olist-customer-ltv-prediction

November 17, 2025

1 Project Overview

Title: Customer Lifetime Value Prediction for an Online Retail Store Objective: Predict each customer's future revenue (LTV) based on past purchase behavior (Recency, Frequency, Monetary, AOV, Tenure etc.) and use it for targeted marketing & segmentation. Tools: Python, Pandas, Scikit-Learn, XGBoost, Matplotlib/Seaborn, Excel/CSV outputs.

[1]:
```
# Imports & Config
import pandas as pd
import numpy as np

base_path = r"D:\intern\project"

customers = pd.read_csv(base_path + r"\olist_customers_dataset.csv")
orders    = pd.read_csv(base_path + r"\olist_orders_dataset.csv")
items     = pd.read_csv(base_path + r"\olist_order_items_dataset.csv")
payments  = pd.read_csv(base_path + r"\olist_order_payments_dataset.csv")

customers.head(), orders.head(), items.head(), payments.head()
```

[1]:
```
(                        customer_id                customer_unique_id  \
 0  06b8999e2fba1a1fbc88172c00ba8bc7  861eff4711a542e4b93843c6dd7febb0
 1  18955e83d337fd6b2def6b18a428ac77  290c77bc529b7ac935b93aa66c333dc3
 2  4e7b3e00288586ebd08712fdd0374a03  060e732b5b29e8181a18229c7b0b2b5e
 3  b2b6027bc5c5109e529d4dc6358b12c3  259dac757896d24d7702b9acbbff3f3c
 4  4f2d8ab171c80ec8364f7c12e35b23ad  345ecd01c38d18a9036ed96c73b8d066

    customer_zip_code_prefix          customer_city customer_state
 0                     14409                 franca             SP
 1                      9790  sao bernardo do campo             SP
 2                      1151              sao paulo             SP
 3                      8775        mogi das cruzes             SP
 4                     13056               campinas             SP  ,
                           order_id                        customer_id  \
 0  e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
 1  53cdb2fc8bc7dce0b6741e2150273451  b0830fb4747a6c6d20dea0b8c802d7ef
 2  47770eb9100c2d0c44946d9cf07ec65d  41ce2a54c0b03bf3443c3d931a367089
 3  949d5b44dbf5de918fe9c16f97b45f8a  f88197465ea7920adcdbec7375364d82
```

```
4  ad21c59c0840e6cb83a9ceb5573f8159  8ab97904e6daea8866dbdbc4fb7aad2c

  order_status order_purchase_timestamp    order_approved_at  \
0    delivered      2017-10-02 10:56:33  2017-10-02 11:07:15
1    delivered      2018-07-24 20:41:37  2018-07-26 03:24:27
2    delivered      2018-08-08 08:38:49  2018-08-08 08:55:23
3    delivered      2017-11-18 19:28:06  2017-11-18 19:45:59
4    delivered      2018-02-13 21:18:39  2018-02-13 22:20:29

  order_delivered_carrier_date order_delivered_customer_date  \
0          2017-10-04 19:55:00           2017-10-10 21:25:13
1          2018-07-26 14:31:00           2018-08-07 15:27:45
2          2018-08-08 13:50:00           2018-08-17 18:06:29
3          2017-11-22 13:39:59           2017-12-02 00:28:42
4          2018-02-14 19:46:34           2018-02-16 18:17:02

  order_estimated_delivery_date
0           2017-10-18 00:00:00
1           2018-08-13 00:00:00
2           2018-09-04 00:00:00
3           2017-12-15 00:00:00
4           2018-02-26 00:00:00  ,
                          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
0  2017-09-19 09:45:35   58.90          13.29
1  2017-05-03 11:05:13  239.90          19.93
2  2018-01-18 14:48:30  199.00          17.87
3  2018-08-15 10:10:18   12.99          12.79
4  2017-02-13 13:57:51  199.90          18.14  ,
                          order_id  payment_sequential payment_type  \
0  b81ef226f3fe1789b1e8b2acac839d17                   1  credit_card
1  a9810da82917af2d9aefd1278f1dcfa0                   1  credit_card
2  25e8ea4e93396b6fa0d3dd708e76c1bd                   1  credit_card
3  ba78997921bbcdc1373bb41e913ab953                   1  credit_card
```

```
4  42fdf880ba16b47b59251dd489d4441a                    1  credit_card

   payment_installments  payment_value
0                     8          99.33
1                     1          24.39
2                     1          65.71
3                     8         107.78
4                     2         128.45  )
```

1.0.1 Basic cleaning + date conversion

[2]:
```
# Convert order_purchase_timestamp to datetime
orders["order_purchase_timestamp"] = pd.
 ↪to_datetime(orders["order_purchase_timestamp"], errors="coerce")

# Keep only delivered orders (optional but usually better for CLV)
orders = orders[orders["order_status"] == "delivered"]

orders.shape
```

[2]:
```
(96478, 8)
```

1.0.2 Build a revenue table by order

[3]:
```
# Revenue per order_id from items
items["price"] = pd.to_numeric(items["price"], errors="coerce").fillna(0)
items["freight_value"] = pd.to_numeric(items["freight_value"], errors="coerce").
 ↪fillna(0)

order_revenue = (
    items
    .groupby("order_id", as_index=False)
    .agg(
        product_revenue = ("price", "sum"),
        freight_total   = ("freight_value", "sum")
    )
)

order_revenue["order_revenue"] = order_revenue["product_revenue"] +␣
 ↪order_revenue["freight_total"]
order_revenue.head()
```

[3]:
```
                          order_id  product_revenue  freight_total  \
0  00010242fe8c5a6d1ba2dd792cb16214            58.90          13.29
1  00018f77f2f0320c557190d7a144bdd3           239.90          19.93
2  000229ec398224ef6ca0657da4fc703e           199.00          17.87
```

```
3  00024acbcdf0a6daa1e931b038114c75            12.99          12.79
4  00042b26cf59d7ce69dfabb4e55b4fd9           199.90          18.14

   order_revenue
0          72.19
1         259.83
2         216.87
3          25.78
4         218.04
```

aggregate payments

[4]:
```
payments["payment_value"] = pd.to_numeric(payments["payment_value"],␣
 ↪errors="coerce").fillna(0)

order_payments = (
    payments
    .groupby("order_id", as_index=False)
    .agg(total_payment = ("payment_value", "sum"))
)

order_payments.head()
```

[4]:
```
                          order_id  total_payment
0  00010242fe8c5a6d1ba2dd792cb16214          72.19
1  00018f77f2f0320c557190d7a144bdd3         259.83
2  000229ec398224ef6ca0657da4fc703e         216.87
3  00024acbcdf0a6daa1e931b038114c75          25.78
4  00042b26cf59d7ce69dfabb4e55b4fd9         218.04
```

1.0.3 Join: customers → orders → revenue

[5]:
```
# Join orders with revenue
orders_rev = orders.merge(order_revenue, on="order_id", how="left")

# If you want to use payment instead:
orders_rev = orders_rev.merge(order_payments, on="order_id", how="left")

# Decide final revenue column to use:
# Prefer payment if available, else computed revenue
orders_rev["revenue"] = orders_rev["total_payment"].
 ↪fillna(orders_rev["order_revenue"])

orders_rev.head()
```

[5]:
```
                          order_id                       customer_id  \
0  e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
```

```
1  53cdb2fc8bc7dce0b6741e2150273451  b0830fb4747a6c6d20dea0b8c802d7ef
2  47770eb9100c2d0c44946d9cf07ec65d  41ce2a54c0b03bf3443c3d931a367089
3  949d5b44dbf5de918fe9c16f97b45f8a  f88197465ea7920adcdbec7375364d82
4  ad21c59c0840e6cb83a9ceb5573f8159  8ab97904e6daea8866dbdbc4fb7aad2c

  order_status order_purchase_timestamp    order_approved_at  \
0    delivered      2017-10-02 10:56:33  2017-10-02 11:07:15
1    delivered      2018-07-24 20:41:37  2018-07-26 03:24:27
2    delivered      2018-08-08 08:38:49  2018-08-08 08:55:23
3    delivered      2017-11-18 19:28:06  2017-11-18 19:45:59
4    delivered      2018-02-13 21:18:39  2018-02-13 22:20:29

  order_delivered_carrier_date order_delivered_customer_date  \
0          2017-10-04 19:55:00           2017-10-10 21:25:13
1          2018-07-26 14:31:00           2018-08-07 15:27:45
2          2018-08-08 13:50:00           2018-08-17 18:06:29
3          2017-11-22 13:39:59           2017-12-02 00:28:42
4          2018-02-14 19:46:34           2018-02-16 18:17:02

  order_estimated_delivery_date  product_revenue  freight_total  \
0           2017-10-18 00:00:00            29.99           8.72
1           2018-08-13 00:00:00           118.70          22.76
2           2018-09-04 00:00:00           159.90          19.22
3           2017-12-15 00:00:00            45.00          27.20
4           2018-02-26 00:00:00            19.90           8.72

   order_revenue  total_payment  revenue
0          38.71          38.71    38.71
1         141.46         141.46   141.46
2         179.12         179.12   179.12
3          72.20          72.20    72.20
4          28.62          28.62    28.62
```

1.0.4 aggregate at customer_unique_id

[6]:
```
# Join orders with customers to get customer_unique_id
orders_cust = orders_rev.merge(
    customers[["customer_id", "customer_unique_id"]],
    on="customer_id",
    how="left"
)

orders_cust.head()
```

[6]:
```
                          order_id                       customer_id  \
0  e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
1  53cdb2fc8bc7dce0b6741e2150273451  b0830fb4747a6c6d20dea0b8c802d7ef
```

```
2  47770eb9100c2d0c44946d9cf07ec65d  41ce2a54c0b03bf3443c3d931a367089
3  949d5b44dbf5de918fe9c16f97b45f8a  f88197465ea7920adcdbec7375364d82
4  ad21c59c0840e6cb83a9ceb5573f8159  8ab97904e6daea8866dbdbc4fb7aad2c

  order_status order_purchase_timestamp    order_approved_at  \
0    delivered      2017-10-02 10:56:33  2017-10-02 11:07:15
1    delivered      2018-07-24 20:41:37  2018-07-26 03:24:27
2    delivered      2018-08-08 08:38:49  2018-08-08 08:55:23
3    delivered      2017-11-18 19:28:06  2017-11-18 19:45:59
4    delivered      2018-02-13 21:18:39  2018-02-13 22:20:29

  order_delivered_carrier_date order_delivered_customer_date  \
0          2017-10-04 19:55:00           2017-10-10 21:25:13
1          2018-07-26 14:31:00           2018-08-07 15:27:45
2          2018-08-08 13:50:00           2018-08-17 18:06:29
3          2017-11-22 13:39:59           2017-12-02 00:28:42
4          2018-02-14 19:46:34           2018-02-16 18:17:02

  order_estimated_delivery_date  product_revenue  freight_total  \
0           2017-10-18 00:00:00            29.99           8.72
1           2018-08-13 00:00:00           118.70          22.76
2           2018-09-04 00:00:00           159.90          19.22
3           2017-12-15 00:00:00            45.00          27.20
4           2018-02-26 00:00:00            19.90           8.72

   order_revenue  total_payment  revenue                customer_unique_id
0          38.71          38.71    38.71  7c396fd4830fd04220f754e42b4e5bff
1         141.46         141.46   141.46  af07308b275d755c9edb36a90c618231
2         179.12         179.12   179.12  3a653a41f6f9fc3d2a113cf8398680e8
3          72.20          72.20    72.20  7c142cf63193a1473d2e66489a9ae977
4          28.62          28.62    28.62  72632f0f9dd73dfee390c9b22eb56dd6
```

1.1 Define snapshot_date and build RFM features

[7]:
```
snapshot_date = orders_cust["order_purchase_timestamp"].max() + pd.
 ↪Timedelta(days=1)
print("Snapshot date:", snapshot_date)
```

```
Snapshot date: 2018-08-30 15:00:37
```

1.1.1 Now aggregate by customer_unique_id:

[8]:
```
cust_agg = (
    orders_cust
    .groupby("customer_unique_id")
    .agg(
        FirstPurchaseDate = ("order_purchase_timestamp", "min"),
```

```
        LastPurchaseDate  = ("order_purchase_timestamp", "max"),
        NumOrders         = ("order_id", "nunique"),
        Monetary          = ("revenue", "sum")
    )
    .reset_index()
)

cust_agg["RecencyDays"] = (snapshot_date - cust_agg["LastPurchaseDate"]).dt.days
cust_agg["TenureDays"]  = (snapshot_date - cust_agg["FirstPurchaseDate"]).dt.
 ↪days
cust_agg["TenureDays"]  = cust_agg["TenureDays"].replace(0, 1)  # avoid 0

cust_agg["AOV"]        = cust_agg["Monetary"] / cust_agg["NumOrders"]
cust_agg["OrderRate"]  = cust_agg["NumOrders"] / (cust_agg["TenureDays"] / 30.
 ↪0)  # orders per month

cust_agg.head()
```

[8]:

	customer_unique_id	FirstPurchaseDate	LastPurchaseDate	\
0	0000366f3b9a7992bf8c76cfdf3221e2	2018-05-10 10:56:27	2018-05-10 10:56:27	
1	0000b849f77a49e4a4ce2b2a4ca5be3f	2018-05-07 11:11:27	2018-05-07 11:11:27	
2	0000f46a3911fa3c0805444483337064	2017-03-10 21:05:03	2017-03-10 21:05:03	
3	0000f6ccb0745a6a4b88665a16c9f078	2017-10-12 20:29:41	2017-10-12 20:29:41	
4	0004aac84e0df4da2b147fca70cf8255	2017-11-14 19:45:42	2017-11-14 19:45:42	

	NumOrders	Monetary	RecencyDays	TenureDays	AOV	OrderRate
0	1	141.90	112	112	141.90	0.267857
1	1	27.19	115	115	27.19	0.260870
2	1	86.22	537	537	86.22	0.055866
3	1	43.62	321	321	43.62	0.093458
4	1	196.89	288	288	196.89	0.104167

1.2 Define LTV target

[9]:
```
cust_agg["LTV"] = cust_agg["Monetary"]

# Log-transform due to skew
cust_agg["LTV_log"] = np.log1p(cust_agg["LTV"])
cust_agg["Monetary_log"] = np.log1p(cust_agg["Monetary"])

cust_agg[["customer_unique_id", "LTV", "LTV_log"]].head()
```

[9]:

	customer_unique_id	LTV	LTV_log
0	0000366f3b9a7992bf8c76cfdf3221e2	141.90	4.962145
1	0000b849f77a49e4a4ce2b2a4ca5be3f	27.19	3.338967
2	0000f46a3911fa3c0805444483337064	86.22	4.468434

```
3  0000f6ccb0745a6a4b88665a16c9f078   43.62  3.798182
4  0004aac84e0df4da2b147fca70cf8255  196.89  5.287711
```

1.3 rain-test split & model training

[10]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

feature_cols = ["RecencyDays", "NumOrders", "Monetary_log", "AOV",␣
 ↪"TenureDays", "OrderRate"]

X = cust_agg[feature_cols].fillna(0)
y = cust_agg["LTV_log"]

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

X_train.shape, X_test.shape
```

[10]:
```
((74686, 6), (18672, 6))
```

1.4 Random Forest model

[11]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Make sure X_train, X_test, y_train, y_test are already defined from␣
 ↪train_test_split

# 1) Create and train the model
rf = RandomForestRegressor(
    n_estimators=300,
    random_state=42,
    n_jobs=-1
)
rf.fit(X_train, y_train)

# 2) Predict on test set (log scale)
y_pred_log_rf = rf.predict(X_test)

# 3) Convert back from log-scale to actual LTV
y_test_actual = np.expm1(y_test)              # true values in original scale
```

```
y_pred_actual_rf = np.expm1(y_pred_log_rf)    # predicted values in original␣
 ↪scale

# 4) Metrics
mae_rf = mean_absolute_error(y_test_actual, y_pred_actual_rf)

mse_rf = mean_squared_error(y_test_actual, y_pred_actual_rf)
rmse_rf = np.sqrt(mse_rf)

r2_rf = r2_score(y_test_actual, y_pred_actual_rf)

print(f"RF - MAE: {mae_rf:,.2f}  RMSE: {rmse_rf:,.2f}  R²: {r2_rf:.4f}")
```

```
RF - MAE: 0.58  RMSE: 52.64  R²: 0.9502
```

[12]:
```
import xgboost as xgb

xgb_model = xgb.XGBRegressor(
    n_estimators=400,
    max_depth=5,
    learning_rate=0.05,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    n_jobs=-1
)

xgb_model.fit(X_train, y_train)

y_pred_log_xgb = xgb_model.predict(X_test)
y_pred_actual_xgb = np.expm1(y_pred_log_xgb)

mae_xgb = mean_absolute_error(y_test_actual, y_pred_actual_xgb)
mse_xgb = mean_squared_error(y_test_actual, y_pred_actual_xgb)
rmse_xgb = np.sqrt(mse_xgb)
r2_xgb = r2_score(y_test_actual, y_pred_actual_xgb)

print(f"XGB - MAE: {mae_xgb:,.2f}  RMSE: {rmse_xgb:,.2f}  R²: {r2_xgb:.4f}")
```

```
XGB - MAE: 3.73  RMSE: 101.98  R²: 0.8129
```

1.5 XGBoost model

[13]:
```
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
```

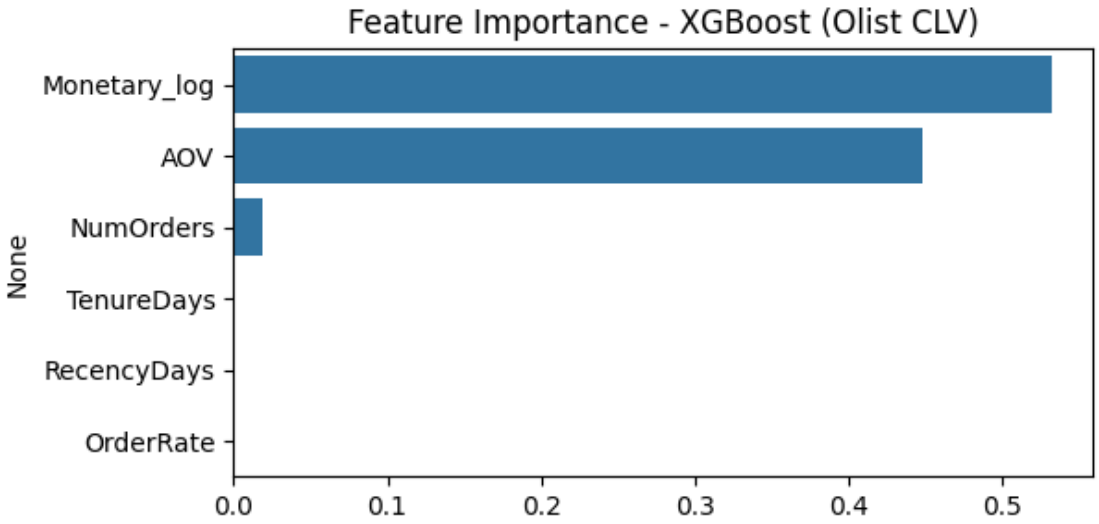
```
xgb_model = xgb.XGBRegressor(
    n_estimators=400,
    max_depth=5,
    learning_rate=0.05,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    n_jobs=-1
)

# Older xgboost: don't pass early_stopping_rounds here
# Simple fit without early stopping:
xgb_model.fit(X_train, y_train)

# Predictions
y_pred_log_xgb = xgb_model.predict(X_test)
y_pred_actual_xgb = np.expm1(y_pred_log_xgb)

# Metrics (no 'squared' arg in older sklearn)
mae_xgb = mean_absolute_error(y_test_actual, y_pred_actual_xgb)

mse_xgb = mean_squared_error(y_test_actual, y_pred_actual_xgb)
rmse_xgb = np.sqrt(mse_xgb)

r2_xgb = r2_score(y_test_actual, y_pred_actual_xgb)

print(f"XGB - MAE: {mae_xgb:,.2f}  RMSE: {rmse_xgb:,.2f}  R²: {r2_xgb:.4f}")
```

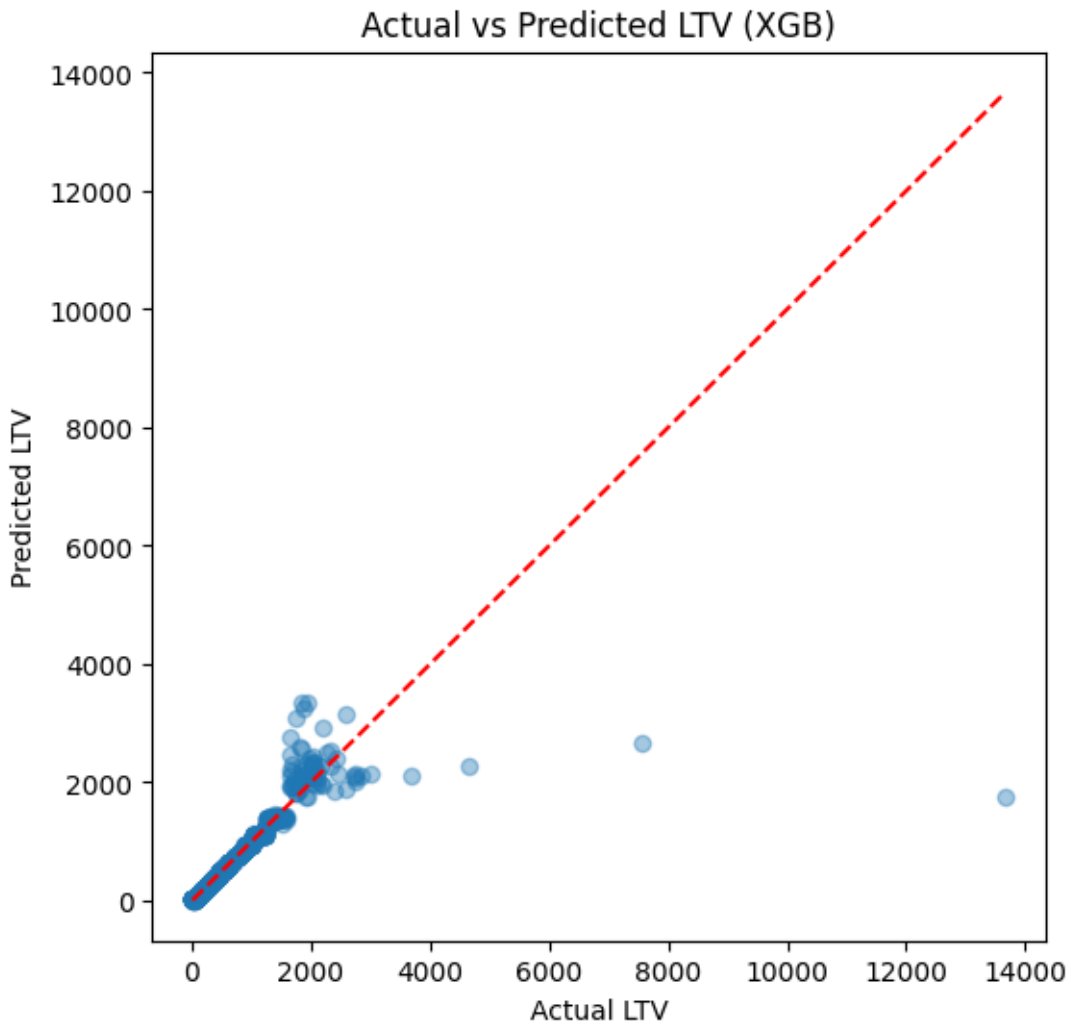
```
XGB - MAE: 3.73  RMSE: 101.98  R²: 0.8129
```

1.6 Feature importance & actual vs predicted plot

[14]:
```
import matplotlib.pyplot as plt
import seaborn as sns

fi = pd.Series(xgb_model.feature_importances_, index=feature_cols).
 ↪sort_values(ascending=False)

plt.figure(figsize=(6, 3))
sns.barplot(x=fi.values, y=fi.index)
plt.title("Feature Importance - XGBoost (Olist CLV)")
plt.show()

plt.figure(figsize=(6, 6))
plt.scatter(y_test_actual, y_pred_actual_xgb, alpha=0.4)
plt.plot([y_test_actual.min(), y_test_actual.max()],
         [y_test_actual.min(), y_test_actual.max()], "r--")
```

```
plt.xlabel("Actual LTV")
plt.ylabel("Predicted LTV")
plt.title("Actual vs Predicted LTV (XGB)")
plt.show()
```





1.6.1 Predict LTV for all customers + segment

[15]:
```
cust_agg["LTV_pred_log"] = xgb_model.predict(X.fillna(0))
cust_agg["LTV_pred"] = np.expm1(cust_agg["LTV_pred_log"])

# 3 segments: Low / Medium / High
q33, q66 = cust_agg["LTV_pred"].quantile([0.33, 0.66])

def seg3(x):
    if x <= q33:
        return "Low"
    elif x <= q66:
        return "Medium"
    else:
        return "High"
```

```
cust_agg["LTV_segment"] = cust_agg["LTV_pred"].apply(seg3)

cust_agg[["customer_unique_id", "LTV", "LTV_pred", "LTV_segment"]].head()
```

[15]:

	customer_unique_id	LTV	LTV_pred	LTV_segment
0	0000366f3b9a7992bf8c76cfdf3221e2	141.90	141.697128	Medium
1	0000b849f77a49e4a4ce2b2a4ca5be3f	27.19	26.996445	Low
2	0000f46a3911fa3c0805444483337064	86.22	86.104973	Medium
3	0000f6ccb0745a6a4b88665a16c9f078	43.62	43.490223	Low
4	0004aac84e0df4da2b147fca70cf8255	196.89	196.322250	High

1.7 Export final CSV + save models

[16]:
```
import joblib

output_cols = [
    "customer_unique_id",
    "FirstPurchaseDate", "LastPurchaseDate",
    "NumOrders", "Monetary", "RecencyDays", "TenureDays",
    "AOV", "OrderRate",
    "LTV", "LTV_pred", "LTV_segment"
]

cust_agg[output_cols].to_csv("olist_customer_ltv_predictions.csv", index=False)
print("Saved -> olist_customer_ltv_predictions.csv")

joblib.dump(rf, "olist_rf_ltv_model.joblib")
joblib.dump(xgb_model, "olist_xgb_ltv_model.joblib")
print("Saved -> olist_rf_ltv_model.joblib, olist_xgb_ltv_model.joblib")
```

```
Saved -> olist_customer_ltv_predictions.csv
Saved -> olist_rf_ltv_model.joblib, olist_xgb_ltv_model.joblib
```