```
"""Re-analysis of VHF/Lassa fever admissions, MAUTHY isolation ward register (de-identified register-wide dataset, October 2026 version).

Run: python3 -I analysis.py <path to the register-wide dataset .xlsx>

Writes results.json (every number quoted in the revised manuscript) and per-record classification CSV."""

import sys, json, re, itertools, math

import numpy as np, pandas as pd

from scipy import stats

from firth import firth


f = sys.argv[1]

df = pd.read_excel(f, sheet_name='Data')

L0 = df[df.disease == 'Lassa fever'].copy()

# Exclusion (identified by register text, because record ids differ between dataset versions):

# 'Pulmonary Oedema, LVHF ppted by non drug compliance R/o Severe covid-19' -- LVHF = left ventricular

# heart failure; VHF never recorded as working or differential diagnosis -> does not meet inclusion criterion.

EXCL_TEXT = 'LVHF ppted by non drug'

excl_mask = L0.diagnosis_verbatim.astype(str).str.contains(EXCL_TEXT, regex=False)

assert excl_mask.sum() == 1, 'expected exactly one LVHF record'

R = {}

R['dataset_lassa_group_n'] = len(L0)

R['excluded'] = L0[excl_mask][['id', 'admission_year', 'outcome', 'diagnosis_verbatim']].to_dict('records')

L = L0[~excl_mask].copy().reset_index(drop=True)
```

```
N = len(L); R['N'] = N
R['isolation_total'] = len(df)
R['disease_counts'] = df.disease.value_counts().to_dict()
R['register_first_date'] = str(pd.to_datetime(df.admission_date).min().date())
R['register_last_date'] = str(pd.to_datetime(df.admission_date).max().date())


def wilson(k, n, z=1.959964):
    if n == 0: return (np.nan, np.nan)
    p = k/n; den = 1+z*z/n
    c = (p + z*z/(2*n))/den; h = z*math.sqrt(p*(1-p)/n + z*z/(4*n*n))/den
    return (max(0, c-h)*100, min(1, c+h)*100)


# ---------------- outcomes ----------------
L['outcome_c'] = L.outcome.fillna('Not recorded')
R['outcome_counts'] = L.outcome_c.value_counts().to_dict()
det = L[L.outcome_c.isin(['Died', 'Discharged'])].copy()
det['y'] = (det.outcome_c == 'Died').astype(int)
D = int(det.y.sum()); n_det = len(det)
R['deaths'] = D; R['determinate'] = n_det
R['cfr'] = D/n_det*100; R['cfr_ci'] = wilson(D, n_det)
R['cfr_all_admissions'] = D/N*100
R['cfr_upper_bound'] = (D + (N-n_det))/N*100
R['cfr_transfer_as_survivor'] = D/(N - (L.outcome_c == 'Not recorded').sum())*100
```

```
# ---------------- demographics ----------------
R['age_missing'] = int(L.age.isna().sum()); R['sex_missing'] = int(L.sex.isna().sum())
a = L.age.dropna()
R['age_mean'] = a.mean(); R['age_sd'] = a.std(); R['age_median'] = a.median()
R['age_q1'] = a.quantile(.25); R['age_q3'] = a.quantile(.75); R['age_min'] = a.min();
R['age_max'] = a.max()
R['male'] = int((L.sex == 'Male').sum()); R['female'] = int((L.sex == 'Female').sum())
bins = [0, 15, 30, 45, 60, 200]; labs = ['0-14', '15-29', '30-44', '45-59', '>=60']
L['ag'] = pd.cut(L.age, bins=bins, right=False, labels=labs)
det['ag'] = pd.cut(det.age, bins=bins, right=False, labels=labs)
R['agegroup_counts'] = L.ag.value_counts().reindex(labs).fillna(0).astype(int).to_dict()


# ---------------- exact tests ----------------
def fisher_freeman_halton(table):
  """Exact r x c test for 2 x k table (rows: died, survived). Sum probs of tables with P <=
P_obs."""
  t = np.asarray(table); cols = t.sum(0); r0 = t[0].sum(); n = t.sum()
  logC = lambda n_, k_: math.lgamma(n_+1)-math.lgamma(k_+1)-math.lgamma(n_-k_+1)
  lden = logC(n, r0)
  def lp(x): return sum(logC(c, xi) for c, xi in zip(cols, x)) - lden
  pobs = lp(t[0]); tot = 0.0
  def rec(i, rem, acc):
    nonlocal tot
    if i == len(cols)-1:
      if 0 <= rem <= cols[i]:
```

```
            x = acc+[rem]; v = lp(x)
            if v <= pobs + 1e-7: tot += math.exp(v)
        return
    for xi in range(0, min(cols[i], rem)+1):
        rec(i+1, rem-xi, acc+[xi])
  rec(0, r0, [])
  return min(tot, 1.0)


# Sex
tab_sex = [[int(((det.sex == 'Male') & (det.y == 1)).sum()), int(((det.sex == 'Male') & (det.y == 0)).sum())],
       [int(((det.sex == 'Female') & (det.y == 1)).sum()), int(((det.sex == 'Female') & (det.y == 0)).sum())]]
R['sex_table'] = tab_sex
orr, p = stats.fisher_exact(tab_sex); R['sex_fisher_p'] = p
R['sex_cfr'] = {'Male': (tab_sex[0][0], sum(tab_sex[0])), 'Female': (tab_sex[1][0], sum(tab_sex[1]))}


# Age group
ag_rows = {}
for g in labs:
  s = det[det.ag == g]
  ag_rows[g] = dict(adm=int((L.ag == g).sum()), deaths=int(s.y.sum()), det=len(s))
R['agegroup'] = ag_rows
groups_present = [g for g in labs if ag_rows[g]['det'] > 0]
```

```
tab_age = [[ag_rows[g]['deaths'] for g in groups_present], [ag_rows[g]['det']-
ag_rows[g]['deaths'] for g in groups_present]]

R['age_ffh_p'] = fisher_freeman_halton(tab_age)

for g in groups_present:

  ag_rows[g]['cfr'] = ag_rows[g]['deaths']/ag_rows[g]['det']*100

  ag_rows[g]['ci'] = wilson(ag_rows[g]['deaths'], ag_rows[g]['det'])


# ---------------- Firth logistic regression ----------------

m = det.dropna(subset=['age', 'sex']).copy()

R['model_n'] = len(m); R['model_events'] = int(m.y.sum())

X = np.c_[np.ones(len(m)), (m.sex == 'Male').astype(float),

      (m.ag == '30-44').astype(float), (m.ag == '45-59').astype(float), (m.ag ==
'>=60').astype(float)]

names = ['Intercept', 'Male sex', 'Age 30-44', 'Age 45-59', 'Age >=60']

res, glob = firth(X, m.y.values.astype(float), names)

R['firth_model1'] = res; R['firth_model1_global'] = glob

# univariable Firth (crude) for each covariate

crude = {}

Xs = np.c_[np.ones(len(m)), (m.sex == 'Male').astype(float)]

crude['Male sex'] = firth(Xs, m.y.values.astype(float), ['Intercept', 'Male sex'])[0][1]

Xa = np.c_[np.ones(len(m)), (m.ag == '30-44').astype(float), (m.ag == '45-59').astype(float),
(m.ag == '>=60').astype(float)]

ra = firth(Xa, m.y.values.astype(float), ['Intercept', 'Age 30-44', 'Age 45-59', 'Age >=60'])[0]

for r_ in ra[1:]: crude[r_['term']] = r_

R['firth_crude'] = crude
```

```
# sensitivity model 2: age continuous per 10 years + sex
X2 = np.c_[np.ones(len(m)), (m.sex == 'Male').astype(float), m.age.values/10.0]
res2, glob2 = firth(X2, m.y.values.astype(float), ['Intercept', 'Male sex', 'Age per 10 years'])
R['firth_model2'] = res2; R['firth_model2_global'] = glob2
# conventional ML for comparison (statsmodels)
import statsmodels.api as sm
try:
    ml = sm.Logit(m.y.values, X).fit(disp=0)
    R['ml_model1'] = [dict(term=n_, OR=float(np.exp(b)), lo=float(np.exp(ci[0])),
hi=float(np.exp(ci[1])), p=float(pv))
                      for n_, b, ci, pv in zip(names, ml.params, ml.conf_int(), ml.pvalues)]
except Exception as e:
    R['ml_model1'] = str(e)

# ---------------- laboratory ----------------
R['lab_counts'] = L.lab.value_counts().to_dict()
L['lab_c'] = L.lab.map({'Positive': 'Confirmed', 'Negative': 'Negative', 'Probable': 'Probable',
'Not recorded': 'Not recorded'})
tested = L[L.lab.isin(['Positive', 'Negative', 'Probable'])]
R['tested'] = len(tested); R['confirmed'] = int((L.lab == 'Positive').sum())
labout = {}
for lv in ['Positive', 'Negative', 'Probable', 'Not recorded']:
    s = L[L.lab == lv]
    labout[lv] = s.outcome_c.value_counts().to_dict()
    sd = s[s.outcome_c.isin(['Died', 'Discharged'])]
```

```
    labout[lv]['deaths'] = int((sd.outcome_c == 'Died').sum()); labout[lv]['det'] = len(sd)
R['lab_outcomes'] = labout
# confirmed vs tested non-confirmed (negative+probable)
c_d = labout['Positive']['deaths']; c_n = labout['Positive']['det']
nc_d = labout['Negative']['deaths']+labout['Probable']['deaths']; nc_n = labout['Negative']['det']+labout['Probable']['det']
R['lab_fisher'] = dict(conf=(c_d, c_n), nonconf=(nc_d, nc_n), p=stats.fisher_exact([[c_d, c_n-c_d], [nc_d, nc_n-nc_d]])[1])
pos = L[L.lab == 'Positive']
R['confirmed_male'] = int((pos.sex == 'Male').sum()); R['confirmed_age_median'] = pos.age.median()
R['confirmed_by_year'] = pos.admission_year.value_counts().sort_index().to_dict()

# ---------- length of stay ----------
los = L.length_of_stay_days
R['los_n'] = int(los.notna().sum()); R['los_median'] = los.median(); R['los_q1'] = los.quantile(.25); R['los_q3'] = los.quantile(.75)
R['los_min'] = los.min(); R['los_max'] = los.max()
by = {}
for o in ['Died', 'Discharged', 'Transferred', 'Not recorded']:
    s = L[L.outcome_c == o].length_of_stay_days.dropna()
    by[o] = dict(n=len(s), median=float(s.median()) if len(s) else None, q1=float(s.quantile(.25)) if len(s) else None, q3=float(s.quantile(.75)) if len(s) else None)
R['los_by_outcome'] = by
d_los = L[L.outcome_c == 'Died'].length_of_stay_days.dropna(); s_los = L[L.outcome_c == 'Discharged'].length_of_stay_days.dropna()
```

```
u = stats.mannwhitneyu(d_los, s_los, alternative='two-sided'); R['los_mwu'] =
dict(U=float(u.statistic), p=float(u.pvalue))

R['los_confirmed'] = dict(n=int(pos.length_of_stay_days.notna().sum()),
median=float(pos.length_of_stay_days.median()))

R['los_over_21'] = L[L.length_of_stay_days > 21][['id', 'length_of_stay_days', 'outcome_c',
'diagnosis_verbatim']].to_dict('records')


# ---------- temporal ----------

L['date'] = pd.to_datetime(L.admission_date)

dated = L[L.date.notna()]

R['dated_n'] = len(dated); R['undated_n'] = int(L.date.isna().sum())

R['undated_records'] = L[L.date.isna()][['id', 'outcome_c', 'lab',
'diagnosis_verbatim']].to_dict('records')

R['first_date'] = str(dated.date.min().date()); R['last_date'] = str(dated.date.max().date())

years = list(range(2020, 2027)); annual = {}

for yv in years:

  s = L[L.admission_year == yv]; sd = s[s.outcome_c.isin(['Died', 'Discharged'])]

  k = int((sd.outcome_c == 'Died').sum()); n_ = len(sd)

  annual[yv] = dict(adm=len(s), deaths=k, det=n_, transferred=int((s.outcome_c ==
'Transferred').sum()),

          notrec=int((s.outcome_c == 'Not recorded').sum()), confirmed=int((s.lab ==
'Positive').sum()),

          cfr=(k/n_*100 if n_ else None), ci=wilson(k, n_))

R['annual'] = annual

ref = 2024

for yv in years:
```

```
    if yv == ref: annual[yv]['p_vs_ref'] = None; continue
    a_ = annual[yv]; r_ = annual[ref]
    annual[yv]['p_vs_ref'] = stats.fisher_exact([[a_['deaths'], a_['det']-a_['deaths']],
[r_['deaths'], r_['det']-r_['deaths']]])[1]
tab_year = [[annual[y]['deaths'] for y in years], [annual[y]['det']-annual[y]['deaths'] for y in
years]]
R['year_ffh_p'] = fisher_freeman_halton(tab_year)
# original manuscript's 2022 vs 2023 comparison, recomputed (0/9 vs 5/10 as published)
R['orig_2022_vs_2023_p'] = stats.fisher_exact([[0, 9], [5, 5]])[1]
R['new_2022_vs_2023_p'] = stats.fisher_exact([[annual[2022]['deaths'],
annual[2022]['det']-annual[2022]['deaths']],
                                              [annual[2023]['deaths'], annual[2023]['det']-
annual[2023]['deaths']]])[1]
# Cochran-Armitage trend in CFR across years (exploratory)
def cochran_armitage(k, n, sc):
    k = np.array(k, float); n = np.array(n, float); sc = np.array(sc, float)
    N_ = n.sum(); p = k.sum()/N_
    T = np.sum(sc*(k - n*p)); V = p*(1-p)*(np.sum(n*sc**2) - np.sum(n*sc)**2/N_)
    z = T/np.sqrt(V); return z, 2*stats.norm.sf(abs(z))
z, pz = cochran_armitage([annual[y]['deaths'] for y in years], [annual[y]['det'] for y in years],
years)
R['cfr_trend'] = dict(z=z, p=pz)
# monthly
dated = dated.copy(); dated['m'] = dated.date.dt.month
R['jan_apr'] = int(dated.m.isin([1, 2, 3, 4]).sum())
R['jan_apr_pct'] = R['jan_apr']/len(dated)*100
```

```
R['nov_apr'] = int(dated.m.isin([11, 12, 1, 2, 3, 4]).sum()); R['nov_apr_pct'] =
R['nov_apr']/len(dated)*100

mc = dated.groupby(dated.date.dt.to_period('M')).size().sort_values(ascending=False)

R['peak_months'] = {str(k): int(v) for k, v in mc.head(5).items()}

R['month_of_year'] = dated.m.value_counts().sort_index().to_dict()


# ---------------- clinical diagnosis categories (explicit keyword rules) ----------------

rules = [

  ('Hepatic failure / acute hepatitis', r'hepatic\s*fail|hepatitis|hepatid'),

  ('Other liver disease (CLD, ALD, varices)', r'\bcld\b|\bald\b|verices|varices'),

  ('Neurological (encephalitis, stroke, SAH, meningitis)',
r'encephal|stroke|\bsah\b|mening|miningitis'),

  ("Weil's disease (leptospirosis), clinical diagnosis", r'weil'),

  ('Haematological malignancy / marrow failure', r'leuk|\baml\b|\ball\b|aplastic|a
plastic|marrow'),

  ('Obstetric / puerperal', r'peuperal|puerperal|iufd|\bdil\b'),

  ('Kidney involvement (AKI, renal impairment, CKD)', r'\baki\b|renal|ckd'),

  ('Sepsis', r'sepsis|septic'),

  ('Gastrointestinal bleeding / bleeding disorder', r'ugib|ubib|bleeding'),

  ('Malaria', r'malaria'),

]

def cats(t):

  t = str(t).lower(); out = [c for c, rx in rules if re.search(rx, t)]

  return out

L['cats'] = L.diagnosis_verbatim.apply(cats)
```

```
L['n_cats'] = L.cats.apply(len)

cat_tab = {}

for c, _ in rules:

  s = L[L.cats.apply(lambda x: c in x)]; sd = s[s.outcome_c.isin(['Died', 'Discharged'])]

  cat_tab[c] = dict(n=len(s), deaths=int((s.outcome_c == 'Died').sum()), det=len(sd),

            transferred=int((s.outcome_c == 'Transferred').sum()), notrec=int((s.outcome_c
== 'Not recorded').sum()),

            ids=s.id.tolist())

s = L[L.n_cats == 0]; sd = s[s.outcome_c.isin(['Died', 'Discharged'])]

cat_tab['No other diagnosis recorded (VHF/Lassa fever only)'] = dict(n=len(s),
deaths=int((s.outcome_c == 'Died').sum()), det=len(sd),

  transferred=int((s.outcome_c == 'Transferred').sum()), notrec=int((s.outcome_c == 'Not
recorded').sum()), ids=s.id.tolist())

s = L[L.n_cats >= 1]; sd = s[s.outcome_c.isin(['Died', 'Discharged'])]

R['any_other_dx'] = dict(n=len(s), deaths=int((s.outcome_c == 'Died').sum()), det=len(sd))

R['no_other_dx'] = cat_tab['No other diagnosis recorded (VHF/Lassa fever only)']

p_ = stats.fisher_exact([[R['any_other_dx']['deaths'], R['any_other_dx']['det']-
R['any_other_dx']['deaths']],

            [R['no_other_dx']['deaths'], R['no_other_dx']['det']-
R['no_other_dx']['deaths']]])[1]

R['otherdx_fisher_p'] = p_

R['categories'] = cat_tab

R['weil_record'] = L[L.diagnosis_verbatim.str.contains('weil', case=False, na=False)][['id',
'lab', 'outcome_c', 'diagnosis_verbatim']].to_dict('records')


# ---------- missing data ----------
```

```
R['missing'] = {
  'Age': int(L.age.isna().sum()), 'Sex': int(L.sex.isna().sum()), 'Admission date': int(L.date.isna().sum()),
  'Laboratory result': int((L.lab == 'Not recorded').sum()), 'Outcome': int((L.outcome_c == 'Not recorded').sum()),
  'Length of stay': int(L.length_of_stay_days.isna().sum()),
  'Final vital status (transferred or not recorded)': int(N - n_det),
}

# per-record classification file (supplementary)
L['categories'] = L.cats.apply(lambda x: '; '.join(x) if x else 'None')
L[['id', 'admission_date', 'admission_year', 'sex', 'age', 'ag', 'lab', 'outcome_c', 'length_of_stay_days', 'categories', 'diagnosis_verbatim']].rename(
  columns={'ag': 'age_group_analysis', 'outcome_c': 'outcome'}).to_csv('./vhf_cohort_classified.csv', index=False)
L.to_pickle('./L.pkl')

def conv(o):
  if isinstance(o, (np.integer,)): return int(o)
  if isinstance(o, (np.floating,)): return float(o)
  if isinstance(o, tuple): return list(o)
  return str(o)
json.dump(R, open('./results.json', 'w'), indent=1, default=conv)
print(json.dumps(R, indent=1, default=conv))
```