f"Loaded {len(lice_df):,} lice rows from {lice_source}.")
lice_df.head(5)
```

```
Token request successful
Loaded 52 lice rows from BarentsWatch API.
```

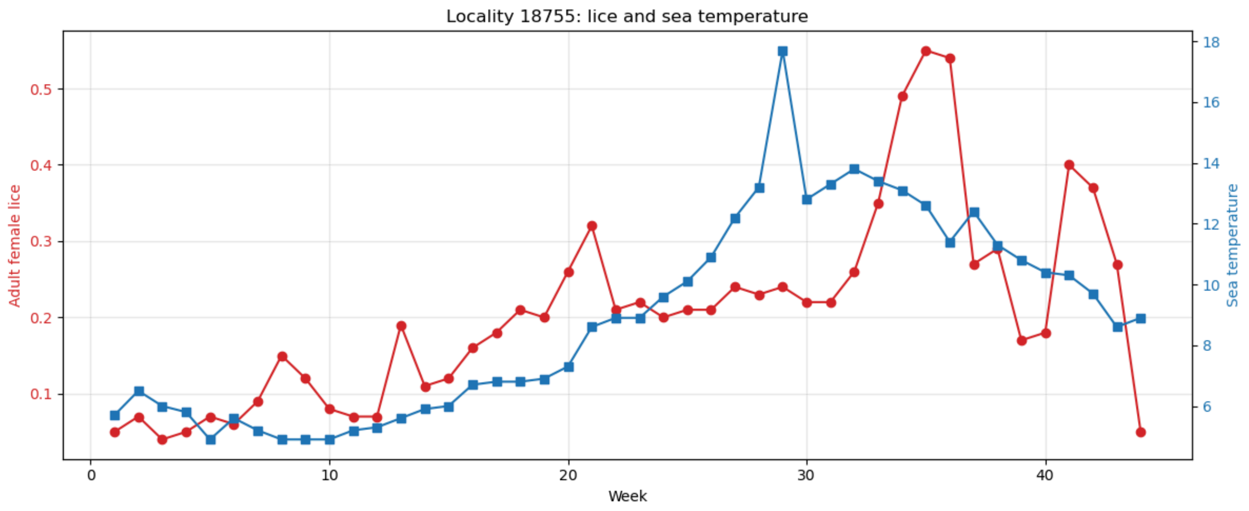
Out[44]:

	id	year	week	hasreportedlice	hasMechanicalRemoval	hasBathTreatn
0	1343568	2022	1	True	False	I
1	1347137	2022	2	True	False	I
2	1349662	2022	3	True	False	I
3	1352712	2022	4	True	False	I
4	1355238	2022	5	True	False	I

5 rows × 25 columns

In [45]:

```
required_lice = ["week", "avgadultfemalelice"]
missing_lice = [c for c in required_lice if c not in lice_df.columns]
if missing_lice:
    raise ValueError(f"Missing required lice columns: {missing_lice}")

fig, ax1 = plt.subplots(figsize=(12, 5))

ax1.plot(
    lice_df["week"],
    lice_df["avgadultfemalelice"],
    color="tab:red",
    marker="o",
    label="Adult female lice",
)
ax1.set_xlabel("Week")
ax1.set_ylabel("Adult female lice", color="tab:red")
ax1.tick_params(axis="y", labelcolor="tab:red")
ax1.grid(alpha=0.3)

if "seatemperature" in lice_df.columns:
    ax2 = ax1.twinx()
    ax2.plot(
        lice_df["week"],
        lice_df["seatemperature"],
        color="tab:blue",
        marker="s",
        label="Sea temperature",
    )
    ax2.set_ylabel("Sea temperature", color="tab:blue")
    ax2.tick_params(axis="y", labelcolor="tab:blue")

plt.title(f"Locality {TARGET_LOCALITY}: lice and sea temperature")
plt.tight_layout()
plt.show()
```



3) Cassandra write

Runs only if `BW_USE_CASSANDRA=1` and Cassandra is reachable on `localhost:9042`.

In [46]:
```
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 locality_summary_cv (
                localityno int,
                week int,
                year int,
                name text,
                municipality text,
                avgadultfemalelice double,
                PRIMARY KEY ((localityno), week, year)
            )
            """
        )

        sample = (
            localities_df[["localityno", "week", "year", "name", "municipality
            .dropna(subset=["localityno", "week", "year"])
            .head(200)
        )

        statement = session.prepare(
            """
            INSERT INTO locality_summary_cv
            (localityno, week, year, name, municipality, avgadultfemalelice)
            VALUES (?, ?, ?, ?, ?, ?)
            """
        )

        inserted = 0
        for row in sample.itertuples(index=False):
            session.execute(
                statement,
                (
                    int(row.localityno),
                    int(row.week),
                    int(row.year),
                    str(row.name) if pd.notna(row.name) else "",
                    str(row.municipality) if pd.notna(row.municipality) else "
                    float(row.avgadultfemalelice) if pd.notna(row.avgadultfema
                ),
            )
            inserted += 1

        print(f"Inserted {inserted} sample rows into fishhealth.locality_summa

    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 200 sample rows into fishhealth.locality_summary_cv
```

In [47]:

```
locality_out = OUTPUT_DIR / f"localities_{TARGET_YEAR}.csv"
lice_out = OUTPUT_DIR / f"lice_locality_{TARGET_LOCALITY}_{TARGET_YEAR}.csv"

localities_df.to_csv(locality_out, index=False)
lice_df.to_csv(lice_out, index=False)

print(f"Saved: {locality_out}")
print(f"Saved: {lice_out}")
```

```
Saved: /Users/endreasgard/NMBU/IND320/IND320/CA1/outputs/localities_2022.csv
Saved: /Users/endreasgard/NMBU/IND320/IND320/CA1/outputs/lice_locality_18755_20
22.csv
```