**Problem Statement (Refined)**

Predict the **number** and **total amount** of **high-value claims** a member will make in the **next 2 years**, using historical membership and claim data.

**🧩 Typical Data You’ll Have**

You mentioned:

* **Membership Data** — likely includes:
  + member\_id
  + demographics (age, gender, location, etc.)
  + plan type / coverage level
  + join\_date, tenure, active\_flag
  + premium, risk score, etc.
* **Claim Data** — likely includes:
  + claim\_id
  + member\_id
  + claim\_date
  + claim\_amount
  + diagnosis / procedure codes
  + claim\_type (inpatient, outpatient, etc.)

**🧠 Key Objective**

We need to predict for each member:

1. **Claim Count Prediction (Frequency Model)** → how many high-value claims (e.g., > ₹X or > $X) in next 2 years
2. **Claim Amount Prediction (Severity Model)** → total or average cost of those claims

Then you can combine both for an **expected future cost model**:

**⚙️ Recommended Approach**

**Step 1: Define “High Claim”**

* Decide a threshold (e.g., top 10% claim amount, or claims > ₹50,000).
* Label historical claims accordingly.

**Step 2: Create a Training Dataset**

For each member:

* Create **observation windows** (e.g., use 2 years of past data to predict next 2 years).
* Aggregate claim & member data within that window:
  + Claim count, total amount, avg claim amount
  + Time since last claim
  + Number of high claims in past
  + Member’s demographics / plan features

**Example:**

| **member\_id** | **age** | **plan\_type** | **prev\_2yr\_claim\_count** | **prev\_2yr\_total\_amount** | **prev\_2yr\_high\_claims** | **target\_high\_claim\_count\_next\_2yr** | **target\_high\_claim\_amount\_next\_2yr** |
| --- | --- | --- | --- | --- | --- | --- | --- |

**Step 3: Feature Engineering Ideas**

* **Demographics**: age, gender, location risk factor
* **Utilization history**: past claim count, avg amount, last claim gap
* **Behavioral trend**: claim frequency trend over time
* **Plan features**: coverage type, deductible
* **Temporal**: seasonality, year-on-year change

**Step 4: Model Building**

You’ll likely use **two models**:

**(A) Frequency Model**

* **Target:** number of high claims in next 2 years
* **Type:** count data → use
  + Poisson Regression / Negative Binomial Regression
  + or ML regressors (XGBoost, Random Forest, etc.)

**(B) Severity Model**

* **Target:** total/average amount of high claims
* **Type:** regression (continuous target) →
  + Linear Regression, Gamma Regression, or Gradient Boosting

**Step 5: Combine (Optional)**

Expected Future High Claim Cost per member:

Expected Cost=Count×AverageAmount

**Step 6: Evaluate**

* **Frequency model:** MAE / RMSE / Poisson Deviance
* **Severity model:** RMSE / MAPE / R²
* **Business validation:** compare predicted vs actual total high claim cost

**Step 7: Forecast for Future (2-Year Horizon)**

Use the **latest available data** for each member (as of now) → feed into trained models → get 2-year projections.

**🧰 Tools & Libraries**

* **Python:** pandas, scikit-learn, statsmodels, xgboost, lightgbm
* **Visualization:** seaborn, matplotlib
* **Optional:** lifelines (for survival analysis if predicting *when* a high claim will occur)

**💡 Extensions**

* Predict **when** a high claim will occur (survival / time-to-event model).
* Segment members by risk score for **premium optimization** or **preventive intervention**.
* Use **Explainable AI** (e.g., SHAP) to understand key drivers of high claims.

Would you like me to help you with the **data preparation pipeline** (i.e., how to create training features from membership and claim data in Python)?  
I can show a ready-to-run example with code and structure.

import numpy as np

import pandas as pd

from datetime import datetime, timedelta

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, r2\_score

from xgboost import XGBRegressor

# --------------------------

# 1️⃣ Generate Synthetic Data

# --------------------------

np.random.seed(42)

n\_members = 2000

start\_date = datetime(2015, 1, 1)

end\_date = datetime(2025, 1, 1)

# Membership data

members = pd.DataFrame({

"member\_id": range(1, n\_members + 1),

"age": np.random.randint(20, 75, n\_members),

"gender": np.random.choice(["M", "F"], n\_members),

"plan\_type": np.random.choice(["Silver", "Gold", "Platinum"], n\_members, p=[0.5, 0.3, 0.2]),

"join\_date": [start\_date + timedelta(days=np.random.randint(0, 365\*5)) for \_ in range(n\_members)],

})

# Claim data

n\_claims = 10000

claim\_dates = [start\_date + timedelta(days=np.random.randint(0, (end\_date - start\_date).days)) for \_ in range(n\_claims)]

claims = pd.DataFrame({

"claim\_id": range(1, n\_claims + 1),

"member\_id": np.random.choice(members["member\_id"], n\_claims),

"claim\_date": claim\_dates,

"claim\_amount": np.random.exponential(scale=15000, size=n\_claims).astype(int)

})

# --------------------------

# 2️⃣ Define high-claim threshold

# --------------------------

threshold = claims["claim\_amount"].quantile(0.9)

claims["is\_high\_claim"] = (claims["claim\_amount"] > threshold).astype(int)

# --------------------------

# 3️⃣ Create historical (past 2 years) features & future (next 2 years) targets

# --------------------------

cutoff\_date = datetime(2021, 1, 1) # pretend we are training as of Jan 2021

past\_window\_start = cutoff\_date - timedelta(days=365\*2)

future\_window\_end = cutoff\_date + timedelta(days=365\*2)

# Past 2 years (feature window)

past\_claims = claims[(claims["claim\_date"] >= past\_window\_start) & (claims["claim\_date"] < cutoff\_date)]

# Future 2 years (target window)

future\_claims = claims[(claims["claim\_date"] >= cutoff\_date) & (claims["claim\_date"] < future\_window\_end)]

# Aggregate past claims

past\_features = past\_claims.groupby("member\_id").agg(

past\_claim\_count=("claim\_id", "count"),

past\_total\_amount=("claim\_amount", "sum"),

past\_high\_claims=("is\_high\_claim", "sum"),

avg\_claim\_amount=("claim\_amount", "mean"),

last\_claim\_date=("claim\_date", "max")

).reset\_index()

# Fill missing (members with no past claims)

past\_features["days\_since\_last\_claim"] = (cutoff\_date - past\_features["last\_claim\_date"]).dt.days

past\_features.drop(columns="last\_claim\_date", inplace=True)

past\_features.fillna({"days\_since\_last\_claim": 9999, "avg\_claim\_amount": 0,

"past\_claim\_count": 0, "past\_total\_amount": 0, "past\_high\_claims": 0}, inplace=True)

# Target: future high claim count & amount

future\_targets = future\_claims[future\_claims["is\_high\_claim"] == 1].groupby("member\_id").agg(

future\_high\_claim\_count=("claim\_id", "count"),

future\_high\_claim\_amount=("claim\_amount", "sum")

).reset\_index()

# Merge features + targets + member info

data = members.merge(past\_features, on="member\_id", how="left").merge(future\_targets, on="member\_id", how="left")

data.fillna({"future\_high\_claim\_count": 0, "future\_high\_claim\_amount": 0}, inplace=True)

# --------------------------

# 4️⃣ Encode categorical variables

# --------------------------

data = pd.get\_dummies(data, columns=["gender", "plan\_type"], drop\_first=True)

# --------------------------

# 5️⃣ Split Train/Test

# --------------------------

X = data.drop(columns=["member\_id", "future\_high\_claim\_count", "future\_high\_claim\_amount", "join\_date"])

y\_count = data["future\_high\_claim\_count"]

y\_amount = data["future\_high\_claim\_amount"]

X\_train, X\_test, y\_count\_train, y\_count\_test, y\_amount\_train, y\_amount\_test = train\_test\_split(

X, y\_count, y\_amount, test\_size=0.2, random\_state=42

)

# --------------------------

# 6️⃣ Train Models

# --------------------------

count\_model = XGBRegressor(objective='count:poisson', n\_estimators=200, learning\_rate=0.05, random\_state=42)

count\_model.fit(X\_train, y\_count\_train)

amount\_model = XGBRegressor(objective='reg:squarederror', n\_estimators=200, learning\_rate=0.05, random\_state=42)

amount\_model.fit(X\_train, y\_amount\_train)

# --------------------------

# 7️⃣ Evaluate Models

# --------------------------

count\_pred = count\_model.predict(X\_test)

amount\_pred = amount\_model.predict(X\_test)

print("=== Claim COUNT Model ===")

print("MAE:", mean\_absolute\_error(y\_count\_test, count\_pred))

print("R² :", r2\_score(y\_count\_test, count\_pred))

print("\n=== Claim AMOUNT Model ===")

print("MAE:", mean\_absolute\_error(y\_amount\_test, amount\_pred))

print("R² :", r2\_score(y\_amount\_test, amount\_pred))

# --------------------------

# 8️⃣ Predict future for all members

# --------------------------

data["pred\_future\_high\_claim\_count"] = count\_model.predict(X)

data["pred\_future\_high\_claim\_amount"] = amount\_model.predict(X)

data["predicted\_total\_high\_claim\_cost"] = data["pred\_future\_high\_claim\_count"] \* data["pred\_future\_high\_claim\_amount"]

# Display sample output

print("\nSample predictions:")

print(data[["member\_id", "pred\_future\_high\_claim\_count", "pred\_future\_high\_claim\_amount", "predicted\_total\_high\_claim\_cost"]].head(10))