ZIP CODE-Level Home Price Forecasting & Material-Decline Risk

Stakeholder: Mortgage lender (portfolio risk / underwriting)
Horizon: 1 month ahead

Targets (supervised):

1) **Regression:** predict next month home value (price level)
2) **Classification (derived):** alert if next month return ≤ **-1%** (“material decline”)

This notebook demonstrates clear business framing, leakage-safe preparation, multiple models (baseline (ARMA) + ARMA + ARIMA + SARIMAX), evaluation against baseline (ARMA), and a demo that works for any ZIP.

Domain grounding sources (Independent Learning):

- Zillow Research Data (ZHVI-style): (https://www.zillow.com/research/data/)
- FHFA House Price Index (HPI): (https://www.fhfa.gov/DataTools/Downloads/Pages/House-Price-Index.aspx)

1) Business Understanding

Mortgage lenders are exposed to changes in collateral values. A ZIP-level forecast can support:

- underwriting adjustments (risk-based pricing / tighter requirements)
- portfolio monitoring (identify weakening ZIPs early)
- reserves and risk reporting

Predictive question: Using historical monthly home values, can we forecast next month’s value for each ZIP? From that forecast, can we flag when the next month is likely to drop by **≥ 1%**?

Because missing a true decline is costly, the alert logic leans toward **recall**.

In [1]:
```
# Imports & global settings (single import cell for reproducibility)
import warnings
import logging

import numpy as np
import pandas as pd
import re
import matplotlib.pyplot as plt


from sklearn.metrics import mean_absolute_error, mean_squared_error, confusion_ma

from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tools.sm_exceptions import ConvergenceWarning

# --- Warning & logging control (keeps notebook output clean) ---
warnings.filterwarnings("ignore")
warnings.filterwarnings("ignore", category=ConvergenceWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)
logging.getLogger("statsmodels").setLevel(logging.ERROR)

# Global settings
RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)

pd.set_option("display.max_columns", 200)
pd.set_option("display.max_rows", 200)

%matplotlib inline
```

In [2]:
```
from matplotlib.ticker import FuncFormatter

def dollars_thousands(x, pos=None):
    """Format axis ticks so that 546 -> $546,000"""
    try:
        return f"${int(x):,}000"
    except Exception:
        return x

def apply_currency_axis(ax, axis='y'):
    if axis == 'y':
        ax.yaxis.set_major_formatter(FuncFormatter(dollars_thousands))
    else:
        ax.xaxis.set_major_formatter(FuncFormatter(dollars_thousands))
```

2) Data Understanding

The CSV is in a wide format (ZIP rows × monthly columns). I load the file, confirm the basic schema, then separate ID columns from date columns.

In [3]:
```
DATA_PATH = "https://raw.githubusercontent.com/rahroy82/home_price_forecast/refs/
raw = pd.read_csv(DATA_PATH)
raw.head(3)
```

Out[3]:

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	
0	84654	60657	Chicago	IL	Chicago	Cook	1	334200.0	335400.0	3
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	236900.0	2
2	91982	77494	Katy	TX	Houston	Harris	3	210400.0	212200.0	2

3 rows × 272 columns

In [4]:
```
# Identify ID columns vs time columns (YYYY-MM)
id_cols = [c for c in raw.columns if not str(c)[:4].isdigit()]
date_cols = [c for c in raw.columns if str(c)[:4].isdigit()]

dates = pd.to_datetime(date_cols, errors="coerce").to_period("M").to_timestamp()
valid = ~pd.isna(dates)
date_cols = list(pd.Series(date_cols)[valid].iloc[np.argsort(dates[valid])])
dates = pd.to_datetime(date_cols).to_period("M").to_timestamp()

print("ZIP rows:", raw.shape[0])
print("Monthly columns:", len(date_cols))
print("Date range:", dates.min().date(), "→", dates.max().date())
```

```
ZIP rows: 14723
Monthly columns: 265
Date range: 1996-04-01 → 2018-04-01
```

3) Data Preparation

I reshape wide → long so time splitting is straightforward. Then I create a chronological train/validation/test split and handle missingness with forward-fill that does not use future values.

In [5]:
```
# Wide → long reshape
long_df = raw.melt(id_vars=id_cols, value_vars=date_cols, var_name="date", value_
long_df["date"] = pd.to_datetime(long_df["date"], errors="coerce").dt.to_period('
long_df = long_df.dropna(subset=["date"])

# Standardize ZIP formatting
long_df = long_df.rename(columns={"RegionName":"zip"})
long_df["zip"] = long_df["zip"].astype(str).str.replace(r"\.0$", "", regex=True)

TRAIN_END = pd.Timestamp("2014-12-01")
VAL_END   = pd.Timestamp("2016-12-01")

long_df["split"] = np.where(long_df["date"] <= TRAIN_END, "train",
                    np.where(long_df["date"] <= VAL_END, "val", "test"))

long_df = long_df.sort_values(["zip","date"])
long_df.head()
```

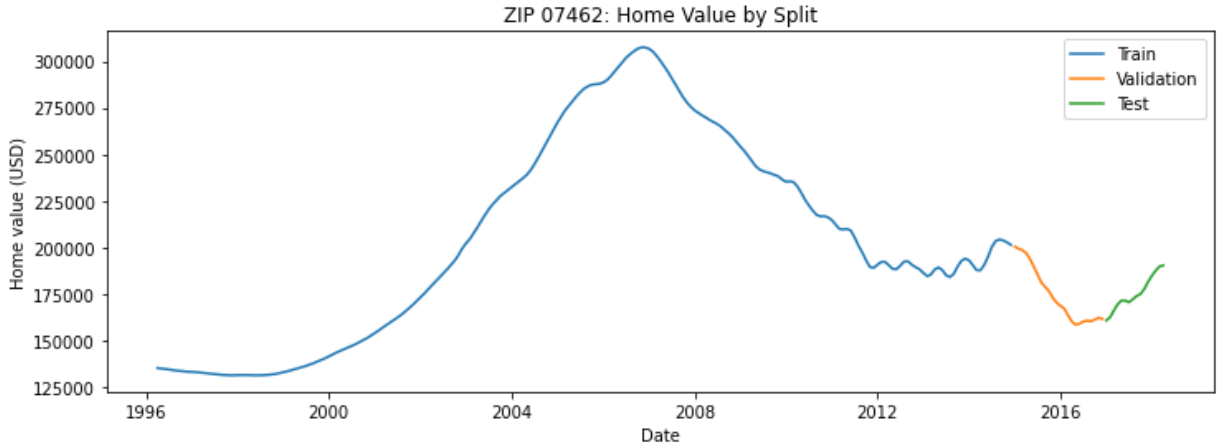
Out[5]:

	RegionID	zip	City	State	Metro	CountyName	SizeRank	date	value	split
5850	58196	01001	Agawam	MA	Springfield	Hampden	5851	1996-04-01	113100.0	train
20573	58196	01001	Agawam	MA	Springfield	Hampden	5851	1996-05-01	112800.0	train
35296	58196	01001	Agawam	MA	Springfield	Hampden	5851	1996-06-01	112600.0	train
50019	58196	01001	Agawam	MA	Springfield	Hampden	5851	1996-07-01	112300.0	train
64742	58196	01001	Agawam	MA	Springfield	Hampden	5851	1996-08-01	112100.0	train

In [6]:
```
# Train-only completeness (used for demo ZIP pool)
train_completeness = (long_df[long_df["split"].eq("train")]
                      .groupby("zip")["value"]
                      .apply(lambda s: s.notna().mean())
                      .sort_values(ascending=False))

zip_pool = train_completeness.head(200).index.tolist()
EXAMPLE_ZIP = "07462"
EXAMPLE_ZIP
```

Out[6]: '07462'

In [7]:
```
def split_series_by_zip(zip_code: str):
    """Return (train, val, test) series for the selected ZIP."""
    df = long_df[long_df["zip"].eq(str(zip_code).zfill(5))][["date","value","spli
    df = df.set_index("date").sort_index()
    train = df[df["split"].eq("train")]["value"]
    val   = df[df["split"].eq("val")]["value"]
    test  = df[df["split"].eq("test")]["value"]
    return train, val, test

def ffill_no_leak_levels(train: pd.Series, val: pd.Series, test: pd.Series):
    """Forward-fill with leakage control: seed val with last train value, seed te
    train_f = train.ffill()
    val_f = pd.concat([train_f.iloc[[-1]], val]).ffill().iloc[1:] if not train_f.
    test_f = pd.concat([val_f.iloc[[-1]], test]).ffill().iloc[1:] if not val_f.dr
    return train_f, val_f, test_f

train_s, val_s, test_s = split_series_by_zip(EXAMPLE_ZIP)
train_f, val_f, test_f = ffill_no_leak_levels(train_s, val_s, test_s)

plt.figure(figsize=(12,4))
plt.plot(train_f.index, train_f.values, label="Train")
plt.plot(val_f.index, val_f.values, label="Validation")
plt.plot(test_f.index, test_f.values, label="Test")

plt.title(f"ZIP {EXAMPLE_ZIP}: Home Value by Split")
plt.xlabel("Date")
plt.ylabel("Home value (USD)")
plt.legend()
plt.show()
```



4) Time Series Exploration - Stationarity + ACF/PACF

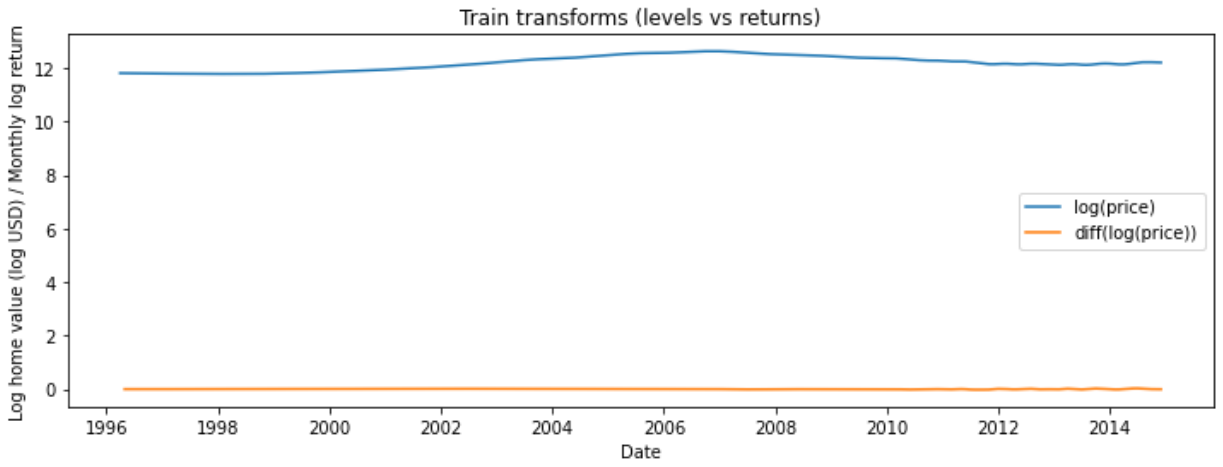
I check stationarity using ADF and KPSS and then visualize ACF/PACF on **diff(log(price))**, which is commonly used as a return-like stationary series. These steps are the diagnostics that lead into ARMA/ARIMA/SARIMAX modeling.

In [8]:
```
# Train transforms
train_log = np.log(train_f.dropna())
train_dlog = train_log.diff().dropna()

adf_log_p = adfuller(train_log, autolag="AIC")[1]
adf_dlog_p = adfuller(train_dlog, autolag="AIC")[1]

try:
    kpss_log_p = kpss(train_log, regression="c", nlags="auto")[1]
except Exception:
    kpss_log_p = np.nan

try:
    kpss_dlog_p = kpss(train_dlog, regression="c", nlags="auto")[1]
except Exception:
    kpss_dlog_p = np.nan

pd.DataFrame({
    "Series": ["log(price)", "diff(log(price))"],
    "ADF p-value": [adf_log_p, adf_dlog_p],
    "KPSS p-value": [kpss_log_p, kpss_dlog_p],
})
```

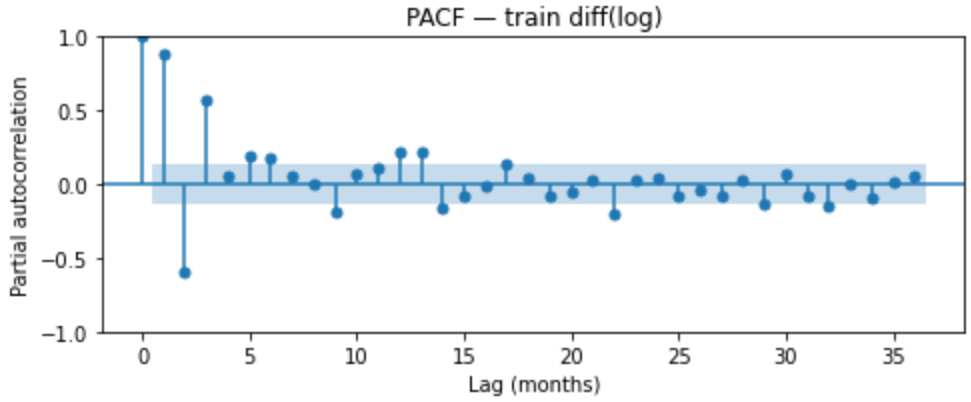
Out[8]:

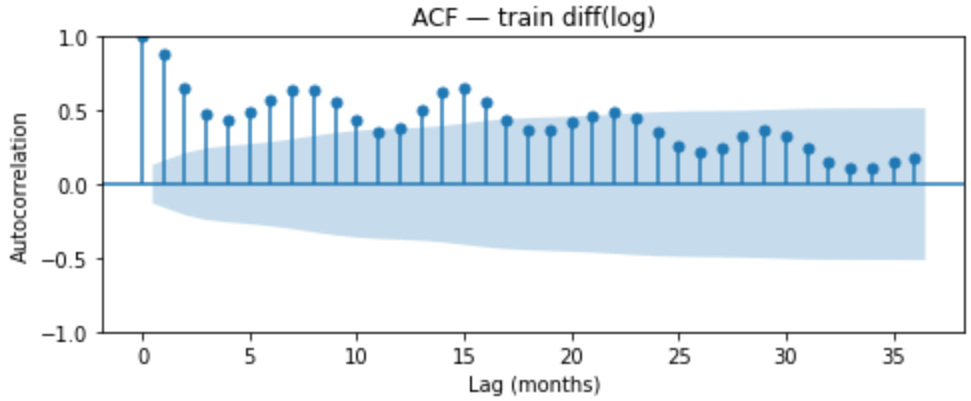
	Series	ADF p-value	KPSS p-value
0	log(price)	0.023021	0.01
1	diff(log(price))	0.665565	0.01

In [9]:
```
plt.figure(figsize=(12,4))
plt.plot(train_log.index, train_log.values, label="log(price)")
plt.plot(train_dlog.index, train_dlog.values, label="diff(log(price))")

plt.title("Train transforms (levels vs returns)")
plt.xlabel("Date")
plt.ylabel("Log home value (log USD) / Monthly log return")
plt.legend()
plt.show()
```



In [10]:
```
#from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

lags = min(36, len(train_dlog) - 1)

# --- PACF ---,
fig, ax = plt.subplots(figsize=(7,3))
plot_pacf(train_dlog, lags=lags, method="ywm", ax=ax)
ax.set_title("PACF — train diff(log)")
ax.set_xlabel("Lag (months)")
ax.set_ylabel("Partial autocorrelation")
plt.tight_layout()
plt.show()

# --- ACF ---
#plt.figure(figsize=(12,3))
fig, ax = plt.subplots(figsize=(7,3))
plot_acf(train_dlog, lags=lags, ax=ax)
ax.set_title("ACF — train diff(log)")
ax.set_xlabel("Lag (months)")
ax.set_ylabel("Autocorrelation")
plt.tight_layout()
plt.show()
```





ACF and PACF summarize how strongly the series relates to its own past values at different lags. The shaded band is the ~95% confidence interval; bars outside the band suggest statistically meaningful autocorrelation at that lag.

PACF: “Shows correlation between returns and a given lag after removing effects of shorter lags; helps choose AR order p.”

5) Modeling: ARMA(Baseline) → ARIMA → SARIMAX

I begin with an ARMA model as the baseline, establishing short-term dynamics without trend or seasonality.

I then extend the baseline by fitting ARIMA models to capture trend through integration, and SARIMAX models to capture seasonality.

Small parameter grids are used to keep runtime reasonable while still demonstrating a complete, iterative modeling process.

This decomposition breaks the (log) home value series into trend, seasonal, and residual components. It helps confirm whether there is a repeating annual pattern worth capturing with seasonal terms in a SARIMAX model. Helps assess whether there is evidence of a repeating annual pattern that may be worth capturing with seasonal terms in a SARIMAX model.

In [11]:
```
def evaluate_regression_aligned(y_true: pd.Series, y_pred: pd.Series) -> dict:
    """Align timestamps, drop missing pairs, compute MAE/RMSE."""
    df = pd.concat([y_true.rename("y"), y_pred.rename("yhat")], axis=1).dropna()
    return {
        "MAE": float(mean_absolute_error(df["y"], df["yhat"])),
        "RMSE": float(np.sqrt(mean_squared_error(df["y"], df["yhat"]))),
        "n": int(len(df))
    }
```

These metrics quantify forecast accuracy (MAE, RMSE) after aligning timestamps. Residual diagnostics (trend inspection and ACF plots) are examined separately to assess whether the model leaves systematic structure unmodeled.

In [12]:
```
# ARMA grid on diff(log(price))
arma_grid = [(1,0,1), (1,0,2), (2,0,1)]
best_arma = None
best_arma_order = None
best_arma_pred = None

tr_log = np.log(train_f.dropna())
va_log = np.log(val_f.dropna())
dlog = tr_log.diff().dropna()

for order in arma_grid:
    try:
        m = ARIMA(dlog, order=order, enforce_stationarity=False, enforce_invertib
        pred_ret = pd.Series(m.forecast(steps=len(va_log)), index=va_log.index)
        pred_log = tr_log.iloc[-1] + pred_ret.cumsum()
        pred_price = np.exp(pred_log)

        met = evaluate_regression_aligned(val_f.loc[pred_price.index], pred_price
        if best_arma is None or met["RMSE"] < best_arma["RMSE"]:
            best_arma = met
            best_arma_order = order
            best_arma_pred = pred_price
    except Exception:
        continue

best_arma_order, best_arma
```

Out[12]:
```
((1, 0, 1), {'MAE': 25744.089797974768, 'RMSE': 30246.40358704504, 'n': 24})
```

In [13]:
```
# ARIMA grid on log(price)
arima_grid = [(0,1,1), (1,1,1), (1,1,2)]
best_arima = None
best_arima_order = None
best_arima_pred = None

for order in arima_grid:
    try:
        m = ARIMA(tr_log, order=order, enforce_stationarity=False, enforce_invert
        pred_log = pd.Series(m.forecast(steps=len(va_log)), index=va_log.index)
        pred_price = np.exp(pred_log)

        met = evaluate_regression_aligned(val_f.loc[pred_price.index], pred_price
        if best_arima is None or met["RMSE"] < best_arima["RMSE"]:
            best_arima = met
            best_arima_order = order
            best_arima_pred = pred_price
    except Exception:
        continue

best_arima_order, best_arima
```

Out[13]:
```
((1, 1, 1), {'MAE': 20915.078136523032, 'RMSE': 24611.292250686114, 'n': 24})
```

In [14]:
```
# SARIMA grid on log(price) with yearly seasonality (s=12)
# Using simple_differencing=False ensures forecast is in log(level) space (safe t

sarima_grid = [
    ((0,1,1), (1,0,1,12)),
    ((1,1,1), (1,0,1,12)),
    ((1,1,2), (1,0,1,12)),
]

best_sarima = None
best_sarima_params = None
best_sarima_pred = None

for order, seas in sarima_grid:
    try:
        m = SARIMAX(tr_log, order=order, seasonal_order=seas,
                    enforce_stationarity=False, enforce_invertibility=False,
                    simple_differencing=False).fit(disp=False, maxiter=50)

        pred_log = pd.Series(m.forecast(steps=len(va_log)), index=va_log.index)
        pred_price = np.exp(pred_log)

        met = evaluate_regression_aligned(val_f.loc[pred_price.index], pred_price
        if best_sarima is None or met["RMSE"] < best_sarima["RMSE"]:
            best_sarima = met
            best_sarima_params = (order, seas)
            best_sarima_pred = pred_price
    except Exception:
        continue

best_sarima_params, best_sarima
```

Out[14]:
```
(((1, 1, 1), (1, 0, 1, 12)),
 {'MAE': 15633.456796442706, 'RMSE': 18214.075682119124, 'n': 24})
```

This chart compares the final selected model's test forecasts against the actual values for the same months. The gap between the lines is the month-ahead forecast error the lender would experience if using the model for this ZIP.

In [15]:
```
# Validation comparison (baseline = ARMA, then ARIMA, then SARIMAX)
val_table = pd.DataFrame([
    {"Model": f"ARMA {best_arma_order} (Baseline)", "Val_RMSE": best_arma["RMSE"]
    {"Model": f"ARIMA {best_arima_order}", "Val_RMSE": best_arima["RMSE"], "Val_M
    {"Model": f"SARIMAX {best_sarima_params[0]}×{best_sarima_params[1]}", "Val_RM
]).sort_values("Val_RMSE").reset_index(drop=True)

val_table
```

Out[15]:

	Model	Val_RMSE	Val_MAE
0	SARIMAX (1, 1, 1)×(1, 0, 1, 12)	18214.075682	15633.456796
1	ARIMA (1, 1, 1)	24611.292251	20915.078137
2	ARMA (1, 0, 1) (Baseline)	30246.403587	25744.089798

6) Final Evaluation (Holdout Test)

I choose the best model from validation and evaluate on the holdout test with a rolling 1-step-ahead forecast for the last 24 months.

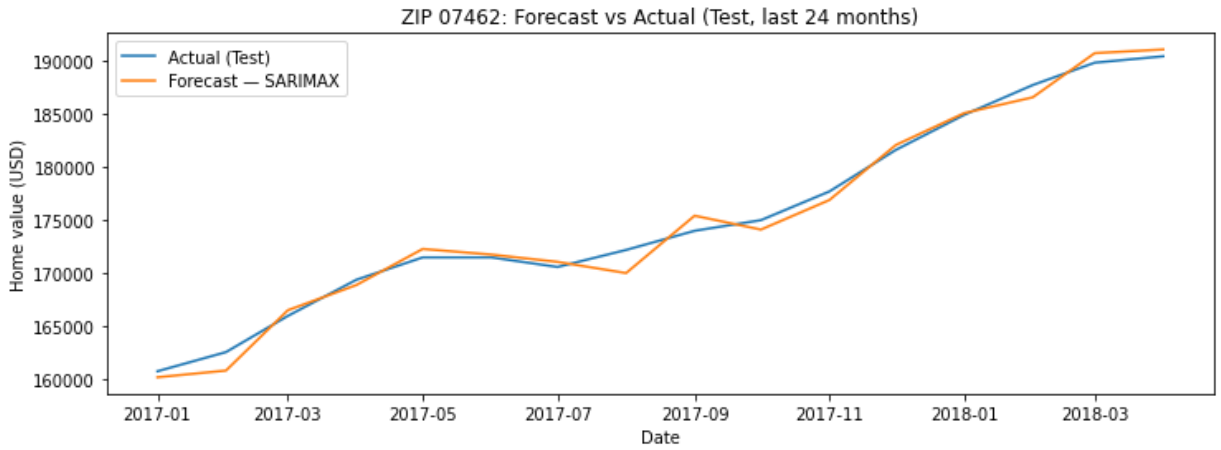
In [16]:
```
# Pick the best validation model (lowest RMSE) and carry it into holdout testing
best_row = val_table.sort_values("Val_RMSE").iloc[0]
best_model = best_row["Model"]

if best_model.startswith("SARIMAX"):
    FINAL_FAMILY = "SARIMAX"
    FINAL_PARAMS = {"order": best_sarima_params[0], "seasonal": best_sarima_param
elif best_model.startswith("ARIMA"):
    FINAL_FAMILY = "ARIMA"
    FINAL_PARAMS = {"order": best_arima_order, "maxiter": 50}
else:
    # ARMA is the baseline family
    FINAL_FAMILY = "ARMA"
    FINAL_PARAMS = {"order": best_arma_order, "maxiter": 50}

FINAL_FAMILY, FINAL_PARAMS
```

Out[16]:
```
('SARIMAX', {'order': (1, 1, 1), 'seasonal': (1, 0, 1, 12), 'maxiter': 50})
```

In [17]:
```
def rolling_one_step_forecast_price(train_price: pd.Series, test_price: pd.Series
                                    family: str, params: dict, last_n: int = 24
    """Rolling 1-step forecast on price levels. Fits a fresh model each step (sim
    test_price = test_price.dropna()
    if last_n is not None and len(test_price) > last_n:
        test_price = test_price.iloc[-last_n:]

    hist_log = np.log(train_price.dropna()).copy()
    test_log = np.log(test_price)
    preds = []

    for t in range(len(test_log)):
        if family == "SARIMAX":
            m = SARIMAX(hist_log, order=params["order"], seasonal_order=params["s
                        enforce_stationarity=False, enforce_invertibility=False,
                        simple_differencing=False).fit(disp=False, maxiter=params
            next_log = float(m.forecast(steps=1).iloc[0])
        elif family == "ARIMA":
            m = ARIMA(hist_log, order=params["order"],
                      enforce_stationarity=False, enforce_invertibility=False).fi
            next_log = float(m.forecast(steps=1).iloc[0])
        elif family == "ARMA":
            dlog = hist_log.diff().dropna()
            m = ARIMA(dlog, order=params["order"],
                      enforce_stationarity=False, enforce_invertibility=False).fi
            ret_hat = float(m.forecast(steps=1).iloc[0])
            next_log = float(hist_log.iloc[-1] + ret_hat)
        else:
            next_log = float(hist_log.iloc[-1])

        preds.append(np.exp(next_log))
        hist_log = pd.concat([hist_log, test_log.iloc[[t]]])

    return pd.Series(preds, index=test_price.index)

trainval = pd.concat([train_f, val_f]).sort_index()
test_pred = rolling_one_step_forecast_price(trainval, test_f, FINAL_FAMILY, FINAL
test_y = test_f.dropna().loc[test_pred.index]

test_metrics = evaluate_regression_aligned(test_y, test_pred)
test_metrics
```

Out[17]:
```
{'MAE': 841.4162859339558, 'RMSE': 991.836886793269, 'n': 16}
```

In [18]:
```
plt.figure(figsize=(12,4))
plt.plot(test_y.index, test_y.values, label="Actual (Test)")
plt.plot(test_pred.index, test_pred.values, label=f"Forecast — {FINAL_FAMILY}")
plt.title(f"ZIP {EXAMPLE_ZIP}: Forecast vs Actual (Test, last 24 months)")
plt.xlabel("Date")
plt.ylabel("Home value (USD)")
plt.legend()
plt.show()
```



This chart shows rolling 12-month volatility of monthly returns. Higher volatility periods tend to be riskier for lenders because prices can move more sharply, so forecast error and decline risk can increase in these regimes.

In [19]:
```
# Residual diagnostics: if p-values are small, residuals may still be autocorrela
trainval_log = np.log(trainval.dropna())

if FINAL_FAMILY == "SARIMA":
    fitted = SARIMAX(trainval_log, order=FINAL_PARAMS["order"], seasonal_order=FI
                     enforce_stationarity=False, enforce_invertibility=False,
                     simple_differencing=False).fit(disp=False, maxiter=FINAL_PAR
elif FINAL_FAMILY == "ARIMA":
    fitted = ARIMA(trainval_log, order=FINAL_PARAMS["order"],
                   enforce_stationarity=False, enforce_invertibility=False).fit(r
else:
    fitted = None

if fitted is not None:
    resid = pd.Series(fitted.resid).dropna()
    acorr_ljungbox(resid, lags=[12,24], return_df=True)
else:
    print("Residual diagnostics skipped for baseline.")
```

```
Residual diagnostics skipped for baseline.
```

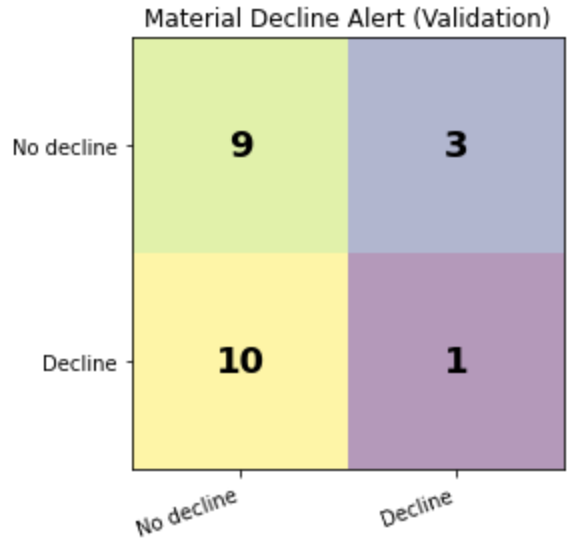
7) Material Decline Alert (≥1% Drop)

I convert the regression outputs into a simple alert: next-month return ≤ -1%. The alert threshold is tuned on validation to emphasize recall via the F2 score.

In [20]:
```
TAU = -0.01  # -1% material decline

val_ret = val_f.pct_change().dropna()
y_true = (val_ret <= TAU).astype(int)

if FINAL_FAMILY == "SARIMA":
    val_pred_price = best_sarima_pred
elif FINAL_FAMILY == "ARIMA":
    val_pred_price = best_arima_pred
else:
    val_pred_price = best_arma_pred

pred_ret = (val_pred_price / val_f.shift(1) - 1).dropna()
cls_df = pd.concat([y_true.rename("y"), pred_ret.rename("pred_ret")], axis=1).dro

thresholds = np.linspace(-0.05, 0.02, 60)
best_thr, best_f2 = None, -1
for thr in thresholds:
    yhat = (cls_df["pred_ret"] <= thr).astype(int)
    f2 = fbeta_score(cls_df["y"], yhat, beta=2, zero_division=0)
    if f2 > best_f2:
        best_f2, best_thr = f2, float(thr)

yhat_best = (cls_df["pred_ret"] <= best_thr).astype(int)
precision = precision_score(cls_df["y"], yhat_best, zero_division=0)
recall = recall_score(cls_df["y"], yhat_best, zero_division=0)
cm = confusion_matrix(cls_df["y"], yhat_best)

best_thr, {"precision": precision, "recall": recall, "F2": best_f2}, cm
```

Out[20]:
```
(0.006949152542372883,
 {'precision': 0.25, 'recall': 0.09090909090909091, 'F2': 0.10416666666666669},
 array([[ 9,  3],
        [10,  1]]))
```

In [21]:
```
plt.figure(figsize=(5,4))
plt.imshow(cm, alpha=0.4)
plt.title("Material Decline Alert (Validation)")
plt.xticks([0,1],["No decline","Decline"], rotation=20, ha="right")
plt.yticks([0,1],["No decline","Decline"])
for (i,j), v in np.ndenumerate(cm):
    plt.text(j,i,str(v),
             ha="center",va="center",
            fontsize=18, fontweight="bold")
plt.tight_layout()
plt.show()
```



ZIP forecast demo (choose any ZIP)

The dataset contains thousands of ZIP codes. The workflow below lets you pick a ZIP code and generate:

- Actual vs forecasted prices (test period)
- MAE / RMSE by model (Baseline → ARMA → ARIMA → SARIMAX)
- A 1-month-ahead “material decline” flag (forecasted return ≤ -1%)

This is designed for quick, repeatable demos. Change `ZIP_CODE` and re-run the next cell.

In [22]:
```
# --- ZIP forecast demo (choose any ZIP) ---
# Produces:
# 1) Actual vs Forecast (Test) for ARMA (Baseline) → ARIMA → SARIMA
# 2) MAE/RMSE by model (per ZIP)
# 3) A simple “material decline” alert: predicted next-month return <= TAU

def _align_drop(y_true, y_pred):
    """Align on timestamp and drop missing pairs."""
    return pd.concat([y_true.rename("y"), y_pred.rename("yhat")], axis=1).dropna(

def evaluate_regression_aligned(y_true, y_pred):
    """Compute MAE/RMSE after aligning timestamps and dropping missing pairs."""
    df = _align_drop(y_true, y_pred)
    return {
        "MAE": float(mean_absolute_error(df["y"], df["yhat"])),
        "RMSE": float(np.sqrt(mean_squared_error(df["y"], df["yhat"]))),
        "n": int(len(df))
    }

def split_series_by_zip(zip_code):
    """Return raw (train, val, test) price series for one ZIP."""
    z = str(zip_code).zfill(5)
    df = long_df[long_df["zip"].eq(z)][["date", "value", "split"]].copy()
    df = df.set_index("date").sort_index()
    train = df[df["split"].eq("train")]["value"]
    val   = df[df["split"].eq("val")]["value"]
    test  = df[df["split"].eq("test")]["value"]
    return train, val, test

def ffill_no_leak_levels(train, val, test):
    """Forward-fill without leaking future values into earlier splits."""
    train_f = train.ffill()

    if not train_f.dropna().empty:
        val_f = pd.concat([train_f.dropna().iloc[[-1]], val]).ffill().iloc[1:]
    else:
        val_f = val.ffill()

    if not val_f.dropna().empty:
        test_f = pd.concat([val_f.dropna().iloc[[-1]], test]).ffill().iloc[1:]
    else:
        test_f = test.ffill()

    return train_f, val_f, test_f

def rolling_one_step_arima_log(train_log, test_log, order):
    """Walk-forward 1-step forecasts in log space for ARIMA."""
    history = train_log.copy()
    preds = []
    for t in range(len(test_log)):
        m = ARIMA(history, order=order, enforce_stationarity=False, enforce_inver
        preds.append(float(m.forecast(steps=1).iloc[0]))
        history = pd.concat([history, test_log.iloc[[t]]])
    return pd.Series(preds, index=test_log.index)

def rolling_one_step_sarima_log(train_log, test_log, order, seasonal_order):
    """Walk-forward 1-step forecasts in log space for SARIMA."""
```

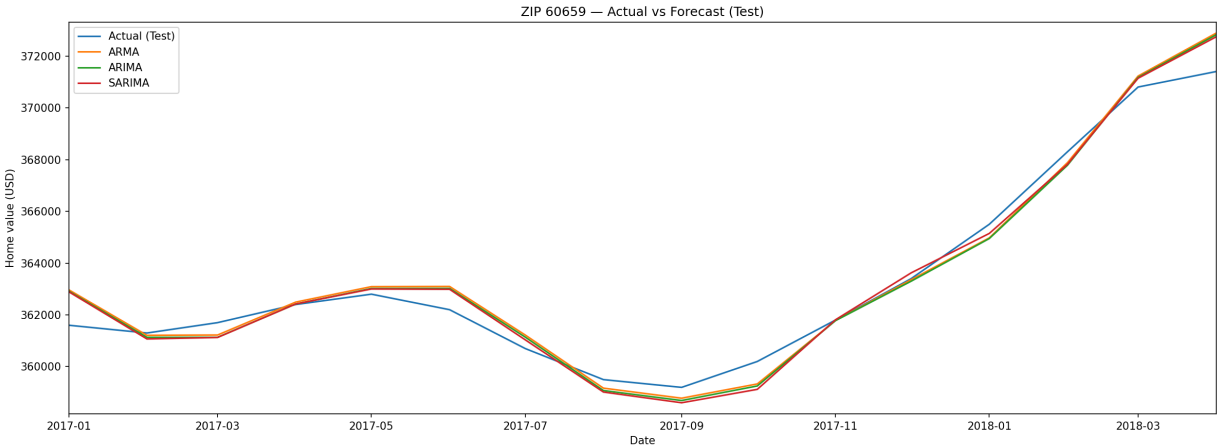
```
    history = train_log.copy()
    preds = []
    for t in range(len(test_log)):
        m = SARIMAX(
            history,
            order=order,
            seasonal_order=seasonal_order,
            enforce_stationarity=False,
            enforce_invertibility=False
        ).fit(disp=False)
        preds.append(float(m.forecast(steps=1).iloc[0]))
        history = pd.concat([history, test_log.iloc[[t]]])
    return pd.Series(preds, index=test_log.index)

def rolling_one_step_arma_log(train_log, test_log, order_pq):
    """Walk-forward 1-step forecasts for ARMA on diff(log), integrated back to lo
    p, q = order_pq
    history_log = train_log.copy()
    preds = []
    for t in range(len(test_log)):
        dlog = history_log.diff().dropna()
        m = ARIMA(dlog, order=(p, 0, q), enforce_stationarity=False, enforce_inve
        pred_dlog = float(m.forecast(steps=1).iloc[0])
        pred_log = float(history_log.iloc[-1]) + pred_dlog
        preds.append(pred_log)
        history_log = pd.concat([history_log, test_log.iloc[[t]]])
    return pd.Series(preds, index=test_log.index)

def _grid_search_on_val(train_log, val_log, family, arima_pdq=None, sarima_specs=
    """Small grid search on validation to pick a reasonable config for ONE ZIP."""
    arima_pdq = arima_pdq or []
    sarima_specs = sarima_specs or []
    arma_pq = arma_pq or []

    best = {"family": family, "spec": None, metric: np.inf}

    if family == "ARIMA":
        for spec in arima_pdq:
            try:
                pred_val_log = rolling_one_step_arima_log(train_log, val_log, or
                m = evaluate_regression_aligned(np.exp(val_log), np.exp(pred_val_
                if m < best[metric]:
                    best.update({"spec": spec, metric: m})
            except Exception:
                continue

    elif family == "SARIMA":
        for order, seas in sarima_specs:
            try:
                pred_val_log = rolling_one_step_sarima_log(train_log, val_log, or
                m = evaluate_regression_aligned(np.exp(val_log), np.exp(pred_val_
                if m < best[metric]:
                    best.update({"spec": (order, seas), metric: m})
            except Exception:
                continue

    elif family == "ARMA":
```

```
        for pq in arma_pq:
            try:
                pred_val_log = rolling_one_step_arma_log(train_log, val_log, orde
                m = evaluate_regression_aligned(np.exp(val_log), np.exp(pred_val_
                if m < best[metric]:
                    best.update({"spec": pq, metric: m})
            except Exception:
                continue

    return best

def run_zip_demo(zip_code, tau=-0.01, tune_on_val=True, show_plot=True):
    """Run ARMA (Baseline) → ARIMA → SARIMA for a ZIP and return a comparison tab
    zip_code = str(zip_code).zfill(5)
    train, val, test = split_series_by_zip(zip_code)
    train_f, val_f, test_f = ffill_no_leak_levels(train, val, test)

    train_log = np.log(train_f.dropna())
    val_log   = np.log(val_f.dropna())
    test_log  = np.log(test_f.dropna())

    # Require enough history for seasonal signal
    if len(train_log) < 36 or len(val_log) < 12 or len(test_log) < 6:
        raise ValueError(f"ZIP {zip_code}: not enough data (train={len(train_log)

    # Defaults
    arma_best = (1, 1)
    arima_best = (1, 1, 1)
    sarima_best = ((1, 1, 1), (1, 0, 1, 12))

    if tune_on_val:
        arima_grid = [(0,1,1), (1,1,0), (1,1,1), (2,1,1)]
        sarima_grid = [((1,1,1), (1,0,1,12)),
                       ((1,1,0), (1,0,1,12)),
                       ((0,1,1), (1,0,1,12))]
        arma_grid = [(1,0), (1,1), (2,1)]

        arma_pick = _grid_search_on_val(train_log, val_log, "ARMA", arma_pq=arma_
        arima_pick = _grid_search_on_val(train_log, val_log, "ARIMA", arima_pdq=a
        sarima_pick = _grid_search_on_val(train_log, val_log, "SARIMA", sarima_sp

        if arma_pick["spec"] is not None:
            arma_best = arma_pick["spec"]
        if arima_pick["spec"] is not None:
            arima_best = arima_pick["spec"]
        if sarima_pick["spec"] is not None:
            sarima_best = sarima_pick["spec"]

    sar_order, sar_seas = sarima_best

    y_test = np.exp(test_log)

    # Walk-forward forecasts on test using train+val history
    trainval_log = pd.concat([train_log, val_log]).sort_index()
    pred_log_arma  = rolling_one_step_arma_log(trainval_log, test_log, order_pq=a
    pred_log_arima = rolling_one_step_arima_log(trainval_log, test_log, order=ari
    pred_log_sar   = rolling_one_step_sarima_log(trainval_log, test_log, order=sa
```

```
pred_arma  = np.exp(pred_log_arma)
pred_arima = np.exp(pred_log_arima)
pred_sar   = np.exp(pred_log_sar)

rows = [
    {"model": f"ARMA{arma_best} (Baseline) on diff(log)", **evaluate_regressi
    {"model": f"ARIMA{arima_best} on log", **evaluate_regression_aligned(y_te
    {"model": f"SARIMA{sar_order}x{sar_seas} on log", **evaluate_regression_a
]
comp = pd.DataFrame(rows).sort_values("RMSE").reset_index(drop=True)

# Material-decline alert (simple rule)
actual_ret = y_test.pct_change().dropna()
y_true = (actual_ret <= tau).astype(int)

pred_ret = (pred_sar / y_test.shift(1) - 1).dropna()
aligned = pd.concat([y_true.rename("y_true"), pred_ret.rename("pred_ret")], a
y_pred = (aligned["pred_ret"] <= tau).astype(int)

prec = precision_score(aligned["y_true"], y_pred, zero_division=0)
rec  = recall_score(aligned["y_true"], y_pred, zero_division=0)
f1   = fbeta_score(aligned["y_true"], y_pred, beta=1, zero_division=0)
cm   = confusion_matrix(aligned["y_true"], y_pred)

print(f"ZIP {zip_code} | Material-decline threshold: {tau:.0%}")
print(f"Decline alert (SARIMA signal) — Precision: {prec:.3f} | Recall: {rec:
print("Confusion matrix [[TN, FP],[FN, TP]]:\n", cm)


if show_plot:
    fig, ax = plt.subplots(figsize=(16, 6), dpi=150)  # bigger + sharper

    ax.plot(y_test.index, y_test.values, label="Actual (Test)")
    ax.plot(pred_arma.index, pred_arma.values, label="ARMA")
    ax.plot(pred_arima.index, pred_arima.values, label="ARIMA")
    ax.plot(pred_sar.index, pred_sar.values, label="SARIMA")

    ax.set_title(f"ZIP {zip_code} — Actual vs Forecast (Test)")
    ax.set_xlabel("Date")
    ax.set_ylabel("Home value (USD)")

    # keep the whole graph, but reduce empty space around it
    ax.set_xlim(y_test.index.min(), y_test.index.max())

    y_all = np.concatenate([
        y_test.values,
        pred_arma.values,
        pred_arima.values,
        pred_sar.values
    ])
    pad = 0.03 * (y_all.max() - y_all.min())  # 3% padding
    ax.set_ylim(y_all.min() - pad, y_all.max() + pad)

    ax.margins(x=0)          # no extra x padding
    ax.legend(loc="best")
    fig.tight_layout()
```

```
        plt.show()

    return comp

# ---- Run the demo: change ZIP_CODE and re-run this cell ----
ZIP_CODE = "60659"
comparison_table = run_zip_demo(ZIP_CODE, tau=-0.01, tune_on_val=True, show_plot=
comparison_table
```

```
ZIP 60659 | Material-decline threshold: -1%
Decline alert (SARIMA signal) — Precision: 0.000 | Recall: 0.000 | F1: 0.000
Confusion matrix [[TN, FP],[FN, TP]]:
 [[15]]
```



Out[22]:

	model	MAE	RMSE	n
0	SARIMA(1, 1, 1)x(1, 0, 1, 12) on log	521.617679	656.538703	16
1	ARIMA(1, 1, 1) on log	525.750766	664.834330	16
2	ARMA(1, 1) (Baseline) on diff(log)	516.949359	668.943555	16

This chart overlays the actual test-period home values with each model's forecast for the same months. If a model tracks the blue "Actual" line more closely, it is producing smaller month-ahead errors for this ZIP.

Reproducibility notes

- Expected data file: zillow_data.csv (https://raw.githubusercontent.com/rahroy82/home_price_forecast/refs/heads/main/zillow_data.c in the same folder as this notebook, or kaggle (https://www.kaggle.com/datasets/zhenyufan/zillow-housing-price) (archived zip)
- Run order: **Kernel → Restart & Run All**
- Recommended: include `requirements.txt` (or `environment.yml`) in the GitHub repo
- Platform: tested on WSL/Ubuntu (local laptop)

10) Conclusions & Next Steps

This project demonstrates a complete supervised time-series ML pipeline with:

- baseline (ARMA) + multiple ARIMA-family models,
- explicit validation-based model selection,
- holdout testing, diagnostics, and a risk-oriented decline alert.

Next steps: add exogenous macro variables (rates, unemployment), scale evaluation across many ZIPs, and improve the alert calibration for higher recall.

Portfolio view (optional): evaluate multiple ZIPs you choose

Metrics in this project are computed **per ZIP** because each ZIP has its own time series and its own model fit.

In the next cell, you can paste a list of ZIPs (comma-separated) to evaluate.
It will compute **test RMSE** for Baseline vs SARIMAX and a simple **next-month risk flag** based on the SARIMAX forecast.

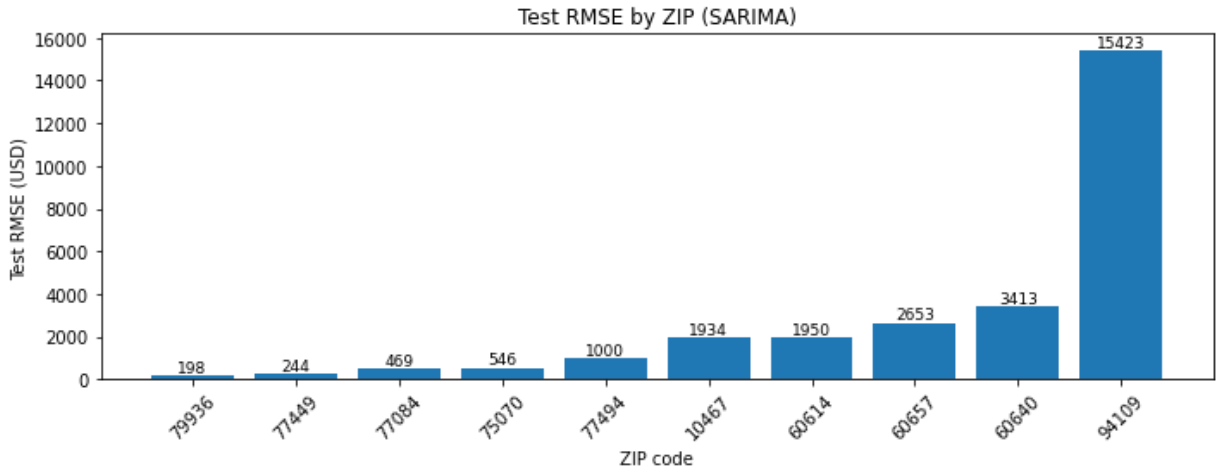
In [24]:
```
# --- Portfolio scan (user-selected ZIPs) ---
# Goal:
# - You choose ZIPs (comma-separated)
# - For each ZIP, compute test RMSE for ARMA (Baseline) vs SARIMA + next-month ri
# - Plot a chart where EACH BAR corresponds to ONE ROW (ONE ZIP) — not a histogra

import re  # needed for parse_zip_list

# Choose ZIP codes here (comma-separated). Example: "60659, 75070, 77494"
ZIP_LIST_TEXT = "60657, 75070, 77494, 60614, 79936, 77084, 10467, 60640, 77449, 9

# Material-decline threshold (e.g., -1% next month)
TAU = -0.01

# Set True to run the scan
RUN_PORTFOLIO_SCAN = True


def parse_zip_list(text):
    """Parse comma-separated ZIPs into zero-padded 5-digit strings."""
    zips = []
    for part in str(text).split(","):
        p = part.strip()
        if not p:
            continue
        # digits only (handles accidental spaces/characters)
        p = re.sub(r"[^0-9]", "", p)
        if p:
            zips.append(p.zfill(5))
    return zips


def portfolio_scan(zip_list,
                   tau=-0.01,
                   arma_pq=(1, 1),
                   sarima_order=(1, 1, 1),
                   sarima_seasonal=(1, 0, 1, 12),
                   min_train=36,
                   min_val=12,
                   min_test=6,
                   verbose=False):
    """Evaluate ARMA (baseline) vs SARIMA on the test split for a list of ZIPs.

    Notes:
    - ARMA baseline is fit on diff(log) (returns) and integrated back to log leve
    - SARIMA is fit directly on log(levels) with seasonal_order (annual seasonali
    """
    results = []
    skipped = {"short_history": 0, "error": 0}

    for z in zip_list:
        try:
            train, val, test = split_series_by_zip(z)
            train_f, val_f, test_f = ffill_no_leak_levels(train, val, test)

            train_log = np.log(train_f.dropna())
            val_log   = np.log(val_f.dropna())
```

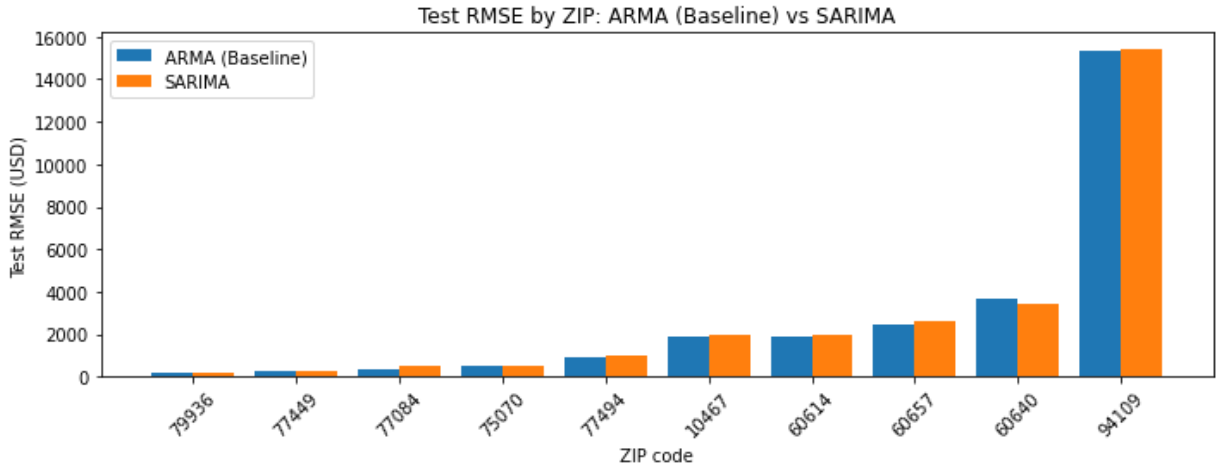
```
            test_log  = np.log(test_f.dropna())

            # skip ZIPs with short history
            if len(train_log) < min_train or len(val_log) < min_val or len(test_l
                skipped["short_history"] += 1
                continue

            # --- Rolling 1-step forecasts on test using train+val history ---
            trainval_log = pd.concat([train_log, val_log]).sort_index()

            # ARMA (Baseline)
            pred_log_arma = rolling_one_step_arma_log(trainval_log, test_log, ord
            pred_arma = np.exp(pred_log_arma)

            # SARIMA
            pred_log_sar = rolling_one_step_sarima_log(
                trainval_log, test_log,
                order=sarima_order,
                seasonal_order=sarima_seasonal
            )
            pred_sar = np.exp(pred_log_sar)

            y_test = np.exp(test_log)

            met_arma = evaluate_regression_aligned(y_test, pred_arma)
            met_sar  = evaluate_regression_aligned(y_test, pred_sar)

            # --- Next-month risk (forecast 1 month ahead from end of full histor
            full_log = np.log(pd.concat([train_f, val_f, test_f]).dropna())
            m = SARIMAX(
                full_log,
                order=sarima_order,
                seasonal_order=sarima_seasonal,
                enforce_stationarity=False,
                enforce_invertibility=False
            ).fit(disp=False)

            next_log = float(m.forecast(steps=1).iloc[0])
            next_price = float(np.exp(next_log))
            last_price = float(np.exp(full_log.iloc[-1]))
            next_ret = next_price / last_price - 1
            risk_flag = int(next_ret <= tau)

            results.append({
                "zip": z,
                "rmse_arma": met_arma["RMSE"],
                "rmse_sarima": met_sar["RMSE"],
                "rmse_improvement": met_arma["RMSE"] - met_sar["RMSE"],
                "next_month_pred_ret": next_ret,
                "risk_flag_decline_le_1pct": risk_flag,
                "n_test": met_sar["n"]
            })

        except Exception as e:
            skipped["error"] += 1
            if verbose:
                print(f"ZIP {z} skipped due to error: {e}")
```

```
            continue

    df = pd.DataFrame(results)
    if df.empty and verbose:
        print("Portfolio scan produced no results.")
        print("Skipped counts:", skipped)
    return df


if RUN_PORTFOLIO_SCAN:
    zip_list = parse_zip_list(ZIP_LIST_TEXT)

    if len(zip_list) == 0:
        print("No ZIPs provided. Add ZIPs to ZIP_LIST_TEXT and re-run.")
    else:
        portfolio_df = portfolio_scan(
            zip_list,
            tau=TAU,
            arma_pq=(1, 1),
            sarima_order=(1, 1, 1),
            sarima_seasonal=(1, 0, 1, 12),
            verbose=False
        )

        if portfolio_df.empty:
            print("No ZIPs produced results (often due to short histories or miss
        else:
            portfolio_df = portfolio_df.sort_values("rmse_sarima").reset_index(dr
            display(portfolio_df)

            # --- Chart 1: ONE bar per ZIP (each bar maps to one table row) ---
            plot_df = portfolio_df.copy()

            plt.figure(figsize=(10, 4))
            plt.bar(plot_df["zip"].astype(str), plot_df["rmse_sarima"])
            plt.title("Test RMSE by ZIP (SARIMA)")
            plt.xlabel("ZIP code")
            plt.ylabel("Test RMSE (USD)")

            # Optional: show exact RMSE above each bar
            for i, v in enumerate(plot_df["rmse_sarima"].values):
                plt.text(i, v, f"{v:.0f}", ha="center", va="bottom", fontsize=9)

            plt.xticks(rotation=45)
            plt.tight_layout()
            plt.show()

            # --- Chart 2 (optional): ARMA (Baseline) vs SARIMA per ZIP ---
            x = np.arange(len(plot_df))
            width = 0.4

            plt.figure(figsize=(10, 4))
            plt.bar(x - width/2, plot_df["rmse_arma"],   width, label="ARMA (Base
            plt.bar(x + width/2, plot_df["rmse_sarima"], width, label="SARIMA")
            plt.title("Test RMSE by ZIP: ARMA (Baseline) vs SARIMA")
            plt.xlabel("ZIP code")
            plt.ylabel("Test RMSE (USD)")
```

```
plt.xticks(x, plot_df["zip"].astype(str), rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```

	zip	rmse_arma	rmse_sarima	rmse_improvement	next_month_pred_ret	risk_flag_decline_le
0	79936	182.410622	198.334526	-15.923904	0.003124	
1	77449	238.720523	243.612748	-4.892225	0.002350	
2	77084	333.460712	468.949369	-135.488658	0.004024	
3	75070	542.194777	546.174452	-3.979675	0.000854	
4	77494	901.733999	1000.258140	-98.524141	0.005991	
5	10467	1913.407465	1934.226866	-20.819401	0.007789	
6	60614	1916.357857	1949.921784	-33.563928	-0.003193	
7	60657	2460.521909	2652.528497	-192.006588	-0.005494	
8	60640	3686.932739	3413.444973	273.487766	-0.002668	
9	94109	15302.744562	15423.438025	-120.693463	0.007043	





This histogram summarizes how forecast error varies across the ZIPs you selected. Each bar counts how many ZIPs fall into a given test RMSE range for the SARIMAX model, which helps compare portfolio risk hotspots vs more stable areas.