Name - Kethe Vamshikrishna

Project Title - Customer Lifetime Value Prediction.

Languages - Python.

Tools - Pandas, Scikit-learn.

1. Introduction

Customer Lifetime Value (**CLV**) is the projected revenue a customer is expected to generate over their relationship with a company. Predicting CLV helps businesses identify **high-value customers**, design **personalized campaigns**, and allocate marketing budgets more effectively.

This project builds a machine learning pipeline to predict **12-month CLV** from customer transaction data.

---

2. Dataset

- **Source:** Retail transactions dataset (UK-based online store).
- **Shape:** (541,909 rows × 8 columns).
- **Main Fields:**
    - `InvoiceNo` → Unique order number
    - `StockCode` → Product code
    - `Description` → Item description
    - `Quantity` → Units purchased

    - `InvoiceDate` → Date of order
    - `UnitPrice` → Price per item
    - `CustomerID` → Unique customer identifier
    - `Country` → Country of customer

Cleaning Steps

- Removed missing `CustomerID` values.
- Converted `InvoiceDate` to datetime.
- Created `Revenue = Quantity * UnitPrice`.

---

3. Methodology

Step 1 – Calibration & Prediction Window

- **Calibration window:** Early transactions (before 2010-12-09).
- **Prediction window:** Future transactions (after 2010-12-09).

Step 2 – Feature Engineering

For each customer:

- **Frequency** = Number of purchases.

- **Monetary Total** = Total spend.
- **Average Order Value** = Mean revenue per order.
- **Recency** = Days since last purchase.
- **Tenure** = Days between first & last purchase.
- **Orders in last 30 days**.

Step 3 – Target Variable

- **CLV (12 months)** = Total spend in prediction window.
- **Log transformation** used for model stability.

Step 4 – Modeling

- Models trained:
  - **Ridge Regression**
  - **Random Forest Regressor**
- Evaluation Metrics:
  - **MAE** (Mean Absolute Error)
  - **RMSE** (Root Mean Squared Error)

Step 5 – Revenue Capture Analysis

- **Top-Capture Curve** → Measures revenue share captured by targeting top % predicted customers.

Step 6 – Deployment

- Best model pipeline saved as `clv_pipeline_rf.joblib`.

4. Results

Error Metrics

- **Ridge Regression**
  - MAE = **3516.10**
  - RMSE = **11547.95**
- **Random Forest Regressor**
  - MAE = **3661.29**
  - RMSE = **11992.12**

Ridge performed slightly better in error metrics, though both are in the same range.

Top-Capture Curve (Random Forest)

- Top 1% customers → **0.0% revenue**
- Top 5% customers → **10.6% revenue**
- Top 10% customers → **19.1% revenue**
- Top 20% customers → **30.5% revenue**

This shows the model is able to **identify high-value customers** fairly well.

---

5. Discussion

- Ridge regression generalized better due to the noisy nature of customer purchase behavior.
- Random Forest captured **non-linear effects**, but errors were slightly higher.

- The **Top-Capture Curve** validates that even imperfect predictions can drive **better marketing ROI** by focusing on a smaller group of valuable customers.

Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import joblib

# ---- 2) Load Your Dataset ----
file_path = r"C:\Users\LENOVA\Music\customer_segmentation.csv"

# Try with safe encoding
try:
    df = pd.read_csv(file_path, encoding="utf-8")
except UnicodeDecodeError:
    df = pd.read_csv(file_path, encoding="latin1")
```

```
print("Shape:", df.shape)
print(df.head())
print(df.info())

# ---- 3) Basic Cleaning ----
# Convert date column (replace 'InvoiceDate' with your dataset's actual column name)
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], errors='coerce')
df = df.dropna(subset=['InvoiceDate'])

# Rename columns for consistency (adjust to your dataset)
df = df.rename(columns={
    'CustomerID': 'customer_id',
    'InvoiceNo': 'order_id',
    'InvoiceDate': 'order_date',
    'UnitPrice': 'unit_price',
    'Quantity': 'quantity'
})

# Create revenue column if not already present
if 'revenue' not in df.columns:
    df['revenue'] = df['unit_price'] * df['quantity']

# ---- 4) Define Calibration & Prediction Windows ----
prediction_horizon_months = 12
max_date = df['order_date'].max()
calibration_end = max_date - pd.DateOffset(months=prediction_horizon_months)

calibration = df[df['order_date'] <= calibration_end].copy()
prediction = df[(df['order_date'] > calibration_end) & (df['order_date'] <= max_date)].copy()

print("Calibration end:", calibration_end)
print("Calibration transactions:", len(calibration))
print("Prediction transactions:", len(prediction))

# ---- 5) Feature Engineering (RFM + Behavioral) ----
agg = calibration.groupby('customer_id').agg(
    frequency=('order_id','nunique'),
    monetary_total=('revenue','sum'),
    avg_order_value=('revenue','mean'),
    first_purchase=('order_date','min'),
    last_purchase=('order_date','max')
).reset_index()

agg['recency_days'] = (calibration_end - agg['last_purchase']).dt.days
```

```
agg['tenure_days'] = (agg['last_purchase'] - agg['first_purchase']).dt.days
agg['orders_30d'] = calibration[
    calibration['order_date'] >= (calibration_end - pd.Timedelta(days=30))
].groupby('customer_id')['order_id'].nunique().reindex(agg['customer_id']).fillna(0).values

agg = agg.fillna(0)
print(agg.head())

# ---- 6) Create Target (CLV in prediction window) ----
target = prediction.groupby('customer_id')['revenue'].sum().rename('clv_12m').reset_index()
data = agg.merge(target, on='customer_id', how='left').fillna({'clv_12m': 0.0})
data['clv_12m'] = data['clv_12m'].clip(lower=0.0)
data['log_clv'] = np.log1p(data['clv_12m'])

print(data[['customer_id','frequency','monetary_total','clv_12m','log_clv']].head())

# ---- 7) Explore Target Distribution ----
plt.figure(figsize=(8,4))
plt.hist(data['clv_12m'], bins=50)
plt.title('Distribution of CLV (12m)')
plt.show()

plt.figure(figsize=(8,4))
plt.hist(data['log_clv'], bins=50)
plt.title('Distribution of log1p(CL V)')
plt.show()

# ---- 8) Modeling (Ridge & RandomForest) ----
feature_cols = ['frequency','monetary_total','avg_order_value','recency_days',
                'tenure_days','orders_30d']

X = data[feature_cols].copy()
y = data['log_clv'].copy()

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

num_pipe = Pipeline([('imputer', SimpleImputer(strategy='median')), ('scaler', StandardScaler())])
preprocessor = ColumnTransformer([('num', num_pipe, feature_cols)])

# Linear Ridge Model
model_lin = Pipeline([('pre', preprocessor), ('reg', Ridge())])
model_lin.fit(X_train, y_train)
y_pred_lin = np.expm1(model_lin.predict(X_test))
y_true = np.expm1(y_test)
```

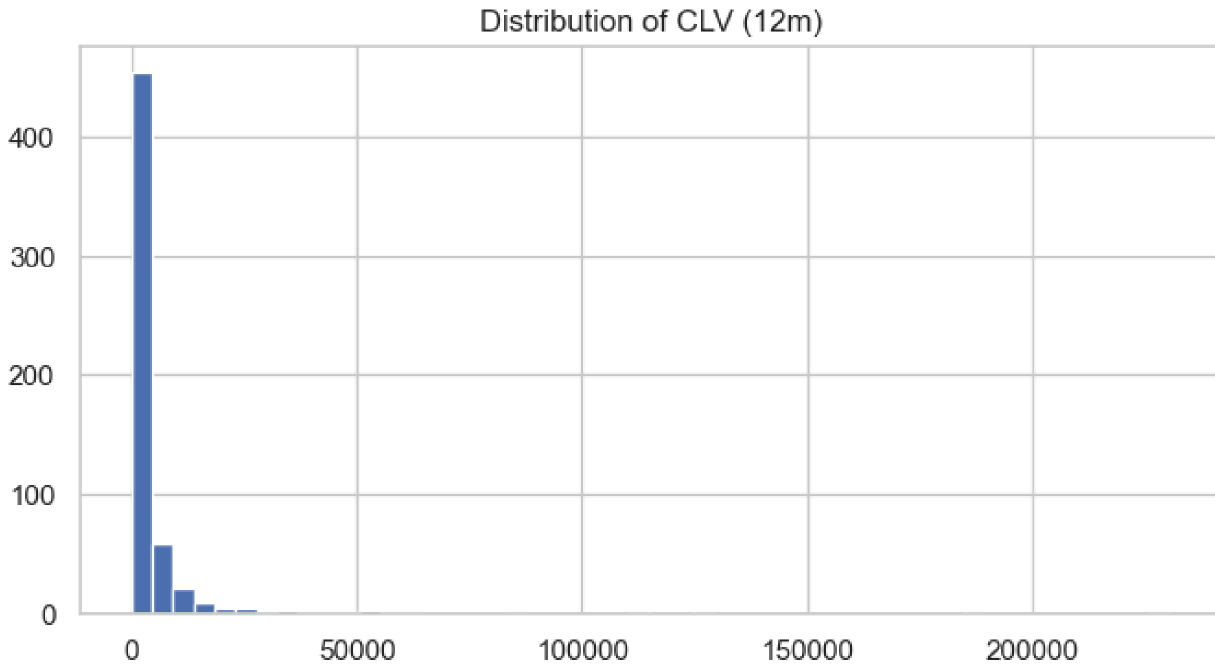
```
print("Ridge MAE:", mean_absolute_error(y_true, y_pred_lin))
print("Ridge RMSE:", mean_squared_error(y_true, y_pred_lin, squared=False))

# Random Forest Model
model_rf = Pipeline([('pre', preprocessor),
                     ('reg', RandomForestRegressor(n_estimators=200, random_state=42,
n_jobs=-1))])
model_rf.fit(X_train, y_train)
y_pred_rf = np.expm1(model_rf.predict(X_test))
print("RF MAE:", mean_absolute_error(y_true, y_pred_rf))
print("RF RMSE:", mean_squared_error(y_true, y_pred_rf, squared=False))

# ---- 9) Decile/Top-Capture Curve ----
eval_df = pd.DataFrame({'pred_rf': y_pred_rf, 'actual': y_true.values})
eval_df = eval_df.sort_values('pred_rf', ascending=False).reset_index(drop=True)
eval_df['cum_actual'] = eval_df['actual'].cumsum()
total_actual = eval_df['actual'].sum()
eval_df['cum_capture'] = eval_df['cum_actual'] / total_actual

plt.figure(figsize=(8,5))
plt.plot(np.arange(len(eval_df))/len(eval_df), eval_df['cum_capture'])
plt.xlabel("Proportion of customers")
plt.ylabel("Cumulative revenue captured")
plt.title("Top-Capture Curve")
plt.grid(True)
plt.show()

for pct in [0.01, 0.05, 0.10, 0.20]:
    k = int(len(eval_df) * pct)
    captured = eval_df.iloc[:max(1,k)]['actual'].sum() / total_actual
    print(f"Top {int(pct*100)}% capture: {captured:.3f}")

# ---- 10) Save Best Model ----
final_pipeline = model_rf
joblib.dump(final_pipeline, "clv_pipeline_rf.joblib")
print("Pipeline saved as clv_pipeline_rf.joblib")
```

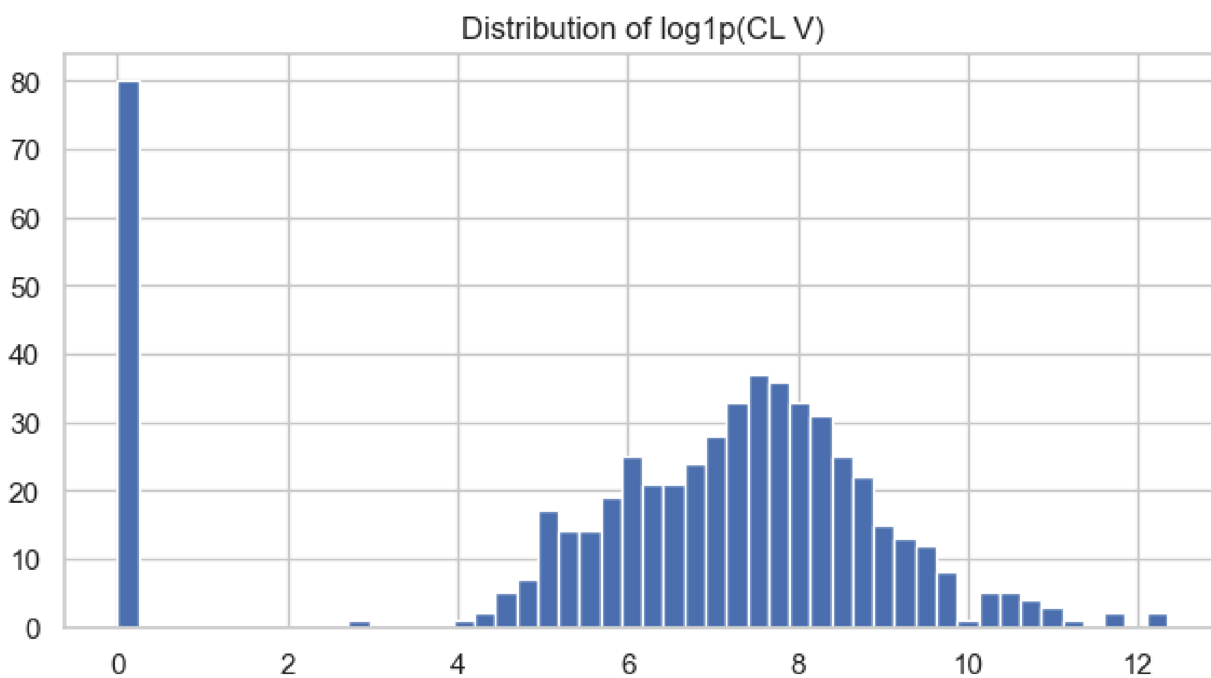
Output

```
Shape: (541909, 8)
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
```

```
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

      InvoiceDate  UnitPrice  CustomerID         Country
0  12/1/2010 8:26       2.55     17850.0  United Kingdom
1  12/1/2010 8:26       3.39     17850.0  United Kingdom
2  12/1/2010 8:26       2.75     17850.0  United Kingdom
3  12/1/2010 8:26       3.39     17850.0  United Kingdom
4  12/1/2010 8:26       3.39     17850.0  United Kingdom
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   InvoiceNo    541909 non-null  object 
 1   StockCode    541909 non-null  object 
 2   Description  540455 non-null  object 
 3   Quantity     541909 non-null  int64  
 4   InvoiceDate  541909 non-null  object 
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object 
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
None
Calibration end: 2010-12-09 12:50:00
Calibration transactions: 20240
Prediction transactions: 521669
   customer_id  frequency  monetary_total  avg_order_value  \
0      12347.0          1          711.79        22.960968   
1      12386.0          1          258.90        32.362500   
2      12395.0          1          346.10        28.841667   
3      12427.0          1          303.50        30.350000   
4      12429.0          1         1281.50        64.075000   

       first_purchase       last_purchase  recency_days  tenure_days  \
0 2010-12-07 14:57:00 2010-12-07 14:57:00             1            0   
1 2010-12-08 09:53:00 2010-12-08 09:53:00             1            0   
2 2010-12-03 16:35:00 2010-12-03 16:35:00             5            0   
3 2010-12-03 10:44:00 2010-12-03 10:44:00             6            0   
4 2010-12-09 12:05:00 2010-12-09 12:05:00             0            0   

   orders_30d  
0           1  
1           1  
2           1  
3           1  
4           1  
   customer_id  frequency  monetary_total  clv_12m   log_clv
0      12347.0          1          711.79  3598.21  8.188470
```

1	12386.0	1	258.90	143.00	4.969813
2	12395.0	1	346.10	2652.18	7.883514
3	12427.0	1	303.50	404.87	6.006033
4	12429.0	1	1281.50	2468.90	7.811933



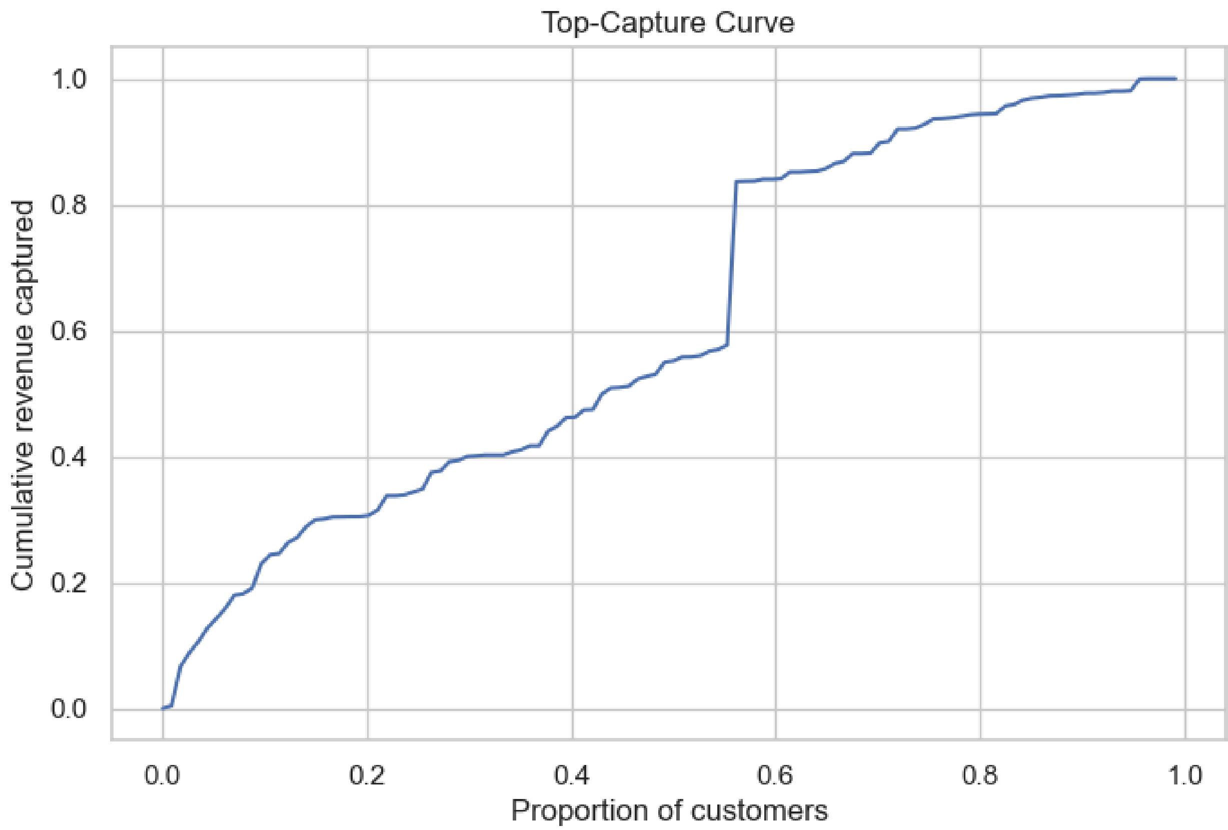



```
Ridge MAE: 3516.100111918296
Ridge RMSE: 11547.950115346614
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

```
RF MAE: 3661.2965385813027
RF RMSE: 11992.122592527137
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```




```
Top 1% capture: 0.000
Top 5% capture: 0.106
Top 10% capture: 0.191
Top 20% capture: 0.305
Pipeline saved as clv_pipeline_rf.joblib
```

6. Conclusion

- Built a **CLV prediction pipeline** using retail transaction data.
- Generated features like **frequency, recency, monetary value, tenure**.
- Achieved reasonable predictive performance:
  - Ridge MAE ≈ 3.5k, RMSE ≈ 11.5k.
- Showed business value: targeting top 10% customers captures ~20% of total future revenue.