02_data_quality_validation_and_imputation.ipynb

```
#%%
import os
import pandas as pd
from pathlib import Path
import missingno as msno
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error

#%%
repo_root = Path.cwd()
data_dir  = (repo_root / "data") if (repo_root / "data").exists() else repo_root

print("repo_root:", repo_root)
print("data_dir :", data_dir.resolve())

filenames = ['sales.csv', 'products.csv', 'categories.csv', 'stores.csv', 'vendors.csv']

paths = {name: (data_dir / name) for name in filenames}
for name, p in paths.items():
    print(f"{name}: {p.exists()} -> {p.resolve()}")

#%%
paths = {
    'sales': Path('sales.csv'),
    'products': Path('products.csv'),
    'categories': Path('categories.csv'),
    'stores': Path('stores.csv'),
    'vendors': Path('vendors.csv'),
}

def read_csv_smart(p):
    cols = pd.read_csv(p, nrows=0).columns.str.lower()
    maybedates = [c for c in cols if 'date' in c or 'datetime' in c]
    df = pd.read_csv(p, parse_dates=[c for c in maybedates if c in cols], low_memory=False)
    df.columns = df.columns.str.lower().str.strip()
    return df

sales_df = read_csv_smart(paths['sales'])
products_df = read_csv_smart(paths['products'])
categories_df = read_csv_smart(paths['categories'])
stores_df = read_csv_smart(paths['stores'])
vendors_df = read_csv_smart(paths['vendors'])

for name, df in [('sales', sales_df), ('products', products_df), ('categories', categories_df), ('stores', stores_df), ('vendors', vendors_df)]:
    print(f"\n{name}: {df.shape}")
    display(df.head(3))
    display(df.dtypes.to_frame('dtype').T)

#%%
dfs = {
    'sales': sales_df,
    'products': products_df,
    'categories': categories_df,
    'stores': stores_df,
    'vendors': vendors_df
}

for name, df in dfs.items():
    print(f"\n{name.upper()}  shape={df.shape}")
    print(df.isna().sum())
    print(df.dtypes)

msno.matrix(sales_df, figsize=(10,4))

#%%
dup_txn_df = sales_df[sales_df.duplicated(['txn_id'], keep=False)]
print('dup txn_id rows:', len(dup_txn_df))

orph_prod_df = (sales_df[['product_id']].drop_duplicates()
                .merge(products_df[['product_id']], on='product_id', how='left', indicator=True)
                .query('_merge == "left_only"').drop(columns=['_merge']))
print('orphan product_id:', len(orph_prod_df))

orph_store_df = (sales_df[['store_id']].drop_duplicates()
                 .merge(stores_df[['store_id']], on='store_id', how='left', indicator=True)
                 .query('_merge == "left_only"').drop(columns=['_merge']))
print('orphan store_id:', len(orph_store_df))

display(dup_txn_df.head(5))
display(orph_prod_df.head(5))
display(orph_store_df.head(5))

#%%
valid_prod = set(products_df['product_id'])
valid_store = set(stores_df['store_id'])

mask = sales_df['product_id'].isin(valid_prod) & sales_df['store_id'].isin(valid_store)
```

```
sales_fix_df = sales_df.loc[mask].copy()

print('sales before:', sales_df.shape, 'after:', sales_fix_df.shape, 'dropped:', len(sales_df) - len(sales_fix_df))

bad_rows_df = sales_df.loc[~mask, ['txn_id','store_id','product_id']].drop_duplicates()
display(bad_rows_df.head())

#%%
exact_dups_df = sales_fix_df[sales_fix_df.duplicated(keep=False)].sort_values(list(sales_fix_df.columns))
print('exact dup rows:', len(exact_dups_df))

tx_counts_df = sales_fix_df['txn_id'].value_counts().rename_axis('txn_id').reset_index(name='n')
print(tx_counts_df.head())

sales_nodup_df = sales_fix_df.drop_duplicates().copy()
sales_nodup_df['line_no'] = sales_nodup_df.groupby('txn_id').cumcount() + 1

dup_key_n = sales_nodup_df.duplicated(['txn_id','line_no']).sum()
print('dup (txn_id,line_no):', dup_key_n)

display(sales_nodup_df.head())

#%%
imp_df = sales_nodup_df.copy()

g_date = imp_df.groupby('txn_id')['txn_date'].transform(lambda s: s.dropna().iloc[0] if s.notna().any() else pd.NaT)
imp_df['txn_date'] = imp_df['txn_date'].fillna(g_date)
store_med_date = imp_df.groupby('store_id')['txn_date'].transform(lambda s: s.dropna().median())
imp_df['txn_date'] = imp_df['txn_date'].fillna(store_med_date)
imp_df['txn_date'] = imp_df['txn_date'].fillna(imp_df['txn_date'].median())

prod_qty_med = imp_df.groupby('product_id')['qty'].transform('median')
store_qty_med = imp_df.groupby('store_id')['qty'].transform('median')
imp_df['qty'] = imp_df['qty'].fillna(prod_qty_med).fillna(store_qty_med).fillna(imp_df['qty'].median())

prod_price_med = imp_df.groupby('product_id')['unit_price'].transform('median')
store_price_med = imp_df.groupby('store_id')['unit_price'].transform('median')
imp_df['unit_price'] = (imp_df['unit_price']
                        .fillna(prod_price_med).fillna(store_price_med).fillna(imp_df['unit_price'].median()))

calc = (imp_df['qty'] * imp_df['unit_price']).round(2)
bad = (imp_df['line_amount'].fillna(-9999).round(2) != calc) | (imp_df['line_amount'].isna())
imp_df.loc[bad, 'line_amount'] = calc

print(imp_df[['txn_date','qty','unit_price','line_amount']].isna().sum())
imp_df.head()

#%%
fig, axs = plt.subplots(1,2, figsize=(12,4))

sales_nodup_df['qty'].hist(ax=axs[0], bins=30, alpha=0.6, label='pre')
imp_df['qty'].hist(ax=axs[0], bins=30, alpha=0.6, label='post')
axs[0].set_title('qty pre vs post')
axs[0].legend()

sales_nodup_df['unit_price'].hist(ax=axs[1], bins=30, alpha=0.6, label='pre')
imp_df['unit_price'].hist(ax=axs[1], bins=30, alpha=0.6, label='post')
axs[1].set_title('unit_price pre vs post')
axs[1].legend()

plt.show()

#%%
daily_df = (imp_df.groupby(['txn_date','store_id'])['line_amount']
            .sum().reset_index())

daily_df['dow'] = daily_df['txn_date'].dt.dayofweek

X = daily_df[['dow']]
y = daily_df['line_amount']

cut = int(len(daily_df)*0.8)
X_train, X_test = X[:cut], X[cut:]
y_train, y_test = y[:cut], y[cut:]

m = LinearRegression().fit(X_train, y_train)
y_pred = m.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
print('baseline mae:', round(mae,2))

#%%
REPO_ROOT = Path.cwd()

DATA_DIR = (REPO_ROOT / "data") if (REPO_ROOT / "data").exists() else REPO_ROOT

print("repo root:", REPO_ROOT)
print("data dir :", DATA_DIR)

#%%
```