ICU Occupancy Change Prediction in California

Allyson Totpal

DSC680

February 15, 2026

Business Problem

Hospital capacity strain poses ongoing operational challenges for healthcare systems, particularly in large and demographically diverse states such as California. While hospitals may routinely monitor occupancy metrics, capacity strain is often recognized after critical thresholds have already been exceeded. Reactive responses limit administrators' ability to proactively adjust staffing, coordinate transfers, or activate surge plans. This project evaluates whether respiratory emergency department (ED) surveillance signals can improve forecasting of medium-term (7-week) changes in ICU occupancy in California.

Background/History

Forecasting hospital capacity became critically important during the COVID-19 pandemic, but most capacity forecasting models rely heavily on autoregressive time-series structure, short-horizon projections, and mechanistic epidemiological assumptions. However, less attention has been given to medium-term strain dynamics outside pandemic contexts, particularly using syndromic surveillance indicators such as ED visit percentages for respiratory illnesses. Respiratory viruses contribute substantially to seasonal inpatient and ICU utilization and emergency department visit percentages for influenza and RSV may provide earlier signals of upstream demand pressures.

Data Explanation

Hospital utilization and emergency department surveillance datasets were obtained from publicly available federal reporting systems and integrated at the weekly level for California. The U.S. Department of Health and Human Services (HHS) hospital utilization dataset was filtered to include only California facilities and aggregated from the hospital level to statewide weekly totals. ICU occupancy was calculated as the ratio of staffed adult ICU beds used to total staffed

adult ICU beds, using 7-day average reporting fields to reduce daily volatility. To address reporting artifacts and extreme values potentially caused by delayed submissions or corrections, ICU occupancy was consolidated at the 99th percentile.

Emergency department surveillance data were obtained from the CDC National Syndromic Surveillance Program (NSSP). Smoothed weekly percentages of ED visits attributed to influenza and RSV were used to reduce short-term noise while preserving seasonal signal patterns. All date fields were standardized to a consistent week-start format to ensure temporal alignment across datasets.

Feature engineering included the creation of lag variables (1- and 2-week lags) for ICU occupancy and ED surveillance signals to capture delayed relationships between respiratory illness activity and ICU strain. The primary outcome variable was constructed as the 7-week forward change in ICU occupancy. A binary directional target indicating whether ICU occupancy increased over the subsequent 7 weeks was also created for classification modeling. Rows with missing lagged or target values were excluded to maintain chronological integrity. A time-aware 80/20 chronological train-test split was used to prevent lookahead bias.

Data Dictionary

Field name	Description	Source	Type
`week`	Standardized weekly time index	Derived	Datetime
`icu_occ`	Weekly ICU occupancy ratio (7-day average)	HHS Hospital Utilization Data	Float
`icu_lag1`	ICU occupancy lagged 1 week	Derived	Float
`icu_lag2`	ICU occupancy lagged 2 weeks	Derived	Float
`percent_visits_smoothed_influenza`	Smoothed weekly % of ED visits attributed to influenza	CDC NSSP ED Surveillance	Float
`percent_visits_smoothed_rsv`	Smoothed weekly % of ED visits attributed to RSV	CDC NSSP ED Surveillance	Float

flu_lag1	Influenza ED % lagged 1 week	Derived	Float
flu_lag2	Influenza ED % lagged 2 weeks	Derived	Float
rsv_lag1	RSV ED % lagged 2 weeks	Derived	Float
rsv_lag2	RSV ED % lagged 2 weeks	Derived	Float
icu_change_7w	7-week forward change in ICU occupancy	Derived	Float
increase_7w	Binary indicator of ICU occupancy increase within 7 weeks	Derived	Integer

Methods

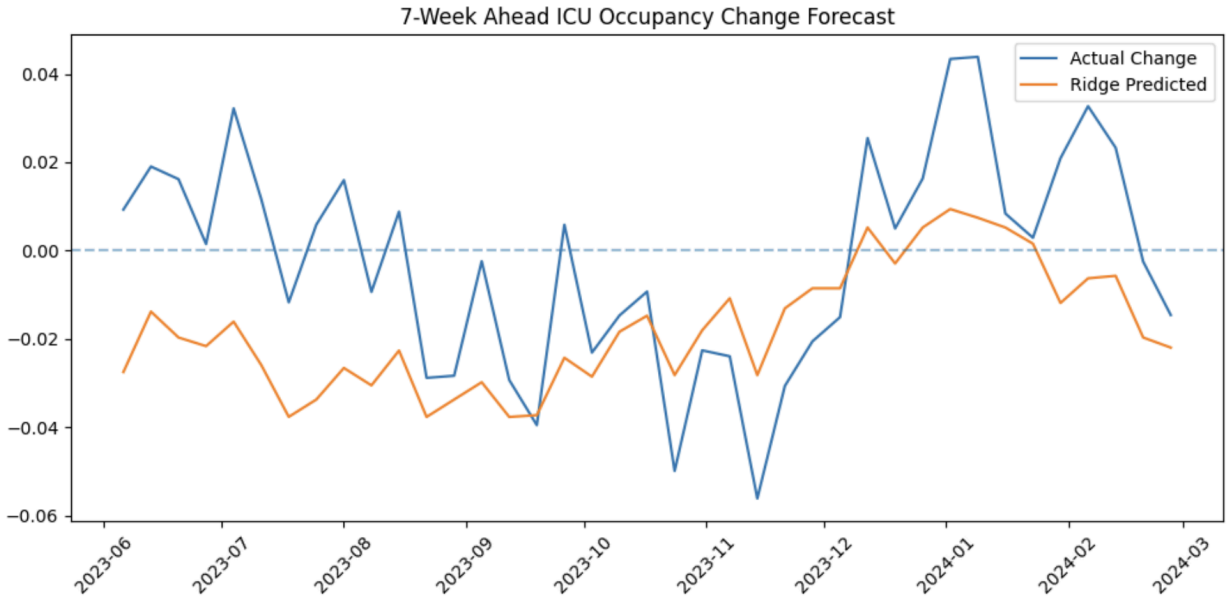
This study applied supervised learning methods to evaluate whether respiratory ED surveillance signals improve forecasting of 7-week changes in ICU occupancy in California. Two modeling tasks were defined: (1) regression to predict the magnitude of ICU occupancy change and (2) classification to predict whether ICU occupancy would increase within the 7-week horizon. Data were sorted chronologically and split using a time-aware 80/20 train-test framework to prevent lookahead bias.

Two models were evaluated: Ridge regression as a linear baseline and Random Forest as a nonlinear ensemble method capable of capturing interaction effects among lagged predictors. Baseline comparisons included a zero-change model for regression and a majority-class model for classification. Performance was assessed using RMSE and MAE for regression and Accuracy and ROC-AUC for classification. Feature importance from Random Forest models was analyzed to evaluate the relative contribution of ICU lags and ED surveillance variables.

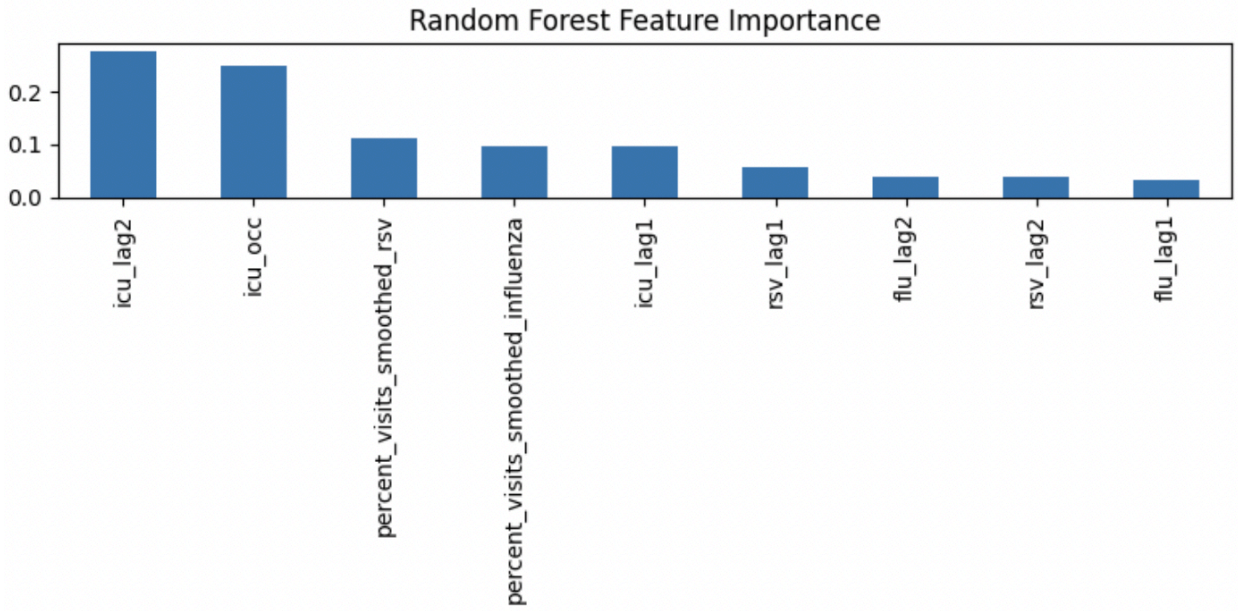
Analysis

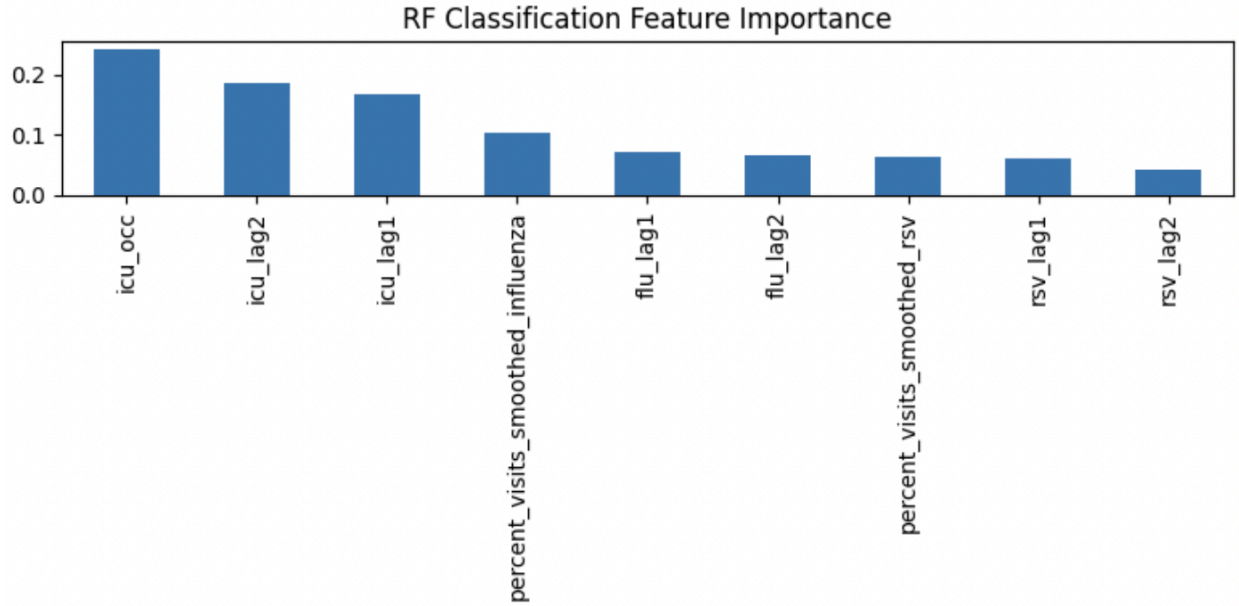
Results indicate that statewide ICU occupancy is strongly autoregressive. Linear regression did not outperform the zero-change baseline when forecasting 7-week change,

suggesting limited linear structure in strain dynamics. However, the Random Forest model reduced RMSE by approximately 8%, indicating meaningful nonlinear relationships between lagged ICU levels and respiratory ED surveillance signals.



Feature importance analysis showed that while lagged ICU occupancy remained the dominant predictor, influenza and RSV ED variables collectively contributed substantial predictive value. Directional classification outperformed continuous change forecasting, achieving strong discrimination in identifying future increases in ICU occupancy. Although the evaluation window was limited, findings suggest that nonlinear modeling of surveillance signals can improve medium-term hospital strain forecasting beyond autoregressive trends alone.





Conclusion

ICU occupancy levels are highly autoregressive at the statewide level. However, changes in ICU occupancy exhibit nonlinear dynamics that can be partially anticipated using lagged respiratory ED surveillance signals. Nonlinear models outperform linear models in forecasting

medium-term change. Directional prediction yields stronger operational signals than magnitude forecasting.

Assumptions, Limitations, & Challenges

This analysis assumes consistent reporting quality across HHS and CDC datasets and a stable relationship between respiratory ED surveillance signals and ICU occupancy over time. Because data were aggregated at the statewide level, volatility is reduced and localized strain dynamics may be masked. The relatively small held-out test window limits the stability of performance estimates, particularly for classification metrics. Additionally, the model does not incorporate non-respiratory drivers of ICU utilization, such as trauma, elective procedure surges, staffing shortages, or policy changes. Structural shifts, such as the emergence of novel pathogens, could weaken predictive relationships. Key technical challenges included multi-source temporal alignment, handling reporting artifacts in ICU occupancy data, and modeling highly persistent time-series behavior in a low-variance aggregate system.

Future Uses, Implementation, &Ethical Assessment

Future applications of this framework include extending modeling to county- or hospital-level forecasting, incorporating additional early-warning signals such as admissions data, wastewater surveillance, or staffing metrics, and deploying real-time dashboards to support operational decision-making. Implementation would require automated weekly data ingestion, routine modeling retraining, performance monitoring, and predefined alert thresholds for rising strain probabilities. Governance structures should ensure transparency in model assumptions and limitations, prevent overreliance on algorithmic outputs in staffing or resource allocation decisions, and communicate uncertainty clearly to decision-makers. Ethical deployment must

prioritize patient privacy, equitable resource distribution, and maintaining human oversight in operational planning.

References

Centers for Disease Control and Prevention, Respiratory Virus Response. (2024). NSSP Emergency Department Visits - COVID-19, Flu, RSV, Sub-state. *CDC.* https://data.cdc.gov/Public-Health-Surveillance/NSSP-Emergency-Department-Visit-Trajectories-by-St/rdmq-nq56/about_data

U.S. Department of Health & Human Services. (2025). COVID-19 Reported Patient Impact and Hospital Capacity by Facility. *Health data.* https://healthdata.gov/Hospital/COVID-19-Reported-Patient-Impact-and-Hospital-Capa/nag-cw7u/about_data

Appendix

```
import pandas as pd
import numpy as np

# read in data
hhs =
pd.read_csv('/Users/smooshii/DSC680/COVID-19_Reported_Patient_Impact_and_Hospital_Capa
city_by_Facility_20260208.csv')
nssp =
pd.read_csv('/Users/smooshii/DSC680/NSSP_Emergency_Department_Visit_Trajectories_by_St
ate_and_Sub_State_Regions-_COVID-19,_Flu,_RSV,_Combined  _20260208.csv')
cdc_dash = pd.read_csv('/Users/smooshii/DSC680/respiratory-virus-dashboard.csv')
# standardize column names
nssp.columns = [c.strip() for c in nssp.columns]
nssp.head()

# parse dates
nssp['week_end'] = pd.to_datetime(nssp['week_end'], errors = 'coerce')

# filter to CA only
nssp = nssp[nssp['geography'].str.lower() == 'california'].copy()

# drop high granularity geography
nssp = nssp.drop(columns = ['county', 'hsa'], errors = 'ignore')
# candidate Ed signal columns
ed_cols = ['percent_visits_influenza', 'percent_visits_rsv',
'percent_visits_combined',
           'percent_visits_smoothed_influenza', 'percent_visits_smoothed_rsv',
'percent_visits_smoothed_combined']
keep_cols = ['week_end'] + [c for c in ed_cols if c in nssp.columns]
nssp = nssp[keep_cols]

# convert to numeric
for c in ed_cols:
    if c in nssp.columns:
        nssp[c] = pd.to_numeric(nssp[c], errors = 'coerce')
# aggregate to one row per week_end
nssp_week = (nssp.groupby('week_end', as_index = False)
             .mean()
             .sort_values('week_end')
            )
nssp_week.rename(columns = {'week_end': 'week_ending'}, inplace = True)
nssp_week.head()
# engineer rolling means + week-over-week deltas
for c in ed_cols:
    if c in nssp_week.columns:
        nssp_week[f'{c}_roll2'] = nssp_week[c].rolling(2, min_periods = 1).mean()
        nssp_week[f'{c}_roll3'] = nssp_week[c].rolling(3, min_periods = 1).mean()
        nssp_week[f'{c}_wow_delta'] = nssp_week[c].diff()
cdc_dash['WEEKENDING'] = pd.to_datetime(cdc_dash['WEEKENDING'], errors = 'coerce')

# identify useful columns
cols_of_interest = ['WEEKENDING', 'FLU_ED_VISITS', 'RSV_ED_VISITS', 'FLU_ADMISSIONS',
'RSV_ADMISSIONS', 'POP']

cdc_dash = cdc_dash[[c for c in cols_of_interest if c in cdc_dash.columns]]
# convert to numeric
for c in cdc_dash.columns:
    if c != 'WEEKENDING':
        cdc_dash[c] = pd.to_numeric(cdc_dash[c], errors = 'coerce')
# aggregate to state-week
```

```
cdc_dash_week = (cdc_dash.groupby('WEEKENDING', as_index = False)
            .sum(numeric_only = True)
            .sort_values('WEEKENDING')
           )
cdc_dash_week.rename(columns = {'WEEKENDING': 'week_ending'}, inplace = True)
cdc_dash_week.head()
# compute per-100k rates if POP exists
if 'POP' in cdc_dash_week.columns:
    pop = cdc_dash_week['POP'].replace(0, np.nan)

    for c in ['FLU_ED_VISITS', 'RSV_ED_VISITS', 'FLU_ADMISSIONS', 'RSV_ADMISSIONS']:
        if c in cdc_dash_week.columns:
            cdc_dash_week[f'{c}_per_100k'] = (cdc_dash_week[c] / pop) * 100_000
# rolling features on rate columns
rate_cols = [c for c in cdc_dash_week.columns if c.endswith('_per_100k')]

for c in rate_cols:
    cdc_dash_week[f'{c}_roll2'] = cdc_dash_week[c].rolling(2, min_periods = 1).mean()
    cdc_dash_week[f'{c}_roll3'] = cdc_dash_week[c].rolling(3, min_periods = 1).mean()
    cdc_dash_week[f'{c}_wow_delta'] = cdc_dash_week[c].diff()
# select columns needed
hhs_cols = ['collection_week', 'state', 'all_adult_hospital_inpatient_beds_7_day_sum',
            'inpatient_beds_used_7_day_sum',
'staffed_adult_icu_bed_occupancy_7_day_avg']
hhs = hhs[hhs_cols].copy()
# parse week & filter to CA
hhs.loc[:, 'collection_week'] = pd.to_datetime(hhs.loc[:, 'collection_week'], errors =
'coerce')
hhs = hhs[hhs['state'] == 'CA'].copy()

# columns to clean
count_cols = ['all_adult_hospital_inpatient_beds_7_day_sum',
'inpatient_beds_used_7_day_sum']
raw_rate_cols = 'staffed_adult_icu_bed_occupancy_7_day_avg'
def _clean_numeric_series(s: pd.Series) -> pd.Series:
    '''robust numeric cleaning'''
    return pd.to_numeric(
        s.astype(str)
        .str.replace(',', '', regex = False)
        .str.replace('%', '', regex = False)
        .str.strip()
        .replace({'': np.nan, 'nan': np.nan, 'None': np.nan}),
        errors = 'coerce'
    )
# clean numeric fields
for c in count_cols + [raw_rate_cols]:
    hhs[c] = _clean_numeric_series(hhs[c])

# aggregate to week
hhs_week = (hhs.groupby('collection_week', as_index = False)
            .agg({**{c: 'sum' for c in count_cols},
                  raw_rate_cols: 'mean'})
            .sort_values('collection_week')
            .rename(columns = {'collection_week': 'week_ending'})
           )
hhs_week['icu_occ_raw'] = hhs_week[raw_rate_cols].abs() / 1_000_000

# winsorize ICU occupancy at p99
cap_99 = hhs_week['icu_occ_raw'].quantile(0.99)
hhs_week['icu_occ_wins'] = hhs_week['icu_occ_raw'].clip(upper = cap_99)

# final ICU occupancy feature
hhs_week['icu_occ'] = hhs_week['icu_occ_wins']
```

```
# inpatient occupancy ratio
hhs_week['inpatient_occ'] = (hhs_week['inpatient_beds_used_7_day_sum'] /

hhs_week['all_adult_hospital_inpatient_beds_7_day_sum'].replace(0, np.nan))

# high strain flag
p95 = hhs_week['icu_occ_wins'].quantile(0.95)
hhs_week['high_strain'] = (hhs_week['icu_occ_wins'] >= p95).astype(int)

# create standardized Monday-start week key for joining
hhs_week['week'] = hhs_week['week_ending'].dt.to_period('W-MON').dt.start_time
print('Weeks:', len(hhs_week))
print('Non-missing icu_occ:', int(hhs_week['icu_occ'].notna().sum()))
print('ICU occ wins cap (p99):', cap_99)
print(hhs_week['icu_occ'].describe())
print('p95 threshold:', p95)
print(hhs_week['high_strain'].value_counts())

hhs_week.head()
# merge on week_ending
hhs_week['week']  = hhs_week['week_ending'].dt.to_period('W-MON').dt.start_time
nssp_week['week'] = nssp_week['week_ending'].dt.to_period('W-MON').dt.start_time
cdc_dash_week['week']  =
cdc_dash_week['week_ending'].dt.to_period('W-MON').dt.start_time
data = (hhs_week
        .merge(nssp_week, on = 'week', how = 'left')
        .sort_values('week')
        .reset_index(drop = True)
       )
# create forecast target (3 weeks ahead)
forecast_horizon = 3
data['icu_occ_t_plus_3w'] = data['icu_occ'].shift(-forecast_horizon)

# drop rows where future target is missing
model_df = data.dropna(subset = ['icu_occ_t_plus_3w']).copy()
print('Model Rows: ', len(model_df))
# select predictors
feature_candidates = ['icu_occ', 'FLU_ED_VISITS_per_100k', 'RSV_ED_VISITS_per_100k',
                      'FLU_ED_VISITS_per_100k_roll2', 'FLU_ED_VISITS_per_100k_roll3',
                      'RSV_ED_VISITS_per_100k_roll2', 'RSV_ED_VISITS_per_100k_roll3']
features = [f for f in feature_candidates if f in model_df.columns]
print('Using features: ', features)
# drop rows with missing predictors
mask = x.notna().all(axis = 1)
x = x[mask]
y = y[mask]

print('Final Modeling Rows: ', len(x))
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error

# ridge regression (standardized)
ridge_model = Pipeline([('scaler', StandardScaler()),
                        ('ridge', Ridge(alpha = 1.0))])
ridge_model.fit(x_train, y_train)
ridge_preds = ridge_model.predict(x_test)

print('Ridge RMSE: ', mean_squared_error(y_test, ridge_preds, squared = False))
print('Ridge MAE: ', mean_absolute_error(y_test, ridge_preds))
from sklearn.ensemble import RandomForestRegressor
```

```
# random forest regression
rf_model = RandomForestRegressor(n_estimators = 300,
                                 max_depth = 6,
                                 random_state = 42)
rf_model.fit(x_train, y_train)
rf_preds = rf_model.predict(x_test)
print('RF RMSE: ', mean_squared_error(y_test, rf_preds, squared = False))
print('RF MAE: ', mean_absolute_error(y_test, rf_preds))
import matplotlib.pyplot as plt

# make sure y_test and predictions are aligned
test_weeks = model_df.loc[y_test.index, 'week']

# plot actual vs predicted change
plt.figure(figsize = (10, 5))
plt.plot(test_weeks, y_test.values, label = 'Actual')
plt.plot(test_weeks, rf_preds, label = 'RF Predicted')
plt.legend()
plt.title('3-Week Ahead ICU Occupancy Forecast')
plt.xticks(rotation = 45)
plt.tight_layout()
plt.show()
# update forecast to 7-weeks
forecast_horizon = 7

data['icu_occ_t_plus_7w'] = data['icu_occ'].shift(-7)
model_df = data.dropna(subset = features + ['icu_occ_t_plus_7w']).copy()
# select predictors
feature_candidates = ['icu_occ', 'FLU_ED_VISITS_per_100k', 'RSV_ED_VISITS_per_100k',
                      'FLU_ED_VISITS_per_100k_roll2', 'FLU_ED_VISITS_per_100k_roll3',
                      'RSV_ED_VISITS_per_100k_roll2', 'RSV_ED_VISITS_per_100k_roll3']
features = [f for f in feature_candidates if f in model_df.columns]
print('Using features: ', features)
x = model_df[features].copy()
y = model_df['icu_occ_t_plus_3w']

# drop rows with missing predictors
mask = x.notna().all(axis = 1)
x = x[mask]
y = y[mask]

print('Final Modeling Rows: ', len(x))

# train/test split
split_idx = int(len(x) * 0.8)

x_train = x.iloc[:split_idx]
x_test = x.iloc[split_idx:]

y_train = y.iloc[:split_idx]
y_test = y.iloc[split_idx:]
# build ridge model
ridge_model = Pipeline([('scaler', StandardScaler()),
                        ('ridge', Ridge(alpha = 1.0))])
ridge_model.fit(x_train, y_train)
ridge_preds = ridge_model.predict(x_test)

print('Ridge RMSE: ', mean_squared_error(y_test, ridge_preds, squared = False))
print('Ridge MAE: ', mean_absolute_error(y_test, ridge_preds))
# build random forest model
rf_model = RandomForestRegressor(n_estimators = 300,
                                 max_depth = 6,
```

```
                                random_state = 42)
rf_model.fit(x_train, y_train)
rf_preds = rf_model.predict(x_test)
print('RF RMSE: ', mean_squared_error(y_test, rf_preds, squared = False))
print('RF MAE: ', mean_absolute_error(y_test, rf_preds))
data = data.sort_values('week').reset_index(drop=True)

# keep only rows where current ICU occupancy exists
data2 = data.dropna(subset = ['icu_occ']).copy()

FORECAST_HORIZON = 7
data2['icu_occ_t_plus_7w'] = data2['icu_occ'].shift(-FORECAST_HORIZON)
data2['icu_change_7w'] = data2['icu_occ_t_plus_7w'] - data2['icu_occ']

print(data2[['week', 'icu_occ', 'icu_occ_t_plus_7w', 'icu_change_7w']].head(12))
print('Rows with target available:', data2['icu_change_7w'].notna().sum())

# sort chronologically
data = data.sort_values('week').reset_index(drop=True)

# keep only rows where ICU occupancy exists
data2 = data.dropna(subset=['icu_occ']).copy()

print('Rows with valid ICU:', len(data2))

# create 7-week horizon change
FORECAST_HORIZON = 7

data2['icu_occ_t_plus_7w'] = data2['icu_occ'].shift(-FORECAST_HORIZON)
data2['icu_change_7w'] = data2['icu_occ_t_plus_7w'] - data2['icu_occ']

# drop rows without future target
model_df = data2.dropna(subset = ['icu_change_7w']).copy()

print('Model rows:', len(model_df))
print('Date range:',
      model_df['week'].min(),
      '→',
      model_df['week'].max())
# resplit
features = ['icu_occ', 'FLU_ED_VISITS_per_100k', 'RSV_ED_VISITS_per_100k']

features = [f for f in features if f in model_df.columns]

split_idx = int(len(model_df) * 0.8)

train_df = model_df.iloc[:split_idx]
test_df  = model_df.iloc[split_idx:]

X_train = train_df[features]
y_train = train_df['icu_change_7w']

X_test  = test_df[features]
y_test  = test_df['icu_change_7w']
from sklearn.metrics import mean_squared_error, mean_absolute_error

# naive baseline: predict zero change
naive_preds = np.zeros(len(y_test))

naive_rmse = mean_squared_error(y_test, naive_preds, squared=False)
naive_mae  = mean_absolute_error(y_test, naive_preds)

print('Naive RMSE:', naive_rmse)
```

```
print('Naive MAE:', naive_mae)
# build new ridge model
ridge_model = Pipeline([('scaler', StandardScaler()),
                        ('ridge', Ridge(alpha = 1.0))])

ridge_model.fit(X_train, y_train)
ridge_preds = ridge_model.predict(X_test)

ridge_rmse = mean_squared_error(y_test, ridge_preds, squared = False)
ridge_mae  = mean_absolute_error(y_test, ridge_preds)

print('Ridge RMSE:', ridge_rmse)
print('Ridge MAE:', ridge_mae)
from sklearn.ensemble import RandomForestRegressor

# build new random forest model
rf_model = RandomForestRegressor(n_estimators = 300,
                                 max_depth = 6,
                                 random_state = 42)

rf_model.fit(X_train, y_train)
rf_preds = rf_model.predict(X_test)

rf_rmse = mean_squared_error(y_test, rf_preds, squared=False)
rf_mae  = mean_absolute_error(y_test, rf_preds)

print('RF RMSE:', rf_rmse)
print('RF MAE:', rf_mae)
# plot actual vs predicted in 7-week ahead model
plt.figure(figsize = (10, 5))
plt.plot(test_df['week'], y_test.values, label = 'Actual Change')
plt.plot(test_df['week'], ridge_preds, label = 'Ridge Predicted')
plt.axhline(0, linestyle = '--', alpha = 0.5)
plt.legend()
plt.title('7-Week Ahead ICU Occupancy Change Forecast')
plt.xticks(rotation = 45)
plt.tight_layout()
plt.show()
# identify ED-rate columns that actually exist
flu_col = next((c for c in model_df.columns if 'flu' in c.lower() and 'per_100k' in
c.lower()), None)
rsv_col = next((c for c in model_df.columns if 'rsv' in c.lower() and 'per_100k' in
c.lower()), None)

print('Detected flu column:', flu_col)
print('Detected rsv column:', rsv_col)

# add lagged ICU levels
model_df = model_df.sort_values('week').reset_index(drop = True)
model_df['icu_lag1'] = model_df['icu_occ'].shift(1)
model_df['icu_lag2'] = model_df['icu_occ'].shift(2)

# add lagged ED signals if columns exist
if flu_col is not None:
    model_df['flu_lag1'] = model_df[flu_col].shift(1)
    model_df['flu_lag2'] = model_df[flu_col].shift(2)

if rsv_col is not None:
    model_df['rsv_lag1'] = model_df[rsv_col].shift(1)
    model_df['rsv_lag2'] = model_df[rsv_col].shift(2)

# drop rows with new NaNs from lags/targets
```

```
model_df = model_df.dropna(subset = ['icu_change_7w', 'icu_occ', 'icu_lag1',
'icu_lag2']).copy()
if flu_col is not None:
    model_df = model_df.dropna(subset = ['flu_lag1', 'flu_lag2']).copy()
if rsv_col is not None:
    model_df = model_df.dropna(subset = ['rsv_lag1', 'rsv_lag2']).copy()

print('Rows after adding lags:', len(model_df))
features = ['icu_occ', 'icu_lag1', 'icu_lag2']

if flu_col is not None:
    features += [flu_col, 'flu_lag1', 'flu_lag2']
if rsv_col is not None:
    features += [rsv_col, 'rsv_lag1', 'rsv_lag2']

print('Features used:', features)
flu_col = 'percent_visits_smoothed_influenza'
rsv_col = 'percent_visits_smoothed_rsv'

print('Using flu column:', flu_col)
print('Using rsv column:', rsv_col)
model_df = model_df.sort_values('week').reset_index(drop=True)

# ICU lags
model_df['icu_lag1'] = model_df['icu_occ'].shift(1)
model_df['icu_lag2'] = model_df['icu_occ'].shift(2)

# ED lags
model_df['flu_lag1'] = model_df[flu_col].shift(1)
model_df['flu_lag2'] = model_df[flu_col].shift(2)

model_df['rsv_lag1'] = model_df[rsv_col].shift(1)
model_df['rsv_lag2'] = model_df[rsv_col].shift(2)

# drop rows with lag-induced NaNs
model_df = model_df.dropna().copy()

print('Rows after adding lags:', len(model_df))
features = ['icu_occ', 'icu_lag1', 'icu_lag2', flu_col,
            'flu_lag1', 'flu_lag2', rsv_col, 'rsv_lag1', 'rsv_lag2']

print('Feature count:', len(features))
split_idx = int(len(model_df) * 0.8)

train_df = model_df.iloc[:split_idx]
test_df  = model_df.iloc[split_idx:]

X_train = train_df[features]
y_train = train_df['icu_change_7w']

X_test  = test_df[features]
y_test  = test_df['icu_change_7w']
naive_preds = np.zeros(len(y_test))

print('Naive RMSE:', mean_squared_error(y_test, naive_preds, squared=False))
print('Naive MAE:', mean_absolute_error(y_test, naive_preds))
ridge_model = Pipeline([('scaler', StandardScaler()),
                        ('ridge', Ridge(alpha = 1.0))])

ridge_model.fit(X_train, y_train)
ridge_preds = ridge_model.predict(X_test)

print('Ridge RMSE:', mean_squared_error(y_test, ridge_preds, squared = False))
```

```
print('Ridge MAE:', mean_absolute_error(y_test, ridge_preds))

rf_model = RandomForestRegressor(n_estimators = 300,
                                 max_depth = 6,
                                 random_state = 42)

rf_model.fit(X_train, y_train)
rf_preds = rf_model.predict(X_test)

print('RF RMSE:', mean_squared_error(y_test, rf_preds, squared = False))
print('RF MAE:', mean_absolute_error(y_test, rf_preds))
importances = pd.Series(rf_model.feature_importances_,
                        index = features).sort_values(ascending = False)

print(importances)

plt.figure(figsize = (8, 4))
importances.plot(kind = 'bar')
plt.title('Random Forest Feature Importance')
plt.tight_layout()
plt.show()
# Create directional target
model_df['increase_7w'] = (model_df['icu_change_7w'] > 0).astype(int)
print(model_df['increase_7w'].value_counts())
features = ['icu_occ', 'icu_lag1', 'icu_lag2', 'percent_visits_smoothed_influenza',
            'flu_lag1', 'flu_lag2', 'percent_visits_smoothed_rsv', 'rsv_lag1',
'rsv_lag2']

model_df = model_df.sort_values('week').reset_index(drop = True)

split_idx = int(len(model_df) * 0.8)

train_df = model_df.iloc[:split_idx]
test_df  = model_df.iloc[split_idx:]

X_train = train_df[features]
y_train = train_df['increase_7w']

X_test  = test_df[features]
y_test  = test_df['increase_7w']
from sklearn.metrics import accuracy_score

majority_class = y_train.mode()[0]
baseline_preds = np.full(len(y_test), majority_class)

print('Baseline Accuracy:', accuracy_score(y_test, baseline_preds))
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (accuracy_score, roc_auc_score, classification_report,
confusion_matrix)

rf_clf = RandomForestClassifier(n_estimators = 400,
                                max_depth = 6,
                                random_state = 42)

rf_clf.fit(X_train, y_train)

rf_probs = rf_clf.predict_proba(X_test)[:, 1]
rf_preds = (rf_probs > 0.5).astype(int)

print('RF Accuracy:', accuracy_score(y_test, rf_preds))
print('RF ROC-AUC:', roc_auc_score(y_test, rf_probs))

print('\nConfusion Matrix:')
```

```
print(confusion_matrix(y_test, rf_preds))

print('\nClassification Report:')
print(classification_report(y_test, rf_preds))
from sklearn.metrics import roc_curve

fpr, tpr, _ = roc_curve(y_test, rf_probs)

plt.figure(figsize = (6, 6))
plt.plot(fpr, tpr, label = f'RF (AUC = {roc_auc_score(y_test, rf_probs):.3f})')
plt.plot([0,1], [0,1], linestyle = '--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve — 7-Week ICU Increase Prediction')
plt.legend()
plt.tight_layout()
plt.show()
clf_importance = pd.Series(rf_clf.feature_importances_,
index=features).sort_values(ascending = False)

print(clf_importance)

clf_importance.plot(kind = 'bar', figsize = (8, 4))
plt.title('RF Classification Feature Importance')
plt.tight_layout()
plt.show()
```