Income Case Study

Data Challenges

Missing Values

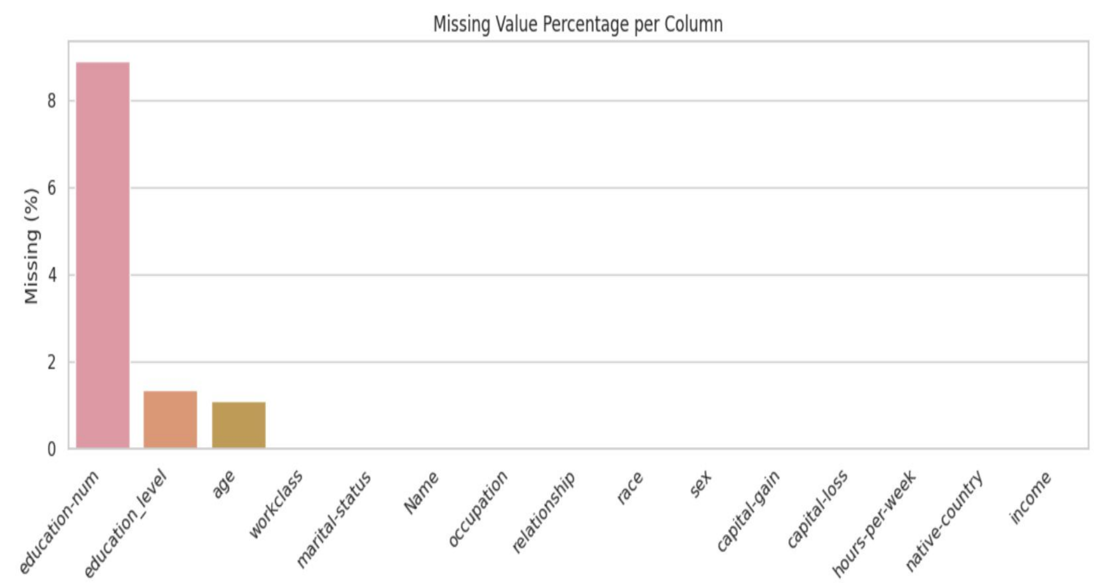
For numeric fields (age, education-num), we used **median imputation** and added **missing-value indicator flags** to preserve information.

For education_level, missing entries were **replaced** with "Unknown" to avoid losing categorical information.

```python
#  Handle missing feature values
if "age" in df.columns:,
    df["age_missing_flag"] = df["age"].isna().astype("int8")
    df["age"] = df["age"].fillna(df["age"].median()).astype("int64")

# education_num: median imputation + int
if "education-num" in df.columns:
    df["edu_num_missing_flag"] = df["education-num"].isna().astype("int8")
    df["education-num"] = (
        df["education-num"]
          .fillna(df["education-num"].median())
          .astype("int64")
    )


# education_level: fill missing with 'Unknown'
if "education_level" in df.columns:
    df["education_level"] = df["education_level"].fillna("Unknown")
```



Imbalanced Classes

- **Catboost cost sensitive learning**

pos_weight = float((len(y_train) - y_train.sum()) / y_train.sum())

cat_model = CatBoostClassifier(…

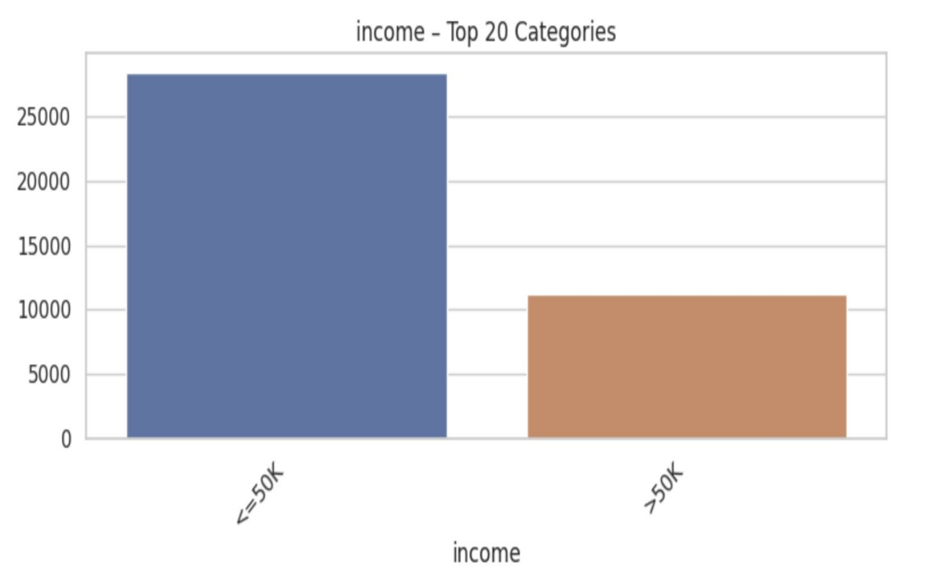
class_weights=[1.0, pos_weight],

…)

- **Evaluation metrics that make sense with imbalance data:**

ROC AUC, PR AUC, Precision & Recall at the %9.5 cutoff

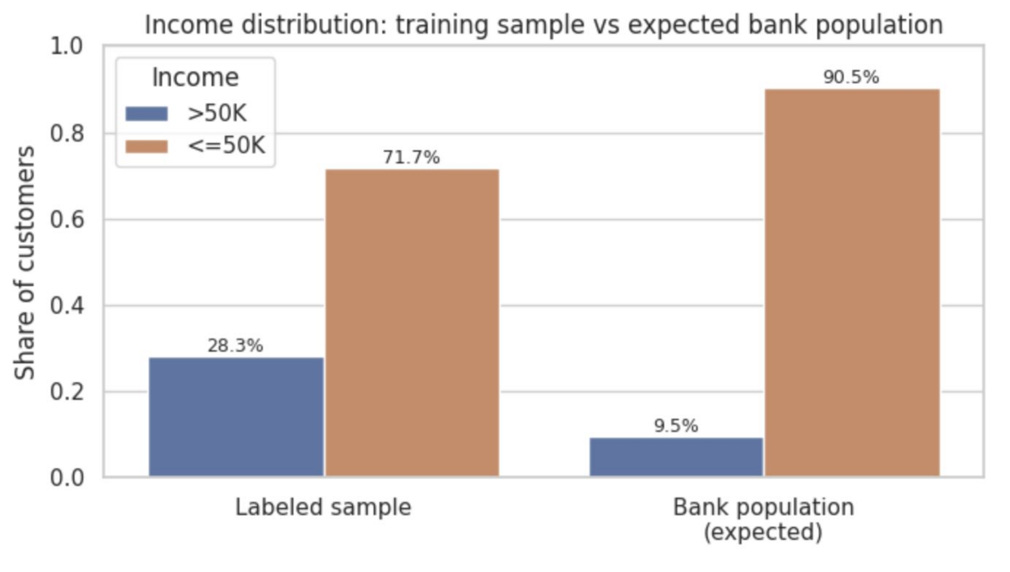
Next Steps: Try calibrated classifier to adjust predicted probabilities



Wealth Bias in the Labeled Sample

“We expect total of 2 million customers that fulfills the target out of the banks 21 million total customer base”

Our labeled sample is much richer than the real bank, so raw probabilities are too optimistic.



Name

Their high-income rates are extreme:

- Bernie: ~97% high income
- Johanna: ~97%
- Audrey, Billie, Karen: 0% high income
- Also, missing income is mostly on Audrey, Billie, Karen; almost never on Bernie/Johanna.

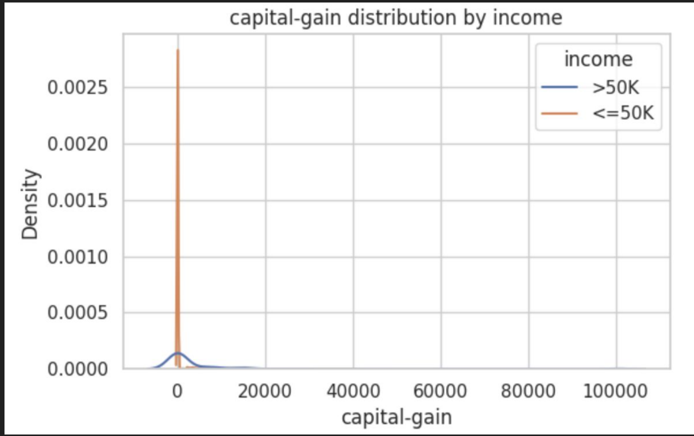
Name encodes the target by construction (**data leak)**. If we keep it, the model will just learn ‘Bernie/Johanna = rich’ and won’t generalise. Therefore, we drop Name entirely before modeling.

Feature Engineering

Capital Gain & Loss

These are very skewed: most people have 0, but when non-zero, it’s a huge signal.

- **capital-gain > 0**
  - ~9% of people
  - high income rate ≈ 66.8%
  - about 21% of all >50K customers.
- **capital-loss > 0**
  - ~5% of people
  - high income rate ≈ 55.7%



Any sign of investment gains/losses is a strong indicator of wealth, but only for a minority of customers.

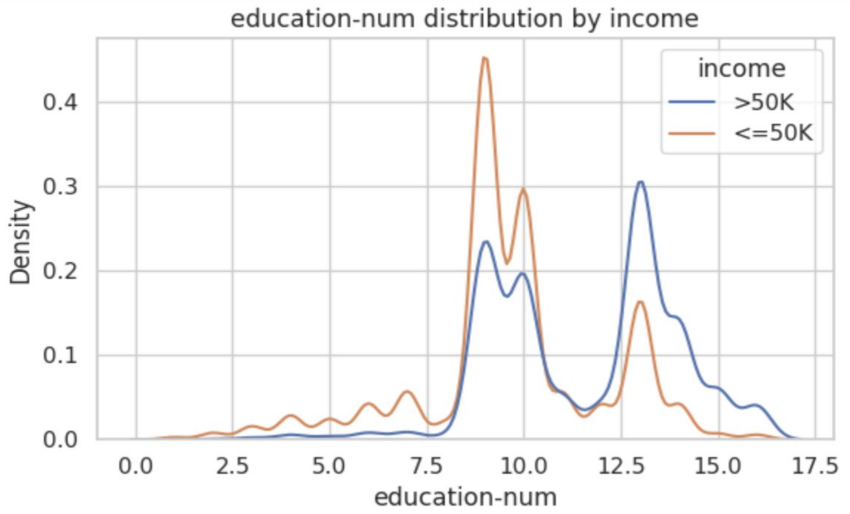
```
# ----- capital features -----
df_model["has_capital_gain"] = (df_model["capital-gain"] > 0).astype(int)
df_model["has_capital_loss"] = (df_model["capital-loss"] > 0).astype(int)
```

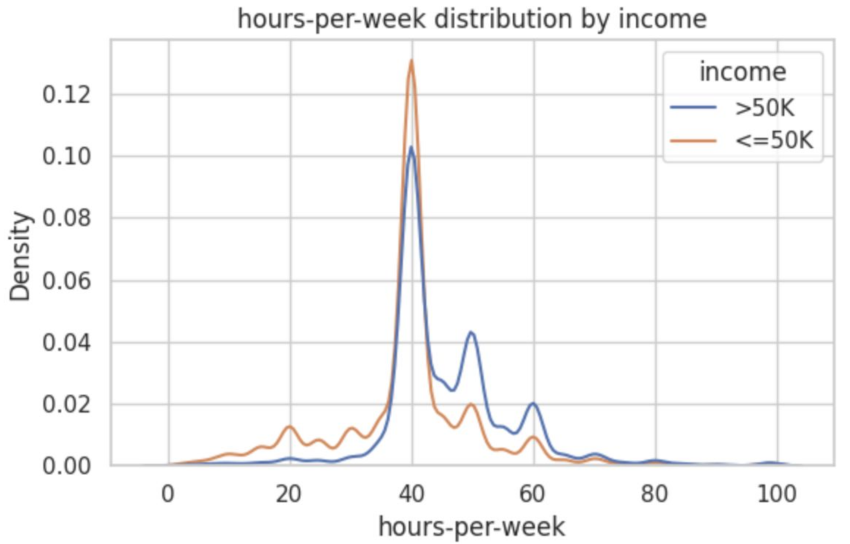
Binned features

Captures meaningful ranges instead of treating every number separately.

```
# ----- age features -----
age_bins   = [0, 25, 35, 45, 55, 65, 120]
age_labels = ["<25", "25-34", "35-44", "45-54", "55-64", "65+"]
df_model["age_group"] = pd.cut(df_model["age"], bins=age_bins, labels=age_labels, right=False)

# ----- hours features -----
hours_bins   = [0, 35, 41, 46, 61, 200]
hours_labels = ["<35", "35-40", "41-45", "46-60", "60+"]
df_model["hours_group"] = pd.cut(df_model["hours-per-week"], bins=hours_bins,
                                 labels=hours_labels, right=False)
df_model["long_hours_flag"] = (df_model["hours-per-week"] >= 46).astype(int)
```

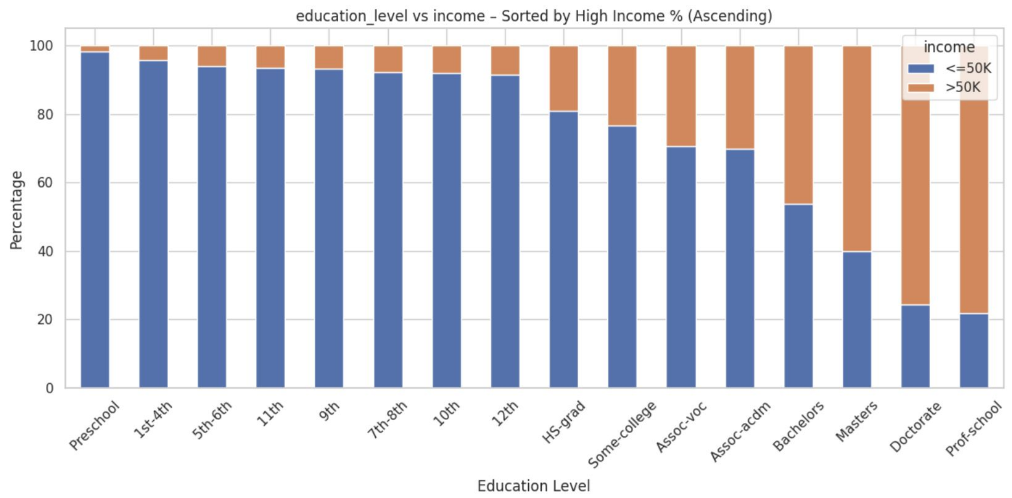




Education score

to express a **strong monotonic relationship**

We convert ordered categories into a single numeric score so the model can learn this trend directly. This reduces noise and improves consistency across similar education levels.




```
# ----- education features -----
edu_order = [
    "Preschool","1st-4th","5th-6th","7th-8th","9th","10th","11th","12th",
    "HS-grad","Some-college","Assoc-voc","Assoc-acdm",
    "Bachelors","Masters","Prof-school","Doctorate"
]
edu_map = {edu: i for i, edu in enumerate(edu_order)}
df_model["education_level"] = df_model["education_level"].fillna("Unknown")
df_model["education_score"] = df_model["education_level"].map(edu_map).fillna(-1)
```

Interaction Effects #1

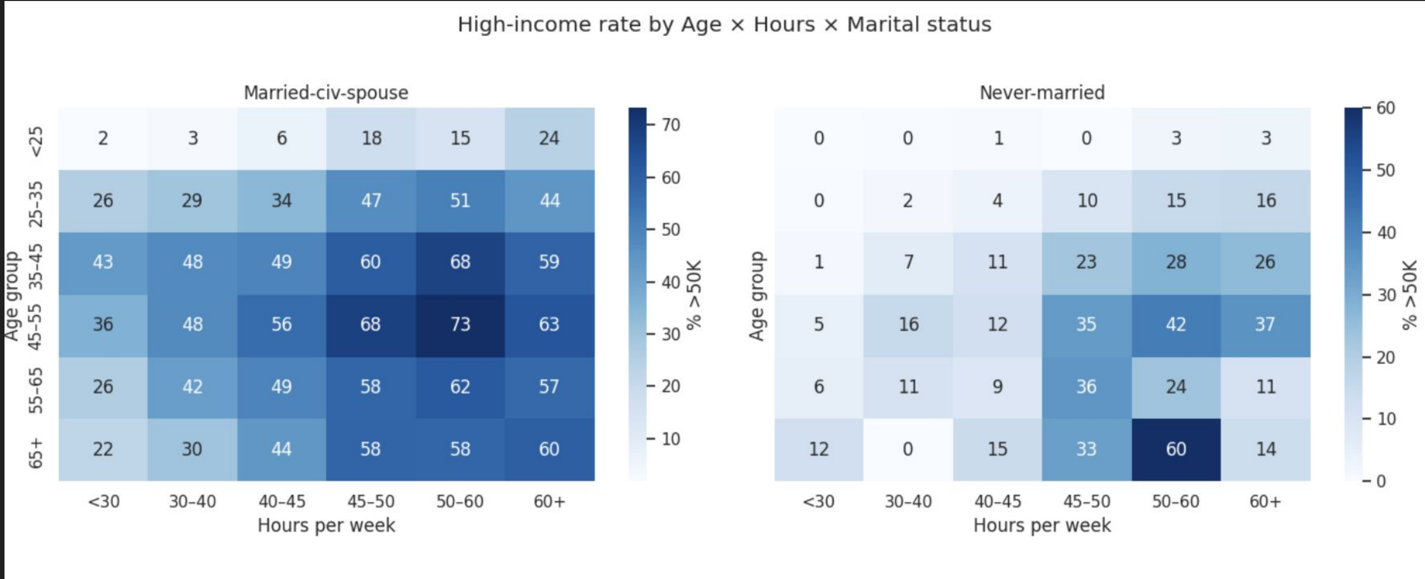
```python
married_statuses = {"Married-civ-spouse", "Married-AF-spouse", "Married-spouse-absent"}

# High-income profile: Married + Age 40–60 + 45+ hours/week
df_model["married_midage_longhours"] = (
    df_model["marital-status"].isin(married_statuses)
    & (df_model["age"] >= 40)
    & (df_model["age"] <= 60)
    & (df_model["hours-per-week"] >= 45)
).astype(int)

# Low-income profile: Never-married + Age <35
df_model["young_never_married"] = (
    (df_model["marital-status"] == "Never-married")
    & (df_model["age"] < 35)
).astype(int)
```

Marital Status + Age + Hours Worked

- Married + Age 40–60 + 45+ hrs/week → **>60%** high-income rate
- Never-married + Age <35 → **<10%**



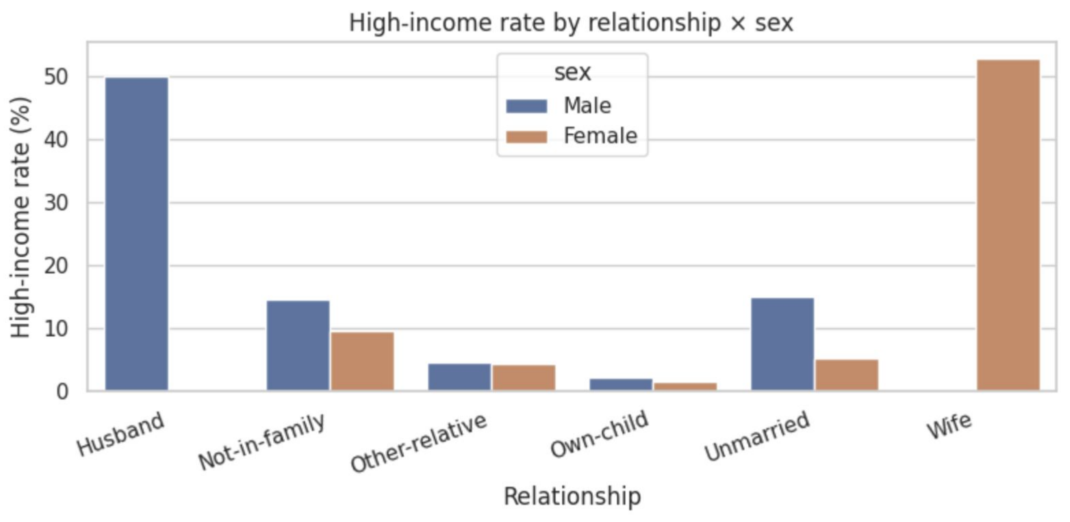
Interaction Effects #2

Relationship + Sex

- “Husband” relationship code → **~45–55%**
- “Own-child” → **<5%**

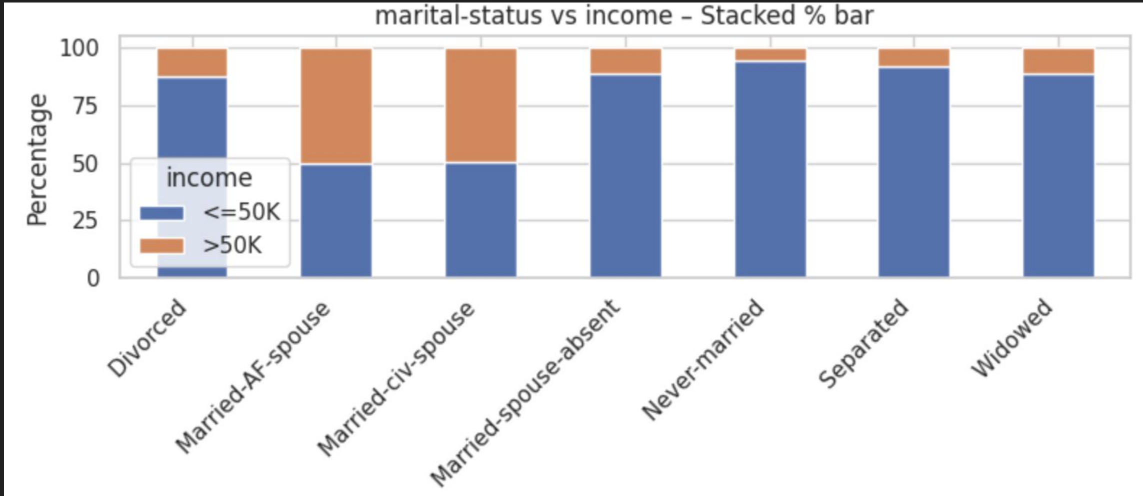
```python
# High-income household heads
df_model["rel_husband_male"] = (
    (df_model["relationship"] == "Husband") &
    (df_model["sex"] == "Male")
).astype(int)

df_model["rel_wife_female"] = (
    (df_model["relationship"] == "Wife") &
    (df_model["sex"] == "Female")
).astype(int)
```



Marital Status

To express similarity add grouped features of "is_married", "is_in_couple" and "has_children".

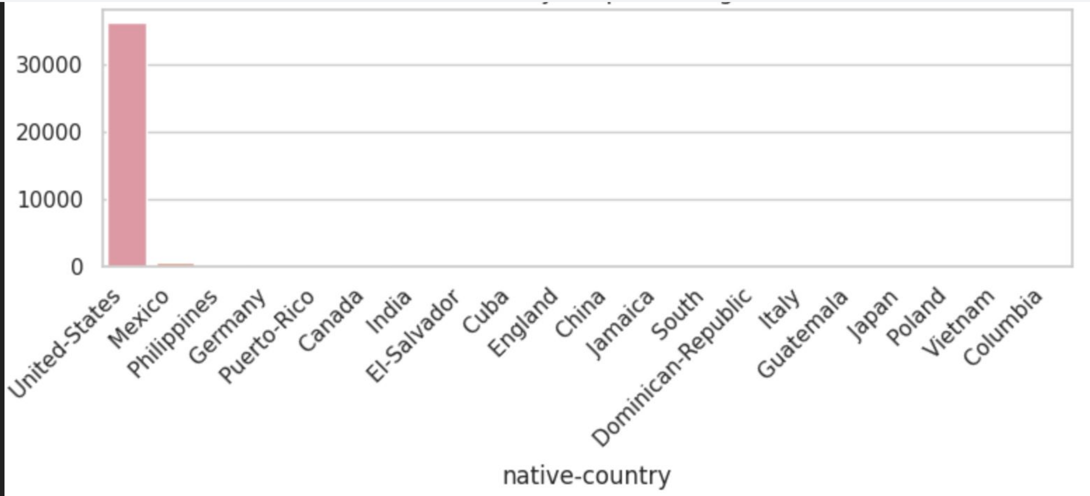



```python
# ----- marital / relationship -----
married_statuses = {"Married-civ-spouse", "Married-AF-spouse", "Married-spouse-absent"}
df_model["is_married"]   = df_model["marital-status"].isin(married_statuses).astype(int)
df_model["is_in_couple"] = df_model["relationship"].isin({"Husband","Wife"}).astype(int)
df_model["has_children"] = df_model["relationship"].isin({"Own-child","Other-relative"}).astype(int)
```

Is American

The native-country distribution is extremely imbalanced toward the U.S.

```python
# ----- geography -----
df_model["is_US"] = (df_model["native-country"] == "United-States").astype(int)
```

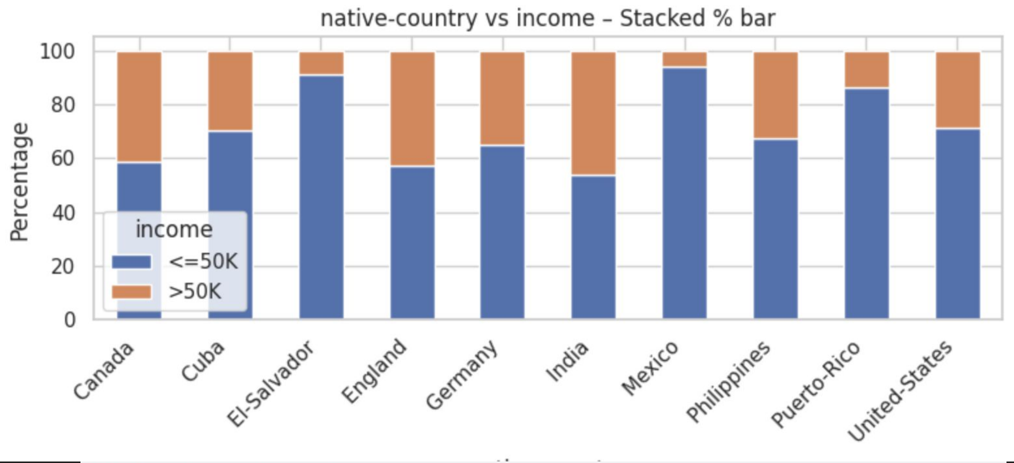


Region

Reduces Noise from Rare Categories

Income Patterns Differ Strongly by Region.

Region feature helps capture macro-economic effects that single-country categories miss.




```
# Simple region grouping example
europe = {"England","Germany","Portugal","France","Italy","Ireland","Scotland","Greece","Holand-Netherlands"}
latin_america = {"Mexico","Cuba","Jamaica","Honduras","Puerto-Rico","Columbia","Ecuador",
                 "Trinadad&Tobago","Peru","Nicaragua","El-Salvador","Guatemala","Haiti","Dominican-Republic"}
asia = {"Philippines","India","Japan","China","Vietnam","Laos","Cambodia","Thailand","Taiwan","Hong"}

def map_region(c):
    if c in europe: return "Europe"
    if c in latin_america: return "LatAm"
    if c in asia: return "Asia"
    if c == "United-States": return "US"
    return "Other"

df_model["country_region"] = df_model["native-country"].map(map_region)
```

Model

Why Catboost?

Easy to use with categorical data

Class-weights feature

“PROUC” performance measure

Hyperparameter Optimization

CV + Grid Search + Optuna

Search space design: We exposed the usual high-leverage CatBoost knobs: depth, learning_rate, iterations, regularization (l2_leaf_reg), and maybe some tree-level constraints / sampling rates. We kept the space reasonably tight for Optuna, based on prior experience and a few manual runs, so we're not wasting budget in obviously bad regions.

Stage 1 – Grid Search: provides sweet starting point for Optuna

Stage 2 – Optuna (Bayesian / TPE + pruning): Provides even better hyperparameters

```
  Best params  : {'depth': 6, 'learning_rate': 0.06284728302324294, 'l2_leaf_reg': 2.3532353602061975,
'iterations': 1179}
```

How we evaluated the model?

Goal: “Predict which customers have annual income > $50k so we can select ~2M of 21M customers (~9.5%) for a premium campaign.”

When we pick the top 10% highest-scoring customers, 96% of them are truly high-income.

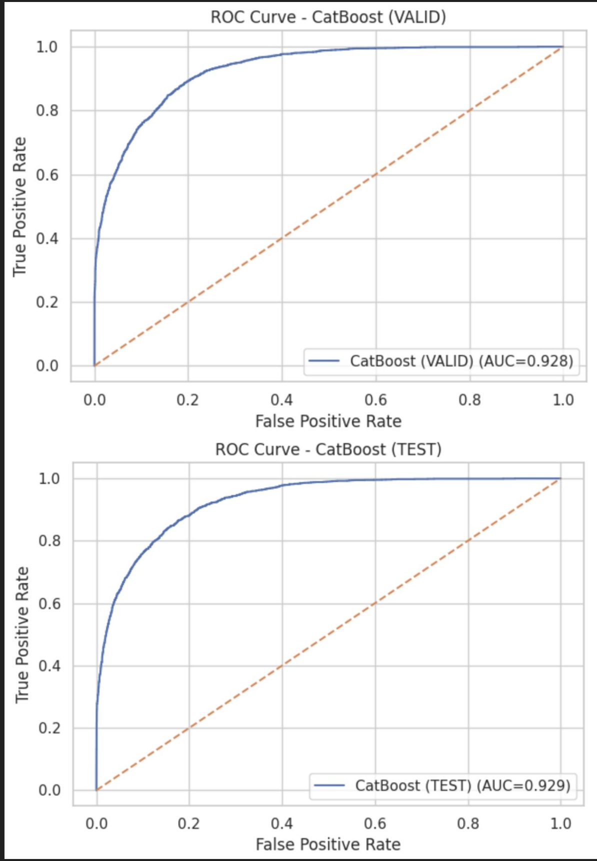
```
Top-K precision / recall (how well we find high-income customers):
 Top  1% -> precision=1.0000, recall=0.0352
 Top  5% -> precision=0.9975, recall=0.1762
 Top 10% -> precision=0.9735, recall=0.3443
 Top 20% -> precision=0.8449, recall=0.5977
```

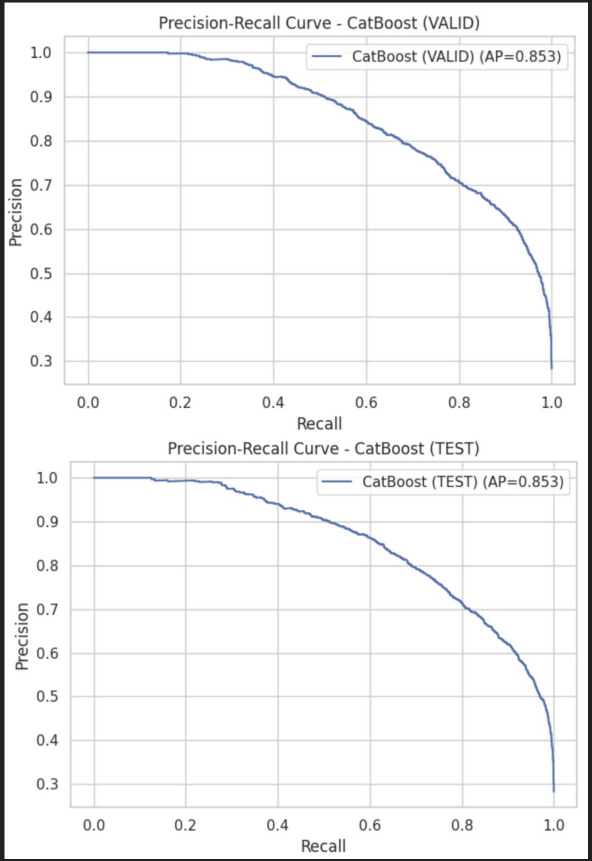
VALIDATION

```
Top-K precision / recall (how well we find high-income customers):
 Top  1% -> precision=1.0000, recall=0.0353
 Top  5% -> precision=0.9924, recall=0.1754
 Top 10% -> precision=0.9622, recall=0.3405
 Top 20% -> precision=0.8588, recall=0.6078
```

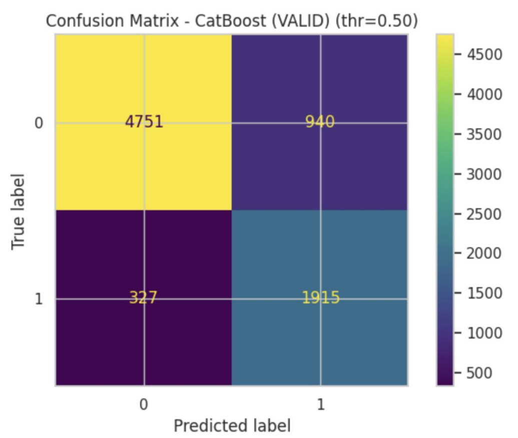
TESTING

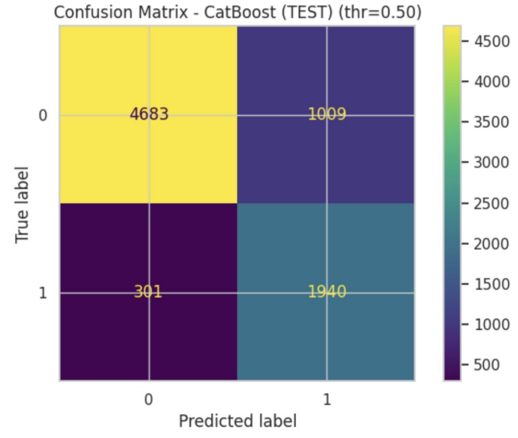
Metrics #1: ROC-AUC, Precision-Recall AUC





Metrics #2: Confusion Matrix, Precision, Recall, F1





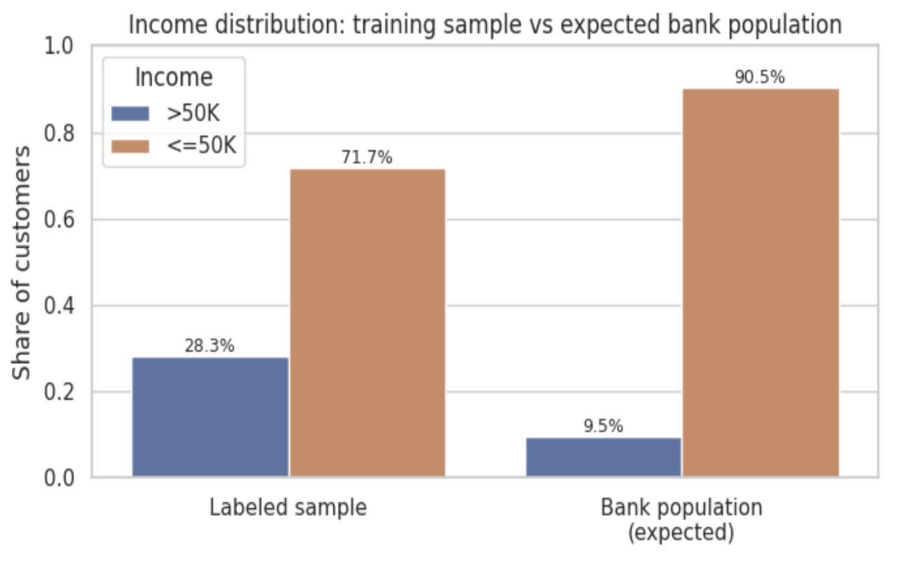
Accuracy	: 0.8403
Precision	: 0.6708
Recall	: 0.8541
F1-score	: 0.7514

Accuracy	: 0.8349
Precision	: 0.6579
Recall	: 0.8657
F1-score	: 0.7476

Wealth Overestimation in Labeled sample

Among the customers where we know income, about 28% are above $50k. But you told us that in the full bank, only about 9.5% are above $50k.

So if we just trained a model on the labeled sample, it would think the world is richer than it really is and would assign probabilities that are too high. We keep the ranking — who looks richer than whom — but we shrink the probabilities so that, on average, they match the 9.5% reality. That's what calibration does.



Pick Customers with the best probability

“We expect total of 2 million customers that fulfills the target out of the banks 21 million total customer base”

%9.5 percent of the customers with the best probability of being high income should be chosen when model is applied to main dataset.

What to improve

Better handling of missing values (e.g. KNN imputer)

Trying other algorithms and comparing metrics

Simulate the real 9.5% positive rate by downsampling in testing part

Simplifying features for easier deployment

Voting Classifier or Ensemble Model