PHASE 7 Counterfactual Uplift & Policy Evaluation

```python
# STEP 7.1 — Load Phase 6 Artifacts

import pandas as pd
import numpy as np

clv_df = pd.read_parquet("phase5_expected_clv.parquet")
state_df = pd.read_parquet("phase2_customer_state.parquet")

# Use latest state per customer
latest_state = (
    state_df.sort_values("InvoiceDate")
            .groupby("Customer ID")
            .tail(1)
            .reset_index(drop=True)
)

df = clv_df.merge(latest_state, on="Customer ID")
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 5881,\n  \"fields\":
[\n    {\n      \"column\": \"Customer ID\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 1715.4297590182248,\
n        \"min\": 12346.0,\n        \"max\": 18287.0,\n
\"num_unique_values\": 5881,\n        \"samples\": [\n
17776.0,\n          17703.0,\n          12546.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"expected_clv\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
13338.156006795914,\n        \"min\": 0.0,\n        \"max\":
401760.5653631753,\n        \"num_unique_values\": 5859,\n
\"samples\": [\n          10381.958679949777,\n
9716.554121162893,\n          2170.4114623373707\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"InvoiceDate\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2009-12-01 09:55:00\",\n        \"max\": \"2011-12-09 12:50:00\",\n
\"num_unique_values\": 5731,\n        \"samples\": [\n
\"2011-11-20 10:15:00\",\n          \"2010-11-04 09:06:00\",\n
\"2011-02-04 14:03:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"recency_days\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 125.85066600962924,\n
\"min\": 0.0,\n        \"max\": 714.0,\n        \"num_unique_values\":
491,\n        \"samples\": [\n          628.0,\n          318.0,\n
394.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"frequency\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 13,\n        \"min\": 0,\n
```

```
\"max\": 399,\n        \"num_unique_values\": 88,\n
\"samples\": [\n          65,\n          11,\n          12\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"monetary_avg\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1214.7556670815925,\n        \"min\":
0.0,\n        \"max\": 84236.25,\n        \"num_unique_values\":
5775,\n        \"samples\": [\n          230.04888888888888,\n
42.22,\n          131.83\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"delta_revenue\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2953.9114568016107,\n
\"min\": -38320.5,\n        \"max\": 168466.7,\n
\"num_unique_values\": 4162,\n        \"samples\": [\n          -
2.8699999999999974,\n          0.8899999999999864,\n          41.25\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"delta_recency\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
139.51796167650065,\n        \"min\": -618.0,\n        \"max\":
701.0,\n        \"num_unique_values\": 694,\n        \"samples\": [\n
120.0,\n          -183.0,\n          -165.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}

# STEP 7.2 — Define Treatment & Outcome (Synthetic but Defensible)
```

Treatment

T = 1 → retention action

T = 0 → no action

```
# STEP 7.3 — Construct Proxy Treatment Effect

# We simulate heterogeneous uplift using behavior


np.random.seed(42)

df["treatment"] = np.random.binomial(1, 0.5, size=len(df))

# Customers with high recency + low frequency benefit more
base_risk = (
    0.6 * (df["recency_days"] / df["recency_days"].max())
    - 0.4 * (df["frequency"] / df["frequency"].max())
)

treatment_effect = 0.2 * (1 - base_risk)

df["outcome"] = (
    1 - base_risk
```

```
    + df["treatment"] * treatment_effect
    + np.random.normal(0, 0.02, size=len(df))
)

# STEP 7.3.1 — Impute state variables before outcome generation

from sklearn.impute import SimpleImputer

state_features = [
    "recency_days",
    "frequency",
    "monetary_avg",
    "delta_revenue",
    "delta_recency"
]

imputer = SimpleImputer(strategy="median")
df[state_features] = imputer.fit_transform(df[state_features])

base_risk = (
    0.6 * (df["recency_days"] / df["recency_days"].max())
    - 0.4 * (df["frequency"] / df["frequency"].max())
)

treatment_effect = 0.2 * (1 - base_risk)

df["outcome"] = (
    1 - base_risk
    + df["treatment"] * treatment_effect
    + np.random.normal(0, 0.02, size=len(df))
)

df["outcome"].isna().sum()

np.int64(0)

treated = df[df["treatment"] == 1]
control = df[df["treatment"] == 0]

model_t.fit(treated[features], treated["outcome"])
model_c.fit(control[features], control["outcome"])

RandomForestRegressor(max_depth=6, random_state=42)

# STEP 7.4 — Uplift Modeling (Two-Model Approach)

# Split treated / control
features = [
    "recency_days",
    "frequency",
    "monetary_avg",
    "delta_revenue",
```

```
    "delta_recency"
]

treated = df[df["treatment"] == 1]
control = df[df["treatment"] == 0]

# Fit outcome models
from sklearn.ensemble import RandomForestRegressor

model_t = RandomForestRegressor(
    n_estimators=100,
    max_depth=6,
    random_state=42
)

model_c = RandomForestRegressor(
    n_estimators=100,
    max_depth=6,
    random_state=42
)

model_t.fit(treated[features], treated["outcome"])
model_c.fit(control[features], control["outcome"])
```

```
RandomForestRegressor(max_depth=6, random_state=42)
```

```
# Predict counterfactuals
mu_1 = model_t.predict(df[features])
mu_0 = model_c.predict(df[features])

df["uplift"] = mu_1 - mu_0
df["uplift"].describe()
```

```
count    5881.000000
mean        0.184900
std         0.020072
min         0.035772
25%         0.182714
50%         0.191182
75%         0.196223
max         0.358401
Name: uplift, dtype: float64
```

```
# STEP 7.5 — Convert Uplift → Incremental CLV (KEY STEP)

df["incremental_clv"] = df["uplift"] * df["expected_clv"]

# STEP 7.6 — Decision Optimization (Re-run with True Uplift)

ACTION_COST = 100
TOTAL_BUDGET = 50000
K = TOTAL_BUDGET // ACTION_COST
```

```
decision_uplift = (
    df.sort_values("incremental_clv", ascending=False)
      .head(K)
)

uplift_value = decision_uplift["incremental_clv"].sum()
uplift_value
```

```
np.float64(2997894.3647211646)
```

```
 # STEP 7.7 — Policy Evaluation vs Heuristics (MANDATORY)

# Heuristic 1 — Frequency only
freq_policy = (
    df.sort_values("frequency", ascending=False)
      .head(K)
)

freq_value = (
    freq_policy["uplift"] * freq_policy["expected_clv"]
).sum()

# Heuristic 2 — CLV only
clv_policy = (
    df.sort_values("expected_clv", ascending=False)
      .head(K)
)

clv_value = (
    clv_policy["uplift"] * clv_policy["expected_clv"]
).sum()

# Comparison table
comparison = pd.DataFrame({
    "Policy": ["Uplift-Optimized (CLV 4.0)", "CLV Only", "Frequency Only"],
    "Total Incremental Value": [uplift_value, clv_value, freq_value]
})

comparison
```


{"summary":"{\n \"name\": \"comparison\",\n \"rows\": 3,\n \"fields\": [\n   {\n   \"column\": \"Policy\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"Uplift-Optimized (CLV 4.0)\",\n   \"CLV Only\",\n   \"Frequency Only\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Total Incremental Value\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 580108.0329351632,\n

```
\"min\": 1989192.2088580416,\n        \"max\": 2997894.3647211646,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
2997894.3647211646,\n          2989996.668483725,\n
1989192.2088580416\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"comparison"}
```