In [56]:
```
import numpy as np
import pandas as pd

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics.pairwise import cosine_similarity

import tensorflow as tf
from tensorflow.keras import layers, Model

np.random.seed(42)
tf.random.set_seed(42)

df = pd.read_csv("./car-details-v3.csv")

df = df[
    [
        "name",
        "year",
        "selling_price",
        "km_driven",
        "fuel",
        "seller_type",
        "transmission",
        "owner",
        "mileage",
        "engine",
        "max_power",
        "torque",
        "seats",
    ]
]

# Parsiranje numerickih vrednosti iz string kolona
df["mileage"] = df["mileage"].astype(str).str.extract(r"(\d+\.?\d*)")[0].
df["engine"] = df["engine"].astype(str).str.extract(r"(\d+\.?\d*)")[0].as
df["max_power"] = df["max_power"].astype(str).str.extract(r"(\d+\.?\d*)")
df["torque"] = df["torque"].astype(str).str.extract(r"(\d+\.?\d*)")[0].as
df["seats"] = df["seats"].astype(float)

df["body_coupe"] = df["name"].str.contains(
    "Coupe|Sports|GT|Roadster|Convertible|Cabrio|TT|Z4|S2000|Mustang",
    case=False
).astype(int)

df["body_sedan"] = df["name"].str.contains(
    "Sedan|Dzire|City|Verna|Civic|Corolla|Passat|Octavia|Jetta|C-Class|S-
    case=False
).astype(int)

df["body_suv"] = df["name"].str.contains(
    "Scorpio|Bolero|Fortuner|Safari|Innova|Jeep|XUV|Endeavour|Creta|Harri
    case=False
).astype(int)

df["is_premium_brand"] = df["name"].str.contains(
    "Mercedes|BMW|Audi|Lexus|Jaguar|Volvo|Porsche|Land Rover",
    case=False
).astype(int)
```

```
df.dropna(inplace=True)
```

In [57]:
```
# Normalizacija i kategorije
NUM_COLS = [
    "year",
    "selling_price",
    "km_driven",
    "mileage",
    "engine",
    "max_power",
    "torque",
    "seats",
]

CAT_COLS = ["fuel", "seller_type", "transmission", "owner"]

scaler = MinMaxScaler()
df[NUM_COLS] = scaler.fit_transform(df[NUM_COLS])

for col in CAT_COLS:
    df[col] = df[col].astype("category")

fuel_cat = df["fuel"].cat.categories
seller_cat = df["seller_type"].cat.categories
trans_cat = df["transmission"].cat.categories
owner_cat = df["owner"].cat.categories

df["fuel"] = df["fuel"].cat.codes
df["seller_type"] = df["seller_type"].cat.codes
df["transmission"] = df["transmission"].cat.codes
df["owner"] = df["owner"].cat.codes

df["is_suv"] = df["name"].str.contains(
    "Scorpio|Bolero|Fortuner|Safari|Sumo|Innova|Jeep|4X4|4WD|Endeavour",
    case=False,
).astype(int)

df["is_sport_model"] = df["name"].str.contains(
    "GTI|GT TSI|TSI|TFSI|vRS|RS\\b|iVTEC|VTEC|Type R|Sports|1.6S|Abarth|T
    "Cooper S|ST Line|AMG|M\\b|M3|M4|M5",
    case=False,
).astype(int)

BIN_COLS = ["is_suv", "is_sport_model"]

num_fuel = len(fuel_cat)
num_seller = len(seller_cat)
num_trans = len(trans_cat)
num_owner = len(owner_cat)

print("fuel:", list(fuel_cat))
print("seller_type:", list(seller_cat))
print("transmission:", list(trans_cat))
print("owner:", list(owner_cat))


def get_code(categories, name, default=0):
    return int(np.where(categories == name)[0][0]) if name in categories
```

```
petrol_code = get_code(fuel_cat, "Petrol", default=0)
diesel_code = get_code(fuel_cat, "Diesel", default=0)

individual_code = 0
dealer_code = 0
for i, c in enumerate(seller_cat):
    if "Individual" in c:
        individual_code = i
    if "Dealer" in c:
        dealer_code = i

manual_code = get_code(trans_cat, "Manual", default=0)
auto_code = get_code(trans_cat, "Automatic", default=manual_code)

first_owner_code = 0
for i, c in enumerate(owner_cat):
    if "First Owner" in c:
        first_owner_code = i
        break

df_items = df[NUM_COLS + CAT_COLS + BIN_COLS].copy()
```

```
fuel: ['CNG', 'Diesel', 'LPG', 'Petrol']
seller_type: ['Dealer', 'Individual', 'Trustmark Dealer']
transmission: ['Automatic', 'Manual']
owner: ['First Owner', 'Fourth & Above Owner', 'Second Owner', 'Test Drive
Car', 'Third Owner']
```

In [67]:

```
def generate_segment_users(n_per_segment=40):
    """
    Generise sinteticke korisnike sa poljem 'segment' (1..6).
    Ti korisnici se koriste za generisanje trening parova.
    """
    users = []

    for _ in range(n_per_segment):
        # 1) Budget Buyer
        users.append(
            {
                "year": np.random.uniform(0.25, 0.55),
                "selling_price": np.random.uniform(0.1, 0.35),
                "km_driven": np.random.uniform(0.3, 0.8),
                "mileage": np.random.uniform(0.6, 1.0),
                "engine": np.random.uniform(0.2, 0.5),
                "max_power": np.random.uniform(0.2, 0.5),
                "torque": np.random.uniform(0.2, 0.5),
                "seats": np.random.uniform(0.3, 0.7),
                "fuel": petrol_code,
                "seller_type": individual_code,
                "transmission": manual_code,
                "owner": first_owner_code,
                "segment": 1,
            }
        )

        # 2) Diesel Commuter
        users.append(
            {
                "year": np.random.uniform(0.45, 0.8),
```

```
            "selling_price": np.random.uniform(0.3, 0.6),
            "km_driven": np.random.uniform(0.2, 0.6),
            "mileage": np.random.uniform(0.5, 0.9),
            "engine": np.random.uniform(0.4, 0.7),
            "max_power": np.random.uniform(0.3, 0.6),
            "torque": np.random.uniform(0.4, 0.8),
            "seats": np.random.uniform(0.4, 0.7),
            "fuel": diesel_code,
            "seller_type": dealer_code,
            "transmission": manual_code,
            "owner": first_owner_code,
            "segment": 2,
        }
    )

    # 3) Family Buyer
    users.append(
        {
            "year": np.random.uniform(0.6, 0.9),
            "selling_price": np.random.uniform(0.4, 0.7),
            "km_driven": np.random.uniform(0.1, 0.4),
            "mileage": np.random.uniform(0.4, 0.8),
            "engine": np.random.uniform(0.4, 0.7),
            "max_power": np.random.uniform(0.4, 0.7),
            "torque": np.random.uniform(0.4, 0.7),
            "seats": np.random.uniform(0.6, 1.0),
            "fuel": np.random.choice([petrol_code, diesel_code]),
            "seller_type": dealer_code,
            "transmission": auto_code,
            "owner": first_owner_code,
            "segment": 3,
        }
    )

    # 4) Sport Enthusiast
    users.append(
        {
            "year": np.random.uniform(0.5, 0.9),
            "selling_price": np.random.uniform(0.6, 0.9),
            "km_driven": np.random.uniform(0.05, 0.3),
            "mileage": np.random.uniform(0.3, 0.7),
            "engine": np.random.uniform(0.6, 0.95),
            "max_power": np.random.uniform(0.7, 1.0),
            "torque": np.random.uniform(0.6, 1.0),
            "seats": np.random.uniform(0.1, 0.3),
            "fuel": petrol_code,
            "seller_type": np.random.choice([individual_code, dealer_
            "transmission": manual_code,
            "owner": first_owner_code,
            "segment": 4,
        }
    )

    # 5) Off-road Utility
    users.append(
        {
            "year": np.random.uniform(0.3, 0.8),
            "selling_price": np.random.uniform(0.4, 0.8),
            "km_driven": np.random.uniform(0.3, 0.8),
            "mileage": np.random.uniform(0.3, 0.7),
```

```
                "engine": np.random.uniform(0.6, 1.0),
                "max_power": np.random.uniform(0.6, 0.9),
                "torque": np.random.uniform(0.7, 1.0),
                "seats": np.random.uniform(0.6, 1.0),
                "fuel": diesel_code,
                "seller_type": np.random.choice([individual_code, dealer_
                "transmission": manual_code,
                "owner": first_owner_code,
                "segment": 5,
            }
        )

        # 6) Premium / Luxury Urban
        users.append(
            {
                "year": np.random.uniform(0.8, 1.0),
                "selling_price": np.random.uniform(0.7, 1.0),
                "km_driven": np.random.uniform(0.0, 0.3),
                "mileage": np.random.uniform(0.3, 0.7),
                "engine": np.random.uniform(0.6, 0.9),
                "max_power": np.random.uniform(0.6, 0.9),
                "torque": np.random.uniform(0.5, 0.85),
                "seats": np.random.uniform(0.5, 0.9),
                "fuel": petrol_code,
                "seller_type": dealer_code,
                "transmission": auto_code,
                "owner": first_owner_code,
                "segment": 6,
            }
        )

    return pd.DataFrame(users)


users_df = generate_segment_users(40)
print("Users shape:", users_df.shape)
```

```
Users shape: (240, 13)
```

In [68]:

```
def score_items_for_segment(user_row, cars_df: pd.DataFrame):
    """
    Segment-based scoring funkcija
    """

    seg = int(user_row["segment"])

    cars = cars_df
    n = len(cars)

    user_num = user_row[NUM_COLS].values.astype("float32")
    car_num = cars[NUM_COLS].values.astype("float32")
    diff = np.abs(car_num - user_num)
    base_sim = 1.0 - diff
    base_sim = np.clip(base_sim, 0.0, 1.0)
    base_score = base_sim.sum(axis=1)

    fuel = cars["fuel"].values
    seller = cars["seller_type"].values
    trans = cars["transmission"].values
    owner = cars["owner"].values
```

```
    is_suv = cars["is_suv"].values
    is_sport = cars["is_sport_model"].values

    year = cars["year"].values
    price = cars["selling_price"].values
    km = cars["km_driven"].values
    mileage = cars["mileage"].values
    engine = cars["engine"].values
    power = cars["max_power"].values
    torque = cars["torque"].values
    seats = cars["seats"].values

    score = np.zeros(n, dtype="float32")

    # BUDGET
    if seg == 1:
        score += 2.5 * (1 - np.abs(price - user_row["selling_price"]))
        score += 1.0 * (1 - np.abs(year - user_row["year"]))
        score += 1.5 * (1 - np.abs(km - user_row["km_driven"]))
        score += 2.0 * (1 - np.abs(mileage - user_row["mileage"]))
        score += 1.0 * (fuel == user_row["fuel"])
        score += 1.0 * (seller == user_row["seller_type"])
        score += 0.5 * (owner == user_row["owner"])
        score += base_score

    # DIESEL COMMUTER
    elif seg == 2:
        score += 3.0 * (fuel == diesel_code)
        score += 2.0 * (1 - np.abs(mileage - user_row["mileage"]))
        score += 1.5 * (1 - np.abs(km - user_row["km_driven"]))
        score += 1.5 * (1 - np.abs(price - user_row["selling_price"]))
        score -= 1.5 * is_suv
        score += 1.0 * (trans == manual_code)
        score += base_score

    # FAMILY BUYER
    elif seg == 3:
        score += 2.0 * (seats >= 0.6).astype("float32")   # 5+ sedišta
        score += 1.5 * (1 - np.abs(km - user_row["km_driven"]))
        score += 1.5 * (1 - np.abs(year - user_row["year"]))
        score += 1.0 * (trans == auto_code)
        score += 1.0 * (seller == dealer_code)
        score += 0.5 * is_suv  # SUV-ovi blagi plus
        score += base_score

    # SPORT ENTHUSIAST
    elif seg == 4:
        score += 4.0 * power
        score += 3.0 * engine
        score += 2.0 * torque
        score += 2.0 * is_sport
        score -= 3.0 * is_suv
        score += 2.0 * (seats <= 0.5).astype("float32")
        score += 1.5 * (fuel == petrol_code)
        score += 1.0 * (trans == manual_code)
        score += 10 * df["body_coupe"]
        score -= 8 * df["body_suv"]
        score -= 5 * df["body_sedan"]
        score += 15 * df["is_sport_model"]
        score += base_score
```

```
    # OFF-ROAD
    elif seg == 5:
        score += 4.0 * is_suv
        score += 3.0 * torque
        score += 2.0 * engine
        score += 1.5 * (fuel == diesel_code)
        score += 1.0 * (seats >= 0.6).astype("float32")
        score += 1.0 * (trans == manual_code)
        score += base_score

    # LUXURY / PREMIUM
    elif seg == 6:
        score += 3.0 * (1 - np.abs(price - user_row["selling_price"]))
        score += 2.5 * (1 - np.abs(year - user_row["year"]))
        score += 2.0 * (trans == auto_code)
        score += 1.5 * (seller == dealer_code)
        score += 2.0 * (seats >= 0.6).astype("float32")
        score += 1.5 * (fuel == petrol_code)
        score += 1.0 * power
        score += 8 * df["body_sedan"]
        score += 10 * df["is_premium_brand"]
        score -= 6 * df["body_suv"]
        score -= 20 * (df["is_premium_brand"] == 0)
        score += base_score

    else:
        score += base_score

    return score
```

In [69]:
```
def generate_training_pairs_fast(users_df, cars_df, n_pos=15, n_neg=15):
    u_num_list = []
    u_fuel_list = []
    u_seller_list = []
    u_trans_list = []
    u_owner_list = []

    i_num_list = []
    i_fuel_list = []
    i_seller_list = []
    i_trans_list = []
    i_owner_list = []

    y_list = []

    for _, user in users_df.iterrows():
        scores = score_items_for_segment(user, cars_df)
        idx_sorted = np.argsort(scores)
        pos_idx = idx_sorted[-n_pos:]
        neg_idx = idx_sorted[:n_neg]

        def add_pairs(indices, label):
            for idx in indices:
                car = cars_df.iloc[idx]

                u_num_list.append(user[NUM_COLS].values.astype("float32")
                u_fuel_list.append(int(user["fuel"]))
                u_seller_list.append(int(user["seller_type"]))
                u_trans_list.append(int(user["transmission"]))
```

```
                u_owner_list.append(int(user["owner"]))

                i_num_list.append(car[NUM_COLS].values.astype("float32"))
                i_fuel_list.append(int(car["fuel"]))
                i_seller_list.append(int(car["seller_type"]))
                i_trans_list.append(int(car["transmission"]))
                i_owner_list.append(int(car["owner"]))

                y_list.append(float(label))

        add_pairs(pos_idx, 1.0)
        add_pairs(neg_idx, 0.0)

    u_num = np.stack(u_num_list).astype("float32")
    u_fuel = np.array(u_fuel_list, dtype="int32")
    u_seller = np.array(u_seller_list, dtype="int32")
    u_trans = np.array(u_trans_list, dtype="int32")
    u_owner = np.array(u_owner_list, dtype="int32")

    i_num = np.stack(i_num_list).astype("float32")
    i_fuel = np.array(i_fuel_list, dtype="int32")
    i_seller = np.array(i_seller_list, dtype="int32")
    i_trans = np.array(i_trans_list, dtype="int32")
    i_owner = np.array(i_owner_list, dtype="int32")

    y = np.array(y_list, dtype="float32")

    return (
        u_num,
        u_fuel,
        u_seller,
        u_trans,
        u_owner,
        i_num,
        i_fuel,
        i_seller,
        i_trans,
        i_owner,
        y,
    )


(
    u_num,
    u_fuel,
    u_seller,
    u_trans,
    u_owner,
    i_num,
    i_fuel,
    i_seller,
    i_trans,
    i_owner,
    y,
) = generate_training_pairs_fast(users_df, df_items, n_pos=15, n_neg=15)

print("u_num:", u_num.shape)
print("i_num:", i_num.shape)
print("y:", y.shape)
```

```
u_num: (7200, 8)
i_num: (7200, 8)
y: (7200,)
```

In [70]:

```
# Two-tower model
embedding_dim = 32
num_numeric = len(NUM_COLS)

# USER tower
user_numeric_in = layers.Input(shape=(num_numeric,), name="user_num")
user_fuel_in = layers.Input(shape=(), dtype="int32", name="user_fuel")
user_seller_in = layers.Input(shape=(), dtype="int32", name="user_seller"
user_trans_in = layers.Input(shape=(), dtype="int32", name="user_trans")
user_owner_in = layers.Input(shape=(), dtype="int32", name="user_owner")

uf_emb = layers.Embedding(num_fuel, 8)(user_fuel_in)
us_emb = layers.Embedding(num_seller, 8)(user_seller_in)
ut_emb = layers.Embedding(num_trans, 8)(user_trans_in)
uo_emb = layers.Embedding(num_owner, 8)(user_owner_in)

u_concat = layers.Concatenate()(
    [
        user_numeric_in,
        layers.Flatten()(uf_emb),
        layers.Flatten()(us_emb),
        layers.Flatten()(ut_emb),
        layers.Flatten()(uo_emb),
    ]
)

u_hidden = layers.Dense(128, activation="relu")(u_concat)
u_hidden = layers.Dropout(0.2)(u_hidden)
u_hidden = layers.Dense(64, activation="relu")(u_hidden)
u_vec = layers.Dense(embedding_dim)(u_hidden)

user_tower = Model(
    inputs=[user_numeric_in, user_fuel_in, user_seller_in, user_trans_in,
    outputs=u_vec,
)

# ITEM tower
item_numeric_in = layers.Input(shape=(num_numeric,), name="item_num")
item_fuel_in = layers.Input(shape=(), dtype="int32", name="item_fuel")
item_seller_in = layers.Input(shape=(), dtype="int32", name="item_seller"
item_trans_in = layers.Input(shape=(), dtype="int32", name="item_trans")
item_owner_in = layers.Input(shape=(), dtype="int32", name="item_owner")

if_emb = layers.Embedding(num_fuel, 8)(item_fuel_in)
is_emb = layers.Embedding(num_seller, 8)(item_seller_in)
it_emb = layers.Embedding(num_trans, 8)(item_trans_in)
io_emb = layers.Embedding(num_owner, 8)(item_owner_in)

i_concat = layers.Concatenate()(
    [
        item_numeric_in,
        layers.Flatten()(if_emb),
        layers.Flatten()(is_emb),
        layers.Flatten()(it_emb),
        layers.Flatten()(io_emb),
    ]
```

```
)

i_hidden = layers.Dense(128, activation="relu")(i_concat)
i_hidden = layers.Dropout(0.2)(i_hidden)
i_hidden = layers.Dense(64, activation="relu")(i_hidden)
i_vec = layers.Dense(embedding_dim)(i_hidden)

item_tower = Model(
    inputs=[item_numeric_in, item_fuel_in, item_seller_in, item_trans_in,
    outputs=i_vec,
)

dot_score = layers.Dot(axes=1)([u_vec, i_vec])

model = Model(
    inputs=[
        user_numeric_in,
        user_fuel_in,
        user_seller_in,
        user_trans_in,
        user_owner_in,
        item_numeric_in,
        item_fuel_in,
        item_seller_in,
        item_trans_in,
        item_owner_in,
    ],
    outputs=dot_score,
)

model.compile(optimizer="adam", loss="binary_crossentropy")
model.summary()
```

Model: "functional_11"

Layer (type)	Output Shape	Param #	Connected t
user_fuel (InputLayer)	(None)	0	-
user_seller (InputLayer)	(None)	0	-
user_trans (InputLayer)	(None)	0	-
user_owner (InputLayer)	(None)	0	-
item_fuel (InputLayer)	(None)	0	-
item_seller (InputLayer)	(None)	0	-
item_trans (InputLayer)	(None)	0	-
item_owner (InputLayer)	(None)	0	-
embedding_24 (Embedding)	(None, 8)	32	user_fuel[(
embedding_25 (Embedding)	(None, 8)	24	user_selle
embedding_26 (Embedding)	(None, 8)	16	user_trans
embedding_27 (Embedding)	(None, 8)	40	user_owner
embedding_28 (Embedding)	(None, 8)	32	item_fuel[(
embedding_29 (Embedding)	(None, 8)	24	item_selle
embedding_30 (Embedding)	(None, 8)	16	item_trans
embedding_31 (Embedding)	(None, 8)	40	item_owner
user_num (InputLayer)	(None, 8)	0	-
flatten_24 (Flatten)	(None, 8)	0	embedding_2
flatten_25 (Flatten)	(None, 8)	0	embedding_2
flatten_26 (Flatten)	(None, 8)	0	embedding_2
flatten_27	(None, 8)	0	embedding_2

(Flatten)			
item_num (InputLayer)	(None, 8)	0	-
flatten_28 (Flatten)	(None, 8)	0	embedding_2
flatten_29 (Flatten)	(None, 8)	0	embedding_2
flatten_30 (Flatten)	(None, 8)	0	embedding_3
flatten_31 (Flatten)	(None, 8)	0	embedding_3
concatenate_6 (Concatenate)	(None, 40)	0	user_num[0] flatten_24 flatten_25 flatten_26 flatten_27
concatenate_7 (Concatenate)	(None, 40)	0	item_num[0] flatten_28 flatten_29 flatten_30 flatten_31
dense_18 (Dense)	(None, 128)	5,248	concatenate
dense_21 (Dense)	(None, 128)	5,248	concatenate
dropout_6 (Dropout)	(None, 128)	0	dense_18[0]
dropout_7 (Dropout)	(None, 128)	0	dense_21[0]
dense_19 (Dense)	(None, 64)	8,256	dropout_6[0
dense_22 (Dense)	(None, 64)	8,256	dropout_7[0
dense_20 (Dense)	(None, 32)	2,080	dense_19[0]
dense_23 (Dense)	(None, 32)	2,080	dense_22[0]
dot_3 (Dot)	(None, 1)	0	dense_20[0] dense_23[0]

Total params: 31,392 (122.62 KB)

Trainable params: 31,392 (122.62 KB)

Non-trainable params: 0 (0.00 B)

In [71]:
```
# Trening
history = model.fit(
    [u_num, u_fuel, u_seller, u_trans, u_owner, i_num, i_fuel, i_seller,
    y,
    epochs=10,
    batch_size=64,
    verbose=1,
)
```

```
def build_item_inputs_from_df(cars_df: pd.DataFrame):
    num = cars_df[NUM_COLS].values.astype("float32")
    fuel = cars_df["fuel"].values.astype("int32")
    seller = cars_df["seller_type"].values.astype("int32")
    trans = cars_df["transmission"].values.astype("int32")
    owner = cars_df["owner"].values.astype("int32")
    return num, fuel, seller, trans, owner


item_num_all, item_fuel_all, item_seller_all, item_trans_all, item_owner_
item_embeddings = item_tower.predict(
    [item_num_all, item_fuel_all, item_seller_all, item_trans_all, item_o
    verbose=0,
)
```

```
Epoch 1/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 5s 24ms/step - loss: 0.5078
Epoch 2/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - loss: 0.2846
Epoch 3/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - loss: 0.4055
Epoch 4/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - loss: 0.1344
Epoch 5/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - loss: 0.0557
Epoch 6/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - loss: 0.0306
Epoch 7/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - loss: 1.5994
Epoch 8/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - loss: 1.3567
Epoch 9/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - loss: 0.2550
Epoch 10/10
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - loss: 0.3338
```

In [72]:

```
def recommend_for_user(user_pref: dict, top_n=10):
    user_num = np.array([[user_pref[c] for c in NUM_COLS]], dtype="float3
    user_fuel = np.array([user_pref["fuel"]], dtype="int32")
    user_seller = np.array([user_pref["seller_type"]], dtype="int32")
    user_trans = np.array([user_pref["transmission"]], dtype="int32")
    user_owner = np.array([user_pref["owner"]], dtype="int32")

    u_emb = user_tower.predict(
        [user_num, user_fuel, user_seller, user_trans, user_owner],
        verbose=0,
    )

    scores = cosine_similarity(u_emb, item_embeddings)[0]
    sorted_idx = np.argsort(scores)[::-1]

    seen_names = set()
    selected_idx = []

    for idx in sorted_idx:
        name = df.iloc[idx]["name"]
        if name not in seen_names:
            seen_names.add(name)
            selected_idx.append(idx)
        if len(selected_idx) == top_n:
```

```
                break

    return df.iloc[selected_idx][
        ["name", "year", "selling_price", "km_driven", "fuel", "transmiss
    ]
```

In [73]:
```
sport_user = {
    "year": 0.6,
    "selling_price": 0.8,
    "km_driven": 0.2,
    "mileage": 0.4,
    "engine": 1,
    "max_power": 1,
    "torque": 0.85,
    "seats": 0.1,
    "fuel": petrol_code,
    "seller_type": dealer_code,
    "transmission": manual_code,
    "owner": first_owner_code,
}

print("sport user:")
print(recommend_for_user(sport_user, top_n=10))
```

```
sport user:
                                              name      year  selling_pric
e  \
170                    Volvo XC90 T8 Excellence BSIV  0.884615       1.00000
0
1338  Ford Ecosport 1.0 Ecoboost Titanium Optional  0.730769       0.04002
0
5399                           Hyundai Verna SX Opt  0.807692       0.05115
4
6543                      Ford Figo Aspire Titanium  0.807692       0.04714
2
6289                            Honda City 1.5 V MT  0.730769       0.04463
4
5402                         Maruti Baleno Delta 1.2  0.884615       0.05717
2
93                    Volkswagen Vento Petrol Highline  0.692308       0.05065
2
345                     Mahindra XUV300 W8 Option BSIV  0.961538       0.10631
9
4380                 Skoda Octavia Elegance 1.8 TSI AT  0.884615       0.15747
3
4948                          Hyundai i20 Sportz 1.2  0.769231       0.04312
9

      km_driven  fuel  transmission  owner
170    0.012709     3             0      0
1338   0.020801     3             1      2
5399   0.014827     3             1      2
6543   0.007625     3             1      2
6289   0.023300     3             1      2
5402   0.014404     3             1      2
93     0.005083     3             1      2
345    0.006990     3             1      0
4380   0.013980     3             0      2
4948   0.017167     3             1      2
```

In [74]:
```
family_user = {
    "year": 0.75,
    "selling_price": 0.6,
    "km_driven": 0.25,
    "mileage": 0.6,
    "engine": 0.6,
    "max_power": 0.6,
    "torque": 0.6,
    "seats": 0.8,
    "fuel": diesel_code,
    "seller_type": dealer_code,
    "transmission": auto_code,
    "owner": first_owner_code,
}

print("\nfamily user:")
print(recommend_for_user(family_user, top_n=10))
```

```
family user:
                                     name      year  selling_price  \
1870          Maruti Vitara Brezza ZDi Plus  0.884615       0.082247
1864                  Maruti Ciaz 1.3 Alpha  0.923077       0.089769
5900         Mahindra Bolero Pik-Up FB 1.7T  1.000000       0.065095
6629        Mahindra Bolero Pik-Up CBC 1.7T  0.961538       0.069408
6720  Jeep Compass 2.0 Longitude Option BSIV  0.961538       0.182548
2838   Toyota Innova 2.5 G (Diesel) 8 Seater  0.769231       0.065196
6288     Toyota Innova 2.5 V Diesel 7-seater  0.730769       0.072217
56     Toyota Innova 2.5 G (Diesel) 7 Seater  0.846154       0.092277
110          Mahindra XUV500 W11 Option AWD  0.961538       0.167503
3991             Mahindra Scorpio SLE BS IV  0.692308       0.045136

      km_driven  fuel  transmission  owner
1870   0.022715     1             1      0
1864   0.010045     1             1      0
5900   0.002118     1             1      0
6629   0.033891     1             1      0
6720   0.005931     1             1      0
2838   0.040246     1             1      0
6288   0.040246     1             1      0
56     0.041941     1             1      0
110    0.013556     1             1      0
3991   0.036210     1             1      0
```

In [75]:
```
budget_user = {
    "year": 0.4,
    "selling_price": 0.2,
    "km_driven": 0.5,
    "mileage": 0.8,
    "engine": 0.4,
    "max_power": 0.4,
    "torque": 0.4,
    "seats": 0.5,
    "fuel": petrol_code,
    "seller_type": individual_code,
    "transmission": manual_code,
    "owner": first_owner_code,
}
```

```
print("\nbudget user:")
print(recommend_for_user(budget_user, top_n=10))
```

```
budget user:
                              name      year  selling_price  km_driven  \
2183                Maruti Alto LXi  0.538462       0.009830   0.008049
2770                   Tata Nano Cx  0.576923       0.004012   0.006354
479           Hyundai i10 Magna 1.1L  0.538462       0.020060   0.021182
2342              Hyundai i10 Magna  0.538462       0.019057   0.014827
6129                Hyundai i10 Era  0.576923       0.013039   0.014827
3388                 Maruti Alto LX  0.576923       0.009027   0.007202
7121                Maruti Alto STD  0.538462       0.009930   0.025418
4310  Maruti Zen Estilo 1.1 LXI BSIII  0.500000       0.012036   0.014827
1466  Maruti Zen Estilo 1.1 VXI BSIII  0.538462       0.009027   0.016945
3893           Maruti Alto LXi BSIII  0.576923       0.010532   0.021182

      fuel  transmission  owner
2183     3             1      0
2770     3             1      0
479      3             1      0
2342     3             1      0
6129     3             1      0
3388     3             1      0
7121     3             1      0
4310     3             1      0
1466     3             1      0
3893     3             1      0
```

In [76]:
```
offroad_user = {
    "year": 0.6,
    "selling_price": 0.6,
    "km_driven": 0.5,
    "mileage": 0.5,
    "engine": 0.9,
    "max_power": 0.8,
    "torque": 0.9,
    "seats": 0.8,
    "fuel": diesel_code,
    "seller_type": dealer_code,
    "transmission": manual_code,
    "owner": first_owner_code,
}

print("\noffroad user:")
print(recommend_for_user(offroad_user, top_n=10))
```

```
offroad user:
                                    name      year  selling_price  \
1870            Maruti Vitara Brezza ZDi Plus  0.884615       0.082247
1864                    Maruti Ciaz 1.3 Alpha  0.923077       0.089769
4900               Ford Endeavour 2.5L 4X2 MT  0.384615       0.037111
3604                  Hyundai Santa Fe 2WD MT  0.769231       0.107322
5953            Mahindra Scorpio 1.99 S10 4WD  0.538462       0.042126
7495                 Jeep Compass 2.0 Limited  0.884615       0.152457
5355          Jeep Compass 2.0 Longitude BSIV  0.884615       0.152457
2795          Jeep Compass 2.0 Limited Option  0.884615       0.129890
5828   Jeep Compass 2.0 Longitude Option BSIV  0.884615       0.152457
4592                 Chevrolet Captiva 2.2 LT  0.730769       0.063892

      km_driven  fuel  transmission  owner
1870   0.022715     1             1      0
1864   0.010045     1             1      0
4900   0.039575     1             1      0
3604   0.033891     1             1      0
5953   0.033044     1             1      0
7495   0.050837     1             1      0
5355   0.037069     1             1      0
2795   0.012709     1             1      0
5828   0.021182     1             1      0
4592   0.027537     1             1      0
```

In [77]:
```
diesel_commuter_user = {
    "year": 0.7,
    "selling_price": 0.5,
    "km_driven": 0.4,
    "mileage": 0.8,
    "engine": 0.6,
    "max_power": 0.5,
    "torque": 0.7,
    "seats": 0.6,
    "fuel": diesel_code,
    "seller_type": dealer_code,
    "transmission": manual_code,
    "owner": first_owner_code,
}

print("\ndiesel commuter user:")
print(recommend_for_user(diesel_commuter_user, top_n=10))
```

```
diesel commuter user:
                                   name      year  selling_price  km_drive
n  \
7037                Chevrolet Cruze LTZ  0.615385       0.027081   0.02541
8
8085                 Chevrolet Cruze LT  0.653846       0.027081   0.01694
5
1880             Skoda Yeti Ambition 4WD  0.653846       0.057172   0.08557
6
2777                Hyundai Verna 1.6 SX  0.653846       0.047142   0.03389
1
8093              Hyundai Verna 1.6 CRDI  0.653846       0.042126   0.04236
4
863           Hyundai Verna 1.6 VGT CRDi  0.653846       0.027081   0.03389
1
2399        Chevrolet Optra Magnum 2.0 LS  0.653846       0.038616   0.03322
7
5693        Hyundai Verna 1.6 SX CRDi (O)  0.653846       0.036108   0.06481
8
4608     Volkswagen Vento Diesel Highline  0.653846       0.031093   0.03812
8
6356    Volkswagen Vento Diesel Trendline  0.653846       0.019559   0.03812
8

      fuel  transmission  owner
7037     1             1      0
8085     1             1      0
1880     1             1      0
2777     1             1      0
8093     1             1      0
863      1             1      0
2399     1             1      0
5693     1             1      0
4608     1             1      0
6356     1             1      0
```

In [ ]:

In [ ]: