Best Seller Product Prediction

Team B09: Aryan Jain, Ahrar Karim, Drishti Chulani, Linh Le, Sai Leela Rahul Pujari

Dataset: Olist Brazilian E-commerce Public Dataset (2016–2018)

1. Problem Statement

For a marketplace where a small share of products drives most of the revenue, the business needs not only to predict which new products will become top performers within their first few days of launch, but also to understand how they build that performance over time. Using the Olist Brazilian e-commerce dataset, this project builds a product-level model that predicts whether a new product will fall into the top 10% by units sold in its first 60 days and segments products into early-momentum profiles based on the ratio of sales, using only features available in the first 7–14 days.

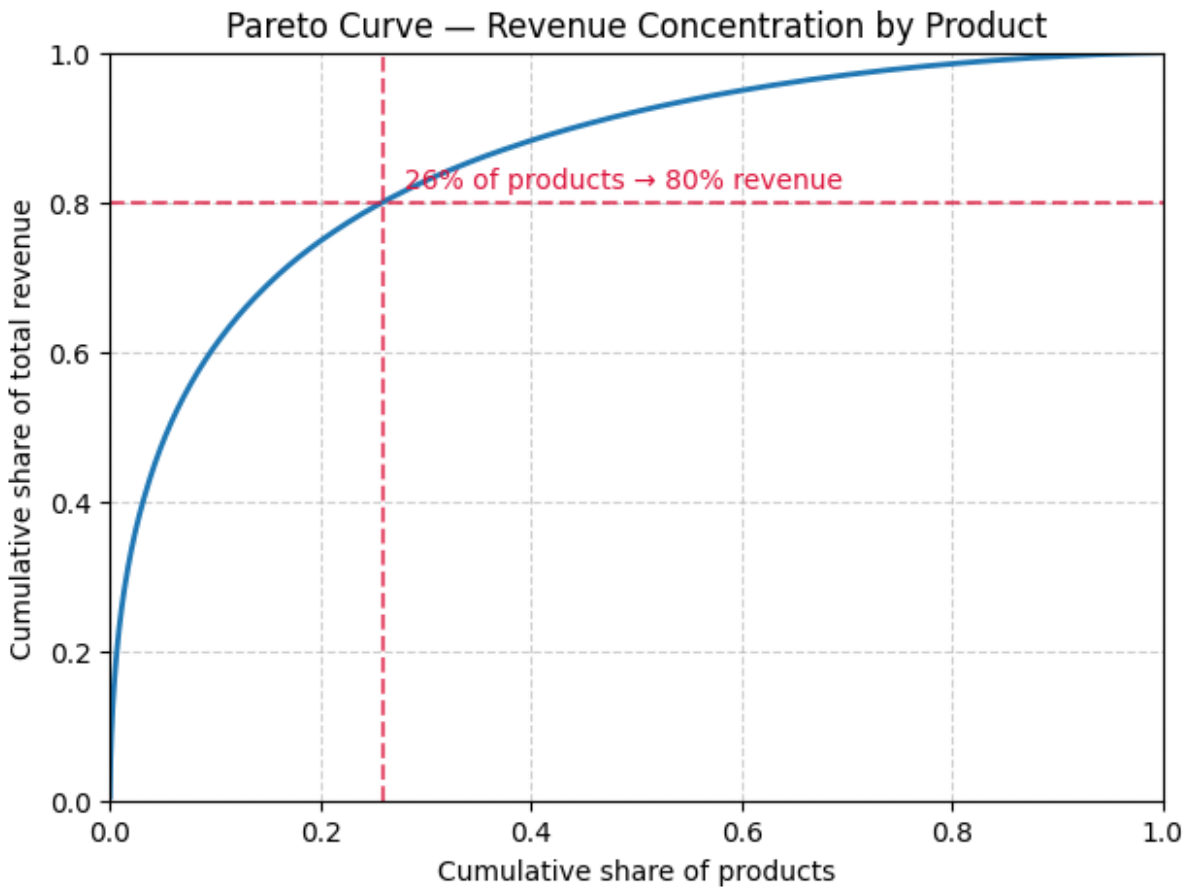
2. Exploratory Data Analysis

Pareto curve

This chart plots the cumulative share of total revenue (y-axis) against the cumulative share of products (x-axis), sorted from highest- to lowest-earning.

It's a visualization of the Pareto principle (80/20 rule) — the idea that a small portion of products generate most of the revenue.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

items = pd.read_csv("/content/olist_order_items_dataset.csv")
orders = pd.read_csv("/content/olist_orders_dataset.csv",
parse_dates=["order_purchase_timestamp"])
reviews = pd.read_csv("/content/olist_order_reviews_dataset.csv")
product = pd.read_csv("/content/olist_products_dataset.csv")

file_name = 'https://drive.google.com/uc?
export=download&id=13bQQfKDjiuBA1njsvCEW81c4Q2huRE9z'
data = pd.read_csv(file_name)

# 1) Revenue per product
#    (In Olist, `price` is the line price; summing gives observed
revenue per product.)
revenue = (items.groupby("product_id", as_index=False)
                .agg(revenue=("price","sum")))

# 2) Sort by revenue desc and compute cumulative shares (0..1)
revenue = revenue.sort_values("revenue",
ascending=False).reset_index(drop=True)
revenue["cum_revenue_share"] = revenue["revenue"].cumsum() /
revenue["revenue"].sum()
revenue["cum_product_share"] = (revenue.index + 1) / len(revenue)

# 3) Find x where we hit 80% of revenue
idx80 = int(np.argmax(revenue["cum_revenue_share"].values >= 0.80))
x80 = float(revenue.loc[idx80, "cum_product_share"])

# 4) Plot properly bounded to [0,1]
plt.figure(figsize=(7,5))
plt.plot(revenue["cum_product_share"], revenue["cum_revenue_share"],
lw=2)
plt.axhline(0.80, linestyle="--", color="crimson", alpha=0.7)
plt.axvline(x80, linestyle="--", color="crimson", alpha=0.7)
plt.xlim(0,1); plt.ylim(0,1)
plt.xlabel("Cumulative share of products")
plt.ylabel("Cumulative share of total revenue")
plt.title("Pareto Curve — Revenue Concentration by Product")
plt.grid(True, linestyle="--", alpha=0.6)
plt.text(min(x80+0.02,0.95), 0.82, f"{x80:.0%} of products → 80%
revenue", color="crimson")
plt.show()
```



This significant skew means that a 1% improvement in correctly predicting the top-performing products could unlock an outsized return in revenue by optimizing marketing and inventory.

Why it Motivates

The marketplace is winner-take-most. With only a quarter of the catalog producing the majority of revenue, correctly identifying those “winners” early has huge business value.

Predicting early success can guide decisions. If we can forecast which products will become bestsellers in their first 60 days, we can:

Promote them on the homepage or ads,

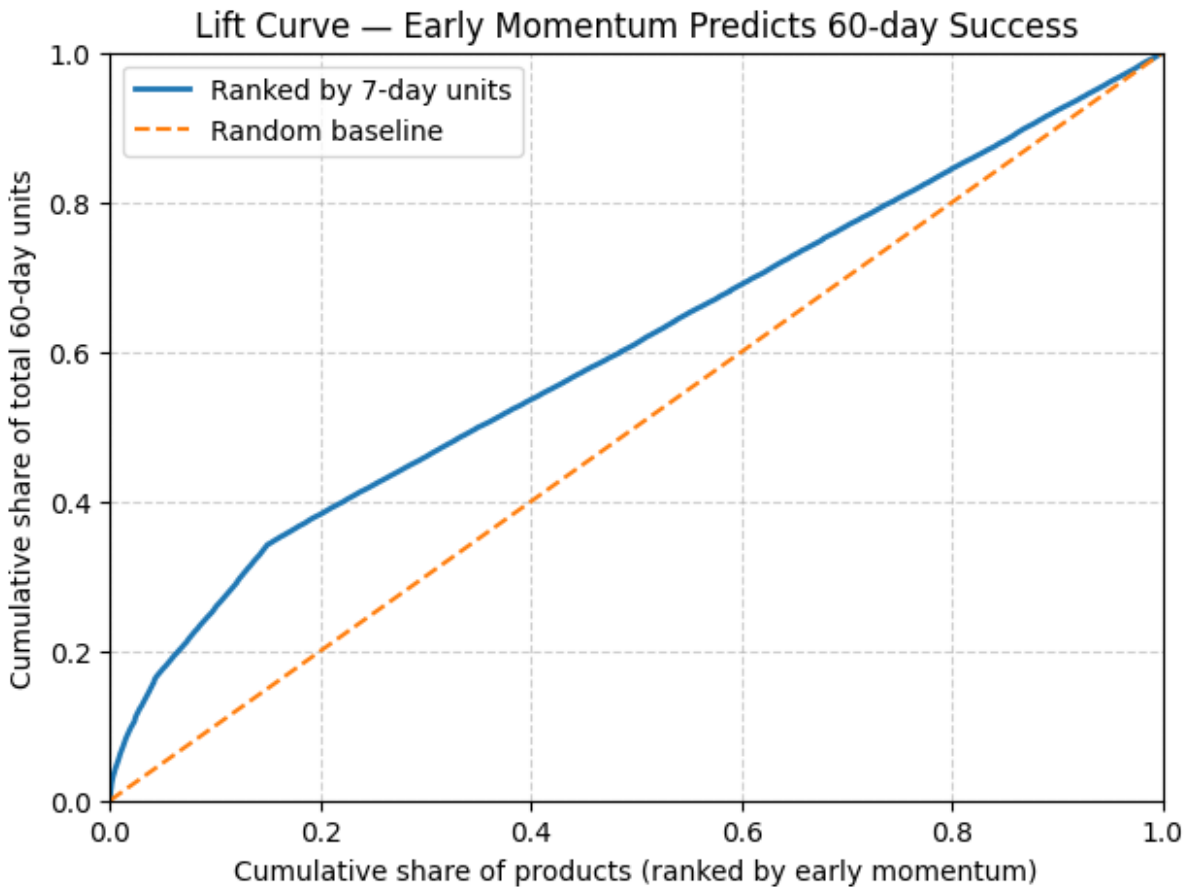
Allocate inventory and logistics efficiently,

Avoid overstocking low-performing “dead” item

Early-Momentum Lift Curve

What it shows: If you rank products by first-7-day units, how much of total 60-day units do the top X% capture? Why it motivates: If the curve is steep above the diagonal baseline, early momentum is predictive—so a model using ≤7-day signals makes sense.

```
# Launch date per product = first purchase timestamp
oi = items.merge(orders[["order_id","order_purchase_timestamp"]],
on="order_id", how="left")
launch = oi.groupby("product_id")
["order_purchase_timestamp"].min().rename("launch_ts")
oi = oi.merge(launch, on="product_id", how="left")

within7  = (oi["order_purchase_timestamp"] >= oi["launch_ts"]) &
(oi["order_purchase_timestamp"] < oi["launch_ts"] +
pd.Timedelta(days=7))
within60 = (oi["order_purchase_timestamp"] >= oi["launch_ts"]) &
(oi["order_purchase_timestamp"] < oi["launch_ts"] +
pd.Timedelta(days=60))

# Early momentum & eventual outcome
u7  = oi[within7].groupby("product_id")
["order_item_id"].count().rename("units7")
u60 = oi[within60].groupby("product_id")
["order_item_id"].count().rename("units60")

df = (pd.concat([u7,u60], axis=1).fillna(0).astype(int)
        .sort_values("units7", ascending=False).reset_index())

# Cumulative capture of 60-day units by top X% ranked on 7-day units
df["cum_units60"] = df["units60"].cumsum()
df["cum_share_products"] = (np.arange(1, len(df)+1)) / len(df)
df["cum_share_units60"]  = df["cum_units60"] / df["units60"].sum()

plt.figure(figsize=(7,5))
plt.plot(df["cum_share_products"], df["cum_share_units60"], lw=2,
label="Ranked by 7-day units")
plt.plot([0,1],[0,1], "--", label="Random baseline")
plt.xlim(0,1); plt.ylim(0,1)
plt.xlabel("Cumulative share of products (ranked by early momentum)")
plt.ylabel("Cumulative share of total 60-day units")
plt.title("Lift Curve — Early Momentum Predicts 60-day Success")
plt.grid(True, linestyle="--", alpha=0.6)
plt.legend()
plt.show()
```



Only a small fraction of products (those that sell quickly in the first 7 days) drive a disproportionate share of long-term sales.

Therefore, if we can predict which products will take off early, we can act before 60 days — promote them, stock them, or adjust prices for laggards.

Customer Satisfaction Distribution

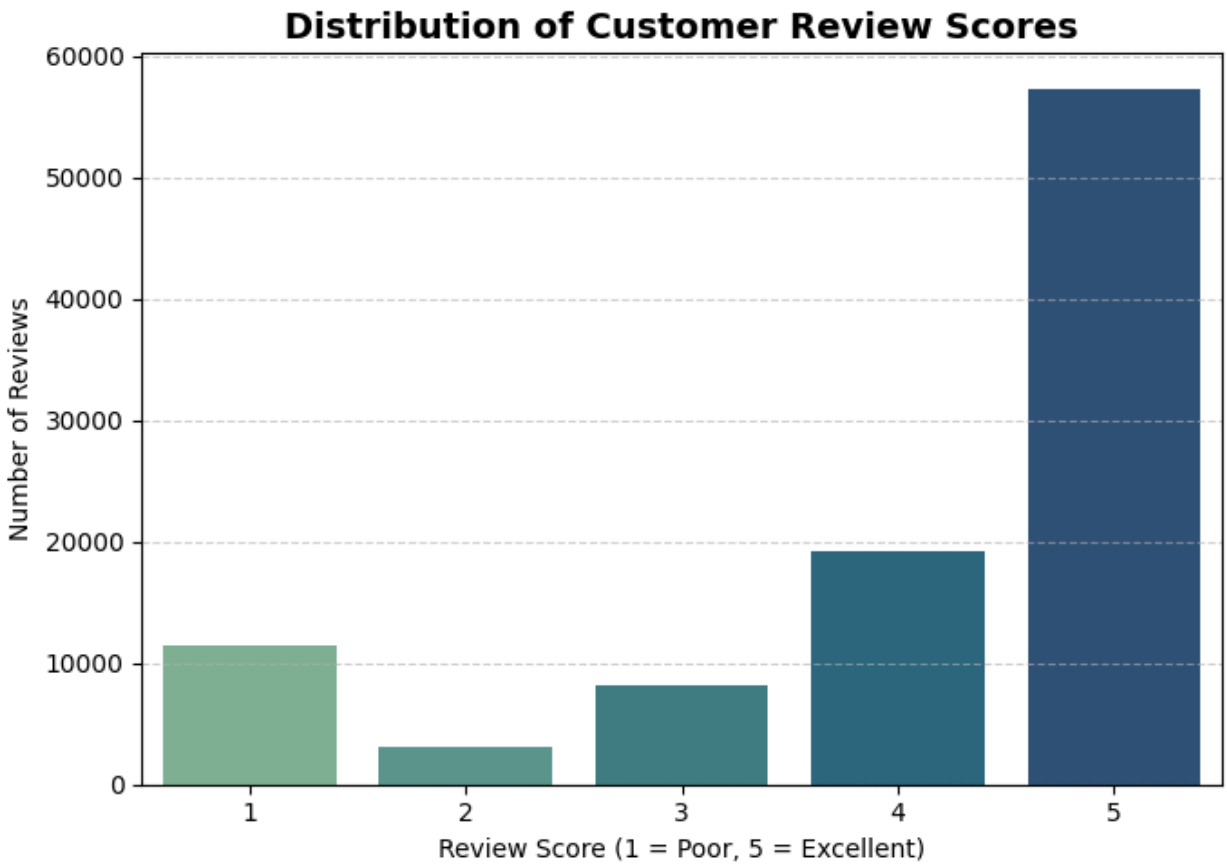
What it shows: This chart displays the frequency of customer review scores (1–5 stars) across all orders in the Olist dataset. Each bar represents how many reviews fall into that rating category. The top 20% of products (ranked by 7-day units) capture approximately 40-45% of the total 60-day units.

Why it motivates: Customer sentiment often correlates with both short-term sales velocity and long-term retention. Products receiving higher ratings early in their lifecycle are more likely to sustain sales momentum and visibility. Understanding the distribution of satisfaction scores also helps interpret noise and bias in feedback-based features since most e-commerce platforms exhibit a strong positive skew toward 4–5 star reviews.

```
#working on Review dataset

# Basic sanity check
print(reviews["review_score"].describe())
```

```python
# Plot histogram of review scores
plt.figure(figsize=(7,5))
sns.countplot(x="review_score", data=reviews, palette="crest")

plt.title("Distribution of Customer Review Scores", fontsize=14,
weight="bold")
plt.xlabel("Review Score (1 = Poor, 5 = Excellent)")
plt.ylabel("Number of Reviews")
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
count    99224.000000
mean         4.086421
std          1.347579
min          1.000000
25%          4.000000
50%          5.000000
75%          5.000000
max          5.000000
Name: review_score, dtype: float64

/tmp/ipython-input-587195617.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x="review_score", data=reviews, palette="crest")
```



Interpretation: As seen in Figure 3, the majority of reviews are 4 or 5 stars, indicating generally high customer satisfaction. However, the presence of low-rated reviews provides variation that can be leveraged as a predictive signal in modeling for instance, linking poor early ratings to reduced best-seller likelihood.

Insights:

- The Business Problem: The Pareto Curve confirms that 26% of products generate 80% of revenue, making the early identification of these 'winners' a high-value business priority.
- The Predictive Feasibility: The Lift Curve demonstrates that early sales (7-day units) are highly predictive of long-term success (60-day units), confirming that signals captured in the first week are sufficient to build an effective forecasting model.
- A Key Signal: The Distribution of Customer Review Scores identifies customer sentiment as a crucial feature, with the variation in rare, low-rated reviews providing powerful, actionable signals for model construction.

Data Information

```
# Merge items with orders and reviews and products

df = (
```

```
    items
    .merge(product[['product_id', 'product_category_name']],
on="product_id", how="left")
    .merge(orders, on="order_id", how="left")
    .merge(reviews[['order_id', 'review_score']], on="order_id",
how="left")
)
# Inspect combined dataset
print(df.shape)
print(df.head())
print(df.columns)
```

```
(113314, 16)
                           order_id  order_item_id  \
0  00010242fe8c5a6d1ba2dd792cb16214              1
1  00018f77f2f0320c557190d7a144bdd3              1
2  000229ec398224ef6ca0657da4fc703e              1
3  00024acbcdf0a6daa1e931b038114c75              1
4  00042b26cf59d7ce69dfabb4e55b4fd9              1

                         product_id                         seller_id
\
0  4244733e06e7ecb4970a6e2683c13e61  48436dade18ac8b2bce089ec2a041202

1  e5f2d52b802189ee658865ca93d83a8f  dd7ddc04e1b6c2c614352b383efe2d36

2  c777355d18b72b67abbeef9df44fd0fd  5b51032eddd242adc84c38acab88f23d

3  7634da152a4610f1595efa32f14722fc  9d7a1d34a5052409006425275ba1c2b4

4  ac6c3623068f30de03045865e4e10089  df560393f3a51e74553ab94004ba5c87


   shipping_limit_date   price  freight_value product_category_name  \
0  2017-09-19 09:45:35   58.90          13.29            cool_stuff
1  2017-05-03 11:05:13  239.90          19.93              pet_shop
2  2018-01-18 14:48:30  199.00          17.87     moveis_decoracao
3  2018-08-15 10:10:18   12.99          12.79            perfumaria
4  2017-02-13 13:57:51  199.90          18.14    ferramentas_jardim

                         customer_id order_status
order_purchase_timestamp  \
0  3ce436f183e68e07877b285a838db11a    delivered      2017-09-13
08:59:02
1  f6dd3ec061db4e3987629fe6b26e5cce    delivered      2017-04-26
10:53:06
2  6489ae5e4333f3693df5ad4372dab6d3    delivered      2018-01-14
14:33:31
3  d4eb9395c8c0431ee92fce09860c5a06    delivered      2018-08-08
10:00:35
```

```
4  58dbd0b2d70206bf40e62cd34e84d795    delivered      2017-02-04
13:57:51

     order_approved_at order_delivered_carrier_date  \
0  2017-09-13 09:45:35          2017-09-19 18:34:16
1  2017-04-26 11:05:13          2017-05-04 14:35:00
2  2018-01-14 14:48:30          2018-01-16 12:36:48
3  2018-08-08 10:10:18          2018-08-10 13:28:00
4  2017-02-04 14:10:13          2017-02-16 09:46:09

  order_delivered_customer_date order_estimated_delivery_date
review_score
0           2017-09-20 23:43:48           2017-09-29 00:00:00
5.0
1           2017-05-12 16:04:24           2017-05-15 00:00:00
4.0
2           2018-01-22 13:19:16           2018-02-05 00:00:00
5.0
3           2018-08-14 13:32:39           2018-08-20 00:00:00
4.0
4           2017-03-01 16:42:31           2017-03-17 00:00:00
5.0
Index(['order_id', 'order_item_id', 'product_id', 'seller_id',
       'shipping_limit_date', 'price', 'freight_value',
       'product_category_name', 'customer_id', 'order_status',
       'order_purchase_timestamp', 'order_approved_at',
       'order_delivered_carrier_date',
'order_delivered_customer_date',
       'order_estimated_delivery_date', 'review_score'],
      dtype='object')

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113314 entries, 0 to 113313
Data columns (total 16 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   order_id                       113314 non-null  object
 1   order_item_id                  113314 non-null  int64
 2   product_id                     113314 non-null  object
 3   seller_id                      113314 non-null  object
 4   shipping_limit_date            113314 non-null  object
 5   price                          113314 non-null  float64
 6   freight_value                  113314 non-null  float64
 7   product_category_name          111702 non-null  object
 8   customer_id                    113314 non-null  object
 9   order_status                   113314 non-null  object
 10  order_purchase_timestamp       113314 non-null  datetime64[ns]
 11  order_approved_at              113299 non-null  object
```

```
 12  order_delivered_carrier_date   112111 non-null  object
 13  order_delivered_customer_date  110839 non-null  object
 14  order_estimated_delivery_date  113314 non-null  object
 15  review_score                   112372 non-null  float64
dtypes: datetime64[ns](1), float64(3), int64(1), object(11)
memory usage: 13.8+ MB
```

```
df.describe()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n{\n      \"column\": \"order_item_id\",\n      \"properties\": {\n\"dtype\": \"number\",\n        \"std\": 40061.190579005735,\n\"min\": 0.707015757274616,\n        \"max\": 113314.0,\n\"num_unique_values\": 5,\n        \"samples\": [\n1.1985279841855376,\n          0.707015757274616,\n          1.0\n],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n}\n    },\n    {\n      \"column\": \"price\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 39762.4222805929,\n        \"min\": 0.85,\n        \"max\": 113314.0,\n\"num_unique_values\": 8,\n        \"samples\": [\n120.47870051361703,\n          134.9,\n          113314.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n}\n    },\n    {\n      \"column\": \"freight_value\",\n\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 40037.73999692956,\n        \"min\": 0.0,\n        \"max\": 113314.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n19.9794275199887,\n          21.15,\n          113314.0\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"order_purchase_timestamp\",\n\"properties\": {\n        \"dtype\": \"date\",\n        \"min\": \"1970-01-01 00:00:00.000113314\",\n        \"max\": \"2018-09-03 09:06:57\",\n        \"num_unique_values\": 7,\n        \"samples\": [\n          \"113314\",\n          \"2017-12-31 15:03:55.044654848\",\n          \"2018-05-04 11:08:28\"\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"review_score\",\n\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 39728.217719771514,\n        \"min\": 1.0,\n        \"max\": 112372.0,\n        \"num_unique_values\": 6,\n        \"samples\": [\n112372.0,\n          4.032472502046773,\n          1.3878487841663931\n        ],\n        \"semantic_type\": \"\",\n\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
# Save the merged dataset to a CSV file
output_path = "/content/olist_combined_dataset.csv"
df.to_csv(output_path, index=False)

print(f"□ Combined dataset saved to: {output_path}")
```

```
□ Combined dataset saved to: /content/olist_combined_dataset.csv
```

```
df.isnull().sum()
```

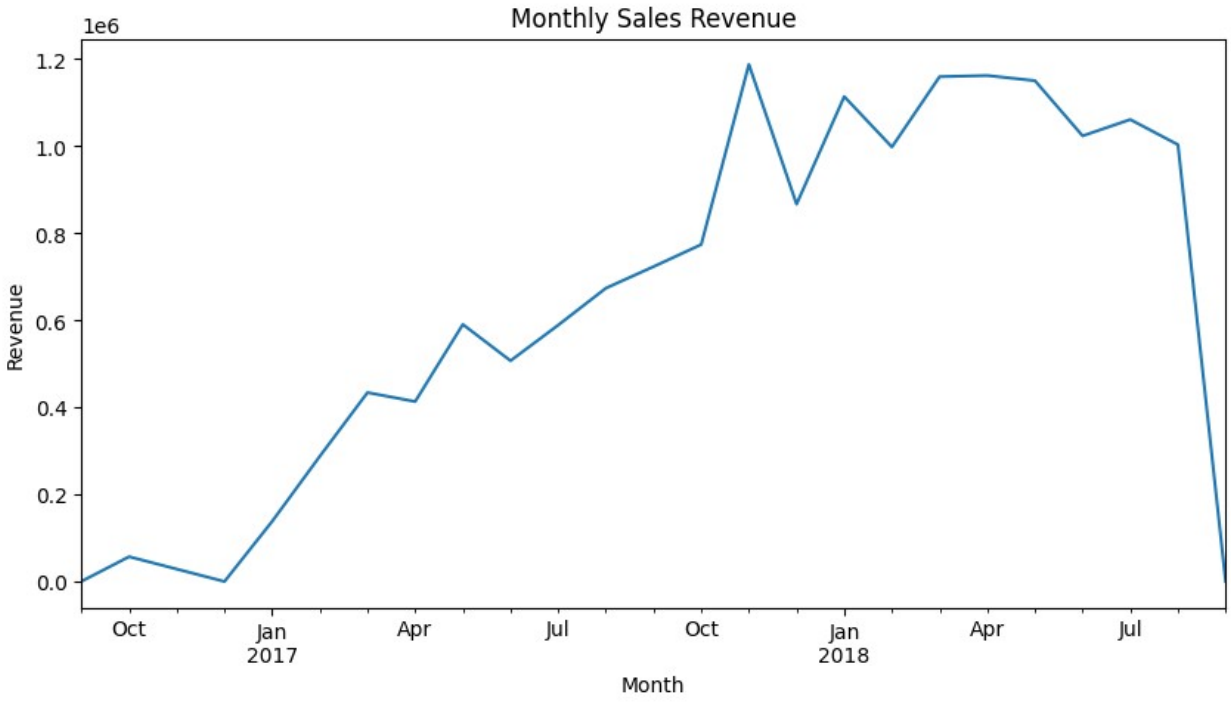
```
order_id                            0
order_item_id                       0
product_id                          0
seller_id                           0
shipping_limit_date                 0
price                               0
freight_value                       0
product_category_name            1612
customer_id                         0
order_status                        0
order_purchase_timestamp            0
order_approved_at                  15
order_delivered_carrier_date     1203
order_delivered_customer_date    2475
order_estimated_delivery_date       0
review_score                      942
dtype: int64
```

```
#changing the Datatype from object to Datetime

date_cols = [
    'shipping_limit_date',
    'order_approved_at',
    'order_delivered_carrier_date',
    'order_delivered_customer_date',
    'order_estimated_delivery_date'
]

for col in date_cols:
    df[col] = pd.to_datetime(df[col], errors='coerce')

df[date_cols].dtypes
```

```
shipping_limit_date             datetime64[ns]
order_approved_at               datetime64[ns]
order_delivered_carrier_date    datetime64[ns]
order_delivered_customer_date   datetime64[ns]
order_estimated_delivery_date   datetime64[ns]
dtype: object
```

```
#Delay in time , actual delivered- estimated

df['delivery_delay_days'] = (df['order_delivered_customer_date'] -
                         df['order_estimated_delivery_date']).dt.days

df['purchase_month'] =
df['order_purchase_timestamp'].dt.to_period('M')
df['purchase_monthyear'] =
df['order_purchase_timestamp'].dt.to_period('M')
```

```
# Delivery time purchase to customer

df['delivery_time_days'] = (
    df['order_delivered_customer_date'] -
df['order_purchase_timestamp']
).dt.days

#Total item revenue

df['item_revenue'] = df['price'] + df['freight_value']

monthly_sales = df.groupby('purchase_month')['item_revenue'].sum()

monthly_sales.plot(figsize=(10,5))
plt.title("Monthly Sales Revenue")
plt.ylabel("Revenue")
plt.xlabel("Month")
plt.show()
```



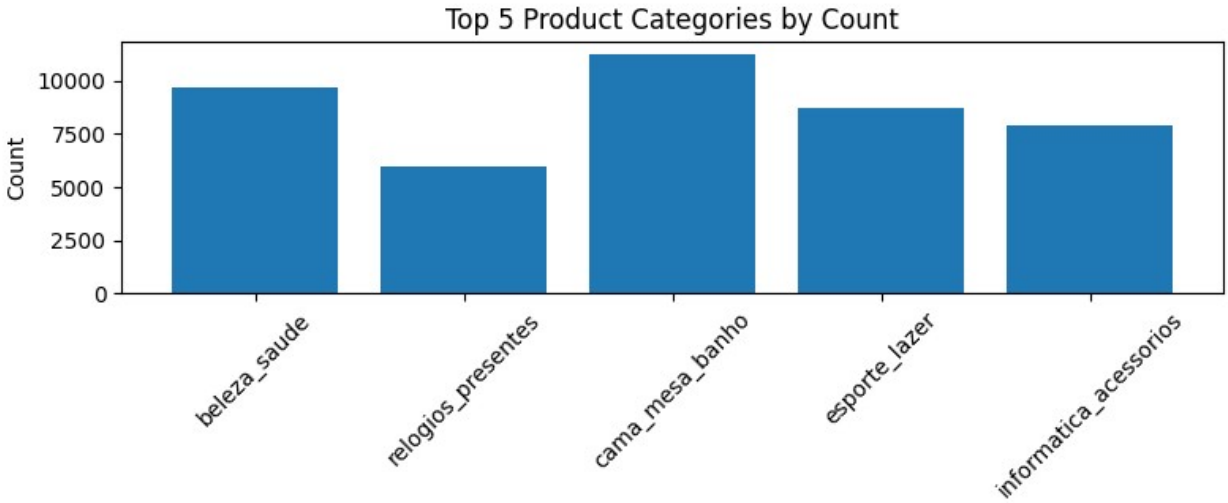

```
df.groupby('purchase_monthyear')['order_id'].count()

purchase_monthyear
2016-09       6
2016-10     367
2016-12       1
2017-01     966
2017-02    1962
2017-03    3020
```

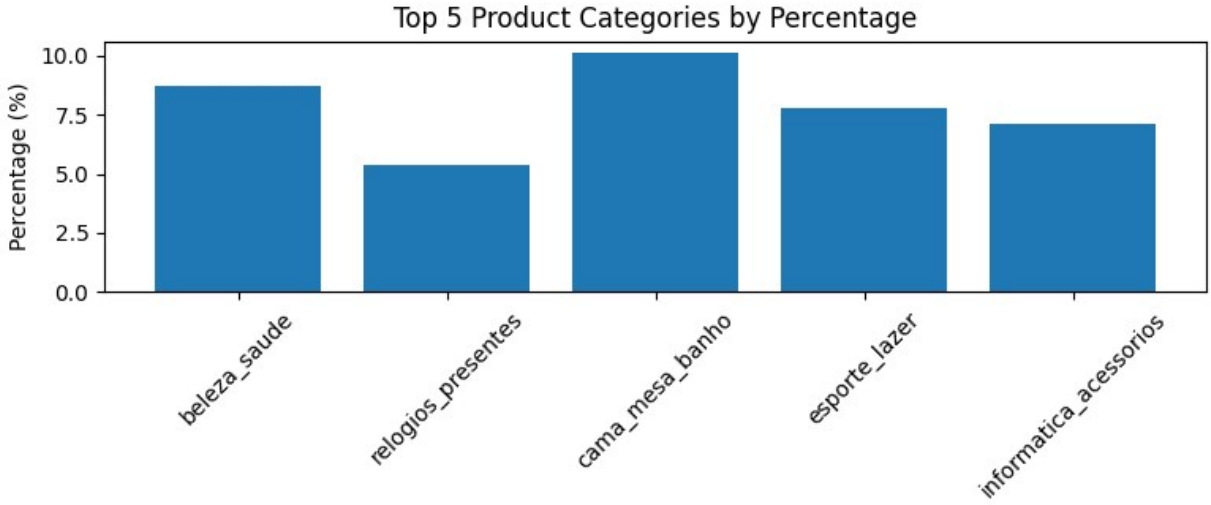
```
2017-04    2693
2017-05    4182
2017-06    3619
2017-07    4565
2017-08    4964
2017-09    4865
2017-10    5369
2017-11    8729
2017-12    6335
2018-01    8263
2018-02    7772
2018-03    8265
2018-04    7986
2018-05    7935
2018-06    7084
2018-07    7115
2018-08    7250
2018-09       1
Freq: M, Name: order_id, dtype: int64
```

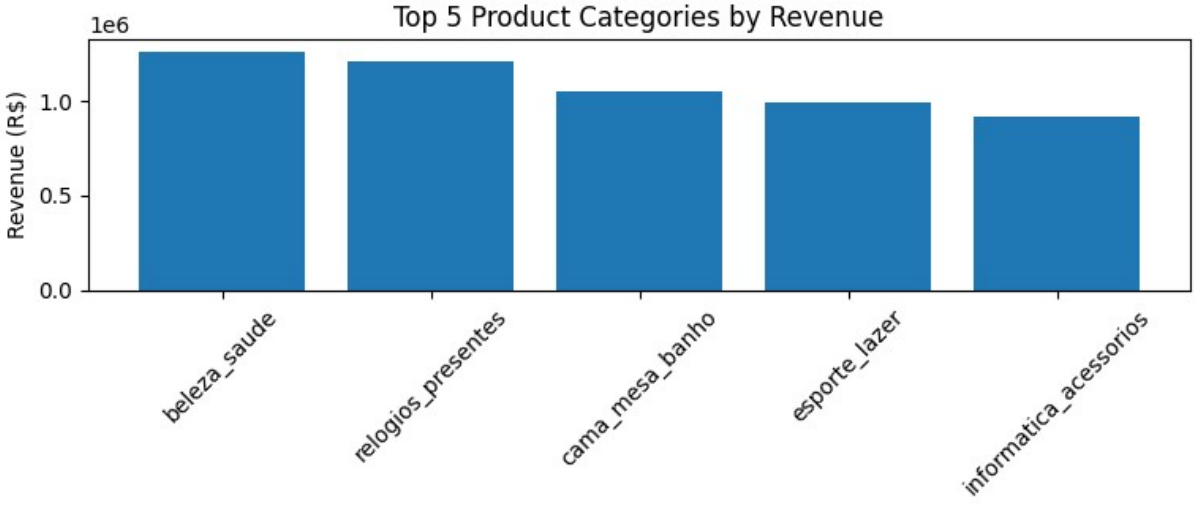
```
# Top 5 Product Categories by count, percentage and Revenue
summary = pd.DataFrame({
    "count": df['product_category_name'].value_counts(),
    "percentage":
df['product_category_name'].value_counts(normalize=True) * 100,
    "revenue": df.groupby('product_category_name')['price'].sum()
})

summary['percentage'] = summary['percentage'].round(2)

# Sort by count and show top 5
top5 = summary.sort_values("revenue", ascending=False).head(5)

print(top5)

plt.figure(figsize=(8, 10))

# Count
plt.subplot(3, 1, 1)
plt.bar(top5.index, top5['count'])
plt.title("Top 5 Product Categories by Count")
plt.ylabel("Count")
plt.xticks(rotation=45)

# Percentage
plt.subplot(3, 1, 2)
plt.bar(top5.index, top5['percentage'])
plt.title("Top 5 Product Categories by Percentage")
plt.ylabel("Percentage (%)")
plt.xticks(rotation=45)
```

```
# Revenue
plt.subplot(3, 1, 3)
plt.bar(top5.index, top5['revenue'])
plt.title("Top 5 Product Categories by Revenue")
plt.ylabel("Revenue (R$)")
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```

product_category_name	count	percentage	revenue
beleza_saude	9727	8.71	1263138.54
relogios_presentes	6001	5.37	1206075.33
cama_mesa_banho	11270	10.09	1050936.61
esporte_lazer	8700	7.79	993656.51
informatica_acessorios	7894	7.07	919640.54







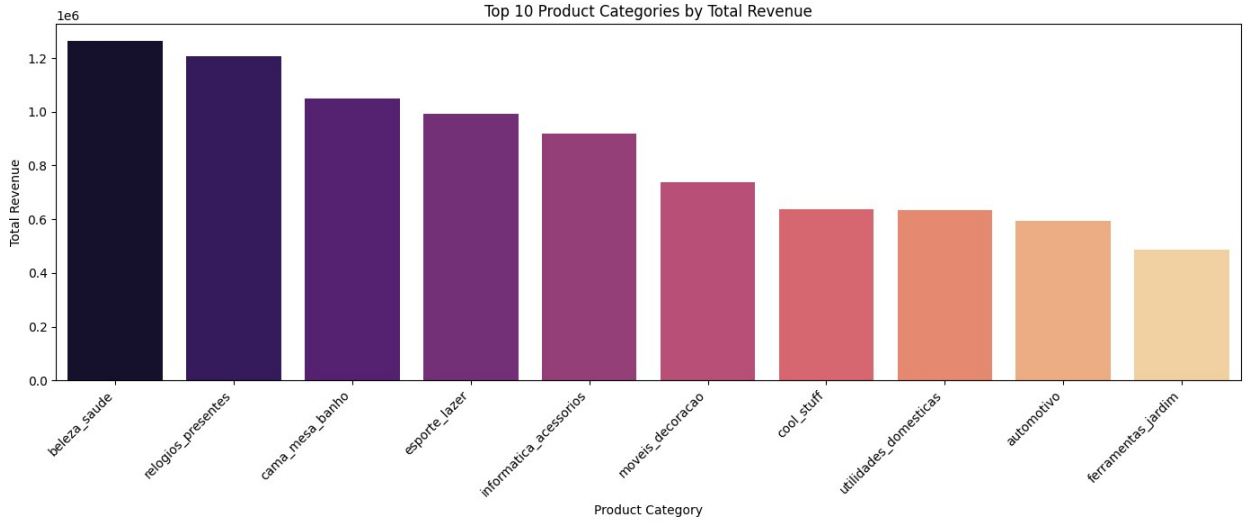
Translation of the 5 product categories are

product_category_name	count	percentage	revenue
beauty_health	9727	8.71	1,263,138.54
watches_gifts	6001	5.37	1,206,075.33
bed_bath_table	11270	10.09	1,050,936.61
sports_leisure	8700	7.79	993,656.51
computers_accessories	7894	7.07	919,640.54

```
# Merge into single df, now including product_category_name
df = (
    items
    .merge(orders, on="order_id", how="left")
    .merge(reviews[["order_id", "review_score"]], on="order_id",
how="left")
    .merge(product[["product_id", "product_category_name"]],
on="product_id", how="left")  # <- change here
)

# Calculate total revenue per category
category_revenue = (
    df.groupby("product_category_name")["price"]
      .sum()
      .reset_index()
)

# Sort by revenue in descending order
top_category_revenue = category_revenue.sort_values(by="price",
ascending=False)

# Get category with highest revenue
highest_rev_cat = top_category_revenue.iloc[0]
print(
    f"The category with the highest revenue is
'{highest_rev_cat['product_category_name']}' "
    f"with a total revenue of ${highest_rev_cat['price']:.2f}.\n"
)

# Plot top 10 categories by revenue
plt.figure(figsize=(14, 6))
sns.barplot(
    x="product_category_name",
    y="price",
    data=top_category_revenue.head(10),
    palette="magma"
)
plt.title("Top 10 Product Categories by Total Revenue")
plt.xlabel("Product Category")
plt.ylabel("Total Revenue")
```

```
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```

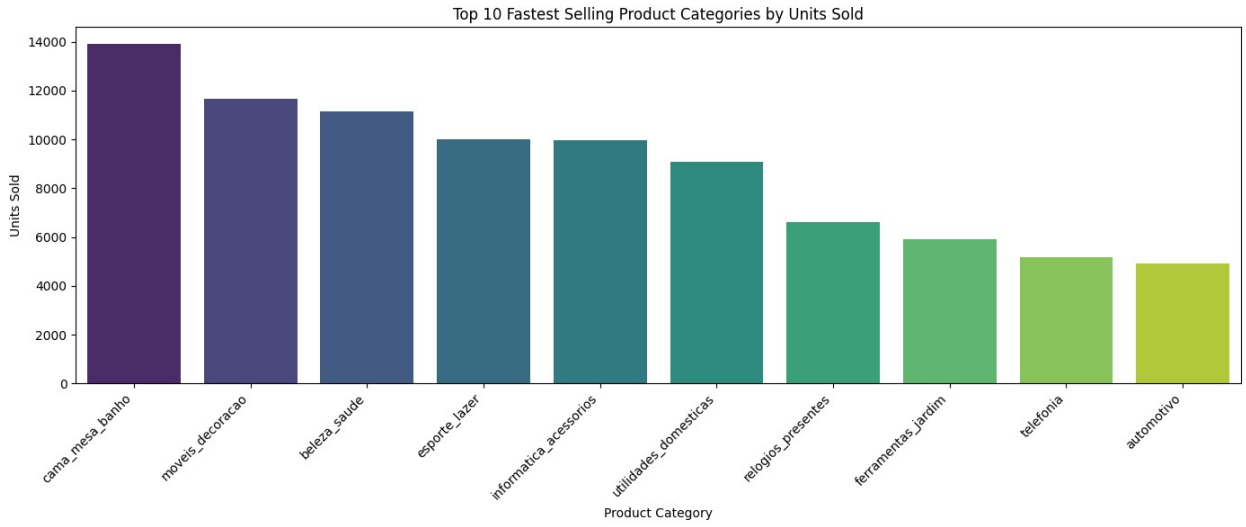
```
The category with the highest revenue is 'beleza_saude' with a total
revenue of $1263138.54.


/tmp/ipython-input-4196354713.py:28: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(
```




```
# Group by category
category_units = (
    df.groupby('product_category_name')['order_item_id']
      .sum()
      .reset_index()
)

# Sort
fastest_selling_categories =
category_units.sort_values(by='order_item_id', ascending=False)

# Print #1
top_cat = fastest_selling_categories.iloc[0]
print(
    f"Fastest selling category: {top_cat['product_category_name']} "
    f"({top_cat['order_item_id']} units sold)\n"
)
```

```
# Plot
plt.figure(figsize=(14, 6))
sns.barplot(
    x='product_category_name',
    y='order_item_id',
    data=fastest_selling_categories.head(10),
    palette='viridis'
)
plt.title('Top 10 Fastest Selling Product Categories by Units Sold')
plt.xlabel('Product Category')
plt.ylabel('Units Sold')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
Fastest selling category: cama_mesa_banho (13911 units sold)


/tmp/ipython-input-2480787699.py:20: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(
```




```
#Freight share of product price

df['freight_ratio'] = df['freight_value'] / df['price']

# Make sure datetime columns are in datetime format
date_cols = [
    'order_delivered_customer_date',
```

```
    'order_estimated_delivery_date'
]

for col in date_cols:
    df[col] = pd.to_datetime(df[col], errors='coerce')

# Compute delivery_delay_days
df['delivery_delay_days'] = (
    df['order_delivered_customer_date'] -
df['order_estimated_delivery_date']
).dt.days


#late flag
df['is_late'] = (df['delivery_delay_days'] > 0).astype(int)

#missing values in the date
df['delivered_flag'] =
df['order_delivered_customer_date'].notnull().astype(int)

#flag the missing review scores
df['has_review'] = df['review_score'].notnull().astype(int)

df['order_purchase_timestamp'] =
pd.to_datetime(df['order_purchase_timestamp'], errors='coerce')
df['order_delivered_customer_date'] =
pd.to_datetime(df['order_delivered_customer_date'], errors='coerce')

# Recompute delivery_time_days (delivery - purchase)
df['delivery_time_days'] = (
    df['order_delivered_customer_date'] -
df['order_purchase_timestamp']
).dt.days


df[(df['delivery_time_days'] > 60) | (df['delivery_time_days'] < 0)]

{"type":"dataframe"}

df

{"type":"dataframe","variable_name":"df"}
```

3. Label Construction

```
import numpy as np
import pandas as pd

# ============================================================
# LABEL CREATION + EARLY-MOMENTUM FEATURES
# ============================================================
```

```python
# 1) Ensure product_id is string everywhere we need it
df["product_id"] = df["product_id"].astype(str)

oi_for_label_construction = df[['product_id', 'order_id',
'order_item_id', 'order_purchase_timestamp']].copy()

# 2) Build launch timestamp per product using the global 'df'
launch = (
    oi_for_label_construction.groupby("product_id")
["order_purchase_timestamp"]
      .min()
      .rename("launch_ts")
)

oi_for_label_construction = oi_for_label_construction.merge(launch,
on="product_id", how="left")

# 3) Helper to compute units sold in the first N days after launch
def units_in_window(days: int) -> pd.Series:
    mask = (
        (oi_for_label_construction["order_purchase_timestamp"] >=
oi_for_label_construction["launch_ts"]) &
        (oi_for_label_construction["order_purchase_timestamp"] <
oi_for_label_construction["launch_ts"] + pd.Timedelta(days=days))
    )
    return (
        oi_for_label_construction[mask]
        .groupby("product_id")["order_item_id"]
        .count()
        .rename(f"units{days}")
    )

# 4) Compute 7, 14, and 60-day unit counts per product
u7  = units_in_window(7)
u14 = units_in_window(14)
u60 = units_in_window(60)

# Combine into a single product-level table
product_sales = (
    pd.concat([u7, u14, u60], axis=1)
      .fillna(0)
      .reset_index()
      .rename(columns={"product_id": "product_id"})
)

product_sales["product_id"] = product_sales["product_id"].astype(str)

# Print a sanity check for product_sales
print("Product sales head after initial computation:")
```

```
display(product_sales.head())
print("Units7 min/max/mean:", product_sales['units7'].min(),
product_sales['units7'].max(), product_sales['units7'].mean())
```

```
Product sales head after initial computation:
```


```
{"summary":"{\n  \"name\": \"print(\\\"Units7 min/max/mean:\\\",
product_sales['units7']\",\n  \"rows\": 5,\n  \"fields\": [\n    {\n
\"column\": \"product_id\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 5,\n
\"samples\": [\n          \"00088930e925c41fd95ebfe695fd2655\",\n
\"000d9be29b5207b54e86aa1b1ac54872\",\n
\"0009406fd7479715e4bef61dd91f2462\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"units7\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 1,\n        \"max\": 1,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"units14\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 2,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"units60\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0,\n        \"min\": 1,\n        \"max\": 2,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          2\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


```
Units7 min/max/mean: 1 29 1.2489454037813723
```

```
# 5) Early momentum features: ratio of early units to 60-day units
product_sales["early_momentum_7"] = np.where(
    product_sales["units60"] > 0,
    product_sales["units7"] / product_sales["units60"],
    0.0,
)

product_sales["early_momentum_14"] = np.where(
    product_sales["units60"] > 0,
    product_sales["units14"] / product_sales["units60"],
    0.0,
)

# 6) MAIN LABEL: Best_Seller_60
#    Top 10% of products ranked by 60-day units
threshold_60 = product_sales["units60"].quantile(0.90)
product_sales["Best_Seller_60"] = (product_sales["units60"] >=
threshold_60).astype(int)
```

```
# 7- and 14-day "top 10%" labels:
threshold_7  = product_sales["units7"].quantile(0.90)
threshold_14 = product_sales["units14"].quantile(0.90)

product_sales["Best_Seller_7"]  = (product_sales["units7"]  >=
threshold_7).astype(int)
product_sales["Best_Seller_14"] = (product_sales["units14"] >=
threshold_14).astype(int)

# Ratio-based labels: 90% of 60-day units sold in first 7/14 days
product_sales["units_sold_7_days_ratio"] = np.where(
    product_sales["units60"] > 0,
    product_sales["units7"] / product_sales["units60"],
    0.0,
)
product_sales["units_sold_14_days_ratio"] = np.where(
    product_sales["units60"] > 0,
    product_sales["units14"] / product_sales["units60"],
    0.0,
)

product_sales["best_seller_label_7"] = (
    product_sales["units_sold_7_days_ratio"] >= 0.90
).astype(int)

product_sales["best_seller_label_14"] = (
    product_sales["units_sold_14_days_ratio"] >= 0.90
).astype(int)

# 7) Merge labels + early-sales features back into main df (order-item
level)
label_cols = [
    "product_id",
    "units7",
    "units14",
    "units60",
    "early_momentum_7",
    "early_momentum_14",
    "Best_Seller_60",
    "Best_Seller_7",
    "Best_Seller_14",
    "best_seller_label_7",
    "best_seller_label_14",
]

df = df.merge(product_sales[label_cols], on="product_id", how="left")

# 8) Quick sanity check
display(
```

```
    df[[
        "product_id",
        "units7", "units14", "units60",
        "early_momentum_7", "early_momentum_14",
        "Best_Seller_60", "Best_Seller_7", "Best_Seller_14",
        "best_seller_label_7", "best_seller_label_14",
    ]].head()
)

# products labeled positive under each definition
for col_name in ["Best_Seller_60", "Best_Seller_14", "Best_Seller_7",
                 "best_seller_label_7", "best_seller_label_14"]:
    if col_name in df.columns:
        n_pos = df[df[col_name] == 1]["product_id"].nunique()
        print(f"Number of products with {col_name} = 1: {n_pos}")
```


```
{"summary":"{\n  \"name\": \"        print(f\\\"Number of products
with {col_name} = 1: {n_pos}\\\")\",\n  \"rows\": 5,\n  \"fields\": [\
n    {\n      \"column\": \"product_id\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 5,\n
\"samples\": [\n          \"e5f2d52b802189ee658865ca93d83a8f\",\n
\"ac6c3623068f30de03045865e4e10089\",\n
\"c777355d18b72b67abbeef9df44fd0fd\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"units7\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 1,\n        \"max\": 1,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"units14\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 2,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"units60\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 1,\n        \"min\": 1,\n        \"max\": 4,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          4\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"early_momentum_7\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.3535533905932738,\n        \"min\": 0.25,\n        \"max\": 1.0,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          0.25\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"early_momentum_14\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.33541019662496846,\n        \"min\": 0.25,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Best_Seller_60\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

```
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Best_Seller_7\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Best_Seller_14\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"best_seller_label_7\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"best_seller_label_14\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


```
Number of products with Best_Seller_60 = 1: 5037
Number of products with Best_Seller_14 = 1: 6562
Number of products with Best_Seller_7 = 1: 5116
Number of products with best_seller_label_7 = 1: 25400
Number of products with best_seller_label_14 = 1: 26395
```

```
display(df[['product_id', 'best_seller_label_14',
'best_seller_label_7']])
```


```
{"type":"dataframe"}
```


```
product_df =
df.sort_values("product_id").drop_duplicates(subset=["product_id"]).co
py()

print("Number of unique products:",
product_df["product_id"].nunique())
```

```
Number of unique products: 32951
```

4. Feature Engineering & Train/Test Split

Clean up early-sales columns

```
# ============================================================
# CLEAN UP DUPLICATE EARLY-SALES COLUMNS
# ============================================================

print("\nQuick sanity check after cleanup:")
display(
    product_df[
        ["product_id", "Best_Seller_60", "units7", "units14",
"units60",
         "early_momentum_7", "early_momentum_14"]
    ].head()
)
```

```
Quick sanity check after cleanup:
```


```
{"summary":"{\n  \"name\": \")\",\n  \"rows\": 5,\n  \"fields\": [\n
{\n      \"column\": \"product_id\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 5,\n
\"samples\": [\n          \"00088930e925c41fd95ebfe695fd2655\",\n
\"000d9be29b5207b54e86aa1b1ac54872\",\n
\"0009406fd7479715e4bef61dd91f2462\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Best_Seller_60\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"units7\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 1,\n        \"max\": 1,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"units14\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 2,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"units60\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0,\n        \"min\": 1,\n        \"max\": 2,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          2\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"early_momentum_7\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.22360679774997896,\n        \"min\": 0.5,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"early_momentum_14\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0,\n        \"min\": 1.0,\n        \"max\": 1.0,\n
```

```
\"num_unique_values\": 1,\n        \"samples\": [\n          1.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```

Train Test Split:

```
from sklearn.model_selection import train_test_split

# ============================================================
# TRAIN/TEST SPLIT (PRODUCT-LEVEL)
# ============================================================

target_col = "Best_Seller_60"
product_df[target_col] = product_df[target_col].fillna(0).astype(int)

feature_cols = [c for c in product_df.columns if c not in
["product_id", target_col]]

X = product_df[feature_cols]
y = product_df[target_col]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print("Train size:", X_train.shape, " Test size:", X_test.shape)
print("Positive rate in train:", y_train.mean(), " Positive rate in
test:", y_test.mean())
```

```
Train size: (26360, 30)  Test size: (6591, 30)
Positive rate in train: 0.152845220030349  Positive rate in test:
0.15293582157487484
```

Define feature lists

```
# ============================================================
# FEATURE LISTS
# ============================================================

numeric_features = [
    "units7",
    "units14",
    "price",
    "freight_value",
    "delivery_time_days",
    "item_revenue",
```

```
    "freight_ratio",
    "product_name_lenght",
    "product_description_lenght",
    "product_photos_qty",
    "product_weight_g",
    "product_length_cm",
    "product_height_cm",
    "product_width_cm",
]

categorical_features = [
    "product_category_name",
    "seller_id",
    "order_status",
]

# Filter to keep only columns that actually exist in X_train
numeric_features = [c for c in numeric_features if c in
X_train.columns]
categorical_features = [c for c in categorical_features if c in
X_train.columns]

print("Numeric features used:", numeric_features)
print("Categorical features used:", categorical_features)
```

```
Numeric features used: ['units7', 'units14', 'price', 'freight_value',
'delivery_time_days', 'freight_ratio']
Categorical features used: ['product_category_name', 'seller_id',
'order_status']
```

5. Modeling (5 models + tuning)

Preprocessing pipeline

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer

# ============================================================
# PREPROCESSING PIPELINE
# ============================================================

numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler()),
])
```

```
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore")),
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ]
)

# Fit on train only (sanity check)
preprocess.fit(X_train)
```

```
ColumnTransformer(transformers=[('num',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),
                                                 ('scaler',
StandardScaler())]),
                                 ['units7', 'units14', 'price',
'freight_value',
                                  'delivery_time_days',
'freight_ratio']),
                                ('cat',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),
                                                 ('onehot',

OneHotEncoder(handle_unknown='ignore'))]),
                                 ['product_category_name',
'seller_id',
                                  'order_status'])])
```

Define 5 baseline models & pipelines

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier

# ============================================================
# BASELINE MODELS
# ============================================================

# Class imbalance ratio for XGBoost (neg/pos)
pos_rate = y_train.mean()
```

```
neg_rate = 1 - pos_rate
scale_pos_weight = neg_rate / pos_rate

models = {
    "log_reg": LogisticRegression(
        max_iter=1000,
        class_weight="balanced",
    ),
    "decision_tree": DecisionTreeClassifier(
        max_depth=None,
        random_state=42,
        class_weight="balanced",
    ),
    "random_forest": RandomForestClassifier(
        n_estimators=200,
        random_state=42,
        n_jobs=-1,
        class_weight="balanced",
    ),
    "svm_rbf": SVC(
        kernel="rbf",
        C=1.0,
        gamma="scale",
        probability=True,
        class_weight="balanced",
        random_state=42,
    ),
    "xgboost": XGBClassifier(
        objective="binary:logistic",
        eval_metric="logloss",
        use_label_encoder=False,
        tree_method="hist",
        scale_pos_weight=scale_pos_weight,
        random_state=42,
        n_estimators=200,
        learning_rate=0.1,
        max_depth=6,
        subsample=0.8,
        colsample_bytree=0.8,
    ),
}

from sklearn.pipeline import Pipeline

pipelines = {
    name: Pipeline(steps=[
        ("preprocess", preprocess),
        ("model", clf),
    ])
```

```
    for name, clf in models.items()
}

from sklearn.model_selection import StratifiedKFold, GridSearchCV

# ============================================================
# CV SETUP & PARAM GRIDS
# ============================================================

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

param_grids = {
    "log_reg": {
        "model__C": [0.01, 0.1, 1.0, 10.0],
        "model__solver": ["liblinear", "lbfgs"],
    },
    "decision_tree": {
        "model__max_depth": [None, 5, 10, 20],
        "model__min_samples_leaf": [1, 5, 10],
        "model__min_samples_split": [2, 10, 20],
    },
    "random_forest": {
        "model__n_estimators": [200, 400],
        "model__max_depth": [None, 10, 20],
        "model__min_samples_leaf": [1, 5, 10],
        "model__max_features": ["sqrt", "log2"],
    },
    "svm_rbf": {
        "model__C": [0.1, 1.0, 10.0],
        "model__gamma": ["scale", "auto"],
    },
    "xgboost": {
        "model__n_estimators": [200, 400],
        "model__max_depth": [4, 6, 8],
        "model__learning_rate": [0.05, 0.1],
        "model__subsample": [0.8, 1.0],
        "model__colsample_bytree": [0.8, 1.0],
    },
}

from sklearn.model_selection import cross_val_score
import pandas as pd

# ============================================================
# GRID SEARCH FOR ALL MODELS
# ============================================================

best_models = {}
cv_summaries = []
```

```
for name, pipe in pipelines.items():
    print(f"\n=== Tuning model: {name} ===")
    param_grid = param_grids[name]

    grid = GridSearchCV(
        estimator=pipe,
        param_grid=param_grid,
        cv=cv,
        scoring="balanced_accuracy",
        n_jobs=-1,
        verbose=1,
    )

    grid.fit(X_train, y_train)

    best_est = grid.best_estimator_
    best_models[name] = best_est

    print(f"Best params for {name}: {grid.best_params_}")
    print(f"Best CV balanced accuracy (GridSearch scoring):
{grid.best_score_:.4f}")

    bal_acc_scores = cross_val_score(
        best_est, X_train, y_train,
        cv=cv,
        scoring="balanced_accuracy",
        n_jobs=-1,
    )
    roc_scores = cross_val_score(
        best_est, X_train, y_train,
        cv=cv,
        scoring="roc_auc",
        n_jobs=-1,
    )

    cv_summaries.append({
        "model": name,
        "best_params": grid.best_params_,
        "cv_bal_acc_mean": bal_acc_scores.mean(),
        "cv_bal_acc_std": bal_acc_scores.std(),
        "cv_roc_auc_mean": roc_scores.mean(),
        "cv_roc_auc_std": roc_scores.std(),
    })

cv_results_df =
pd.DataFrame(cv_summaries).sort_values("cv_bal_acc_mean",
ascending=False)
cv_results_df
```

```
=== Tuning model: log_reg ===
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best params for log_reg: {'model__C': 0.1, 'model__solver': 'lbfgs'}
Best CV balanced accuracy (GridSearch scoring): 0.7962

=== Tuning model: decision_tree ===
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best params for decision_tree: {'model__max_depth': 5,
'model__min_samples_leaf': 10, 'model__min_samples_split': 2}
Best CV balanced accuracy (GridSearch scoring): 0.7952

=== Tuning model: random_forest ===
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best params for random_forest: {'model__max_depth': None,
'model__max_features': 'sqrt', 'model__min_samples_leaf': 5,
'model__n_estimators': 400}
Best CV balanced accuracy (GridSearch scoring): 0.7961

=== Tuning model: svm_rbf ===
Fitting 5 folds for each of 6 candidates, totalling 30 fits

/usr/local/lib/python3.12/dist-packages/joblib/externals/loky/
process_executor.py:782: UserWarning: A worker stopped while some jobs
were given to the executor. This can be caused by a too short worker
timeout or by a memory leak.
  warnings.warn(

Best params for svm_rbf: {'model__C': 10.0, 'model__gamma': 'auto'}
Best CV balanced accuracy (GridSearch scoring): 0.7959

=== Tuning model: xgboost ===
Fitting 5 folds for each of 48 candidates, totalling 240 fits

/usr/local/lib/python3.12/dist-packages/joblib/externals/loky/
process_executor.py:782: UserWarning: A worker stopped while some jobs
were given to the executor. This can be caused by a too short worker
timeout or by a memory leak.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199:
UserWarning: [00:29:58] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)

Best params for xgboost: {'model__colsample_bytree': 0.8,
'model__learning_rate': 0.1, 'model__max_depth': 6,
'model__n_estimators': 400, 'model__subsample': 1.0}
Best CV balanced accuracy (GridSearch scoring): 0.7961
```

{"summary":"{\n \"name\": \"cv_results_df\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"model\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"xgboost\",\n   \"decision_tree\",\n
\"random_forest\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"best_params\",\n   \"properties\": {\n   \"dtype\":
\"object\",\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"cv_bal_acc_mean\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.0004102409829573275,\n   \"min\":
0.7951975088735881,\n   \"max\": 0.7961787328775881,\n
\"num_unique_values\": 5,\n   \"samples\": [\n
0.7961331859794774,\n   0.7951975088735881,\n
0.7961180560196628\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"cv_bal_acc_std\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.00035680133444326753,\n
\"min\": 0.00654746925515235,\n   \"max\": 0.007462192850651754,\
n   \"num_unique_values\": 5,\n   \"samples\": [\n
0.006796729572763691,\n   0.00654746925515235,\n
0.0070402965540029205\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"cv_roc_auc_mean\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.011164713050136228,\n   \"min\":
0.8330569097878463,\n   \"max\": 0.8613102689949919,\n
\"num_unique_values\": 5,\n   \"samples\": [\n
0.8563154364351971,\n   0.8330569097878463,\n
0.8435039475772614\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"cv_roc_auc_std\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.0015161751564632651,\n   \"min\":
0.005882683792517709,\n   \"max\": 0.009819840645355416,\n
\"num_unique_values\": 5,\n   \"samples\": [\n
0.007938346899695976,\n   0.005882683792517709,\n
0.009819840645355416\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"cv_results_df"}

```
from sklearn.metrics import (roc_auc_score, average_precision_score,
accuracy_score, balanced_accuracy_score, f1_score, precision_score,
recall_score, confusion_matrix)

# ============================================================
# TEST-SET EVALUATION FOR ALL TUNED MODELS
# ============================================================

test_summaries = []
```

```python
for name, model in best_models.items():
    print(f"\n=== Evaluating on test set: {name} ===")

    model.fit(X_train, y_train)

    y_proba_test = model.predict_proba(X_test)[:, 1]
    y_pred_test = (y_proba_test >= 0.5).astype(int)

    roc = roc_auc_score(y_test, y_proba_test)
    pr = average_precision_score(y_test, y_proba_test)
    acc = accuracy_score(y_test, y_pred_test)
    bal_acc = balanced_accuracy_score(y_test, y_pred_test)
    f1 = f1_score(y_test, y_pred_test)
    prec = precision_score(y_test, y_pred_test, zero_division=0)
    rec = recall_score(y_test, y_pred_test)

    tn, fp, fn, tp = confusion_matrix(y_test, y_pred_test).ravel()

    test_summaries.append({
        "model": name,
        "test_roc_auc": roc,
        "test_pr_auc": pr,
        "test_accuracy": acc,
        "test_balanced_accuracy": bal_acc,
        "test_f1": f1,
        "test_precision": prec,
        "test_recall": rec,
        "tn": tn,
        "fp": fp,
        "fn": fn,
        "tp": tp,
    })

test_results_df =
pd.DataFrame(test_summaries).sort_values("test_pr_auc",
ascending=False)
test_results_df
```

```
=== Evaluating on test set: log_reg ===

=== Evaluating on test set: decision_tree ===

=== Evaluating on test set: random_forest ===

=== Evaluating on test set: svm_rbf ===

=== Evaluating on test set: xgboost ===

/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199:
UserWarning: [00:33:53] WARNING: /workspace/src/learner.cc:790:
```

```
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```


```
{"summary":"{\n  \"name\": \"test_results_df\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"model\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"log_reg\",\n          \"random_forest\",\n          \"svm_rbf\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"test_roc_auc\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.010907907922283558,\n        \"min\": 0.8470275588592354,\n
\"max\": 0.8737591654370268,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.8737591654370268,\n
0.8556571607686599,\n          0.862131250195463\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"test_pr_auc\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.046279035464762384,\n        \"min\": 0.646987460871393,\n
\"max\": 0.7500242637199841,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.7499428635776009,\n
0.646987460871393,\n          0.739731063069227\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"test_accuracy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.022139984537123902,\n        \"min\": 0.8631467152177211,\n
\"max\": 0.9159459869519041,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.8670914883932636,\n
0.8631467152177211,\n          0.9159459869519041\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"test_balanced_accuracy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.037399306073568055,\n        \"min\": 0.7251984126984127,\n
\"max\": 0.8107866780959204,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.8089542126182373,\n
0.810690456999565,\n          0.7251984126984127\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"test_f1\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.004767458714551645,\n        \"min\": 0.6201550387596899,\n
\"max\": 0.6317611423626136,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.6253207869974337,\n
0.6216442953020134,\n          0.6210670314637483\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"test_precision\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.20180354386620744,\n        \"min\": 0.5385174418604651,\n
\"max\": 1.0,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          0.5496240601503759,\n          0.5385174418604651,\n
```

```
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"test_recall\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.12289460548655971,\n        \"min\":
0.4503968253968254,\n        \"max\": 0.7351190476190477,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.7251984126984127,\n          0.7351190476190477,\n
0.4503968253968254\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"tn\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 269,\n        \"min\": 4948,\n        \"max\": 5583,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          4984,\n
4948,\n          5583\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"fp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 269,\n        \"min\": 0,\n        \"max\": 635,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          599,\n
635,\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"fn\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 123,\n        \"min\": 267,\n        \"max\": 554,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          277,\n
267,\n          554\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"tp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 123,\n        \"min\": 454,\n        \"max\": 741,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          731,\n
741,\n          454\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"test_results_df"}
```


```
#
==========================================================================
======
# MINIMAL SEGMENTATION CODE (QUANTILE-BASED)
#
==========================================================================
=====

# Calculate momentum ratio
product_df['momentum_ratio'] = np.where(
    product_df['units60'] > 0,
    product_df['units7'] / product_df['units60'],
    0.0
)

# Use quantile-based thresholds (adapts to your data)
# Make sure units7 and units60 are numeric for quantile calculation
product_df['units7'] = pd.to_numeric(product_df['units7'],
errors='coerce').fillna(0)
```

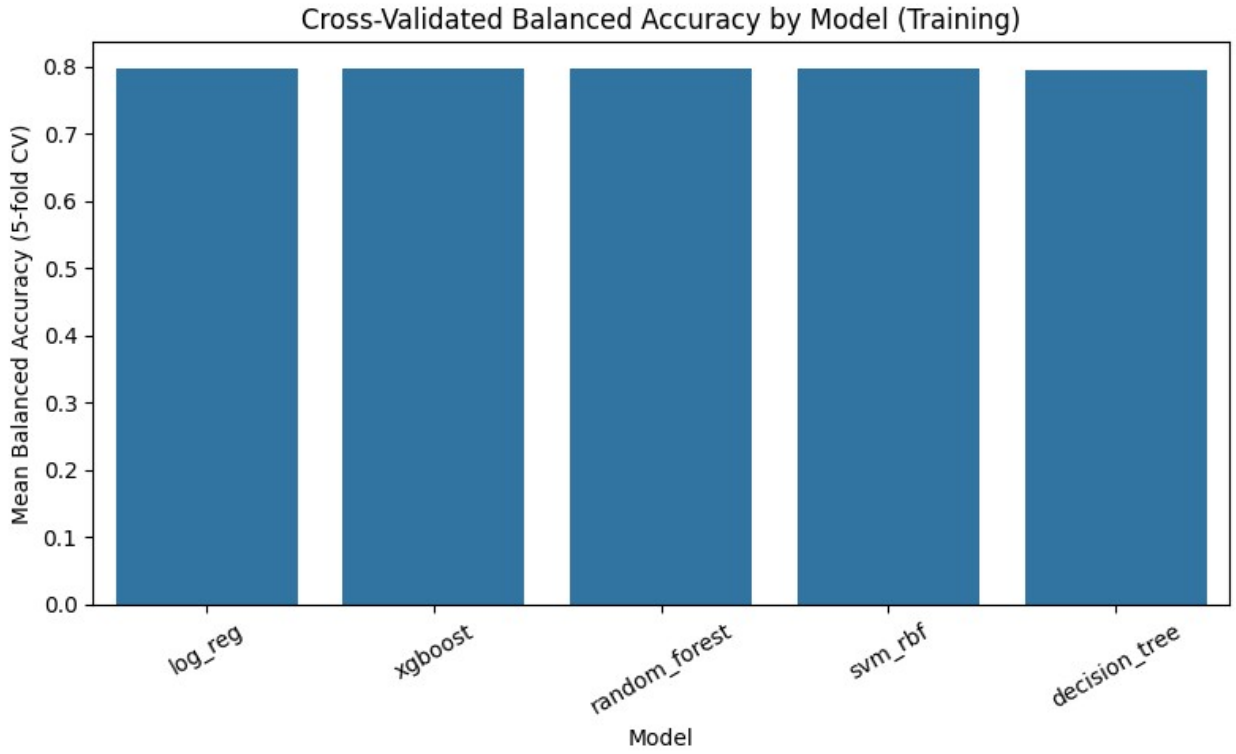
```
product_df['units60'] = pd.to_numeric(product_df['units60'],
errors='coerce').fillna(0)

# Recalculate momentum_ratio after ensuring units are numeric
product_df['momentum_ratio'] = np.where(
    product_df['units60'] > 0,
    product_df['units7'] / product_df['units60'],
    0.0
)

# Only calculate quantiles if there's actual variation in
momentum_ratio
if product_df['momentum_ratio'].nunique() > 1:
    q75 = product_df['momentum_ratio'].quantile(0.75)
    q50 = product_df['momentum_ratio'].quantile(0.50)
else:
    # If all momentum_ratios are the same use default thresholds
    q75 = 0.75
    q50 = 0.35

print(f"\nThresholds based on your data:")
print(f"75th percentile: {q75:.3f}")
print(f"50th percentile: {q50:.3f}")

def segment(ratio):
    if ratio >= q75:
        return 'TRENDING'
    elif ratio >= q50:
        return 'BALANCED'
    else:
        return 'LATE_BLOOMER'

product_df['segment'] = product_df['momentum_ratio'].apply(segment)

# Verify distribution
print("\nSEGMENT DISTRIBUTION:")
print(product_df['segment'].value_counts())

# Print stats
print("\nSEGMENT SUMMARY:")
print(product_df.groupby('segment')[['units7', 'units60',
'Best_Seller_60']].agg({
    'units7': 'mean',
    'units60': 'mean',
    'Best_Seller_60': 'sum'
}).round(1))

# Get predictions and merge with segments
test_product_ids_from_product_df = product_df.loc[X_test.index,
'product_id']
```

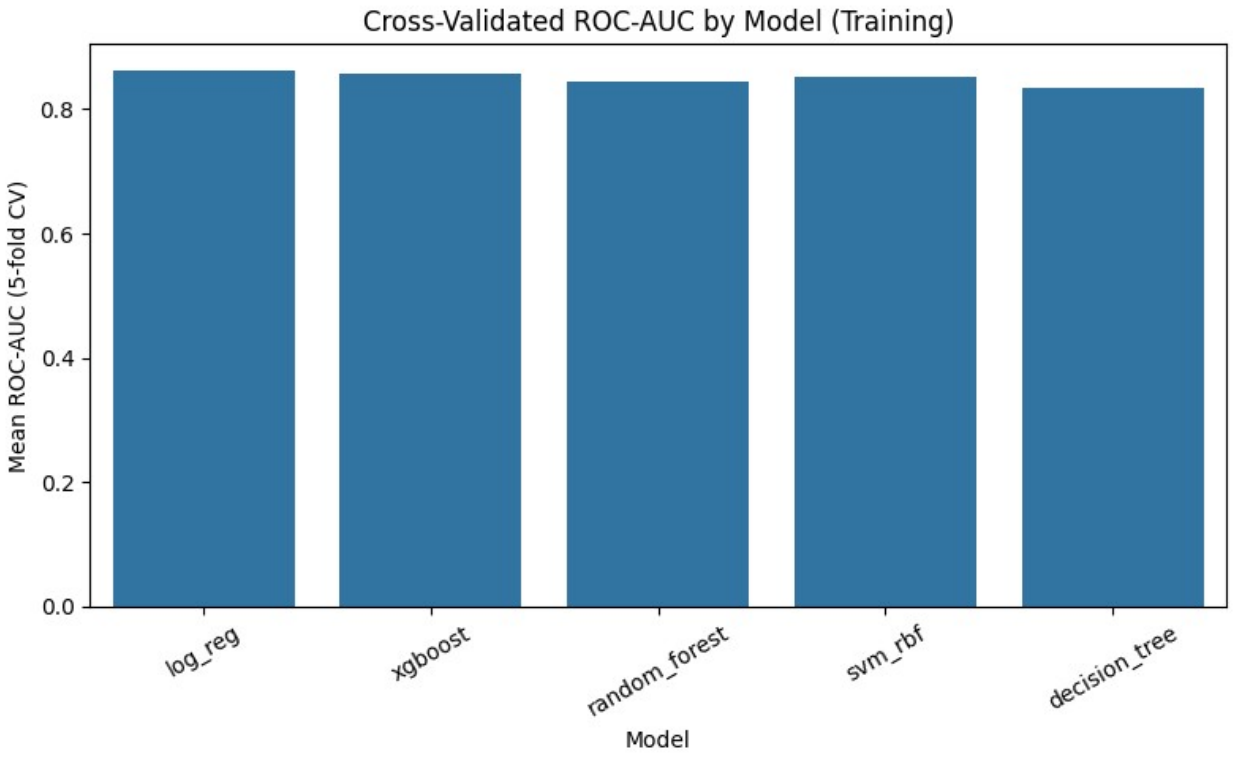
```
pred_df = pd.DataFrame({
    'product_id': test_product_ids_from_product_df.values,
    'pred_prob': model.predict_proba(X_test)[:, 1],
    'actual': y_test.values
})

# Ensure product_id is string type in pred_df before merging
pred_df['product_id'] = pred_df['product_id'].astype(str)

# Merge with product_df using the correct product_id column
pred_df = pred_df.merge(
    product_df[['product_id', 'units7', 'units60', 'momentum_ratio',
'segment']],
    on='product_id',
    how='left'
)

# Show top 3 products per segment
print("\nTOP PRODUCTS BY SEGMENT:")
for seg in ['TRENDING', 'BALANCED', 'LATE_BLOOMER']:
    segment_products = pred_df[pred_df['segment'] == seg]
    if not segment_products.empty:
        top = segment_products.nlargest(3, 'pred_prob')
        print(f"\n{seg} ({len(segment_products)} products):")
        for _, row in top.iterrows():
            print(f"  {row['product_id'][:8]}... | Prob:
{row['pred_prob']:.2f} | Ratio: {row['momentum_ratio']:.3f}")
    else:
        print(f"\n{seg} (0 products): No products in this segment.")

# Export
pred_df.to_csv('/content/segmented_predictions.csv', index=False)
print("\n✓ Saved to segmented_predictions.csv")
```

6. Visualization

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, precision_recall_curve

# CV RESULTS — BALANCED ACCURACY + ROC-AUC

cv_plot_df = cv_results_df.sort_values("cv_bal_acc_mean",
ascending=False).copy()

plt.figure(figsize=(8,5))
sns.barplot(
    data=cv_plot_df,
    x="model",
```

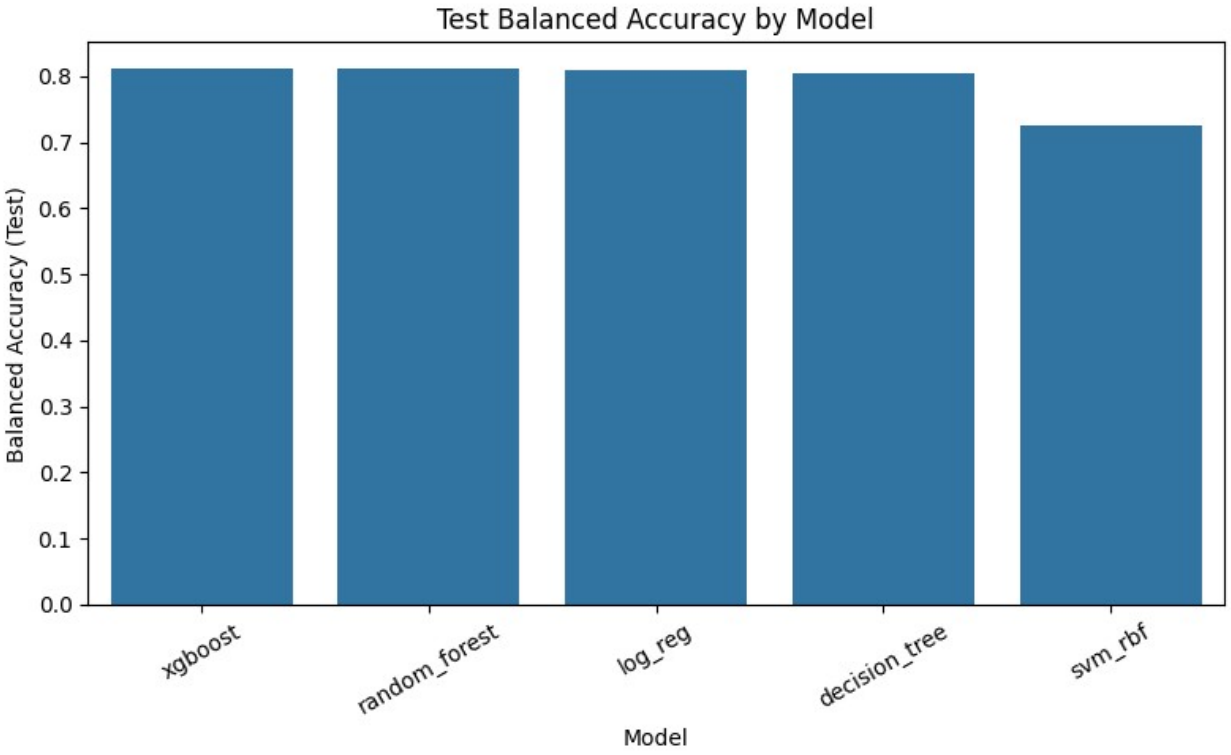
```
    y="cv_bal_acc_mean"
)
plt.title("Cross-Validated Balanced Accuracy by Model (Training)")
plt.ylabel("Mean Balanced Accuracy (5-fold CV)")
plt.xlabel("Model")
plt.xticks(rotation=30)
plt.tight_layout()
plt.show()

plt.figure(figsize=(8,5))
sns.barplot(
    data=cv_plot_df,
    x="model",
    y="cv_roc_auc_mean"
)
plt.title("Cross-Validated ROC-AUC by Model (Training)")
plt.ylabel("Mean ROC-AUC (5-fold CV)")
plt.xlabel("Model")
plt.xticks(rotation=30)
plt.tight_layout()
plt.show()
```

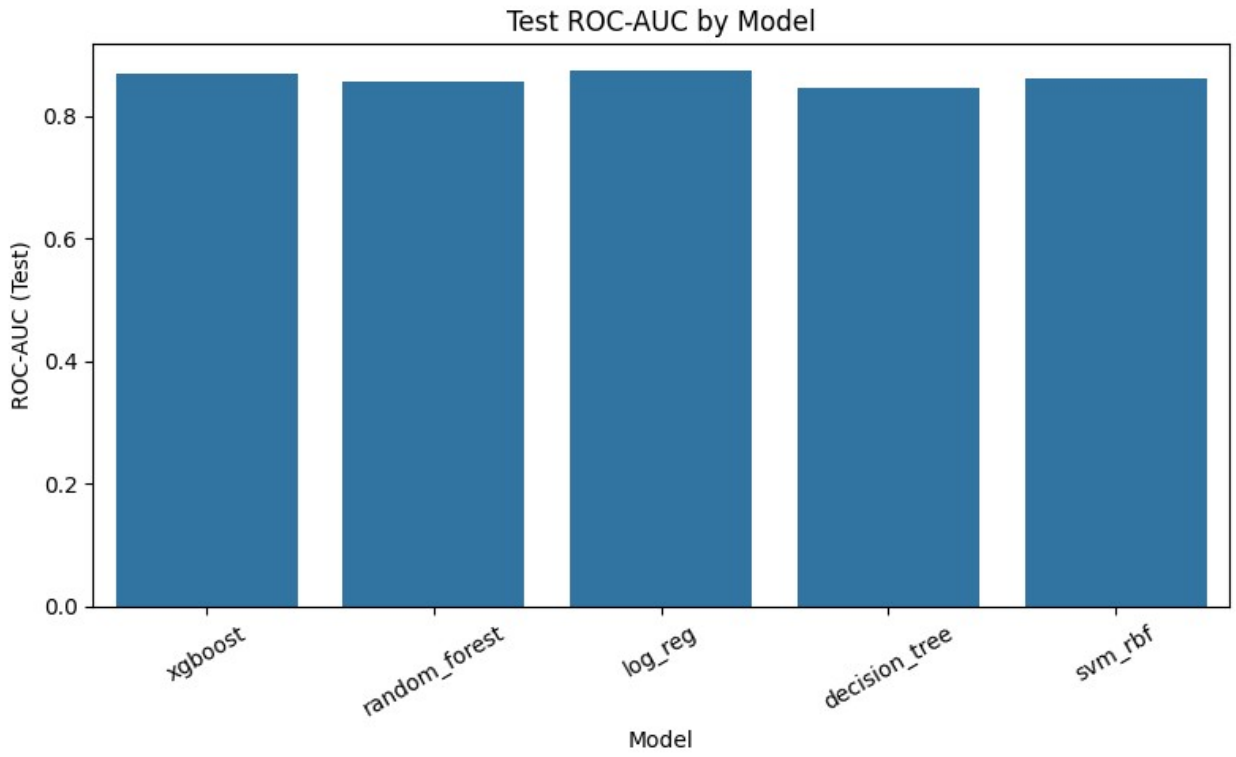





```
# 2) TEST RESULTS — BALANCED ACCURACY + ROC-AUC

test_plot_df = test_results_df.sort_values("test_balanced_accuracy",
ascending=False).copy()

plt.figure(figsize=(8,5))
sns.barplot(
    data=test_plot_df,
    x="model",
    y="test_balanced_accuracy"
)
plt.title("Test Balanced Accuracy by Model")
plt.ylabel("Balanced Accuracy (Test)")
plt.xlabel("Model")
plt.xticks(rotation=30)
plt.tight_layout()
plt.show()

plt.figure(figsize=(8,5))
sns.barplot(
    data=test_plot_df,
    x="model",
    y="test_roc_auc"
)
plt.title("Test ROC-AUC by Model")
plt.ylabel("ROC-AUC (Test)")
```

```
plt.xlabel("Model")
plt.xticks(rotation=30)
plt.tight_layout()
plt.show()
```



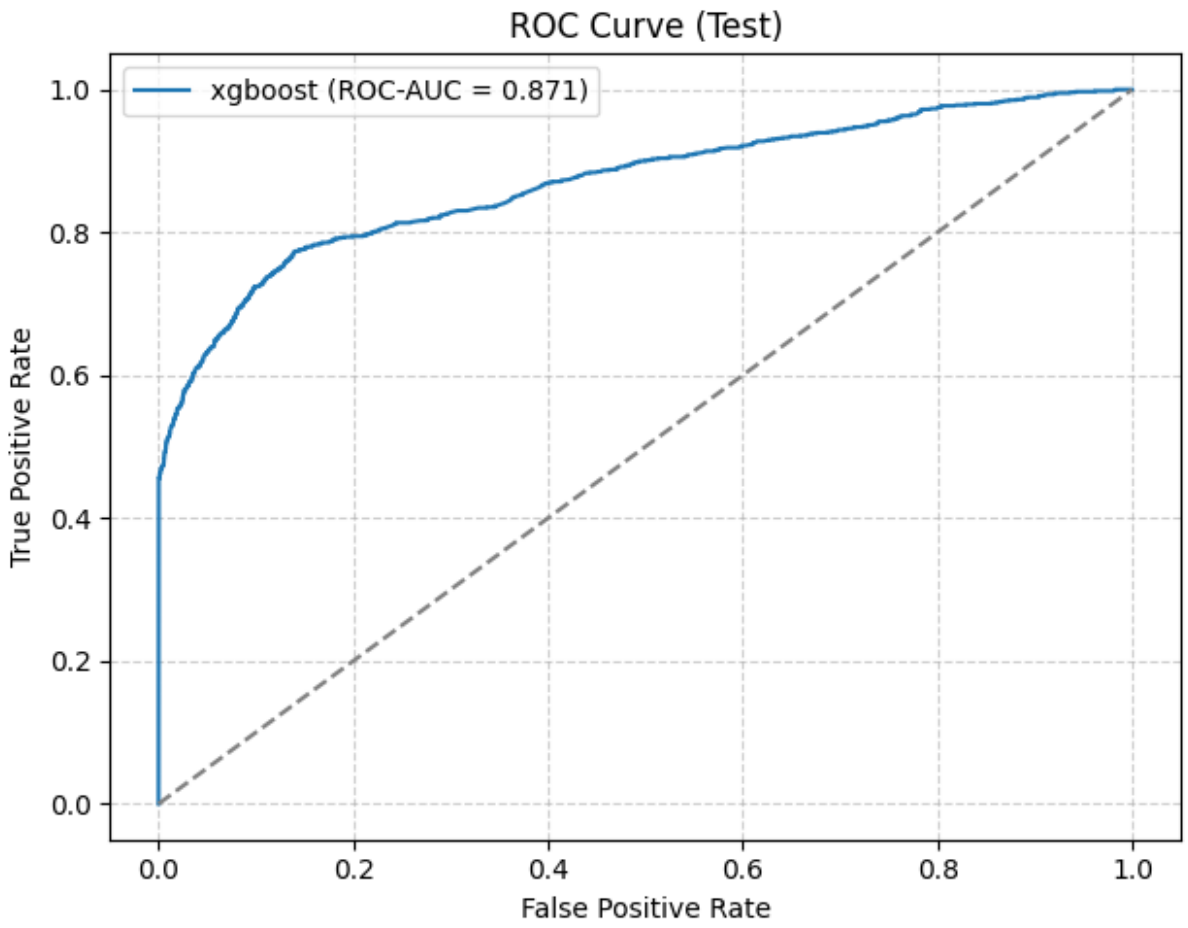



```
# 3) PICK BEST MODEL BY TEST BALANCED ACCURACY

best_model_name = test_results_df.sort_values(
    "test_balanced_accuracy", ascending=False
).iloc[0]["model"]
best_model = best_models[best_model_name]

print("Best model on test (by balanced accuracy):", best_model_name)

# Fit on full training data
best_model.fit(X_train, y_train)

# Get probabilities on test
y_proba_best = best_model.predict_proba(X_test)[:, 1]

# ROC curve
fpr, tpr, roc_thresh = roc_curve(y_test, y_proba_best)

# PR curve (kept as a secondary diagnostic)
precisions, recalls, pr_thresh = precision_recall_curve(y_test,
y_proba_best)

from sklearn.metrics import roc_auc_score, average_precision_score

plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
```

```
plt.plot(
    fpr, tpr,
    label=f"{best_model_name} (ROC-AUC = {roc_auc_score(y_test,
y_proba_best):.3f})"
)
plt.plot([0,1],[0,1],"--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve (Test)")
plt.legend()
plt.grid(alpha=0.6, linestyle="--")

plt.tight_layout()
plt.show()
```

```
Best model on test (by balanced accuracy): xgboost
```

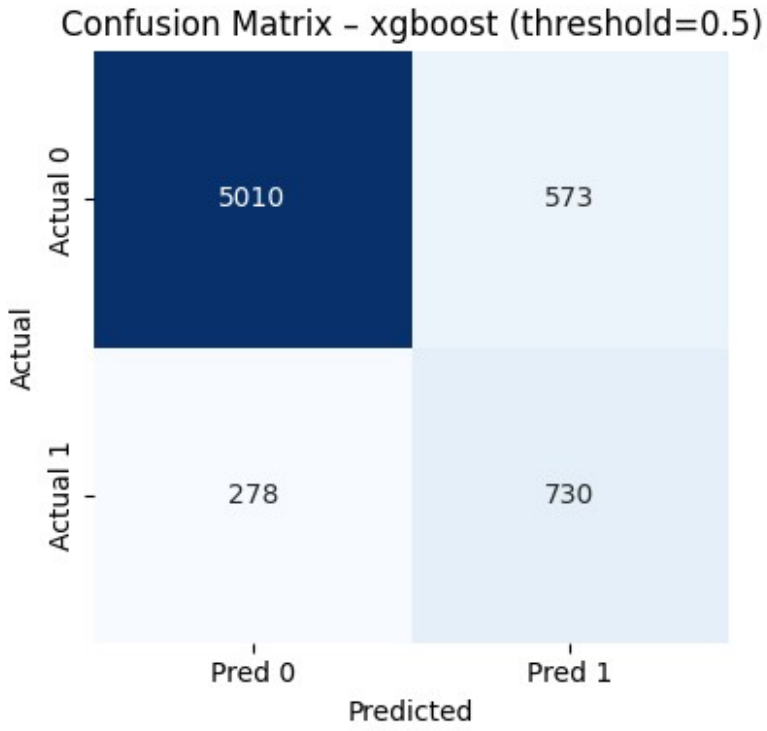
```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199:
UserWarning: [00:33:55] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```




```
# 4) CONFUSION MATRIX AT A CHOSEN THRESHOLD

from sklearn.metrics import confusion_matrix, precision_score,
recall_score, accuracy_score, balanced_accuracy_score

threshold = 0.5
y_pred_best = (y_proba_best >= threshold).astype(int)

cm = confusion_matrix(y_test, y_pred_best)
tn, fp, fn, tp = cm.ravel()

print(f"Confusion matrix for {best_model_name} at
threshold={threshold}:")
print(cm)
print(f"TN={tn}, FP={fp}, FN={fn}, TP={tp}")
print(f"Accuracy:          {accuracy_score(y_test, y_pred_best):.3f}")
print(f"Balanced accuracy: {balanced_accuracy_score(y_test,
y_pred_best):.3f}")
print(f"Precision:         {precision_score(y_test, y_pred_best,
zero_division=0):.3f}")
print(f"Recall:            {recall_score(y_test, y_pred_best):.3f}")
```

```
plt.figure(figsize=(4,4))
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    cbar=False,
    xticklabels=["Pred 0", "Pred 1"],
    yticklabels=["Actual 0", "Actual 1"],
)
plt.title(f"Confusion Matrix – {best_model_name}
(threshold={threshold})")
plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.tight_layout()
plt.show()
```

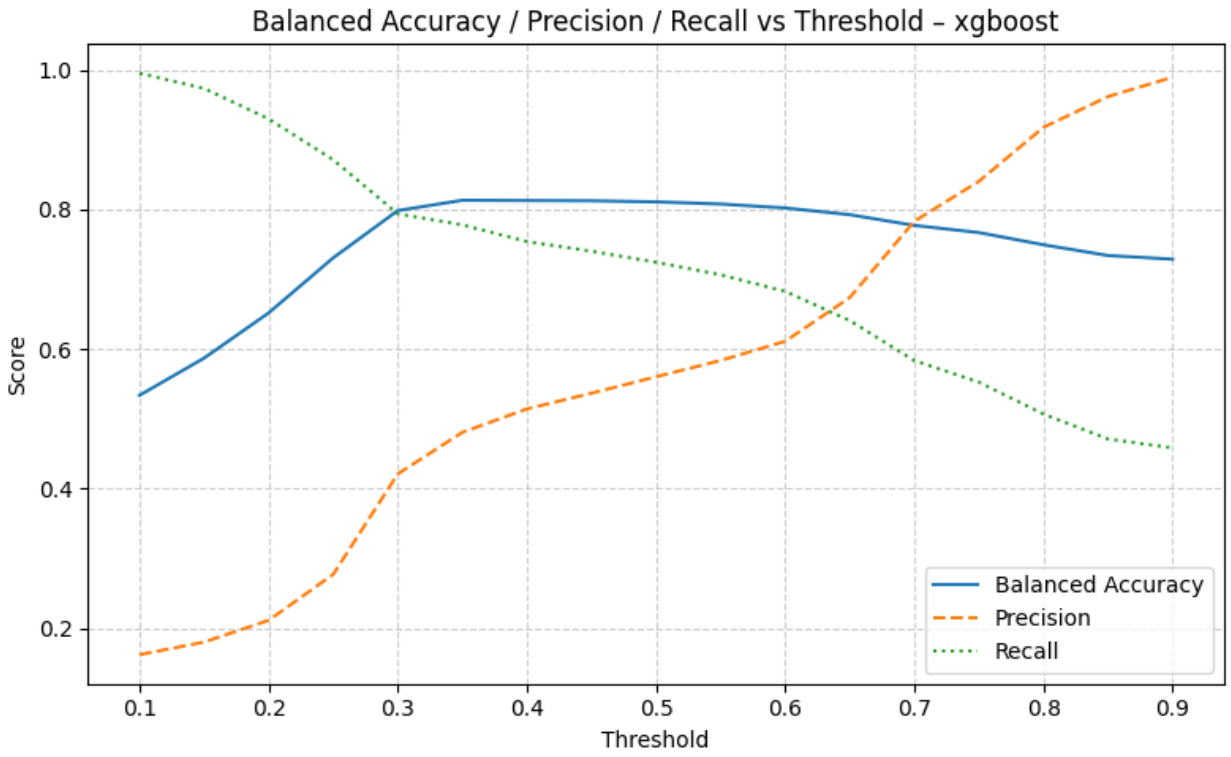
```
Confusion matrix for xgboost at threshold=0.5:
[[5010  573]
 [ 278  730]]
TN=5010, FP=573, FN=278, TP=730
Accuracy:          0.871
Balanced accuracy: 0.811
Precision:         0.560
Recall:            0.724
```



```
# 5) THRESHOLD SWEEP

thresholds = np.linspace(0.1, 0.9, 17)

rows = []
for t in thresholds:
    y_pred_t = (y_proba_best >= t).astype(int)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred_t).ravel()
    rows.append({
        "threshold": t,
        "precision": precision_score(y_test, y_pred_t,
zero_division=0),
        "recall": recall_score(y_test, y_pred_t),
        "accuracy": accuracy_score(y_test, y_pred_t),
        "balanced_accuracy": balanced_accuracy_score(y_test,
y_pred_t),
        "tp": tp,
        "fp": fp,
        "fn": fn,
        "tn": tn,
    })

thr_df = pd.DataFrame(rows)
display(thr_df)

plt.figure(figsize=(8,5))
plt.plot(thr_df["threshold"], thr_df["balanced_accuracy"],
label="Balanced Accuracy")
plt.plot(thr_df["threshold"], thr_df["precision"], label="Precision",
linestyle="--")
plt.plot(thr_df["threshold"], thr_df["recall"], label="Recall",
linestyle=":")
plt.xlabel("Threshold")
plt.ylabel("Score")
plt.title(f"Balanced Accuracy / Precision / Recall vs Threshold —
{best_model_name}")
plt.legend()
plt.grid(alpha=0.6, linestyle="--")
plt.tight_layout()
plt.show()
```


```
{"summary":"{\n  \"name\": \"thr_df\",\n  \"rows\": 17,\n  \"fields\":
[\n    {\n      \"column\": \"threshold\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.25248762345905196,\n
\"min\": 0.1,\n        \"max\": 0.9,\n        \"num_unique_values\":
17,\n        \"samples\": [\n          0.1,\n
0.15000000000000002,\n          0.35\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"precision\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

```
0.2678215857486136,\n        \"min\": 0.16224522808152703,\n
\"max\": 0.9892933618843683,\n        \"num_unique_values\": 17,\n
\"samples\": [\n          0.16224522808152703,\n
0.18023148998713945,\n          0.48068669527896996\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"recall\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.1671014083377848,\n        \"min\": 0.4583333333333333,\n
\"max\": 0.9950396825396826,\n        \"num_unique_values\": 17,\n
\"samples\": [\n          0.9950396825396826,\n
0.9732142857142857,\n          0.7777777777777778\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"accuracy\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.2245841002976137,\n        \"min\": 0.21347291761492945,\n
\"max\": 0.9176149294492489,\n        \"num_unique_values\": 17,\n
\"samples\": [\n          0.21347291761492945,\n
0.31891973903808224,\n          0.8375056895766955\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"balanced_accuracy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.08316897301651181,\n        \"min\": 0.5337011058229489,\n
\"max\": 0.8130336139471013,\n        \"num_unique_values\": 17,\n
\"samples\": [\n          0.5337011058229489,\n
0.5870011962334638,\n          0.8130336139471013\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"tp\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 168,\n        \"min\": 462,\n
\"max\": 1003,\n        \"num_unique_values\": 17,\n
\"samples\": [\n          1003,\n          981,\n          784\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"fp\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1625,\n        \"min\": 5,\n
\"max\": 5179,\n        \"num_unique_values\": 17,\n
\"samples\": [\n          5179,\n          4462,\n          847\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"fn\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 168,\n        \"min\": 5,\n
\"max\": 546,\n        \"num_unique_values\": 17,\n
\"samples\": [\n          5,\n          27,\n          224\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"tn\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1625,\n        \"min\": 404,\n        \"max\": 5578,\n
\"num_unique_values\": 17,\n        \"samples\": [\n          404,\n
1121,\n          4736\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"thr_df"}
```



---

7. Final Results & Interpretation

Evaluation Metric

Because only ~15% of products are best-sellers, plain accuracy would be misleading.
We therefore use **balanced accuracy** as our main evaluation metric:

Balanced Accuracy = (TPR + TNR)/ 2

This treats the positive (best-seller) and negative classes symmetrically and is robust to class imbalance.

Our best model (XGBoost) achieves:

- **Balanced accuracy ≈ 0.81**
- ROC-AUC ≈ 0.87
- Precision ≈ 0.55, Recall ≈ 0.74

We focus on **balanced accuracy** for model comparison, and interpret **recall** as the key business metric (fraction of true best-sellers we successfully flag).

Gen AI Disclosure

We used ChatGPT as a learning aid to clarify concepts related to:

- Designing a product-level prediction model
- Understanding balanced accuracy, ROC-AUC, and class imbalance
- Structuring the modeling pipeline (feature engineering → train/test split → tuning → evaluation)
- Interpreting segmentation logic (Trending vs. Balanced vs. Late-Bloomer products)
- Fixing code issues related to merging data, feature lists, preprocessing pipelines, and GridSearchCV

Prompts Used:

- "Explain whether PR-AUC or balanced accuracy is better for an imbalanced classification problem."
- "Help me debug a KeyError in my preprocessing pipeline where a feature is missing from X_train."
- "Suggest a slide structure that follows this project rubric."
- "Should I apply stacking or voting ensembles for my current model performance?"
- "Explain how to interpret a threshold sweep (precision, recall, balanced accuracy)."