Churn Prediction — Split repo + Notebook

I've created a split, production-ready layout and a runnable notebook. Below are the files included (each file is provided in full inside this document). Save each into your repo root as shown and push to GitHub.

Files added

- `train.py` — training entrypoint (uses `utils.py`)
- `predict.py` — simple prediction CLI for saved models
- `utils.py` — shared helpers: data loading, preprocessing builder, model factory, synthetic data generator
- `tests/test_pipeline.py` — pytest tests that run a quick smoke-training on synthetic data
- `notebook/churn_exploration.py` — a Jupyter-friendly `.py` notebook (with `# %%` cells) that demonstrates dataset, training, evaluation, and sample plots

---

Instruction

1. Create a repo and add these files with the exact filenames above.
2. Run `python train.py --out_dir output --model random_forest` to train (or pass `--data data/churn_data.csv`).
3. Run `python predict.py --model_path models/churn_model_random_forest.pkl --input data/sample_input.csv --out_dir output` to produce predictions.
4. Run tests with `pytest -q`.
5. Open the notebook file in VS Code or Jupyter (it is a `.py` with cell markers).

<details> <summary>train.py (click to expand)</summary>

```
# train.py
"""Train entrypoint for the churn prediction project.
Usage:
  python train.py --data data/churn_data.csv --out_dir output --model
random_forest
"""

import argparse
import logging
import os
from utils import load_data, train_and_evaluate, write_repo_files, init_logging
```

```
def parse_args():
    import argparse
    p = argparse.ArgumentParser()
    p.add_argument('--data', type=str, default=None)
    p.add_argument('--out_dir', type=str, default='output')
    p.add_argument('--model', type=str, default='random_forest',
choices=['random_forest','lightgbm','xgboost'])
    p.add_argument('--apply_smote', action='store_true')
    p.add_argument('--init_repo', action='store_true')
    p.add_argument('--random_state', type=int, default=42)
    return p.parse_args()


def main():
    args = parse_args()
    if args.init_repo:
        write_repo_files('.')
        print('Wrote requirements.txt and README.md')
        return

    logger = init_logging(args.out_dir)
    df = load_data(args.data, logger=logger)
    metrics, model_path = train_and_evaluate(df, target='churn',
out_dir=args.out_dir, model_name=args.model, apply_smote=args.apply_smote,
random_state=args.random_state, logger=logger)
    logger.info('Done. Metrics: %s', metrics)


if __name__ == '__main__':
    main()
```

</details>

<details> <summary>predict.py (click to expand)</summary>

```
# predict.py
"""Simple prediction CLI. Provide model path and an input CSV (rows without the
target) and it writes predictions to out_dir/predictions.csv"""

import argparse
import pandas as pd
import joblib
import os
```

```
def parse_args():
    p = argparse.ArgumentParser()
    p.add_argument('--model_path', type=str, required=True)
    p.add_argument('--input', type=str, required=True)
    p.add_argument('--out_dir', type=str, default='output')
    return p.parse_args()


def main():
    args = parse_args()
    os.makedirs(args.out_dir, exist_ok=True)
    model = joblib.load(args.model_path)
    df = pd.read_csv(args.input)
    preds = model.predict(df)
    proba = model.predict_proba(df)[:,1] if hasattr(model, 'predict_proba') else
None
    out = pd.DataFrame({'y_pred': preds})
    if proba is not None:
        out['y_proba'] = proba
    if 'customer_id' in df.columns:
        out['customer_id'] = df['customer_id']
    out.to_csv(os.path.join(args.out_dir, 'predictions.csv'), index=False)
    print('Wrote predictions to', os.path.join(args.out_dir, 'predictions.csv'))

if __name__ == '__main__':
    main()
```

</details>

<details> <summary>utils.py (click to expand)</summary>

```
# utils.py
"""Shared utilities: data load, preprocessing builder, model factory, synthetic
generator, trainer.
"""

import os
import json
import numpy as np
import pandas as pd
from pathlib import Path
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, RandomizedSearchCV,
StratifiedKFold
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
```

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score, classification_report
from sklearn.utils import check_random_state
import joblib

# Optional imports
try:
    import lightgbm as lgb
    HAS_LGB = True
except Exception:
    HAS_LGB = False
try:
    import xgboost as xgb
    HAS_XGB = True
except Exception:
    HAS_XGB = False
try:
    from imblearn.over_sampling import SMOTE
    HAS_SMOTE = True
except Exception:
    HAS_SMOTE = False

DEFAULT_RANDOM_STATE = 42


def generate_synthetic_data(n_samples=1000, random_state=DEFAULT_RANDOM_STATE):
    rng = check_random_state(random_state)
    df = pd.DataFrame()
    df['customer_id'] = np.arange(1, n_samples+1)
    df['tenure_months'] = rng.poisson(24, n_samples)
    df['monthly_charges'] = np.round(rng.normal(70,30,n_samples).clip(10,300),2)
    df['num_support_tickets'] = rng.poisson(1.2, n_samples)
    df['contract_type'] = rng.choice(['month-to-month','one-year','two-year'],
size=n_samples, p=[0.6,0.25,0.15])
    df['is_senior'] = rng.binomial(1,0.12,size=n_samples)
    logits = -0.02*df['tenure_months'] + 0.015*df['num_support_tickets'] +
0.4*(df['contract_type']=='month-to-month').astype(int)
    probs = 1/(1+np.exp(-logits))
    df['churn'] = (rng.rand(n_samples) < probs).astype(int)
    return df


def load_data(path: str=None, logger=None):
    if path and Path(path).exists():
        df = pd.read_csv(path)
        if logger: logger.info(f'Loaded {len(df)} rows from {path}')
```

```
    else:
        if logger: logger.warning('No data found; generating synthetic dataset')
        df = generate_synthetic_data()
    return df


def build_preprocessor(df: pd.DataFrame):
    numeric = df.select_dtypes(include=['int64','float64']).columns.tolist()
    categorical =
df.select_dtypes(include=['object','category']).columns.tolist()
    numeric = [c for c in numeric if c not in ('churn','customer_id')]
    categorical = [c for c in categorical if c not in ('churn','customer_id')]
    num_pipe = Pipeline([('imputer', SimpleImputer(strategy='median')),
('scaler', StandardScaler())])
    cat_pipe = Pipeline([('imputer', SimpleImputer(strategy='most_frequent')),
('onehot', OneHotEncoder(handle_unknown='ignore', sparse=False))])
    pre = ColumnTransformer([('num', num_pipe, numeric), ('cat', cat_pipe,
categorical)], remainder='drop')
    return pre, numeric, categorical


def get_model(name='random_forest', random_state=DEFAULT_RANDOM_STATE):
    name = name.lower()
    if name=='random_forest':
        return RandomForestClassifier(n_estimators=200,
random_state=random_state, n_jobs=-1)
    if name=='lightgbm' and HAS_LGB:
        return lgb.LGBMClassifier(n_estimators=300, random_state=random_state)
    if name=='xgboost' and HAS_XGB:
        return xgb.XGBClassifier(n_estimators=300, use_label_encoder=False,
eval_metric='logloss', random_state=random_state)
    raise RuntimeError(f'Model {name} not available or its package not
installed')


def quick_param_dist(model_name):
    if model_name=='random_forest':
        return {'clf__n_estimators':[100,200],'clf__max_depth':[None,
8],'clf__min_samples_split':[2,5]}
    return {}


def train_and_evaluate(df, target='churn', out_dir='output',
model_name='random_forest', apply_smote=False,
random_state=DEFAULT_RANDOM_STATE, logger=None):
    os.makedirs(out_dir, exist_ok=True)
    df = df.dropna(subset=[target]).copy()
    ids = df.pop('customer_id') if 'customer_id' in df.columns else None
```

```
    X = df.drop(columns=[target])
    y = df[target].astype(int)
    pre, num_cols, cat_cols = build_preprocessor(X)
    model = get_model(model_name, random_state=random_state)
    pipe = Pipeline([('preprocessor', pre), ('clf', model)])
    X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.2,stratify=y,random_state=random_state)
    param_dist = quick_param_dist(model_name)
    if param_dist:
        cv = StratifiedKFold(3, shuffle=True, random_state=random_state)
        search = RandomizedSearchCV(pipe, param_distributions=param_dist,
n_iter=3, cv=cv, scoring='roc_auc', random_state=random_state, n_jobs=-1)
        search.fit(X_train, y_train)
        best = search.best_estimator_
    else:
        pipe.fit(X_train, y_train)
        best = pipe
    y_pred = best.predict(X_test)
    y_proba = best.predict_proba(X_test)[:,1]
    metrics = {'accuracy': float(accuracy_score(y_test,y_pred)), 'precision':
float(precision_score(y_test,y_pred, zero_division=0)), 'recall':
float(recall_score(y_test,y_pred, zero_division=0)), 'f1':
float(f1_score(y_test,y_pred, zero_division=0)), 'roc_auc':
float(roc_auc_score(y_test,y_proba))}
    with open(os.path.join(out_dir,'metrics.json'),'w') as f:
json.dump(metrics,f,indent=2)
    joblib.dump(best, os.path.join('models', f'churn_model_{model_name}.pkl'))
    pd.DataFrame({'y_true': y_test.values, 'y_pred': y_pred, 'y_proba':
y_proba}).to_csv(os.path.join(out_dir,'predictions.csv'), index=False)
    return metrics
```

</details>

<details> <summary>tests/test_pipeline.py (click to expand)</summary>

```
# tests/test_pipeline.py
import os
import shutil
from utils import generate_synthetic_data, train_and_evaluate

def test_quick_train(tmp_path):
    out = tmp_path / 'out'
    out.mkdir()
    df = generate_synthetic_data(n_samples=200)
    metrics = train_and_evaluate(df, out_dir=str(out),
model_name='random_forest')
```

```
    assert 'accuracy' in metrics
    assert os.path.exists(out / 'predictions.csv')
```

</details>

<details> <summary>notebook/churn_exploration.py (click to expand)</summary>

```
# %%
# churn_exploration.py - a notebook-style script you can open as a notebook in
VSCode or import into Jupyter.

# %%
import pandas as pd
from utils import generate_synthetic_data, train_and_evaluate

# %%
# 1. Load a small dataset and peek
df = generate_synthetic_data(n_samples=500)
print(df.head())

# %%
# 2. Train a quick model (this will write to ./output)
metrics = train_and_evaluate(df, out_dir='notebook_output',
model_name='random_forest')
print('Metrics:', metrics)

# %%
# 3. Plot simple distributions (requires matplotlib)
import matplotlib.pyplot as plt
plt.hist(df['tenure_months'], bins=30)
plt.title('Tenure distribution')
plt.show()
```

</details>

If you'd like, I can now: - commit these files into a single canvas `repo.zip` archive for download, or - create a GitHub Actions CI workflow and add it to the repo, or - push the files directly to a GitHub repo if you give me its name (I can instead show the exact `git` commands to run).

Which should I do next?

GitHub Actions CI workflow

I added a recommended GitHub Actions workflow file you can paste into `.github/workflows/ci.yml`. It runs on push and pull_request, sets up Python, installs requirements, runs tests with pytest, and runs a quick smoke `python -m pyflakes .` (if pyflakes is installed).

`.github/workflows/ci.yml`

```
name: CI

on:
  push:
    branches: [ main, master ]
  pull_request:
    branches: [ main, master ]

jobs:
  build:
    runs-on: ubuntu-latest

    strategy:
      matrix:
        python-version: [3.10, 3.11]

    steps:
    - uses: actions/checkout@v4
    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: ${{ matrix.python-version }}
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
        pip install pytest
        pip install pyflakes || true
    - name: Run tests
      run: |
        pytest -q
    - name: Lint (pyflakes)
      run: |
        pyflakes . || true
```

I've added instructions in the canvas on where to place this file. If you want, I can also add Dependabot config or add a codeql-analysis job.