Education_Dataset_Generator_Notebook

New notebook

SEND + Education Synthetic Dataset Generator (Hybrid LA + MAT)

Outputs CSVs for dimensions + facts:

- dim_date, dim_geography, dim_trust, dim_school, dim_pupil, dim_send_need, dim_intervention
- fact_enrolment, fact_attendance_daily, fact_attainment_termly, fact_behaviour_incidents, fact_send_provision, fact_intervention_participation

Author: Michael Obidey

In [ ]:
```
"""
SEND + Education Synthetic Dataset Generator (Hybrid LA + MAT)
Outputs CSVs for dimensions + facts

Author: Michael Obideyi
"""

from __future__ import annotations
import os
import numpy as np
import pandas as pd
from dataclasses import dataclass
from datetime import date, timedelta
from notebookutils import mssparkutils

# ----------------------------
# Config
# ----------------------------
@dataclass
class Config:
    seed: int = 42
    out_dir: str = "send_education_synthetic"
    start_date: str = "2023-09-01"
    end_date: str = "2025-07-31"
    num_las: int = 3
    num_trusts: int = 3
    num_schools: int = 18
    num_pupils: int = 500

cfg = Config()
rng = np.random.default_rng(cfg.seed)



# Lakehouse Files (visible in UI)
deltafolder = f"Files/{cfg.out_dir}"     #this works as well -> f"Files//lakehouse/
```

```
mssparkutils.fs.mkdirs(deltafolder)

lakehouseFileFolder = "Files/"
deltafolder = lakehouseFileFolder + cfg.out_dir
os.makedirs(deltafolder, exist_ok=True)                                    #os.makedir

# ----------------------------
# Helpers
# ----------------------------
def daterange(start: pd.Timestamp, end: pd.Timestamp):
    d = start
    while d <= end:
        yield d
        d += pd.Timedelta(days=1)

def academic_year(d: pd.Timestamp) -> str:
    # UK academic year: Sept -> Aug
    y = d.year
    if d.month < 9:
        return f"{y-1}/{str(y)[-2:]}"
    return f"{y}/{str(y+1)[-2:]}"

def term_name(d: pd.Timestamp) -> str:
    # simple term mapping
    if d.month in (9,10,11,12):
        return "Autumn"
    if d.month in (1,2,3,4):
        return "Spring"
    return "Summer"

def is_school_day(d: pd.Timestamp) -> int:
    # weekday only; we can refine to exclude holidays in Stage 2 if you want
    return int(d.weekday() < 5)

def weighted_choice(values, weights, size=1):
    return rng.choice(values, size=size, p=np.array(weights)/np.sum(weights))

# ----------------------------
# Dimensions
# ----------------------------
start = pd.to_datetime(cfg.start_date)
end = pd.to_datetime(cfg.end_date)

# dim_date
dates = pd.DataFrame({"Date": pd.date_range(start, end, freq="D")})
dates["DateKey"] = dates["Date"].dt.strftime("%Y%m%d").astype(int)
dates["Year"] = dates["Date"].dt.year
dates["Month"] = dates["Date"].dt.month
dates["Week"] = dates["Date"].dt.isocalendar().week.astype(int)
dates["AcademicYear"] = dates["Date"].apply(academic_year)
dates["Term"] = dates["Date"].apply(term_name)
dates["IsSchoolDay"] = dates["Date"].apply(is_school_day)
dim_date = dates[["DateKey","Date","Year","Month","Week","AcademicYear","Term","IsS

# dim_geography
```

```
la_names = [f"LocalAuthority_{i+1}" for i in range(cfg.num_las)]
wards = []
for la in la_names:
    for w in range(rng.integers(6, 12)):
        wards.append((la, f"{la}_Ward_{w+1}", f"PC{rng.integers(1,30):02d}", "Engla
dim_geo = pd.DataFrame(wards, columns=["LocalAuthority","Ward","PostcodeArea","Regi
dim_geo.insert(0, "GeoKey", np.arange(1, len(dim_geo)+1))

# dim_trust
trusts = []
for i in range(cfg.num_trusts):
    la = rng.choice(la_names)
    trusts.append((f"T{i+1:03d}", f"Trust_{i+1}", rng.choice(["MAT","Single Academy
dim_trust = pd.DataFrame(trusts, columns=["TrustId","TrustName","TrustType","LocalA
dim_trust.insert(0, "TrustKey", np.arange(1, len(dim_trust)+1))

# dim_school
school_types = ["Primary","Secondary","Special"]
schools = []
for i in range(cfg.num_schools):
    st = weighted_choice(school_types, [0.55,0.35,0.10])[0]
    trust = dim_trust.sample(1, random_state=int(rng.integers(1,1e9))).iloc[0]
    geo = dim_geo.sample(1, random_state=int(rng.integers(1,1e9))).iloc[0]
    is_send_specialist = "Y" if st == "Special" else weighted_choice(["Y","N"], [0.
    cap = int(rng.integers(350, 1600)) if st != "Special" else int(rng.integers(80,
    phase = "KS1-2" if st=="Primary" else ("KS3-4" if st=="Secondary" else "Mixed S
    schools.append((
        f"S{i+1:04d}",
        f"School_{i+1}",
        st,
        trust["TrustId"],
        int(geo["GeoKey"]),
        phase,
        is_send_specialist,
        cap
    ))
dim_school = pd.DataFrame(
    schools,
    columns=["SchoolId","SchoolName","SchoolType","TrustId","GeoKey","Phase","IsSen
)
dim_school.insert(0, "SchoolKey", np.arange(1, len(dim_school)+1))

# dim_send_need
need_list = ["ASD","SLCN","SEMH","MLD","SpLD","HI","VI","PD","MSI","OTH"]
need_list_desc = [
    "Autism Spectrum Disorder",
    "Speech, Language and Communication Needs",
    "Social, Emotional and Mental Health",
    "Moderate Learning Difficulty",
    "Specific Learning Difficulty",
    "Hearing Impairment",
    "Visual Impairment",
    "Physical Disability",
    "Multi-Sensory Impairment",
    "Other Difficulty / Disability"
]
```

```
dim_send_need = pd.DataFrame({
    "NeedKey": np.arange(1, len(need_list)+1),
    "PrimaryNeed": need_list,
    "PrimaryNeedDesc": need_list_desc,
    "SecondaryNeed": [""] * len(need_list)   # was getting error void - column of t
})
'''
options to assign empty value to column
"SecondaryNeed": [""] * len(need_list)   # <- not None
dim_send_need["SecondaryNeed"] = dim_send_need["SecondaryNeed"].astype("string")
dim_send_need["SecondaryNeed"] = dim_send_need["SecondaryNeed"].fillna("")
'''

# dim_intervention
interventions = [
    ("Speech&Language","Low"), ("Speech&Language","High"),
    ("SEMH Support","Med"), ("SEMH Support","High"),
    ("OT","Med"), ("OT","High"),
    ("Literacy Catchup","Low"), ("Numeracy Catchup","Low"),
    ("Behaviour Plan","Med"), ("Attendance Mentoring","Low"),
    ("Pastoral Support","Med")
]
dim_intervention = pd.DataFrame(interventions, columns=["InterventionType","Intensi
dim_intervention.insert(0, "InterventionKey", np.arange(1, len(dim_intervention)+1)

# dim_pupil
genders = ["M","F"]
ethnicities = ["White","Black","Asian","Mixed","Other"]
year_groups = list(range(1, 12))  # Y1-Y11
sen_statuses = ["None","SEN Support","EHCP"]

# Assign pupils to schools with capacity-ish distribution
school_probs = dim_school["Capacity"].to_numpy()
school_probs = school_probs / school_probs.sum()
pupil_school_keys = rng.choice(dim_school["SchoolKey"], size=cfg.num_pupils, p=scho

# SEN assignment
sen_flag = rng.random(cfg.num_pupils) < 0.18                # true if less than 18%
# EHCP among SEN                                            # Education, Health, an
ehcp_flag = np.where(sen_flag, rng.random(cfg.num_pupils) < 0.27, False)        # o

sen_status = np.where(~sen_flag, "None", np.where(ehcp_flag, "EHCP", "SEN Support")

dim_pupil = pd.DataFrame({
    "PupilKey": np.arange(1, cfg.num_pupils+1),
    "PupilId": [f"P{i+1:06d}" for i in range(cfg.num_pupils)],
    "UPN_masked": [f"UPN{rng.integers(10**9,10**10-1)}" for _ in range(cfg.num_pupi
    "DOB": pd.to_datetime(rng.integers(
        pd.Timestamp("2007-01-01").value//10**9,
        pd.Timestamp("2018-12-31").value//10**9,
        cfg.num_pupils
    ), unit="s"),
    "Gender": rng.choice(genders, cfg.num_pupils),
    "Ethnicity": rng.choice(ethnicities, cfg.num_pupils, p=[0.6,0.12,0.18,0.06,0.04
    "EAL": rng.choice([0,1], cfg.num_pupils, p=[0.82,0.18]),
    "FSM": rng.choice([0,1], cfg.num_pupils, p=[0.72,0.28]),
```

```
    "LAC_flag": rng.choice([0,1], cfg.num_pupils, p=[0.97,0.03]),
    "SEN_flag": sen_flag.astype(int),
    "SEN_Status": sen_status,
    "SchoolKey": pupil_school_keys,
    "YearGroup": rng.choice(year_groups, cfg.num_pupils, p=[0.09,0.10,0.10,0.10,0.1
})

# ------------------------------
# Facts
# ------------------------------

# fact_enrolment (simple: one current enrolment per pupil; we'll add moves in Stage
enrol_start = pd.to_datetime(cfg.start_date)
fact_enrolment = dim_pupil[["PupilKey","SchoolKey","YearGroup"]].copy()
fact_enrolment["StartDateKey"] = int(enrol_start.strftime("%Y%m%d"))
fact_enrolment["EndDateKey"] = np.nan
fact_enrolment["EntryReason"] = "OnRoll"
fact_enrolment["ExitReason"] = ""   #None
fact_enrolment = fact_enrolment[["PupilKey","SchoolKey","StartDateKey","EndDateKey"

# Attendance generation: daily, school days only
school_days = dim_date[dim_date["IsSchoolDay"]==1]["DateKey"].to_numpy()
n_days = len(school_days)

# base attendance probabilities by SEN_Status
p_present = np.where(dim_pupil["SEN_Status"].eq("None"), 0.95,
             np.where(dim_pupil["SEN_Status"].eq("SEN Support"), 0.92, 0.88))

# create attendance table (this can be large; adjust volume by lowering pupils/date
# We'll sample: generate for all pupils, all school days (portfolio realistic)
att_rows = cfg.num_pupils * n_days
# vectorize sessions
present_am = rng.random(att_rows) < np.repeat(p_present, n_days)
present_pm = rng.random(att_rows) < np.repeat(p_present, n_days)

attendance = pd.DataFrame({
    "PupilKey": np.repeat(dim_pupil["PupilKey"].to_numpy(), n_days),
    "SchoolKey": np.repeat(dim_pupil["SchoolKey"].to_numpy(), n_days),
    "DateKey": np.tile(school_days, cfg.num_pupils),
    "SessionAM": np.where(present_am, "P", "A"),
    "SessionPM": np.where(present_pm, "P", "A"),
})

# Late marks (small probability)
late_am = rng.random(att_rows) < 0.02
late_pm = rng.random(att_rows) < 0.02
attendance.loc[late_am & (attendance["SessionAM"]=="P"), "SessionAM"] = "L"
attendance.loc[late_pm & (attendance["SessionPM"]=="P"), "SessionPM"] = "L"

# AttendanceCode only meaningful when absent
absence_codes = ["Illness","Medical","Unauthorised","Holiday","Other"]
absence_weights_none = [0.55,0.08,0.20,0.10,0.07]
absence_weights_ehcp = [0.45,0.18,0.15,0.10,0.12]
is_absent_any = (attendance["SessionAM"]=="A") | (attendance["SessionPM"]=="A")

codes = []
```

```
abs_idx = np.where(is_absent_any.to_numpy())[0]
# choose weights by SEN status of pupil
sen_map = dim_pupil.set_index("PupilKey")["SEN_Status"].to_dict()
for idx in abs_idx:
    pk = int(attendance.iloc[idx]["PupilKey"])
    st = sen_map[pk]
    w = absence_weights_ehcp if st=="EHCP" else absence_weights_none
    codes.append(weighted_choice(absence_codes, w)[0])

attendance["AttendanceCode"] = ""
attendance.loc[abs_idx, "AttendanceCode"] = codes
fact_attendance_daily = attendance

# Attainment (termly snapshots)
term_ends = dim_date.groupby(["AcademicYear","Term"], as_index=False)["DateKey"].ma
term_end_keys = term_ends["DateKey"].to_list()
subjects = ["English","Maths","Science"]

# baseline attainment by SEN_Status
def grade_band_for_status(status, size):
    if status == "None":
        return weighted_choice(["Below","At","Above"], [0.20,0.60,0.20], size=size)
    if status == "SEN Support":
        return weighted_choice(["Below","At","Above"], [0.35,0.55,0.10], size=size)
    return weighted_choice(["Below","At","Above"], [0.50,0.45,0.05], size=size)

attainment_rows = []
for tk in term_end_keys:
    for subj in subjects:
        bands = []
        for st in dim_pupil["SEN_Status"].to_numpy():
            bands.append(grade_band_for_status(st, 1)[0])
        bands = np.array(bands)
        scores = np.where(bands=="Below", rng.normal(88, 6, cfg.num_pupils),
                 np.where(bands=="At", rng.normal(100, 7, cfg.num_pupils),
                                  rng.normal(112, 6, cfg.num_pupils))).round(0).as
        expected = np.where(bands=="Below", 0, 1)
        attainment_rows.append(pd.DataFrame({
            "PupilKey": dim_pupil["PupilKey"],
            "SchoolKey": dim_pupil["SchoolKey"],
            "DateKey": tk,
            "Subject": subj,
            "GradeBand": bands,
            "ScaledScore": scores,
            "ExpectedProgressFlag": expected
        }))
fact_attainment_termly = pd.concat(attainment_rows, ignore_index=True)

# Behaviour incidents (sparse)
# base incident rate by SEN_Status (annualised-ish)
rate = np.where(dim_pupil["SEN_Status"].eq("None"), 0.12,
        np.where(dim_pupil["SEN_Status"].eq("SEN Support"), 0.18, 0.25))
# number of incidents per pupil (Poisson)
inc_counts = rng.poisson(rate * 6, cfg.num_pupils)  # scaling factor for period
incident_types = ["Disruption","Bullying","Violence","Truancy","Defiance","Other"]
incident_rows = []
```

```
incident_id = 1
date_keys_for_incidents = school_days

for i, cnt in enumerate(inc_counts):
    if cnt <= 0:
        continue
    pk = int(dim_pupil.iloc[i]["PupilKey"])
    sk = int(dim_pupil.iloc[i]["SchoolKey"])
    for _ in range(cnt):
        dk = int(rng.choice(date_keys_for_incidents))
        it = weighted_choice(incident_types, [0.40,0.12,0.08,0.10,0.20,0.10])[0]
        sev = int(np.clip(rng.normal(2.3, 1.1), 1, 5))
        exclusion = int((sev >= 4) and (rng.random() < 0.18))
        ex_type = None
        ex_days = 0
        if exclusion:
            ex_type = weighted_choice(["Fixed","Permanent"], [0.92,0.08])[0]
            ex_days = int(rng.integers(1, 8)) if ex_type=="Fixed" else int(rng.inte
        incident_rows.append((incident_id, pk, sk, dk, it, sev, exclusion, ex_type,
        incident_id += 1

fact_behaviour_incidents = pd.DataFrame(
    incident_rows,
    columns=["IncidentKey","PupilKey","SchoolKey","DateKey","IncidentType","Severit
)

# SEND provision monthly snapshots
month_ends = dim_date.groupby(["AcademicYear","Month"], as_index=False)["DateKey"].

# Assign needs to SEN pupils
sen_pupils = dim_pupil[dim_pupil["SEN_flag"]==1].copy()
need_keys = dim_send_need["NeedKey"].to_numpy()
need_probs = np.array([0.22,0.18,0.20,0.10,0.10,0.05,0.03,0.07,0.01,0.04])
sen_pupils["NeedKey"] = rng.choice(need_keys, size=len(sen_pupils), p=need_probs)

placements = ["Mainstream","Resourced","Special","PRU","Alternative"]
prov_rows = []
for dk in month_ends:
    # subset only SEN pupils get provision rows
    for row in sen_pupils.itertuples(index=False):
        eh = 1 if row.SEN_Status=="EHCP" else 0
        # provision hours
        base_hours = 4 if row.SEN_Status=="SEN Support" else 12
        hours = float(np.clip(rng.normal(base_hours, 2.5), 1, 30))
        # placement type
        if row.SEN_Status=="EHCP":
            placement = weighted_choice(placements, [0.55,0.12,0.25,0.03,0.05])[0]
        else:
            placement = weighted_choice(placements, [0.86,0.10,0.02,0.01,0.01])[0]
        # funding (annual)
        funding = float(np.clip(rng.normal(1800 if row.SEN_Status=="SEN Support" el
        prov_rows.append((row.PupilKey, row.SchoolKey, int(dk), int(row.NeedKey), e

fact_send_provision = pd.DataFrame(
    prov_rows,
    columns=["PupilKey","SchoolKey","DateKey","NeedKey","EHCPFlag","ProvisionHoursP
```

```
)

# Intervention participation (subset)
interv_rows = []
interv_keys = dim_intervention["InterventionKey"].to_numpy()
target_outcomes = ["Attendance","Attainment","Behaviour","Wellbeing"]

# 20% of SEN support, 45% of EHCP get an intervention record
for row in dim_pupil.itertuples(index=False):
    if row.SEN_Status=="None":
        continue
    p = 0.20 if row.SEN_Status=="SEN Support" else 0.45
    if rng.random() > p:
        continue
    ik = int(rng.choice(interv_keys))
    start_dk = int(rng.choice(month_ends))
    # end after 1-4 months (approx)
    end_idx = min(month_ends.index(start_dk) + int(rng.integers(1,5)), len(month_en
    end_dk = int(month_ends[end_idx])
    outcome = rng.choice(target_outcomes, p=[0.35,0.30,0.25,0.10])
    baseline = float(np.clip(rng.normal(50, 15), 0, 100))
    # impact - mixed realistic
    delta = float(rng.normal(6, 8))
    endv = float(np.clip(baseline + delta, 0, 100))
    impact = "High" if delta>12 else ("Med" if delta>5 else ("Low" if delta>1 else
    interv_rows.append((row.PupilKey, row.SchoolKey, ik, start_dk, end_dk, outcome,

fact_intervention_participation = pd.DataFrame(
    interv_rows,
    columns=["PupilKey","SchoolKey","InterventionKey","StartDateKey","EndDateKey","
)

# ------------------------------
# Save to CSV
# Want single CSV file per table name (not folder)?
# Spark will always write a folder. If you want one file named exactly filename.csv
# ------------------------------
'''
def save(df: pd.DataFrame, name: str):
    # path = os.path.join(deltafolder, f"{name}.csv")
    print("Ready to save the datasets dataframes")
    display(df)
    path = f"{deltafolder}/{name}.csv"
    print(path)
    df.to_csv(path, index=False)
    print("Wrote:", path, "rows:", len(df))
'''

def save(df, name: str):
    out_path = f"{deltafolder}/{name}"   # Spark writes a folder

    tmp = f"{deltafolder}/_{name}_tmp"
    final = f"{deltafolder}/{name}.csv"

    # clean old
    if mssparkutils.fs.exists(tmp):
```

```
        mssparkutils.fs.rm(tmp, recurse=True)
    if mssparkutils.fs.exists(final):
        mssparkutils.fs.rm(final, recurse=True)

    print("Ready to save the datasets dataframes")
    display(df)

    sdf = spark.createDataFrame(df)
    (sdf.coalesce(1)
        .write.mode("overwrite")
        .option("header", True)
        .csv(tmp))

    # find the part file and move it
    part = [f.path for f in mssparkutils.fs.ls(tmp) if f.name.startswith("part-") a
    mssparkutils.fs.mv(part, final)
    mssparkutils.fs.rm(tmp, recurse=True)

    print("Wrote:", final)
    print(f"Contents of {deltafolder} folder :", [x.name for x in mssparkutils.fs.l

save(dim_date, "dim_date")
save(dim_geo, "dim_geography")
save(dim_trust, "dim_trust")

save(dim_school, "dim_school")
save(dim_pupil.drop(columns=["SchoolKey","YearGroup"]), "dim_pupil")  # keep enrolm
save(dim_send_need, "dim_send_need")
save(dim_intervention, "dim_intervention")

save(fact_enrolment, "fact_enrolment")
save(fact_attendance_daily, "fact_attendance_daily")
save(fact_attainment_termly, "fact_attainment_termly")
save(fact_behaviour_incidents, "fact_behaviour_incidents")
save(fact_send_provision, "fact_send_provision")
save(fact_intervention_participation, "fact_intervention_participation")

print("\nDone. Next: load these into Fabric/Lakehouse as Bronze tables.")
```