E-Commerce Recommendation System – Full Source Code (Python + SQL)

Below is the **complete, working project** you asked for: full Python source code, SQL schema, sample dataset CSVs, a Streamlit dashboard, model training + hybrid recommender, and export-to-Excel reporting. You can copy this directly into a GitHub repo.

I've organized everything under a root folder named ``. If you prefer the shorter name shown earlier, just rename the folder before committing.

📁 Final Project Layout

```
ecommerce_recommendation_system/
│
├── README.md
├── requirements.txt
├── config_example.yaml
├── main.py                        # CLI: build-models / recommend / export
│
├── data/                          # sample demo data (replace w/your real data)
│   ├── customers.csv
│   ├── products.csv
│   ├── transactions.csv
│   └── sample_reports.xlsx        # generated after export (ignored in git if you
want)
│
├── database/
│   ├── schema.sql                 # DDL for SQLite (works w/other DBs w/ small
tweaks)
│   ├── create_db.py               # build empty DB from schema
│   ├── insert_data.py             # load CSV -> DB
│   └── db_utils.py                # SQLAlchemy engine helper
│
├── models/
│   ├── __init__.py
│   ├── data_loader.py             # unified data access from DB
│   ├── content_based.py           # TF-IDF text similarity
│   ├── collaborative_filtering.py  # implicit item-based CF
│   ├── hybrid_model.py            # weighted blend (collab + content +
popularity)
│   ├── recommender.py             # high-level API wrapper class
```

```
│   ├── metrics.py               # precision@k etc.
│   └── export_reports.py        # Excel multi-sheet export
│
├── utils/
│   ├── paths.py                 # path helpers
│   └── seed_demo_data.py        # optional synthetic data generator (bigger
demo)
│
├── dashboard/
│   ├── app.py                   # Streamlit dashboard
│   ├── charts.py                # Plotly chart helpers
│   └── session_state.py         # convenience state utilities
│
└── tests/
    ├── conftest.py
    ├── test_data_integrity.py
    └── test_recommender_basic.py
```

🔧Configuration File

` – Copy to config.yaml` and edit.

```yaml
# config_example.yaml

database:
  uri: "sqlite:///database/ecommerce.db"  # change to mysql+pymysql://
user:pw@host/db

recommender:
  top_n: 10
  min_interactions_user: 1
  min_interactions_item: 1
  weights:
    collaborative: 0.6
    content: 0.3
    popularity: 0.1

content:
  text_fields: ["product_name", "category", "subcategory", "brand",
"description"]
  max_features: 5000
```

```
export:
  excel_filename: "data/sample_reports.xlsx"
```

📦 requirements.txt

```
pandas
numpy
sqlalchemy
scikit-learn
streamlit
plotly
pyyaml
openpyxl
pytest
```

🗄️ SQL Schema

``

```
PRAGMA foreign_keys = ON;

DROP TABLE IF EXISTS transactions;
DROP TABLE IF EXISTS customers;
DROP TABLE IF EXISTS products;

CREATE TABLE customers (
    customer_id INTEGER PRIMARY KEY,
    customer_name TEXT NOT NULL,
    segment TEXT,
    location TEXT
);

CREATE TABLE products (
    product_id INTEGER PRIMARY KEY,
    product_name TEXT NOT NULL,
    category TEXT,
    subcategory TEXT,
    brand TEXT,
    price REAL,
    description TEXT
```

```
);

CREATE TABLE transactions (
    transaction_id INTEGER PRIMARY KEY,
    customer_id INTEGER NOT NULL,
    product_id INTEGER NOT NULL,
    quantity INTEGER DEFAULT 1,
    unit_price REAL,
    rating REAL,                  -- optional explicit feedback (nullable)
    purchase_timestamp TEXT,
    FOREIGN KEY(customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY(product_id) REFERENCES products(product_id)
);

CREATE INDEX idx_transactions_customer ON transactions(customer_id);
CREATE INDEX idx_transactions_product ON transactions(product_id);
CREATE INDEX idx_transactions_timestamp ON transactions(purchase_timestamp);
```

🛠️database/db_utils.py

```
# database/db_utils.py
import os
from sqlalchemy import create_engine
from sqlalchemy.engine import Engine


def get_engine(db_uri: str = "sqlite:///database/ecommerce.db", echo: bool =
False) -> Engine:
    """Return a SQLAlchemy engine for the configured database URI.

    If SQLite, ensure directory exists.
    """
    if db_uri.startswith("sqlite"):
        # sqlite:///path/to/db.sqlite
        path = db_uri.split("///")[-1]
        dirpath = os.path.dirname(path)
        if dirpath:
            os.makedirs(dirpath, exist_ok=True)
    engine = create_engine(db_uri, echo=echo, future=True)
    return engine
```

🛠️database/create_db.py

```python
# database/create_db.py
import argparse
from pathlib import Path
from sqlalchemy import text

from db_utils import get_engine


def create_db(schema_path: str, db_uri: str):
    engine = get_engine(db_uri)
    schema_sql = Path(schema_path).read_text(encoding="utf-8")
    with engine.begin() as conn:
        conn.execute(text(schema_sql))
    print(f"Database created at {db_uri}.")


def main():
    parser = argparse.ArgumentParser(description="Create DB from schema.sql")
    parser.add_argument("--schema", default="database/schema.sql")
    parser.add_argument("--db-uri", default="sqlite:///database/ecommerce.db")
    args = parser.parse_args()
    create_db(args.schema, args.db_uri)


if __name__ == "__main__":
    main()
```

🛠️database/insert_data.py

```python
# database/insert_data.py
import argparse
import os
import pandas as pd
from sqlalchemy import text

from db_utils import get_engine


def load_schema(engine, schema_path: str):
    with open(schema_path, "r", encoding="utf-8") as f:
```

```
        schema_sql = f.read()
    with engine.begin() as conn:
        conn.execute(text(schema_sql))


def load_csvs(engine, data_dir: str):
    customers = pd.read_csv(os.path.join(data_dir, "customers.csv"))
    products = pd.read_csv(os.path.join(data_dir, "products.csv"))
    transactions = pd.read_csv(os.path.join(data_dir, "transactions.csv"))

    customers.to_sql("customers", engine, if_exists="append", index=False)
    products.to_sql("products", engine, if_exists="append", index=False)
    transactions.to_sql("transactions", engine, if_exists="append", index=False)


def main():
    parser = argparse.ArgumentParser(description="Load CSV data into DB.")
    parser.add_argument("--db-uri", default="sqlite:///database/ecommerce.db",
help="SQLAlchemy DB URI")
    parser.add_argument("--data-dir", default="data",
help="Directory containing CSV files")
    parser.add_argument("--schema", default="database/schema.sql",
help="Path to schema SQL")
    args = parser.parse_args()

    engine = get_engine(args.db_uri)

    load_schema(engine, args.schema)
    load_csvs(engine, args.data_dir)
    print("Data load complete.")


if __name__ == "__main__":
    main()
```

📥Sample Data CSVs

``

```
customer_id,customer_name,segment,location
1,Arjun Kumar,Regular,Chennai
2,Priya Sharma,Prime,Bengaluru
3,Rahul Iyer,Regular,Hyderabad
```

```
4,Meera Joshi,Prime,Mumbai
5,Deepak Reddy,Regular,Pune
```

```

```
product_id,product_name,category,subcategory,brand,price,description
101,Wireless Mouse,Electronics,Accessories,LogiTech,799,Ergonomic 2.4G wireless mouse
102,Gaming Keyboard,Electronics,Accessories,RedStorm,2499,Mechanical RGB gaming keyboard
103,Noise Cancelling Headphones,Electronics,Audio,SonicBeat,4999,Over-ear ANC headphones
104,USB-C Charger,Electronics,Power,ChargeMax,999,Fast charging USB-C wall adapter
105,Fitness Tracker,Wearables,Fitness,FitPulse,3499,Heart rate and sleep monitoring band
106,Cotton T-Shirt,Fashion,Apparel,UrbanWear,599,Unisex soft cotton tee
107,Sports Shoes,Fashion,Footwear,RunPro,2999,Breathable running shoes
108,Coffee Maker,Home,Kitchen,BrewEase,2199,Drip coffee machine with timer
109,Stainless Steel Bottle,Home,Kitchen,SteelSafe,499,Insulated hot & cold bottle
110,Tablet Stand,Electronics,Accessories,FlexiHold,699,Adjustable desk tablet stand
```

```

```
transaction_id,customer_id,product_id,quantity,unit_price,rating,purchase_timestamp
10001,1,101,1,799,,2025-03-01 10:15:00
10002,1,104,1,999,,2025-03-10 09:05:00
10003,1,109,2,499,,2025-03-15 12:20:00
10004,2,103,1,4999,,2025-03-02 14:30:00
10005,2,105,1,3499,,2025-03-18 08:50:00
10006,2,107,1,2999,,2025-04-05 17:10:00
10007,3,106,3,599,,2025-03-07 19:45:00
10008,3,109,1,499,,2025-04-01 11:00:00
10009,3,101,1,799,,2025-04-20 09:55:00
10010,4,103,1,4999,,2025-03-12 16:05:00
10011,4,108,1,2199,,2025-04-10 10:30:00
10012,4,105,1,3499,,2025-04-22 13:25:00
10013,5,107,1,2999,,2025-03-08 15:00:00
10014,5,106,2,599,,2025-03-19 18:00:00
10015,5,110,1,699,,2025-04-12 09:00:00
10016,1,102,1,2499,,2025-04-25 21:10:00
10017,2,104,1,999,,2025-05-01 07:45:00
10018,3,108,1,2199,,2025-05-03 20:35:00
```

```
10019,4,110,1,699,,2025-05-06 23:15:00
10020,5,101,1,799,,2025-05-09 06:45:00
```

📚models/init.py

```python
# models/__init__.py
from .recommender import RecommenderSystem
```

📚models/data_loader.py

```python
# models/data_loader.py
import pandas as pd
from sqlalchemy import text


class DataLoader:
    """Centralized data access from SQL DB into DataFrames."""
    def __init__(self, engine):
        self.engine = engine

    def customers(self) -> pd.DataFrame:
        return pd.read_sql("SELECT * FROM customers", self.engine)

    def products(self) -> pd.DataFrame:
        return pd.read_sql("SELECT * FROM products", self.engine)

    def transactions(self) -> pd.DataFrame:
        df = pd.read_sql("SELECT * FROM transactions", self.engine,
parse_dates=["purchase_timestamp"])
        return df

    def user_history(self, customer_id: int) -> pd.DataFrame:
        q = text(
            """
            SELECT t.*, p.product_name, p.category, p.subcategory, p.brand,
p.price
            FROM transactions t
            JOIN products p ON t.product_id = p.product_id
            WHERE t.customer_id = :cid
            ORDER BY t.purchase_timestamp DESC
```

```
            """
        )
        return pd.read_sql(q, self.engine, params={"cid": customer_id},
parse_dates=["purchase_timestamp"])
```

📚models/content_based.py

```
# models/content_based.py
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity


class ContentBasedModel:
    def __init__(self, text_fields, max_features=5000):
        self.text_fields = text_fields
        self.max_features = max_features
        self.vectorizer = None
        self.matrix = None  # TF-IDF sparse matrix
        self.product_ids = None

    def _combine_text(self, products: pd.DataFrame) -> pd.Series:
        texts = []
        for _, row in products.iterrows():
            parts = [str(row.get(col, "")) for col in self.text_fields]
            texts.append(" ".join(parts))
        return pd.Series(texts, index=products.index)

    def fit(self, products: pd.DataFrame):
        combo = self._combine_text(products)
        self.vectorizer = TfidfVectorizer(max_features=self.max_features,
stop_words="english")
        self.matrix = self.vectorizer.fit_transform(combo)
        self.product_ids = products["product_id"].tolist()
        return self

    def recommend_similar(self, product_id: int, top_n=10, exclude_self=True):
        if self.matrix is None:
            raise ValueError("Model not fit.")
        try:
            idx = self.product_ids.index(product_id)
        except ValueError:
            return []
```

```
        sims = cosine_similarity(self.matrix[idx], self.matrix).flatten()
        order = np.argsort(-sims)
        recs = []
        for i in order:
            pid = self.product_ids[i]
            if exclude_self and pid == product_id:
                continue
            recs.append((pid, float(sims[i])))
            if len(recs) >= top_n:
                break
        return recs

    def similarity_vector(self, product_ids_list):
        idxs = [self.product_ids.index(pid) for pid in product_ids_list if pid
in self.product_ids]
        if not idxs:
            return np.zeros(self.matrix.shape[0])
        sub = self.matrix[idxs]
        sims = cosine_similarity(sub, self.matrix)  # (#purchased x #all)
        mean_sims = np.asarray(sims).mean(axis=0)
        return mean_sims
```

📚models/collaborative_filtering.py

```
# models/collaborative_filtering.py
import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.metrics.pairwise import cosine_similarity


class CollaborativeFiltering:
    """Implicit item-based collaborative filtering using purchase counts
(quantity) or ratings."""
    def __init__(self, min_interactions_user=1, min_interactions_item=1):
        self.min_interactions_user = min_interactions_user
        self.min_interactions_item = min_interactions_item
        self.user_index = {}
        self.item_index = {}
        self.index_user = {}
        self.index_item = {}
        self.matrix = None  # users x items CSR
        self.item_sims = None
```

```
    def _filter(self, interactions: pd.DataFrame):
        u_counts = interactions.groupby("customer_id").size()
        keep_users = u_counts[u_counts >= self.min_interactions_user].index
        i_counts = interactions.groupby("product_id").size()
        keep_items = i_counts[i_counts >= self.min_interactions_item].index
        return interactions[(interactions.customer_id.isin(keep_users)) &
(interactions.product_id.isin(keep_items))]

    def fit(self, interactions: pd.DataFrame):
        # expected cols: customer_id, product_id, quantity OR rating
        df = interactions.copy()
        df = self._filter(df)

        # signal: rating if available else quantity else 1
        if "rating" in df and df["rating"].notna().any():
            signal = df["rating"].fillna(0)
        elif "quantity" in df:
            signal = df["quantity"].fillna(1)
        else:
            signal = 1.0

        users = df.customer_id.unique().tolist()
        items = df.product_id.unique().tolist()
        self.user_index = {u: i for i, u in enumerate(users)}
        self.item_index = {p: i for i, p in enumerate(items)}
        self.index_user = {i: u for u, i in self.user_index.items()}
        self.index_item = {i: p for p, i in self.item_index.items()}

        rows = df.customer_id.map(self.user_index)
        cols = df.product_id.map(self.item_index)
        data = pd.Series(signal).astype(float)
        self.matrix = csr_matrix((data, (rows, cols)), shape=(len(users),
len(items)))

        # cosine similarity item-item
        self.item_sims = cosine_similarity(self.matrix.T)
        return self

    def recommend_for_user(self, customer_id: int, top_n=10,
exclude_items=None):
        if self.matrix is None:
            raise ValueError("Model not fit.")
        if customer_id not in self.user_index:
            return []
        uidx = self.user_index[customer_id]
        user_vector = self.matrix[uidx]  # 1 x items
        scores = user_vector.dot(self.item_sims).A1
```

```
        # exclude purchased
        if exclude_items:
            for pid in exclude_items:
                if pid in self.item_index:
                    scores[self.item_index[pid]] = -np.inf

        order = np.argsort(-scores)
        recs = []
        for i in order:
            if len(recs) >= top_n:
                break
            if scores[i] == -np.inf:
                continue
            pid = self.index_item[i]
            recs.append((pid, float(scores[i])))
        return recs
```

📚models/hybrid_model.py

```
# models/hybrid_model.py
import numpy as np
import pandas as pd


class HybridRecommender:
    """Blend collaborative, content, and popularity scores."""
    def __init__(self, collab_model, content_model, popularity_series,
weights=None):
        if weights is None:
            weights = {"collaborative": 0.6, "content": 0.3, "popularity": 0.1}
        self.w = weights
        self.collab = collab_model
        self.content = content_model
        self.popularity = popularity_series  # pd.Series indexed by product_id

    def recommend(self, customer_id: int, purchased_ids, top_n=10):
        # Collaborative scores
        collab_scores = {}
        if self.collab is not None:
            for pid, score in self.collab.recommend_for_user(customer_id,
top_n=100, exclude_items=purchased_ids):
                collab_scores[pid] = score

        # Content scores
```

```
        content_scores = {}
        if self.content is not None and purchased_ids:
            sims = self.content.similarity_vector(purchased_ids)
            for pid, score in zip(self.content.product_ids, sims):
                if pid in purchased_ids:
                    continue
                content_scores[pid] = float(score)

        # Popularity normalized 0..1
        pop = self.popularity.copy()
        if len(pop) > 0:
            pop = (pop - pop.min()) / (pop.max() - pop.min() + 1e-9)
        pop_scores = pop.to_dict()

        # Combine
        all_ids = set(collab_scores) | set(content_scores) | set(pop_scores)
        combined = []
        for pid in all_ids:
            c = collab_scores.get(pid, 0.0)
            t = content_scores.get(pid, 0.0)
            p = pop_scores.get(pid, 0.0)
            score = self.w["collaborative"] * c + self.w["content"] * t +
self.w["popularity"] * p
            combined.append((pid, float(score)))
        combined.sort(key=lambda x: x[1], reverse=True)
        return combined[:top_n]
```

📚models/metrics.py

```
# models/metrics.py
import numpy as np


def precision_at_k(recommended, actual, k=10):
    recs = [pid for pid, _ in recommended[:k]]
    actual_set = set(actual)
    if not recs:
        return 0.0
    hits = sum(1 for r in recs if r in actual_set)
    return hits / min(k, len(recs))


def recall_at_k(recommended, actual, k=10):
    recs = [pid for pid, _ in recommended[:k]]
```

```
    actual_set = set(actual)
    if not actual_set:
        return 0.0
    hits = sum(1 for r in recs if r in actual_set)
    return hits / len(actual_set)
```

📚models/export_reports.py

```
# models/export_reports.py
import pandas as pd


def export_excel(customers_df, products_df, transactions_df,
recommendations_dict, outpath):
    """Write multi-sheet Excel workbook.

    recommendations_dict: {customer_id:
DataFrame(columns=[rank,product_id,...])}
    """
    with pd.ExcelWriter(outpath, engine="openpyxl") as writer:
        customers_df.to_excel(writer, sheet_name="customers", index=False)
        products_df.to_excel(writer, sheet_name="products", index=False)
        transactions_df.to_excel(writer, sheet_name="transactions", index=False)
        for cid, df in recommendations_dict.items():
            sheet = f"recs_{cid}"
            if len(sheet) > 31:
                sheet = f"c{cid}"
            df.to_excel(writer, sheet_name=sheet, index=False)
    return outpath
```

📚models/recommender.py

```
# models/recommender.py
import pandas as pd

from .data_loader import DataLoader
from .content_based import ContentBasedModel
from .collaborative_filtering import CollaborativeFiltering
from .hybrid_model import HybridRecommender
from .export_reports import export_excel
```

```
class RecommenderSystem:
    def __init__(self, engine=None, config=None):
        self.engine = engine
        self.config = config or {}
        self.data = DataLoader(engine)
        self.content_model = None
        self.collab_model = None
        self.hybrid_model = None
        self.products_df = None
        self.transactions_df = None

    # ---------- Data ----------
    def load_data(self):
        self.products_df = self.data.products()
        self.transactions_df = self.data.transactions()
        return self

    # ---------- Train ----------
    def build_models(self):
        if self.products_df is None or self.transactions_df is None:
            self.load_data()
        # content
        text_fields = self.config.get("content", {}).get("text_fields",
["product_name","category","subcategory","brand","description"])
        max_features = self.config.get("content", {}).get("max_features", 5000)
        self.content_model = ContentBasedModel(text_fields=text_fields,
max_features=max_features)
        self.content_model.fit(self.products_df)
        # collaborative
        min_u = self.config.get("recommender", {}).get("min_interactions_user",
1)
        min_i = self.config.get("recommender", {}).get("min_interactions_item",
1)
        self.collab_model = CollaborativeFiltering(min_u, min_i)
        interactions =
self.transactions_df[["customer_id","product_id","quantity","rating"]].copy()
        self.collab_model.fit(interactions)
        # popularity
        pop = self.transactions_df.groupby("product_id")
["quantity"].sum().sort_values(ascending=False)
        # hybrid
        weights = self.config.get("recommender", {}).get("weights", None)
        self.hybrid_model = HybridRecommender(self.collab_model,
self.content_model, pop, weights)
        return self
```

```
    # ---------- Recommend ----------
    def get_purchase_history(self, customer_id):
        return self.data.user_history(customer_id)

    def recommend_products(self, customer_id, top_n=None):
        if self.hybrid_model is None:
            self.build_models()
        if top_n is None:
            top_n = self.config.get("recommender", {}).get("top_n", 10)
        hist = self.get_purchase_history(customer_id)
        purchased = hist["product_id"].tolist()
        recs = self.hybrid_model.recommend(customer_id, purchased, top_n=top_n)
        # decorate
        out_rows = []
        for rank, (pid, score) in enumerate(recs, start=1):
            prod = self.products_df[self.products_df.product_id == pid].iloc[0]
            out_rows.append({
                "rank": rank,
                "product_id": pid,
                "score": score,
                "product_name": prod.product_name,
                "category": prod.category,
                "subcategory": prod.subcategory,
                "brand": prod.brand,
                "price": prod.price,
            })
        return pd.DataFrame(out_rows)

    # ---------- Export ----------
    def export_all(self, outpath="data/sample_reports.xlsx", top_n=None):
        customers_df = self.data.customers()
        recs_dict = {}
        for cid in customers_df.customer_id.tolist():
            recs_dict[cid] = self.recommend_products(cid, top_n=top_n)
        export_excel(customers_df, self.products_df, self.transactions_df,
recs_dict, outpath)
        return outpath
```

🧹 utils/paths.py

```
# utils/paths.py
from pathlib import Path

ROOT = Path(__file__).resolve().parents[1]
```

```
DATA_DIR = ROOT / "data"
DB_PATH = ROOT / "database" / "ecommerce.db"
CONFIG_DEFAULT = ROOT / "config_example.yaml"
```

🧪 utils/seed_demo_data.py (optional)

```
# utils/seed_demo_data.py
"""Generate a larger synthetic dataset if you want to stress-test the
recommender."""
import numpy as np
import pandas as pd
from pathlib import Path

rng = np.random.default_rng(42)


def generate_customers(n=100):
    return pd.DataFrame({
        "customer_id": range(1, n+1),
        "customer_name": [f"User_{i}" for i in range(1, n+1)],
        "segment": rng.choice(["Regular","Prime"], size=n, p=[0.7,0.3]),
        "location":
rng.choice(["Chennai","Bengaluru","Mumbai","Hyderabad","Delhi"], size=n)
    })


def generate_products(n=200):
    cats = ["Electronics","Fashion","Home","Grocery","Sports"]
    subcats = {
        "Electronics":["Accessories","Audio","Power","Mobile"],
        "Fashion":["Apparel","Footwear","Accessories"],
        "Home":["Kitchen","Decor","Cleaning"],
        "Grocery":["Snacks","Beverages","Staples"],
        "Sports":["Fitness","Outdoor","Gear"],
    }
    rows = []
    for pid in range(1, n+1):
        cat = rng.choice(cats)
        sub = rng.choice(subcats[cat])
        price = float(np.round(rng.uniform(100, 5000), 2))
        rows.append((pid, f"Product_{pid}", cat, sub, f"Brand_{rng.integers(1,
50)}", price, f"Synthetic {cat} {sub} item."))
    return pd.DataFrame(rows,
columns=["product_id","product_name","category","subcategory","brand","price","description"])
```

```
def generate_transactions(num_users, num_products, n=5000):
    rows = []
    tid = 1
    for _ in range(n):
        uid = rng.integers(1, num_users+1)
        pid = rng.integers(1, num_products+1)
        qty = int(rng.choice([1,1,1,2,3]))
        price = float(np.round(rng.uniform(100, 5000), 2))
        ts = pd.Timestamp("2025-01-01") + pd.to_timedelta(int(rng.integers(0,
180)), unit="D")
        rows.append((tid, uid, pid, qty, price, np.nan, ts))
        tid += 1
    return pd.DataFrame(rows,
columns=["transaction_id","customer_id","product_id","quantity","unit_price","rating","purchase_t


def main(outdir="data", n_users=100, n_products=200, n_tx=5000):
    out = Path(outdir)
    out.mkdir(parents=True, exist_ok=True)
    cdf = generate_customers(n_users)
    pdf = generate_products(n_products)
    tdf = generate_transactions(n_users, n_products, n_tx)
    cdf.to_csv(out/"customers.csv", index=False)
    pdf.to_csv(out/"products.csv", index=False)
    tdf.to_csv(out/"transactions.csv", index=False)
    print(f"Demo data written to {out}")


if __name__ == "__main__":
    main()
```

📊 dashboard/charts.py

```
# dashboard/charts.py
import pandas as pd
import plotly.express as px


def category_pie(history_df: pd.DataFrame):
    if history_df.empty:
        return None
    counts = history_df["category"].value_counts().reset_index()
```

```
    counts.columns = ["category","count"]
    fig = px.pie(counts, names="category", values="count", title="Purchase by
Category")
    return fig


def top_products_bar(transactions_df: pd.DataFrame, products_df: pd.DataFrame,
top_n=10):
    counts = transactions_df.groupby("product_id")
["quantity"].sum().reset_index()
    counts = counts.merge(products_df[["product_id","product_name"]],
on="product_id", how="left")
    counts = counts.sort_values("quantity", ascending=False).head(top_n)
    fig = px.bar(counts, x="product_name", y="quantity", title=f"Top {top_n}
Products", text="quantity")
    fig.update_layout(xaxis_tickangle=-45)
    return fig
```

📊dashboard/session_state.py

```
# dashboard/session_state.py
import streamlit as st


def get_state(key, default=None):
    if key not in st.session_state:
        st.session_state[key] = default
    return st.session_state[key]


def set_state(key, value):
    st.session_state[key] = value
```

📊dashboard/app.py

```
# dashboard/app.py
import streamlit as st
import pandas as pd
import yaml
from pathlib import Path
```

```
from models import RecommenderSystem
from database.db_utils import get_engine
from .charts import category_pie, top_products_bar

st.set_page_config(page_title="E-Commerce Recommender", layout="wide")

# ----- Load Config -----
CONFIG_PATH = Path("config.yaml") if Path("config.yaml").exists() else
Path("config_example.yaml")
with open(CONFIG_PATH, "r", encoding="utf-8") as f:
    config = yaml.safe_load(f)

# ----- DB & Recommender -----
engine = get_engine(config["database"]["uri"])
recommender = RecommenderSystem(engine=engine, config=config)
recommender.load_data().build_models()

customers_df = recommender.data.customers()
products_df = recommender.products_df
transactions_df = recommender.transactions_df

st.title("E-Commerce Recommendation System Dashboard")

# Sidebar filters
st.sidebar.header("Select Customer")
selected_customer = st.sidebar.selectbox(
    "Customer", options=customers_df.customer_id.tolist(), format_func=lambda
cid: customers_df.loc[customers_df.customer_id==cid, "customer_name"].values[0]
)

st.sidebar.markdown("---")
export_btn = st.sidebar.button("Export Excel Report")

# Main panels
col1, col2 = st.columns([1,1])

with col1:
    st.subheader("Purchase History")
    hist = recommender.get_purchase_history(selected_customer)
    st.dataframe(hist)

with col2:
    st.subheader("Recommended Products")
    recs = recommender.recommend_products(selected_customer)
    st.dataframe(recs)

st.markdown("---")
```

```
# Charts
c1, c2 = st.columns([1,1])
with c1:
    st.subheader("Category Mix")
    fig = category_pie(hist)
    if fig:
        st.plotly_chart(fig, use_container_width=True)
    else:
        st.info("No purchase history.")

with c2:
    st.subheader("Top Products Overall")
    fig2 = top_products_bar(transactions_df, products_df, top_n=10)
    st.plotly_chart(fig2, use_container_width=True)

# Export
if export_btn:
    outpath = recommender.export_all(outpath=config["export"]["excel_filename"],
top_n=config["recommender"]["top_n"])
    st.success(f"Report exported: {outpath}")
    st.download_button(
        label="Download Excel",
        data=open(outpath, "rb").read(),
        file_name=outpath.split("/")[-1],
        mime="application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet",
    )
```

🧮 main.py (CLI)

```
# main.py
import argparse
import yaml
from pathlib import Path

from database.db_utils import get_engine
from models import RecommenderSystem


def load_config(path):
    with open(path, "r", encoding="utf-8") as f:
        return yaml.safe_load(f)
```

```
def cmd_build_models(args):
    config = load_config(args.config)
    engine = get_engine(config["database"]["uri"])
    recsys = RecommenderSystem(engine=engine, config=config)
    recsys.load_data().build_models()
    print("Models built.")


def cmd_recommend(args):
    config = load_config(args.config)
    engine = get_engine(config["database"]["uri"])
    recsys = RecommenderSystem(engine=engine, config=config)
    recsys.load_data().build_models()
    df = recsys.recommend_products(args.customer_id, top_n=args.top_n)
    print(df)


def cmd_export(args):
    config = load_config(args.config)
    engine = get_engine(config["database"]["uri"])
    recsys = RecommenderSystem(engine=engine, config=config)
    recsys.load_data().build_models()
    outpath = recsys.export_all(outpath=args.outpath, top_n=args.top_n)
    print(f"Exported to {outpath}")


def main():
    parser = argparse.ArgumentParser(description="E-Commerce Recommender CLI")
    parser.add_argument("command", choices=["build-
models","recommend","export"], help="Action")
    parser.add_argument("--config", default="config_example.yaml", help="Config
YAML path")
    parser.add_argument("--customer-id", dest="customer_id", type=int,
default=1, help="Customer ID for recommend")
    parser.add_argument("--top-n", dest="top_n", type=int, default=10, help="#
of recommendations")
    parser.add_argument("--outpath", default="data/sample_reports.xlsx",
help="Excel output path")

    args = parser.parse_args()

    if args.command == "build-models":
        cmd_build_models(args)
    elif args.command == "recommend":
        cmd_recommend(args)
    elif args.command == "export":
        cmd_export(args)
```

```
if __name__ == "__main__":
    main()
```

🔗tests/conftest.py

```
# tests/conftest.py
import pytest
import yaml
from pathlib import Path

from database.db_utils import get_engine
from models import RecommenderSystem


@pytest.fixture(scope="session")
def config():
    path = Path("config_example.yaml")
    with open(path, "r", encoding="utf-8") as f:
        return yaml.safe_load(f)


@pytest.fixture(scope="session")
def engine(config):
    return get_engine(config["database"]["uri"])


@pytest.fixture(scope="session")
def recsys(engine, config):
    rs = RecommenderSystem(engine=engine, config=config)
    rs.load_data().build_models()
    return rs
```

🔗tests/test_data_integrity.py

```
# tests/test_data_integrity.py

def test_customer_count(recsys):
    df = recsys.data.customers()
```

```
    assert len(df) > 0


def test_product_count(recsys):
    assert len(recsys.products_df) > 0
```

🔗tests/test_recommender_basic.py

```
# tests/test_recommender_basic.py

def test_recommend_for_user(recsys):
    df = recsys.recommend_products(1, top_n=5)
    assert df is not None
    assert len(df) <= 5
```

📘README.md (Final Copy)

Below is a ready-to-use README. Replace `<your-username>` and add screenshots.

````
# E-Commerce Recommendation System

A full-featured hybrid recommendation engine built with **Python, SQL, Pandas,
Scikit-learn, and Streamlit**. Generate personalized product suggestions from
customer interactions and purchase history. Includes dashboard, analytics, and
Excel export.

---

## 📌 Project Highlights
- Hybrid model: Collaborative + Content + Popularity.
- SQL-backed data pipeline (customers, products, transactions).
- Interactive Streamlit dashboard for recommendations & analytics.
- Automated Excel export of user-level recommendations and raw data.

---

## 🗜️ Quick Start
```bash
git clone https://github.com/<your-username>/ecommerce_recommendation_system.git
cd ecommerce_recommendation_system
````
````

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

Load Demo Data & Build DB

```
python database/insert_data.py --db-uri sqlite:///database/ecommerce.db --data-dir data
```

Train / Build Models

```
python main.py build-models --config config_example.yaml
```

Launch Dashboard

```
streamlit run dashboard/app.py
```

Then open http://localhost:8501.

Export Excel Reports

```
python main.py export --config config_example.yaml --outpath data/reports.xlsx
```

🗄️ Data Format

customers.csv: customer_id, customer_name, segment, location\ **products.csv**: product_id, product_name, category, subcategory, brand, price, description\ **transactions.csv**: transaction_id, customer_id, product_id, quantity, unit_price, rating, purchase_timestamp

---

🕐 Use Your Own Data

1. Replace the CSVs in `data/` with your real data (keep same headers).
2. Rerun `insert_data.py`.
3. Rebuild models.

📏 Screenshots

Add these to `assets/` and reference below:

```
![Dashboard](assets/dashboard.png)
![Recommendations](assets/recommendations.png)
```

📜 License

MIT – see LICENSE.

Author

Your Name\ GitHub

```
---

# ✅ Next Step
**Ready for packaging.** Say: **`Create ZIP now`** (SQLite default), and I'll
generate all files in a downloadable archive you can push to GitHub.

If you want MySQL/PostgreSQL support out of the box, tell me and I'll prepare
the DB URI + loader tweaks.
```