📦Task 2: Customer Churn Prediction — End-to-End Starter Kit

This kit gives you a complete, reproducible path to build a churn model, generate probabilities, and create an insights dashboard in Power BI. It's designed for telecom or banking churn data.

---

1) Project Structure

```
churn-task2/
├── data/
│   ├── raw/                   # put original CSV here (e.g., telco_churn.csv)
│   └── processed/             # train/test splits, encoded data (auto-created)
├── models/                    # saved models (.joblib)
├── outputs/
│   ├── figures/               # ROC, PR, calibration plots
│   └── exports/               # churn_probabilities.csv for Power BI
├── notebooks/                 # optional EDA
├── src/
│   ├── config.py
│   ├── eda.py                 # optional EDA helpers
│   └── train.py               # MAIN script (run this)
├── requirements.txt
└── README.md
```

`requirements.txt`

```
pandas>=2.0
numpy>=1.24
scikit-learn>=1.3
xgboost>=2.0
matplotlib>=3.7
joblib>=1.3
shap>=0.45
```

> If `xgboost`/`shap` are hard to install on your machine, you can comment out those sections in `train.py` and still complete the task using Logistic Regression or Random Forest.

---

2) Data Assumptions (Customize as needed)

Your dataset should have: - **Target column**: `Churn` (values: `Yes` / `No` or `1` / `0`). - **Customer ID**: `customerID` or `CustomerId` (string). - **Feature types**: - **Categorical**: plan type, contract, internet service, region, gender, etc. - **Numeric**: tenure, monthly charges, total charges, balance, transactions, etc.

Update `config.py` if your column names differ.

Example `src/config.py`

```python
# Basic config — edit to match your file & schema
RAW_DATA_PATH = "data/raw/telco_churn.csv"  # or banking_churn.csv
CUSTOMER_ID_COL = "customerID"
TARGET_COL = "Churn"  # values: Yes/No or 1/0

# Provide lists if you want to force types; leave [] to auto-detect
FORCE_NUMERIC = []
FORCE_CATEGORICAL = []

# Business costs (optional): cost of false negative (missed churn) vs false
positive (unnecessary retention)
COST_FN = 5.0
COST_FP = 1.0

SEED = 42
N_FOLDS = 5
TEST_SIZE = 0.2
```

---

3) Main Training Script (`src/train.py`)

Copy the script below to `src/train.py` . Run with `python src/train.py` .

```python
import os, warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
from pathlib import Path

from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_predict
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import (
    roc_auc_score, average_precision_score, f1_score, accuracy_score,
    precision_recall_curve, roc_curve, brier_score_loss
)
from sklearn.calibration import CalibrationDisplay
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.calibration import CalibratedClassifierCV
import matplotlib.pyplot as plt
from joblib import dump

try:
    import xgboost as xgb
    HAS_XGB = True
except Exception:
    HAS_XGB = False

# ---- Config ----
from config import (
    RAW_DATA_PATH, CUSTOMER_ID_COL, TARGET_COL, FORCE_NUMERIC,
FORCE_CATEGORICAL,
    COST_FN, COST_FP, SEED, N_FOLDS, TEST_SIZE
)

# ---- Paths ----
BASE = Path(__file__).resolve().parents[1]
DATA_DIR = BASE / "data"
PROC_DIR = DATA_DIR / "processed"
MODEL_DIR = BASE / "models"
OUT_DIR = BASE / "outputs"
FIG_DIR = OUT_DIR / "figures"
EXP_DIR = OUT_DIR / "exports"
for d in [PROC_DIR, MODEL_DIR, FIG_DIR, EXP_DIR]:
    d.mkdir(parents=True, exist_ok=True)

# ---- Load ----
df = pd.read_csv(RAW_DATA_PATH)
assert CUSTOMER_ID_COL in df.columns, f"Missing {CUSTOMER_ID_COL}"
assert TARGET_COL in df.columns, f"Missing {TARGET_COL}"

# Normalize target to {0,1}
if df[TARGET_COL].dtype == object:
    df[TARGET_COL] = df[TARGET_COL].str.strip().str.lower().map({"yes":1, "no":
0, "y":1, "n":0, "true":1, "false":0})
    df[TARGET_COL] = df[TARGET_COL].fillna(df[TARGET_COL])
```

```
df[TARGET_COL] = df[TARGET_COL].astype(int)

# Identify feature types
features = [c for c in df.columns if c not in [CUSTOMER_ID_COL, TARGET_COL]]
num_cols = [c for c in features if (np.issubdtype(df[c].dtype, np.number)) or (c
in FORCE_NUMERIC)]
cat_cols = [c for c in features if c not in num_cols or (c in
FORCE_CATEGORICAL)]

# Common data issues
for col in num_cols:
    if df[col].dtype == object:
        # coerce numeric-like text (e.g., ' 100 ') to float
        df[col] = pd.to_numeric(df[col].astype(str).str.replace(",", "",
regex=False).str.strip(), errors='coerce')

# Train/test split
X = df[features]
y = df[TARGET_COL]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=TEST_SIZE, stratify=y, random_state=SEED
)

# Preprocess
numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler(with_mean=False))  # with_mean=False keeps sparse
safety
])

categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore", sparse=True))
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, num_cols),
        ("cat", categorical_transformer, cat_cols)
    ]
)

# Models
models = {
    "logreg": LogisticRegression(max_iter=200, n_jobs=None,
class_weight="balanced"),
    "rf": CalibratedClassifierCV(
        base_estimator=RandomForestClassifier(n_estimators=400,
random_state=SEED, n_jobs=-1, class_weight="balanced"),
        method="isotonic",
        cv=3
    )
}

if HAS_XGB:
    models["xgb"] = xgb.XGBClassifier(
        n_estimators=500,
        learning_rate=0.05,
        max_depth=6,
        subsample=0.8,
        colsample_bytree=0.8,
        eval_metric="logloss",
        tree_method="hist",
        random_state=SEED,
        n_jobs=-1,
        scale_pos_weight=(y_train.value_counts()[0] / y_train.value_counts()[1])
    )

results = []

for name, clf in models.items():
    pipe = Pipeline(steps=[("prep", preprocess), ("clf", clf)])

    # CV probabilities for train set (out-of-fold)
    skf = StratifiedKFold(n_splits=N_FOLDS, shuffle=True, random_state=SEED)
    oof_proba = cross_val_predict(pipe, X_train, y_train, cv=skf,
method="predict_proba", n_jobs=-1)[:, 1]

    # Fit on full train
    pipe.fit(X_train, y_train)

    # Evaluate on test
    test_proba = pipe.predict_proba(X_test)[:, 1]

    # Metrics
    def metrics(y_true, p):
        return {
            "roc_auc": roc_auc_score(y_true, p),
            "pr_auc": average_precision_score(y_true, p),
            "brier": brier_score_loss(y_true, p),
        }

    m_train = metrics(y_train, oof_proba)
    m_test = metrics(y_test, test_proba)

    # Optimal threshold (cost-sensitive)
```

```
prec, rec, thr = precision_recall_curve(y_train, oof_proba)
# Expected cost = FN*C_FN + FP*C_FP → choose threshold minimizing cost
# Approximate FP/FN using prec/rec and class priors
p_pos = y_train.mean()
p_neg = 1 - p_pos
costs = []
for t in np.r_[thr, 1.0]:  # align thresholds length
    pred = (oof_proba >= t).astype(int)
    fn = ((y_train==1) & (pred==0)).sum()
    fp = ((y_train==0) & (pred==1)).sum()
    costs.append(fn*COST_FN + fp*COST_FP)
costs = np.array(costs)
best_idx = int(costs.argmin())
best_thr = float(np.r_[thr, 1.0][best_idx])

# Discrete metrics at threshold
def discrete_scores(y_true, p, t):
    pred = (p >= t).astype(int)
    return {
        "accuracy": accuracy_score(y_true, pred),
        "f1": f1_score(y_true, pred, zero_division=0)
    }

d_train = discrete_scores(y_train, oof_proba, best_thr)
d_test = discrete_scores(y_test, test_proba, best_thr)

results.append({
    "model": name,
    "best_threshold": best_thr,
    **{f"train_{k}": v for k, v in m_train.items()},
    **{f"test_{k}": v for k, v in m_test.items()},
    **{f"train_{k}": v for k, v in d_train.items()},
    **{f"test_{k}": v for k, v in d_test.items()},
})

# Save model
dump(pipe, MODEL_DIR / f"{name}_pipeline.joblib")

# Plots
fpr, tpr, _ = roc_curve(y_test, test_proba)
plt.figure()
plt.plot(fpr, tpr, label=f"{name} ROC")
plt.plot([0,1],[0,1],'--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title(f"ROC Curve — {name}")
plt.legend(loc="lower right")
```

```
    plt.savefig(FIG_DIR / f"roc_{name}.png", dpi=160, bbox_inches='tight')
    plt.close()

    prec, rec, _ = precision_recall_curve(y_test, test_proba)
    plt.figure()
    plt.plot(rec, prec, label=f"{name} PR")
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title(f"Precision-Recall Curve — {name}")
    plt.legend(loc="lower left")
    plt.savefig(FIG_DIR / f"pr_{name}.png", dpi=160, bbox_inches='tight')
    plt.close()

    plt.figure()
    CalibrationDisplay.from_predictions(y_test, test_proba)
    plt.title(f"Calibration — {name}")
    plt.savefig(FIG_DIR / f"calibration_{name}.png", dpi=160,
bbox_inches='tight')
    plt.close()

# Pick best by test PR AUC (handles imbalance well)
res_df = pd.DataFrame(results).sort_values("test_pr_auc", ascending=False)
res_df.to_csv(EXP_DIR / "model_comparison.csv", index=False)

best_model_name = res_df.iloc[0]["model"]
print("Best model:", best_model_name)

# Reload best model and export probabilities for Power BI
from joblib import load
best_model = load(MODEL_DIR / f"{best_model_name}_pipeline.joblib")

# Export churn probabilities for the entire dataset (train+test)
proba_all = best_model.predict_proba(X)[:,1]
export = pd.DataFrame({
    CUSTOMER_ID_COL: df[CUSTOMER_ID_COL],
    "churn_probability": proba_all,
    TARGET_COL: y,
})

# Add risk banding for business consumption
export["risk_band"] = pd.qcut(export["churn_probability"], q=5, labels=["Very
Low","Low","Medium","High","Very High"])

export.to_csv(EXP_DIR / "churn_probabilities.csv", index=False)
print("Saved:", EXP_DIR / "churn_probabilities.csv")

# (Optional) Global feature importance via permutation on test set
from sklearn.inspection import permutation_importance
```

```
perm = permutation_importance(best_model, X_test, y_test, n_repeats=5,
random_state=SEED, n_jobs=-1)
imp = pd.DataFrame({"feature": features, "importance":
perm.importances_mean}).sort_values("importance", ascending=False)
imp.to_csv(EXP_DIR / "feature_importance_permutation.csv", index=False)

# (Optional) SHAP for tree or linear models
try:
    import shap
    explainer = None
    if HAS_XGB and best_model_name == "xgb":
        explainer = shap.TreeExplainer(best_model.named_steps["clf"])
    elif best_model_name == "rf":
        # TreeExplainer works for RF too
        explainer =
shap.TreeExplainer(best_model.named_steps["clf"].base_estimator)
    elif best_model_name == "logreg":
        explainer = shap.LinearExplainer(best_model.named_steps["clf"],
best_model.named_steps["prep"].transform(X_train))
    if explainer is not None:
        # Use a sample to keep it light
        X_sample = X_test.sample(min(1000, len(X_test)), random_state=SEED)
        shap_values =
explainer.shap_values(best_model.named_steps["prep"].transform(X_sample))
        shap.summary_plot(shap_values, show=False)
        plt.title(f"SHAP Summary — {best_model_name}")
        plt.savefig(FIG_DIR / f"shap_summary_{best_model_name}.png", dpi=160,
bbox_inches='tight')
        plt.close()
except Exception as e:
    print("SHAP skipped:", e)
```

---

4) Quick How-To Run

1. **Place your CSV** at `data/raw/` and update `src/config.py` paths/column names.
2. Create a virtual env and install packages:

   ```
   pip install -r requirements.txt
   ```

3. Run training:

   ```
   python src/train.py
   ```

4. Outputs:
5. `outputs/exports/churn_probabilities.csv` → **import this into Power BI**.
6. `outputs/exports/model_comparison.csv` → choose best model.
7. `outputs/exports/feature_importance_permutation.csv`.
8. `outputs/figures/*` → ROC/PR/Calibration/SHAP plots.
9. `models/*_pipeline.joblib` → saved models.

---

5) Power BI Dashboard Blueprint

Data sources - `churn_probabilities.csv` - (Optional) customer dimension tables: demographics, product/plan, usage, complaints, geography - (Optional) campaign cost table for ROI

Recommended pages 1. **Executive Overview** - KPIs: Overall Churn Rate, At-Risk % (prob ≥ threshold), Monthly Trend - Visuals: Churn by Segment (bar), Risk Band Distribution (stacked bar), Map (region) 2. **Drivers & Insights** - Feature Importance (from CSV) - SHAP summary image (import as an image) - Drilldown tables: Top at-risk customers with key drivers 3. **Retention Playbook** - What-If parameter: threshold slider (0.1–0.9) - Measures: predicted churners, expected cost, expected savings - Visual: Gains/Lift chart (create using deciles of probability)

Core DAX measures (adjust table/column names):

```
Churners := SUM('churn_probabilities'[Churn])
Customers := COUNTROWS('churn_probabilities')
Churn Rate := DIVIDE([Churners],[Customers])

At Risk Customers :=
VAR t = SELECTEDVALUE('Threshold'[Value], 0.5)
RETURN COUNTROWS(FILTER('churn_probabilities',
'churn_probabilities'[churn_probability] >= t))

Avg Risk Score := AVERAGE('churn_probabilities'[churn_probability])
```

Gains/Lift Setup - Create deciles by probability using `TopN` or calculated column with `RANKX`. - Plot cumulative % of churn captured vs % of customers targeted.

---

6) Evaluation Criteria (what to report)

- Data description and cleaning steps.
- Class balance and handling (class_weight, scale_pos_weight, thresholding).
- Metrics: **PR AUC**, ROC AUC, Brier, F1/Accuracy at chosen threshold.
- Calibration plot and final threshold rationale (business cost-based).
- Top drivers (Permutation / SHAP) and segment insights.

- Dashboard screenshots with key takeaways.

---

7) Business Recommendations (Template)

- **Segmented offers**: e.g., discounts for high-ARPU high-risk; proactive outreach for new-tenure high-risk.
- **Product fixes**: if churn correlates with specific plan/feature, prioritize stability or pricing adjustments.
- **Onboarding**: if early-tenure customers churn more, strengthen first-90-days education and incentives.
- **Support improvements**: high complaints → churn; reduce resolution time, offer callbacks.
- **Contract strategy**: move month-to-month high-risk customers to flexible but sticky bundles.
- **Experimentation**: A/B test retention actions; track uplift vs control.

ROI Sketch

```
Expected Savings = (# targeted) * (response_rate) * (avg_margin *
retained_months) - (campaign_cost)
```

---

8) Submission Checklist

- [ ] Code repo with `src/`, `requirements.txt`, and `README.md`.
- [ ] `model_comparison.csv`, figures, and `churn_probabilities.csv` exported.
- [ ] Power BI `.pbix` with 3 pages (Overview, Drivers, Playbook).
- [ ] One-pager summary of findings and business recommendations.

---

9) Tips

- Start with **Logistic Regression** baseline; add **Random Forest/XGBoost**.
- Prefer **PR AUC** for imbalanced churn.
- Use **cost-sensitive threshold** (provided) instead of default 0.5.
- Keep a **data dictionary** and track all transformations.

Drop your CSV column names if you want me to tailor `config.py` for you.