ETL_netflix_v3_countries

October 19, 2025

[ ]:
```
try:
    get_ipython().run_line_magic('matplotlib', 'inline')
except Exception:
    pass
import numpy as np, pandas as pd, re, unicodedata, ast
from pathlib import Path
from datetime import datetime

np.random.seed(42)
anio_actual = datetime.now().year

# Rutas
base_dir = Path(r'C:\Users\mario\Downloads\PRUEBA_BI BOOSTER\00_Datos')
out_dir  = base_dir / 'etl_output_v3_countries'
out_dir.mkdir(exist_ok=True, parents=True)
print('Salida ETL v3 + Countries:', out_dir)
```

0.1 1) Lectura tolerante y disponibilidad

[ ]:
```
def leer_csv_tolerante(path):
    encodings = ['utf-8','utf-8-sig','latin-1']
    seps = [',',';','\t']
    ult = None
    for enc in encodings:
        for sep in seps:
            try:
                df = pd.read_csv(path, encoding=enc, sep=sep, engine='python')
                if df.shape[1]==1 and sep!='\t':
                    continue
                return df
            except Exception as e:
                ult = e
                continue
    print(f'AVISO: no se pudo leer {path} → {ult}')
    return None

rutas = {
```

```
    'data_netflix'      : base_dir / 'data_netflix.csv',
    'actores'           : base_dir / 'Actores.csv',
    'mejores_peliculas' : base_dir / 'mejores peliculas Netflix.csv',
    'mejores_shows'     : base_dir / 'mejores Shows Netflix.csv',
}
data_netflix      = leer_csv_tolerante(rutas['data_netflix'])
actores           = leer_csv_tolerante(rutas['actores'])
mejores_peliculas = leer_csv_tolerante(rutas['mejores_peliculas'])
mejores_shows     = leer_csv_tolerante(rutas['mejores_shows'])

print('Cargadas:', [k for k,v in␣
 ↪[('data_netflix',data_netflix),('actores',actores),('mejores_peliculas',mejores_peliculas),
 ↪if v is not None])
if data_netflix is None or actores is None:
    raise SystemExit('Faltan insumos mínimos (data_netflix / Actores)')
```

0.2 2) Auxiliares (normalización, reglas, imputación selectiva, caps)

[ ]:
```
def norm_title(s):
    if pd.isna(s): return s
    s = unicodedata.normalize('NFKD', str(s)).encode('ascii','ignore').
 ↪decode('ascii')
    s = re.sub(r'[^a-z0-9 ]+',' ', s.lower())
    s = re.sub(r'\s+',' ', s).strip()
    return s

def derive_main_genre_from_genres(val):
    if pd.isna(val):
        return pd.NA
    s = str(val)
    try:
        x = ast.literal_eval(s)
        if isinstance(x, (list, tuple)) and len(x)>0:
            g = str(x[0]).strip()
            return g if g else pd.NA
    except Exception:
        parts = re.split(r'[\,;\|\/]', s)
        if parts:
            g = parts[0].strip()
            if g and g.lower() not in {'nan','none',''}:
                return g
    return pd.NA

def vista_estandar(df, mapping):
    d = df.rename(columns={k:v for k,v in mapping.items() if k in df.columns}).
 ↪copy()
```

```
    keep = [c for c in␣
 ↪['title','release_year','imdb_score','imdb_votes','runtime','seasons','main_genre','type','
 ↪if c in d.columns]
    d = d[keep]
    if {'title','release_year'}.issubset(d.columns):
        d['title_norm'] = d['title'].map(norm_title)
    return d

def coerce_tipos(df):
    for c in ['imdb_score','imdb_votes','runtime','seasons','release_year']:
        if c in df.columns:
            df[c] = pd.to_numeric(df[c], errors='coerce')
    return df

def aplicar_reglas(df, anio_actual):
    d = df.copy()
    if 'imdb_score' in d.columns:
        d.loc[~d['imdb_score'].between(0,10) & d['imdb_score'].notna(),␣
 ↪'imdb_score'] = np.nan
    if 'imdb_votes' in d.columns:
        d.loc[(d['imdb_votes']<0) & d['imdb_votes'].notna(), 'imdb_votes'] = np.
 ↪nan
    if 'runtime' in d.columns:
        d.loc[~d['runtime'].between(1,600) & d['runtime'].notna(), 'runtime'] =␣
 ↪np.nan
    if 'seasons' in d.columns:
        d.loc[(d['seasons']<0) & d['seasons'].notna(), 'seasons'] = np.nan
    if 'release_year' in d.columns:
        d.loc[(d['release_year']>anio_actual) & d['release_year'].notna(),␣
 ↪'release_year'] = np.nan
    if {'type','seasons'}.issubset(d.columns):
        d.loc[d['type']!='SHOW', 'seasons'] = np.nan
    return d

def coalesce3(a,b,c):
    return a.combine_first(b).combine_first(c)

def hot_deck_group(df, cols, group_cols, seed=42):
    rng = np.random.default_rng(seed)
    def _grp(g):
        g = g.copy()
        for c in cols:
            if c not in g.columns: continue
            m = g[c].isna()
            if m.any():
                donors = g.loc[~g[c].isna(), c].values
                if donors.size>0:
```

```
                    g.loc[m, c] = rng.choice(donors, size=m.sum(), replace=True)
        return g
    return df.groupby(group_cols, dropna=False, group_keys=False).apply(_grp)

def imputacion_fallback_medianas(df, cols, niveles):
    d = df.copy()
    for c in cols:
        for nivel in niveles:
            if not set(nivel+[c]).issubset(d.columns):
                continue
            d[c] = d[c].fillna(d.groupby(nivel)[c].transform('median'))
        d[c] = d[c].fillna(d[c].median())
    return d

def winsorizar_cap(s, p_low=None, p_high=0.99):
    x = s.copy()
    low = x.quantile(p_low) if p_low is not None else None
    high = x.quantile(p_high) if p_high is not None else None
    if low is not None:
        x = np.where(x < low, low, x)
    if high is not None:
        x = np.where(x > high, high, x)
    return pd.Series(x, index=s.index)
```

0.3 3) Estandarización + derivación de main_genre + reglas + deduplicación

[ ]:
```
dn = data_netflix.copy()
dn['title_norm'] = dn['title'].map(norm_title) if 'title' in dn.columns else np.
 ↪nan
dn = coerce_tipos(dn)
dn = aplicar_reglas(dn, anio_actual)

# Derivar main_genre si es posible
if 'main_genre' not in dn.columns or dn['main_genre'].isna().all():
    if 'genres' in dn.columns:
        dn['main_genre'] = dn['genres'].apply(derive_main_genre_from_genres)
dn['main_genre'] = dn.get('main_genre').astype('string') if 'main_genre' in dn.
 ↪columns else pd.Series(pd.NA, index=dn.index, dtype='string')

dn = dn.drop_duplicates(keep='first').copy()
print('data_netflix listo:', dn.shape)
display(dn.head(3))
```

0.4 4) Donantes y coalesce (métricas + main_genre)

[ ]:
```
v_mp = vista_estandar(mejores_peliculas, {
    'TITLE':'title','RELEASE_YEAR':'release_year','SCORE':'imdb_score',
    'NUMBER_OF_VOTES':'imdb_votes','DURATION':'runtime','MAIN_GENRE':
 ↪'main_genre','TYPE':'type'
}) if mejores_peliculas is not None else None
v_ms = vista_estandar(mejores_shows, {
    'TITLE':'title','RELEASE_YEAR':'release_year','SCORE':'imdb_score',
    'NUMBER_OF_VOTES':'imdb_votes','DURATION':'runtime','NUMBER_OF_SEASONS':
 ↪'seasons',
    'MAIN_GENRE':'main_genre','TYPE':'type'
}) if mejores_shows is not None else None

enri  = dn.copy()
llave = ['title_norm','release_year']
base_cols = ['imdb_score','imdb_votes','runtime','seasons']
extra_cols = ['main_genre']

def merge_donante(base, donante, sufijo):
    if (donante is None) or (not set(llave).issubset(donante.columns)):
        return base
    cols_don = [c for c in base_cols + extra_cols if c in donante.columns]
    if not cols_don:
        return base
    return base.merge(donante[llave + cols_don], on=llave, how='left',␣
 ↪suffixes=('', sufijo))

enri = merge_donante(enri, v_mp, '_mp')
enri = merge_donante(enri, v_ms, '_ms')

for c in base_cols:
    if f'{c}_mp' not in enri.columns: enri[f'{c}_mp'] = np.nan
    if f'{c}_ms' not in enri.columns: enri[f'{c}_ms'] = np.nan
for c in extra_cols:
    if f'{c}_mp' not in enri.columns: enri[f'{c}_mp'] = pd.NA
    if f'{c}_ms' not in enri.columns: enri[f'{c}_ms'] = pd.NA

for c in base_cols:
    a = enri[c] if c in enri.columns else pd.Series(index=enri.index,␣
 ↪dtype=float)
    b = enri[f'{c}_mp']
    d_ = enri[f'{c}_ms']
    final = coalesce3(a, b, d_)
    src   = np.where(~a.isna(),'data_netflix', np.where(~b.
 ↪isna(),'mejores_peliculas', np.where(~d_.isna(),'mejores_shows','missing')))
    enri[f'{c}_final']  = final
```

```
    enri[f'source_{c}'] = src

if 'main_genre' in enri.columns:
    enri['main_genre'] = enri['main_genre'].fillna(enri.get('main_genre_mp')).
 ↪fillna(enri.get('main_genre_ms'))

drop_cols = [col for col in enri.columns if col.endswith('_mp') or col.
 ↪endswith('_ms')]
enri = enri.drop(columns=drop_cols)
print('Coalesce listo:', enri.shape)
display(enri.head(3))
```

0.5 5) Imputaciones sin seasons (hot-deck → medianas) + flags

[ ]:
```
imputable = ['imdb_score','imdb_votes','runtime']  # seasons EXCLUIDO
cols_obj = [f'{c}_final' for c in imputable]
trab = enri.copy()

if {'type','main_genre'}.issubset(trab.columns) and cols_obj:
    trab = hot_deck_group(trab, cols_obj, ['type','main_genre'], seed=42)

niveles = [['type','main_genre'], ['type']]
trab = imputacion_fallback_medianas(trab, cols_obj, niveles)

for c in imputable:
    mask_missing = enri[f'{c}_final'].isna()
    mask_now     = trab[f'{c}_final'].notna()
    trab[f'was_imputed_{c}'] = (mask_missing & mask_now).astype(int)

print('Imputaciones OK (sin seasons):', trab.shape)
display(trab.head(3))
```

0.6 6) Outliers y coherencias finales

[ ]:
```
final = trab.copy()

if 'imdb_votes_final' in final.columns:
    s = pd.to_numeric(final['imdb_votes_final'], errors='coerce')
    p99_votes = s.quantile(0.99) if s.notna().any() else np.nan
    final['was_capped_imdb_votes'] = ((~s.isna()) & (s > p99_votes)).
 ↪astype(int) if s.notna().any() else 0
    final['imdb_votes_final'] = winsorizar_cap(s, p_high=0.99) if s.notna().
 ↪any() else s
else:
    p99_votes = np.nan
```

```
if 'runtime_final' in final.columns:
    r = pd.to_numeric(final['runtime_final'], errors='coerce')
    p99_rt = min(600, r.quantile(0.99) if r.notna().any() else 600)
    final['was_capped_runtime'] = ((~r.isna()) & (r > p99_rt)).astype(int) if r.
 ↪notna().any() else 0
    final['runtime_final'] = np.where(r > p99_rt, p99_rt, r)
else:
    p99_rt = np.nan

if {'seasons_final','type'}.issubset(final.columns):
    final.loc[final['type']!='SHOW','seasons_final'] = np.nan  # sin imputar

print({'p99_votes': float(p99_votes) if pd.notna(p99_votes) else None,␣
 ↪'p99_runtime': float(p99_rt) if pd.notna(p99_rt) else None})
display(final.head(3))
```

0.7 7) Modelo BI (dims/facts/puente)

[ ]:
```
# dim_title
tit_cols = [c for c in␣
 ↪['imdb_id','title','title_norm','type','main_genre','age_certification','release_year','run
 ↪if c in final.columns]
dim_title = final[tit_cols].drop_duplicates().reset_index(drop=True).copy()
dim_title.insert(0,'title_id', np.arange(1, len(dim_title)+1))

# fact_ratings con fallback de title_id
fact_cols = [c for c in [
    'imdb_id','title','title_norm','release_year',
    'imdb_score_final','imdb_votes_final','runtime_final','seasons_final',
    'source_imdb_score','source_imdb_votes','source_runtime','source_seasons',
    'was_imputed_imdb_score','was_imputed_imdb_votes','was_imputed_runtime',
    'was_capped_imdb_votes','was_capped_runtime'
] if c in final.columns]
fact_ratings = final[fact_cols].copy()

key_on = 'imdb_id' if 'imdb_id' in dim_title.columns else None
if key_on and key_on in fact_ratings.columns:
    fact_ratings = fact_ratings.merge(dim_title[['title_id', key_on]],␣
 ↪on=key_on, how='left')
mask_faltante = fact_ratings['title_id'].isna() if 'title_id' in fact_ratings.
 ↪columns else pd.Series(False, index=fact_ratings.index)
if mask_faltante.any() and {'title_norm','release_year'}.issubset(fact_ratings.
 ↪columns):
    aux = dim_title[['title_id','title_norm','release_year']].dropna().
 ↪drop_duplicates()
```

```
    fact_ratings = fact_ratings.merge(aux, on=['title_norm','release_year'],␣
 ↪how='left', suffixes=('','_by_norm'))
    if 'title_id_by_norm' in fact_ratings.columns:
        fact_ratings['title_id'] = fact_ratings['title_id'].
 ↪fillna(fact_ratings['title_id_by_norm'])
        fact_ratings = fact_ratings.drop(columns=[c for c in fact_ratings.
 ↪columns if c.endswith('_by_norm')])
if 'title_id' in fact_ratings.columns:
    fact_ratings = fact_ratings[['title_id'] + [c for c in fact_ratings.columns␣
 ↪if c!='title_id']]

# dim_person
act = actores.copy()
for c in ['name','character','role']:
    if c in act.columns:
        act[c] = act[c].astype(str)
        act[c] = act[c].where(act[c].str.strip().ne(''), other=np.nan)
        act[c] = act[c].fillna('desconocido')
pers_id_cols = [c for c in␣
 ↪['person_id','imdb_person_id','person_imdb_id','imdb_id'] if c in act.
 ↪columns]
text_cols = [c for c in ['name'] if c in act.columns]
dim_person = act[pers_id_cols + text_cols].drop_duplicates().
 ↪reset_index(drop=True).copy()
if 'person_id' not in dim_person.columns:
    dim_person.insert(0,'person_id', np.arange(1, len(dim_person)+1))

# bridge_title_person con crosswalk
btp_cols = [c for c in ['id','imdb_id','person_id','role','character'] if c in␣
 ↪act.columns]
bridge_title_person = act[btp_cols].copy()
cols_cross = [c for c in ['id','imdb_id','title','release_year','title_norm']␣
 ↪if c in dn.columns]
xwalk = dn[cols_cross].dropna(subset=['id']).drop_duplicates()
if 'imdb_id' in bridge_title_person.columns:
    bridge_title_person = bridge_title_person.merge(xwalk[['id','imdb_id']],␣
 ↪on='id', how='left', suffixes=('','_from_dn'))
    bridge_title_person['imdb_id'] = bridge_title_person['imdb_id'].
 ↪fillna(bridge_title_person.get('imdb_id_from_dn'))
    bridge_title_person = bridge_title_person.drop(columns=[c for c in␣
 ↪bridge_title_person.columns if c.endswith('_from_dn')])
else:
    bridge_title_person = bridge_title_person.merge(xwalk[['id','imdb_id']],␣
 ↪on='id', how='left')
bridge_title_person = bridge_title_person.
 ↪merge(dim_title[['title_id','imdb_id']], on='imdb_id', how='left')
```

```
faltan_title_id = bridge_title_person['title_id'].isna() if 'title_id' in␣
 ↪bridge_title_person.columns else pd.Series(False, index=bridge_title_person.
 ↪index)
if faltan_title_id.any() and {'title_norm','release_year'}.issubset(xwalk.
 ↪columns):
    aux_dim = dim_title[['title_id','title_norm','release_year']].dropna().
 ↪drop_duplicates()
    bridge_title_person = bridge_title_person.
 ↪merge(xwalk[['id','title_norm','release_year']], on='id', how='left',␣
 ↪suffixes=('','_xw'))
    bridge_title_person = bridge_title_person.merge(aux_dim,␣
 ↪on=['title_norm','release_year'], how='left', suffixes=('','_by_norm'))
    if 'title_id_by_norm' in bridge_title_person.columns:
        bridge_title_person['title_id'] = bridge_title_person['title_id'].
 ↪fillna(bridge_title_person['title_id_by_norm'])
        bridge_title_person = bridge_title_person.drop(columns=[c for c in␣
 ↪bridge_title_person.columns if c.endswith('_by_norm') or c.endswith('_xw')])

print('dim_title:', dim_title.shape)
print('fact_ratings:', fact_ratings.shape, '| cobertura title_id en fact =',␣
 ↪round(fact_ratings['title_id'].notna().mean()*100,2) if 'title_id' in␣
 ↪fact_ratings.columns else 'N/A')
print('dim_person:', dim_person.shape)
print('bridge_title_person:', bridge_title_person.shape, '| fk_title_id_% =',␣
 ↪round(bridge_title_person['title_id'].notna().mean()*100,2) if 'title_id' in␣
 ↪bridge_title_person.columns else 'N/A')
display(dim_title.head(3))
display(fact_ratings.head(3))
display(dim_person.head(3))
display(bridge_title_person.head(3))
```

0.8 7.5) Países (dim_country + bridge_title_country)

[ ]:
```
# Detección de columna de país
country_col = next((c for c in␣
 ↪['production_countries','origin_country','countries','country'] if c in dn.
 ↪columns), None)

def parse_iso2_list(val):
    if pd.isna(val):
        return []
    s = str(val)
    # Intenta lista Python
    try:
        x = ast.literal_eval(s)
        if isinstance(x, (list, tuple, set)):
```

```
            return [str(i).strip().upper() for i in x if str(i).strip()]
    except Exception:
        pass
    # Separadores comunes
    parts = re.split(r'[,;|/]+', s)
    return [p.strip().upper() for p in parts if p.strip()]

dim_country = None
bridge_title_country = None
cobertura_pais = None

if country_col:
    tmp = dn[['title_norm','release_year', country_col]].copy()
    tmp['country_list'] = tmp[country_col].apply(parse_iso2_list)

    # Explota N:M
    expl = tmp.explode('country_list').rename(columns={'country_list':
 ↪'country_code'})
    expl = expl.dropna(subset=['country_code'])

    # Dim país
    dim_country = expl[['country_code']].drop_duplicates().
 ↪reset_index(drop=True)
    dim_country.insert(0, 'country_id', np.arange(1, len(dim_country)+1))

    # Bridge título-país (fallback por title_norm+release_year)
    aux_keys = dim_title[['title_id','title_norm','release_year']].
 ↪drop_duplicates()
    bridge_title_country = expl.merge(aux_keys,␣
 ↪on=['title_norm','release_year'], how='left')
    bridge_title_country = bridge_title_country.merge(dim_country,␣
 ↪on='country_code', how='left')
    bridge_title_country = bridge_title_country[['title_id','country_id']].
 ↪dropna().drop_duplicates()

    cobertura_pais = round(100 * (expl['country_code'].notna().mean()), 2)
    print('dim_country:', dim_country.shape, '| bridge_title_country:',␣
 ↪bridge_title_country.shape, '| Cobertura país %:', cobertura_pais)
    display(dim_country.head(3))
    display(bridge_title_country.head(3))
else:
    print('No se encontró columna de país en data_netflix; se omite␣
 ↪construcción de mapa.')
```

0.9 8) Exportación y KPIs de QA

[ ]:
```
def save_outputs(df, name):
    csv_path = out_dir / f'{name}.csv'
    pq_path  = out_dir / f'{name}.parquet'
    df.to_csv(csv_path, index=False)
    try:
        df.to_parquet(pq_path, index=False)
    except Exception as e:
        print('Parquet no disponible para', name, ':', e)
    return str(csv_path), str(pq_path)

paths = {}
paths['dim_title'] = save_outputs(dim_title, 'dim_title')
paths['fact_ratings'] = save_outputs(fact_ratings, 'fact_ratings')
paths['dim_person'] = save_outputs(dim_person, 'dim_person')
paths['bridge_title_person'] = save_outputs(bridge_title_person,␣
 ↪'bridge_title_person')

if 'dim_country' in globals() and dim_country is not None and␣
 ↪'bridge_title_country' in globals() and bridge_title_country is not None:
    paths['dim_country'] = save_outputs(dim_country, 'dim_country')
    paths['bridge_title_country'] = save_outputs(bridge_title_country,␣
 ↪'bridge_title_country')

print('Exportado en:', out_dir)
paths

def pct_nulos(s):
    return round(float(s.isna().mean()*100), 2)

kpis = {}
for col in␣
 ↪['imdb_score_final','imdb_votes_final','runtime_final','seasons_final']:
    if col in final.columns:
        kpis[f'nulos_{col}_%'] = pct_nulos(final[col])
if {'title_norm','release_year'}.issubset(final.columns):
    kpis['cobertura_llave_title_norm_release_year_%'] = round(100 *␣
 ↪((final['title_norm'].notna()) & (final['title_norm'].astype(str).str.
 ↪strip()!='') & (final['release_year'].notna())).mean(), 2)
kpis['duplicados_dim_title'] = int(dim_title.duplicated().sum())
kpis['duplicados_fact_ratings'] = int(fact_ratings.duplicated().sum())
kpis['duplicados_bridge_title_person'] = int(bridge_title_person.duplicated().
 ↪sum())
kpis['winsorized_votes_rows'] = int(final.get('was_capped_imdb_votes', pd.
 ↪Series(dtype=int)).sum())
if 'title_id' in fact_ratings.columns:
```

```
    kpis['cobertura_title_id_fact_%'] = round(float(fact_ratings['title_id'].
 ↪notna().mean()*100), 2)
if 'title_id' in bridge_title_person.columns:
    kpis['fk_title_id_bridge_%'] = round(float(bridge_title_person['title_id'].
 ↪notna().mean()*100), 2)
if 'dim_country' in globals() and dim_country is not None and␣
 ↪'bridge_title_country' in globals() and bridge_title_country is not None:
    kpis['cobertura_country_%'] = cobertura_pais

print('KPIs de calidad ETL v3 + Countries:')
for k,v in kpis.items():
    print('-', k, ':', v)
kpis
```