End-to-End ETL Pipeline for Data Analysis and Visualization

Authors: Aniket Gulab Khedkar
Contact: aniketgk@umich.edu

0) Stack

- **Source**: Kaggle Olist CSVs
- **DB**: Microsoft SQL Server (on-prem or Azure SQL)
- **Ingest**: Python (pandas + SQLAlchemy + pyodbc, `fast_executemany=True`)
- **Modeling**: dbt-sqlserver (T-SQL) to build dimensions & facts
- **Quality**: lightweight checks (Python)
- **Viz**: Power BI / Tableau (+ optional Streamlit app)

1) Project Layout

```
olist-mssql-etl/
  ├── configs/
  │   ├── .env.example
  │   └── dbt_profile.yml
  ├── sql/
  │   ├── create_schemas.sql
  │   ├──
create_staging_tables.sql
  │   └── helper_indexes.sql
  ├── dbt_project/
  │   ├── dbt_project.yml
  │   └── models/
  │       ├── sources.yml
  │       ├── staging/
  │       │   ├── stg_orders.sql
  │       │   ├──
stg_order_items.sql
  │       │   ├── stg_payments.sql
  │       │   ├──
stg_customers.sql
  │       │   ├── stg_products.sql
  │       │   ├── stg_sellers.sql
  │       │   ├──
stg_geolocation.sql
  │       │   └── stg_reviews.sql
  │       ├── warehouse/
  │       │   ├── dim_date.sql
  │       │   ├── dim_customer.sql
  │       │   ├── dim_product.sql
  │       │   ├── dim_seller.sql
  │       │   ├──
fact_order_items.sql
  │       │   ├──
fact_payments.sql
  │       │   └── fact_orders.sql
  │       └── marts/
  │           └──
sales_summary.sql
  ├── scripts/
  │   ├── kaggle_download.sh
  │   ├── load_csvs_to_staging.py
  │   ├── run_quality_checks.py
  │   └── streamlit_app.py
  ├── requirements.txt
  └── README.md
```

2) Install & Configure

2.1 Python env

```
python -m venv .venv && source
.venv/bin/activate   # (Windows:
.venv\Scripts\activate)
pip install -r requirements.txt
```

requirements.txt

```
python-dotenv>=1.0
pandas>=2.2
pyodbc>=5.1
SQLAlchemy>=2.0
tqdm>=4.66
# dbt for SQL Server
dbt-core>=1.7
dbt-sqlserver>=1.7
# dashboard
streamlit>=1.37
plotly>=5.22
```

Install **ODBC Driver 18 for SQL Server** on your OS.

2.2 Kaggle CLI

- Create/deploy your Kaggle API token: https://www.kaggle.com/settings/account
- Place `kaggle.json` under `~/.kaggle/` (Linux/Mac) or `%USERPROFILE%\.kaggle\` (Windows).

2.3 `.env` (copy from `configs/.env.example`)

```
MSSQL_SERVER=localhost
MSSQL_PORT=1433
MSSQL_DATABASE=OlistDB
MSSQL_USERNAME=sa
MSSQL_PASSWORD=YourStrong!Passw0rd
MSSQL_DRIVER=ODBC Driver 18 for
SQL Server
```

---

3) Download Olist CSVs (Kaggle)

scripts/kaggle_download.sh

```
#!/usr/bin/env bash
set -euo pipefail
mkdir -p data/raw
kaggle datasets download -d
olistbr/brazilian-ecommerce -p
data/raw
unzip -o data/raw/brazilian-
ecommerce.zip -d data/raw
```

Run:

```
bash scripts/kaggle_download.sh
```

---

4) Create Schemas & Staging Tables (SQL Server)

4.1 Schemas

```
-- sql/create_schemas.sql
USE OlistDB;
IF NOT EXISTS (SELECT * FROM
sys.schemas WHERE name='stg')
EXEC('CREATE SCHEMA stg');
IF NOT EXISTS (SELECT * FROM
sys.schemas WHERE name='wh')
EXEC('CREATE SCHEMA wh');
```

4.2 Staging tables (1:1 with Kaggle CSVs)

```
-- sql/create_staging_tables.sql
USE OlistDB;

CREATE TABLE stg.olist_orders (
  order_id
VARCHAR(50) PRIMARY KEY,
  customer_id
VARCHAR(50),
  order_status
VARCHAR(32),
  order_purchase_timestamp
DATETIME2,
  order_approved_at
DATETIME2,
  order_delivered_carrier_date
DATETIME2,
  order_delivered_customer_date
DATETIME2,
  order_estimated_delivery_date
DATETIME2
);

CREATE TABLE stg.olist_order_items
(
  order_id        VARCHAR(50),
  order_item_id   INT,
  product_id      VARCHAR(50),
  seller_id       VARCHAR(50),
  shipping_limit_date DATETIME2,
  price           DECIMAL(18,2),
  freight_value   DECIMAL(18,2)
);

CREATE TABLE
stg.olist_order_payments (
  order_id          VARCHAR(50),
  payment_sequential INT,
  payment_type      VARCHAR(32),
  payment_installments INT,
  payment_value     DECIMAL(18,2)
);

CREATE TABLE stg.olist_customers (
  customer_id          VARCHAR(50)
PRIMARY KEY,
  customer_unique_id  VARCHAR(50),
  customer_zip_code_prefix INT,
  customer_city
NVARCHAR(100),
  customer_state      NVARCHAR(10)
);

CREATE TABLE stg.olist_products (
  product_id
VARCHAR(50) PRIMARY KEY,
  product_category_name
NVARCHAR(100),
  product_name_length
INT,
  product_description_length
INT,
  product_photos_qty
INT,
  product_weight_g
INT,
  product_length_cm
INT,
  product_height_cm
INT,
  product_width_cm             INT
);

CREATE TABLE stg.olist_sellers (
  seller_id
VARCHAR(50) PRIMARY KEY,
  seller_zip_code_prefix INT,
  seller_city
NVARCHAR(100),
  seller_state
NVARCHAR(10)
);

CREATE TABLE stg.olist_geolocation
(
  geolocation_zip_code_prefix INT,
  geolocation_lat FLOAT,
  geolocation_lng FLOAT,
  geolocation_city NVARCHAR(100),
  geolocation_state NVARCHAR(10)
);

CREATE TABLE
stg.product_category_name_translat
ion (
  product_category_name
NVARCHAR(100),
  product_category_name_english
NVARCHAR(100)
);

CREATE TABLE
stg.olist_order_reviews (
  review_id          VARCHAR(50)
PRIMARY KEY,
  order_id           VARCHAR(50),
  review_score       INT,
  review_comment_title
NVARCHAR(255),
  review_comment_message
NVARCHAR(MAX),
  review_creation_date DATETIME2,
  review_answer_timestamp
DATETIME2
);
```

4.3 Helpful indexes

```
-- sql/helper_indexes.sql
CREATE INDEX IX_orders_purchase_ts
ON
stg.olist_orders(order_purchase_ti
mestamp);
CREATE INDEX IX_items_order ON
stg.olist_order_items(order_id);
CREATE INDEX IX_payments_order ON
stg.olist_order_payments(order_id)
;
CREATE INDEX IX_customers_unique
ON
stg.olist_customers(customer_uniqu
e_id);
CREATE INDEX IX_geo_zip ON
stg.olist_geolocation(geolocation_
zip_code_prefix);
```

---

5) Load CSVs → Staging (Python bulk upsert)

scripts/load_csvs_to_staging.py

```
import os, glob, pandas as pd
from sqlalchemy import
create_engine, text
from dotenv import load_dotenv

load_dotenv()
SERVER=os.getenv('MSSQL_SERVER');
PORT=os.getenv('MSSQL_PORT','1433'
)
DB=os.getenv('MSSQL_DATABASE');
USER=os.getenv('MSSQL_USERNAME');
PWD=os.getenv('MSSQL_PASSWORD')
DRV=os.getenv('MSSQL_DRIVER','ODBC
Driver 18 for SQL Server')

odbc=f"DRIVER={{{DRV}}};SERVER={SE
RVER},{PORT};DATABASE={DB};UID={US
ER};PWD={PWD};Encrypt=yes;TrustSer
verCertificate=yes;".replace('
','+')
engine=create_engine(f"mssql+pyodb
c:///?odbc_connect={odbc}",
fast_executemany=True)

CSV_MAP={
  'olist_orders_dataset.csv':
('stg.olist_orders', {
```

```
'order_purchase_timestamp':'dateti
me64[ns]',

'order_approved_at':'datetime64[ns
]','order_delivered_carrier_date':
'datetime64[ns]',

'order_delivered_customer_date':'d
atetime64[ns]','order_estimated_de
livery_date':'datetime64[ns]'}),
  'olist_order_items_dataset.csv':
('stg.olist_order_items',
{'shipping_limit_date':'datetime64
[ns]'}),

'olist_order_payments_dataset.csv'
: ('stg.olist_order_payments',
{}),
  'olist_customers_dataset.csv':
('stg.olist_customers', {}),
  'olist_products_dataset.csv':
('stg.olist_products', {}),
  'olist_sellers_dataset.csv':
('stg.olist_sellers', {}),
  'olist_geolocation_dataset.csv':
('stg.olist_geolocation', {}),

'olist_order_reviews_dataset.csv':
('stg.olist_order_reviews', {

'review_creation_date':'datetime64
[ns]','review_answer_timestamp':'d
atetime64[ns]'}),

'product_category_name_translation
.csv':
('stg.product_category_name_transl
ation', {})
}

with engine.begin() as con:
    for fname,(table,dtypes) in
CSV_MAP.items():

path=os.path.join('data','raw',fna
me)
        df=pd.read_csv(path,
dtype=str, keep_default_na=False)
        # Cast specific dtypes
        for col,dt in
dtypes.items():
            if col in df.columns:

df[col]=pd.to_datetime(df[col],
errors='coerce')
        # Numeric casts
        for c in
['order_item_id','price','freight_
value','payment_sequential','payme
nt_installments','payment_value',

'product_name_length','product_des
cription_length','product_photos_q
ty',

'product_weight_g','product_length
_cm','product_height_cm','product_
width_cm',

'customer_zip_code_prefix','seller
_zip_code_prefix','review_score',

'geolocation_zip_code_prefix','geo
location_lat','geolocation_lng']:
            if c in df.columns:

df[c]=pd.to_numeric(df[c],
errors='coerce')
        # Load: replace table on
first run for idempotency

df.to_sql(table.split('.')[-1],
con, schema=table.split('.')[0],
if_exists='append', index=False)
        print(f"Loaded {len(df):,}
rows into {table}")
```

Run:

```
python
scripts/load_csvs_to_staging.py
```

---

6) Transform → Warehouse (dbt-sqlserver)

6.1 Configure dbt profile

configs/dbt_profile.yml

```
default:
  target: dev
  outputs:
    dev:
      type: sqlserver
      driver: 'ODBC Driver 18 for
SQL Server'
      server: '{{
env_var("MSSQL_SERVER") }}'
      port: 1433
      user: '{{
env_var("MSSQL_USERNAME") }}'
      password: '{{
env_var("MSSQL_PASSWORD") }}'
      database: '{{
env_var("MSSQL_DATABASE") }}'
      schema: 'wh'
      encrypt: true
      trust_cert: true
```

6.2 Declare sources

dbt_project/models/sources.yml

```
version: 2
sources:
  - name: stg
    schema: stg
    tables:
      - name: olist_orders
      - name: olist_order_items
      - name: olist_order_payments
      - name: olist_customers
      - name: olist_products
      - name: olist_sellers
      - name: olist_geolocation
      - name: olist_order_reviews
      - name:
product_category_name_translation
```

6.3 Staging views (standardize types)

dbt_project/models/staging/stg_orders.sql

```
select
  order_id,
  customer_id,
  order_status,
  cast(order_purchase_timestamp as
datetime2) as order_purchase_ts,

cast(order_estimated_delivery_date
as date) as estimated_delivery_dt,

cast(order_delivered_customer_date
as datetime2) as
delivered_customer_ts
from {{
source('stg','olist_orders') }}
```

dbt_project/models/staging/stg_order_items.sql

```
select
  order_id,
  order_item_id,
  product_id,
  seller_id,
  cast(shipping_limit_date as
datetime2) as shipping_limit_ts,
  cast(price as decimal(18,2)) as
price,
  cast(freight_value as
decimal(18,2)) as freight_value
from {{
source('stg','olist_order_items')
}}
```

dbt_project/models/staging/stg_payments.sql

```
select
  order_id,
  payment_type,
  payment_installments,
  cast(payment_value as
decimal(18,2)) as payment_value
from {{
source('stg','olist_order_payments
') }}
```

dbt_project/models/staging/stg_customers.sql

```
select
  customer_id,
  customer_unique_id,
  customer_zip_code_prefix,
  customer_city,
  customer_state
from {{
source('stg','olist_customers') }}
```

dbt_project/models/staging/stg_products.sql

```
select p.product_id,

coalesce(t.product_category_name_e
nglish, p.product_category_name)
as product_category,
       p.product_weight_g,
p.product_length_cm,
p.product_height_cm,
p.product_width_cm
from {{
source('stg','olist_products') }}
p
left join {{
source('stg','product_category_nam
e_translation') }} t
  on t.product_category_name =
p.product_category_name
```

dbt_project/models/staging/stg_sellers.sql

```
select seller_id,
seller_zip_code_prefix,
seller_city, seller_state
from {{
source('stg','olist_sellers') }}
```

dbt_project/models/staging/stg_reviews.sql

```
select order_id, review_score
from {{
source('stg','olist_order_reviews'
) }}
```

6.4 Warehouse models (star schema)

dbt_project/models/warehouse/dim_date.sql

```
with d as (
  select cast(dateadd(day,
v.number, '2016-01-01') as date)
as d
  from master..spt_values v where
v.type='P' and v.number between 0
and 1826  -- 5 years
)
select cast(format(d,'yyyyMMdd')
as int) as date_key,
       d as date_value,
       year(d) as year,
       datepart(quarter,d) as
quarter,
       month(d) as month,
       day(d) as day,
       datepart(week,d) as
week_of_year
```

dbt_project/models/warehouse/dim_customer.sql

```
select
  c.customer_unique_id as
customer_key,
  any_value(c.customer_id) as
sample_customer_id,
  c.customer_city,
  c.customer_state
from {{ ref('stg_customers') }} c
group by c.customer_unique_id,
c.customer_city, c.customer_state
```

dbt_project/models/warehouse/dim_product.sql

```
select
  p.product_id as product_key,
  p.product_category,
  p.product_weight_g,
  p.product_length_cm,
  p.product_height_cm,
  p.product_width_cm
from {{ ref('stg_products') }} p
```

dbt_project/models/warehouse/dim_seller.sql

```
select
  s.seller_id as seller_key,
  s.seller_city,
  s.seller_state
from {{ ref('stg_sellers') }} s
```

dbt_project/models/warehouse/fact_orders.sql

```
select
  o.order_id,
  dc.customer_key,

cast(format(o.order_purchase_ts,'y
yyyMMdd') as int) as date_key,
  o.order_status,
  o.estimated_delivery_dt,
  o.delivered_customer_ts,
  i.total_items,
  i.items_value,
  i.freight_value,
  p.payments_value,
  r.avg_review_score
from {{ ref('stg_orders') }} o
left join (
  select order_id,
         sum(price) as
items_value,
         sum(freight_value) as
freight_value,
         count(*) as total_items
  from {{ ref('stg_order_items')
}}
  group by order_id
) i on i.order_id = o.order_id
left join (
  select order_id,
sum(payment_value) as
payments_value
  from {{ ref('stg_payments') }}
  group by order_id
) p on p.order_id = o.order_id
left join (
  select order_id,
avg(review_score*1.0) as
avg_review_score
  from {{ ref('stg_reviews') }}
  group by order_id
) r on r.order_id = o.order_id
left join (
  select customer_id,
customer_unique_id from {{
ref('stg_customers') }}
) cu on cu.customer_id =
o.customer_id
left join {{ ref('dim_customer')
}} dc on dc.customer_key =
cu.customer_unique_id
```

dbt_project/models/warehouse/fact_order_items.sql

```
select
  oi.order_id,

cast(format(o.order_purchase_ts,'y
yyyMMdd') as int) as date_key,
  dp.product_key,
  ds.seller_key,
  oi.order_item_id,
  oi.price,
  oi.freight_value
from {{ ref('stg_order_items') }}
oi
left join {{ ref('stg_orders') }}
o on o.order_id = oi.order_id
left join {{ ref('dim_product') }}
dp on dp.product_key =
oi.product_id
left join {{ ref('dim_seller') }}
ds on ds.seller_key = oi.seller_id
```

dbt_project/models/marts/sales_summary.sql

```
select
  dd.year,
  dd.month,
  dp.product_category,
  sum(fo.items_value) as revenue,
  sum(fo.freight_value) as
freight,
  sum(fo.total_items) as items,
  avg(fo.avg_review_score) as
avg_review
from {{ ref('fact_orders') }} fo
join {{ ref('dim_date') }} dd on
dd.date_key = fo.date_key
join {{ ref('dim_product') }} dp
on dp.product_key in (
  select product_key from {{
ref('fact_order_items') }} foi
where foi.order_id = fo.order_id
)
group by dd.year, dd.month,
dp.product_category
```

Run dbt:

```
export DBT_PROFILES_DIR=./configs
cd dbt_project && dbt run && dbt
test
```

---

7) Quick Data Quality Checks (Python)

scripts/run_quality_checks.py

```
from sqlalchemy import
create_engine
import pandas as pd, os
from dotenv import load_dotenv
load_dotenv()
engine=create_engine(

f"mssql+pyodbc:///?odbc_connect=DR
IVER={{ODBC Driver 18 for SQL
Server}};SERVER={os.getenv('MSSQL_
SERVER')},{os.getenv('MSSQL_PORT',
'1433')};DATABASE={os.getenv('MSSQ
L_DATABASE')};UID={os.getenv('MSSQ
L_USERNAME')};PWD={os.getenv('MSSQ
L_PASSWORD')};Encrypt=yes;TrustSer
verCertificate=yes;".replace('
','+')
)

checks={

'stg.olist_orders':['order_id','cu
stomer_id','order_purchase_timesta
mp'],

'stg.olist_order_items':['order_id
','order_item_id','product_id','pr
ice'],

'stg.olist_order_payments':['order
_id','payment_value'],
}

with engine.begin() as con:
  for table, cols in
checks.items():
    df=pd.read_sql(f"select * from
{table}", con)
    missing=[c for c in cols if c
not in df.columns or
df[c].isna().any()]
    if missing: raise
AssertionError(f"{table} failed
not-null check on {missing}")
print("Quality checks passed")
```

Run:

```
python
scripts/run_quality_checks.py
```

---

8) Dashboards

Power BI (recommended)

1. Connect → SQL Server → Database: **OlistDB**.
2. Import: `wh.dim_date`, `wh.dim_customer`, `wh.dim_product`, `wh.dim_seller`, `wh.fact_orders`, `wh.fact_order_items`, and view `wh.sales_summary` (if materialized as view).
3. Relationships:
   - `fact_orders.date_key → dim_date.date_key`
   - `fact_order_items.date_key → dim_date.date_key`
   - `fact_order_items.product_key → dim_product.product_key`
   - `fact_order_items.seller_key → dim_seller.seller_key`

DAX Measures

```
Total Revenue :=
SUMX(wh_fact_orders,
wh_fact_orders[items_value])
Total Freight :=
SUM(wh_fact_orders[freight_value])
Total Orders :=
DISTINCTCOUNT(wh_fact_orders[order
_id])
Total Items  :=
SUM(wh_fact_orders[total_items])
AOV := DIVIDE([Total Revenue],
[Total Orders])
OnTime Delivery % := DIVIDE(
  CALCULATE([Total Orders],
FILTER(wh_fact_orders,
wh_fact_orders[delivered_customer_
ts] <=
wh_fact_orders[estimated_delivery_
dt])),
  [Total Orders]
)
Avg Review :=
AVERAGE(wh_fact_orders[avg_review_
score])
Repeat Customer % :=
VAR t = SUMMARIZE(wh_dim_customer,
wh_dim_customer[customer_key],
"orders",
CALCULATE(DISTINCTCOUNT(wh_fact_or
ders[order_id])))
RETURN DIVIDE(COUNTROWS(FILTER(t,
[orders] > 1)), COUNTROWS(t))
```

Streamlit (optional)

scripts/streamlit_app.py

```
import os, pandas as pd,
plotly.express as px
from sqlalchemy import
create_engine
import streamlit as st

st.set_page_config(page_title="Oli
st Sales Analytics",
layout="wide")
engine = create_engine(

f"mssql+pyodbc:///?odbc_connect=DR
IVER={{ODBC Driver 18 for SQL
Server}};SERVER={os.getenv('MSSQL_
SERVER')},{os.getenv('MSSQL_PORT',
'1433')};DATABASE={os.getenv('MSSQ
L_DATABASE')};UID={os.getenv('MSSQ
L_USERNAME')};PWD={os.getenv('MSSQ
L_PASSWORD')};Encrypt=yes;TrustSer
verCertificate=yes;".replace('
','+')
)

@st.cache_data
def load(sql):
    with engine.begin() as con:
        return pd.read_sql(sql,
con)

df = load("select * from
wh.sales_summary")
st.title("Olist — Sales Summary")
col1, col2 = st.columns(2)
col1.plotly_chart(px.line(df,
x="month", y="revenue",
color="year", title="Revenue by
Month"), use_container_width=True)
col2.plotly_chart(px.bar(df,
x="product_category", y="revenue",
title="Revenue by Category"),
use_container_width=True)
```

Run:

```
streamlit run
scripts/streamlit_app.py
```

---

9) End-to-End Runbook

```
# 1) env & deps
python -m venv .venv && source
.venv/bin/activate
pip install -r requirements.txt

# 2) download Kaggle data
bash scripts/kaggle_download.sh

# 3) create schemas + staging
tables in SQL Server
sqlcmd -S localhost -d OlistDB -U
sa -P YourStrong!Passw0rd -i
sql/create_schemas.sql
sqlcmd -S localhost -d OlistDB -U
sa -P YourStrong!Passw0rd -i
sql/create_staging_tables.sql
sqlcmd -S localhost -d OlistDB -U
sa -P YourStrong!Passw0rd -i
sql/helper_indexes.sql

# 4) load CSVs to staging
python
scripts/load_csvs_to_staging.py

# 5) build warehouse with dbt
export DBT_PROFILES_DIR=./configs
cd dbt_project && dbt run && dbt
test && cd ..

# 6) (optional) quality checks
python
scripts/run_quality_checks.py

# 7) (optional) run Streamlit
dashboard
streamlit run
scripts/streamlit_app.py
```

---

10) Notes

- **Revenue** uses sum of `order_items.price` (payments can be used for cross-checks).
- **On-time delivery** compares actual delivered timestamp vs. estimated delivery date.
- **Repeat customers** defined via `customer_unique_id` (stable across orders).
- Adjust indexes for large deployments; partition by date if needed.
