devops-ca1-binary-bank-classification

This canvas contains the full file scaffold and ready-to-paste contents for your GitHub repo. Copy each file into your local repo exactly as shown (create directories as needed), then run the git commands provided at the end of the document.

Repository structure

```
devops-ca1-binary-bank-classification/
│
├── .github/workflows/ml-pipeline.yml
├── ansible/playbook.yml
├── ansible/inventory.ini
├── data/  (download Kaggle files here)
│   ├── train.csv  (DO NOT upload to GitHub — add instruction)
│   ├── test.csv
│   └── sample_submission.csv
├── docker/Dockerfile
├── docker/deployment.yaml
├── docker/service.yaml
├── monitoring/prometheus.yml
├── monitoring/grafana_dashboard.json
├── notebooks/binary_classification.ipynb  (notebook text summary)
├── src/
│   ├── data_preprocessing.py
│   ├── model_training.py
│   ├── evaluate.py
│   ├── predict.py
│   └── model.joblib  (ignored — generated by training)
├── requirements.txt
├── .gitignore
├── README.md
├── submission.csv
└── scripts/create_env.sh
```

File: .gitignore

```
__pycache__/
.ipynb_checkpoints/
venv/
```

```
*.pyc
*.pyo
*.pyd
.env
.DS_Store
data/
src/model.joblib
secrets.yml
```

File: requirements.txt

```
pandas
numpy
scikit-learn
joblib
flask
gunicorn
matplotlib
seaborn
pyyaml
ansible
prometheus-client
requests
```

File: README.md

(Complete README with CA-friendly sections — pasted below)

```
# Binary Classification with a Bank Dataset

**Kaggle Playground Series — Season 5 Episode 8**

Team: Yogesh Meena (22070122258)

## Objective
Build and deploy a reproducible ML pipeline to solve the Kaggle binary
classification task. This repo contains preprocessing scripts, training code, a
prediction API, containerization and Kubernetes manifests, Ansible playbook for
environment setup, and monitoring configuration.

## Quick start
```

~~~
1. Clone repo
2. Create Python virtualenv and install requirements
   ```bash
   python -m venv venv
   source venv/bin/activate
   pip install -r requirements.txt
   ```
3. Download Kaggle dataset (train.csv, test.csv, sample_submission.csv) into
`data/`.
4. Train model (generates `src/model.joblib`)
   ```bash
   python src/model_training.py --data-dir data --output src/model.joblib
   ```
5. Run Flask prediction API
   ```bash
   cd src
   gunicorn --bind 0.0.0.0:5000 app:app
   ```
6. Build Docker image
   ```bash
   docker build -t bank-ml-app -f docker/Dockerfile .
   ```
7. (Optional) Deploy to Minikube/Kubernetes using manifests in `docker/`.

## What we added for CA1
- Preprocessing and training scripts
- Dockerfile for prediction API
- Kubernetes manifests for deployment
- GitHub Actions CI workflow (build + smoke test)
- Ansible playbook to provision environment
- Monitoring config (Prometheus + Grafana)

## Notes
- **Do not** commit/download Kaggle data to the public repo: add data/
to .gitignore. Put instructions in README for graders to download from Kaggle.

## Files of interest
- `notebooks/binary_classification.ipynb` — notebook used for EDA and model
tuning
- `src/model_training.py` — training script
- `src/predict.py` + `src/app.py` — prediction helper and Flask API
- `docker/Dockerfile` — containerization for the API
- `.github/workflows/ml-pipeline.yml` — CI workflow

## Contact
Yogesh Meena — 22070122258
~~~

File: src/data_preprocessing.py

```python
# src/data_preprocessing.py
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import joblib


def build_preprocessor(df: pd.DataFrame):
    # simple automatic detection
    num_cols = df.select_dtypes(include=["int64", "float64"]).columns.tolist()
    cat_cols = df.select_dtypes(include=["object", "category"]).columns.tolist()
    # drop target if present
    for c in ["target", "label"]:
        if c in num_cols:
            num_cols.remove(c)
        if c in cat_cols:
            cat_cols.remove(c)

    num_pipeline = Pipeline([
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler())
    ])

    cat_pipeline = Pipeline([
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("ohe", OneHotEncoder(handle_unknown="ignore", sparse=False))
    ])

    preprocessor = ColumnTransformer([
        ("num", num_pipeline, num_cols),
        ("cat", cat_pipeline, cat_cols)
    ])
    return preprocessor


def load_data(path):
    return pd.read_csv(path)
```

File: src/model_training.py

```python
# src/model_training.py
import argparse
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.pipeline import Pipeline
import joblib
from data_preprocessing import build_preprocessor


def train(data_dir, output):
    train_df = pd.read_csv(f"{data_dir}/train.csv")
    # assume target column is 'target' or 'label'
    if 'target' in train_df.columns:
        y = train_df['target']
        X = train_df.drop(columns=['target'])
    elif 'label' in train_df.columns:
        y = train_df['label']
        X = train_df.drop(columns=['label'])
    else:
        raise ValueError('No target column found in train.csv')

    preprocessor = build_preprocessor(X)

    # Simple ensemble pipeline (RF as baseline)
    model = RandomForestClassifier(n_estimators=100, random_state=42)

    pipe = Pipeline([
        ('pre', preprocessor),
        ('clf', model)
    ])

    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

    pipe.fit(X_train, y_train)

    # Save pipeline
    joblib.dump(pipe, output)

    # quick validation metric
    score = pipe.score(X_val, y_val)
    print(f"Validation accuracy: {score:.4f}")
```

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--data-dir', default='data')
    parser.add_argument('--output', default='src/model.joblib')
    args = parser.parse_args()
    train(args.data_dir, args.output)
```

File: src/evaluate.py

```
# src/evaluate.py
import joblib
import pandas as pd
from sklearn.metrics import accuracy_score, f1_score, classification_report


def evaluate(model_path, data_path):
    model = joblib.load(model_path)
    df = pd.read_csv(data_path)
    if 'target' in df.columns:
        y = df['target']
        X = df.drop(columns=['target'])
    elif 'label' in df.columns:
        y = df['label']
        X = df.drop(columns=['label'])
    else:
        raise ValueError('No target column found for evaluation')

    preds = model.predict(X)
    print('Accuracy:', accuracy_score(y, preds))
    print('F1:', f1_score(y, preds))
    print(classification_report(y, preds))


if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('--model', default='src/model.joblib')
    parser.add_argument('--data', default='data/train.csv')
    args = parser.parse_args()
    evaluate(args.model, args.data)
```

File: src/predict.py

```
# src/predict.py
import joblib
import pandas as pd


def predict(model_path, input_csv, output_csv):
    model = joblib.load(model_path)
    X = pd.read_csv(input_csv)
    preds = model.predict(X)
    out = pd.DataFrame({'prediction': preds})
    out.to_csv(output_csv, index=False)
    print(f'Wrote predictions to {output_csv}')


if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('--model', default='src/model.joblib')
    parser.add_argument('--input', default='data/test.csv')
    parser.add_argument('--output', default='submission.csv')
    args = parser.parse_args()
    predict(args.model, args.input, args.output)
```

File: src/app.py

```
# src/app.py
from flask import Flask, request, jsonify
import joblib
import pandas as pd
import os

MODEL_PATH = os.environ.get('MODEL_PATH', 'src/model.joblib')
app = Flask(__name__)

model = joblib.load(MODEL_PATH)

@app.route('/predict', methods=['POST'])
def predict():
    payload = request.json
    # expect JSON with list of records
    df = pd.DataFrame(payload['instances'])
    preds = model.predict(df)
    return jsonify({'predictions': preds.tolist()})

@app.route('/health')
def health():
    return jsonify({'status': 'ok'})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

File: docker/Dockerfile

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
COPY src/ ./src
ENV MODEL_PATH=/app/src/model.joblib
EXPOSE 5000
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "src.app:app"]
```

File: docker/deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bank-ml-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bank-ml
  template:
    metadata:
      labels:
        app: bank-ml
    spec:
      containers:
        - name: bank-ml
          image: <your-dockerhub-username>/bank-ml-app:latest
          ports:
            - containerPort: 5000
          env:
            - name: MODEL_PATH
              value: "/app/src/model.joblib"
```

File: docker/service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: bank-ml-service
spec:
  selector:
    app: bank-ml
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
  type: NodePort
```

File: .github/workflows/ml-pipeline.yml

```
name: ML CI

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

jobs:
  build-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
      - name: Run unit checks
        run: |
          echo "No unit tests defined yet"
      - name: Build Docker image
        run: |
          docker build -t bank-ml-app:ci -f docker/Dockerfile .
      - name: Smoke test API
        run: |
          docker run -d -p 5001:5000 --name bank-ml-app-ci bank-ml-app:ci
          sleep 4
          curl -fsS http://localhost:5001/health
```

File: ansible/playbook.yml

```
- hosts: all
  become: yes
  tasks:
    - name: Ensure python3 pip is present
      apt:
        name: python3-pip
        state: present
        update_cache: yes

    - name: Install python packages
      pip:
        name:
          - pip
          - virtualenv
```

File: ansible/inventory.ini

```
[local]
localhost ansible_connection=local
```

File: monitoring/prometheus.yml

```
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'bank-ml-app'
    static_configs:
      - targets: ['host.docker.internal:5000']
```

File: monitoring/grafana_dashboard.json

```
{}
```

File: notebooks/binary_classification.ipynb

(The notebook is long; create one in Jupyter using the scripts in `src/`. Put EDA, model training cells and save.)

File: scripts/create_env.sh

```
#!/usr/bin/env bash
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Quick git commands to initialize and push

```
git init
git checkout -b main
git add .
git commit -m "Initial CA1 scaffold: preprocessing, training, API, Docker, k8s,
CI"
# create repo on GitHub and add remote
git remote add origin https://github.com/<your-username>/devops-ca1-binary-bank-
```

```
classification.git
git push -u origin main
```

Next steps I can do for you now

- Generate the exact `binary_classification.ipynb` notebook content (I can create a runnable .py conversion instead if you prefer).
- Create the Docker Hub push steps and CI secrets example.
- Create a short PR description and Google Sheet entry ready to paste.

Copy files from this canvas into your local repo. If you want, I can also export these into a downloadable ZIP (I cannot create files outside the canvas here, but I can provide exact file contents which you can copy/paste).