🚀 Heart Disease ML Pipeline - Deployment Guide

🌐 Ngrok Deployment Setup

Step 1: Install Ngrok

```bash
# Option 1: Using pip
pip install pyngrok

# Option 2: Download from ngrok.com
# Visit: https://ngrok.com/download
# Extract and add to PATH
```

Step 2: Create Ngrok Account

1. Visit ngrok.com and sign up
2. Get your authentication token from the dashboard
3. Configure ngrok with your token:

```bash
ngrok authtoken YOUR_AUTH_TOKEN
```

Step 3: Deploy the Application

```bash
# Terminal 1: Start the Streamlit application
streamlit run ui/app.py --server.port 8501

# Terminal 2: Create public tunnel
ngrok http 8501
```

Step 4: Access Your Application

- Copy the **https** URL from ngrok output
- Share this URL to allow public access
- Example: `https://abc123.ngrok.io`

🐳 Docker Deployment

Dockerfile

dockerfile

```
FROM python:3.9-slim

WORKDIR /app

# Copy requirements and install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Expose port
EXPOSE 8501

# Health check
HEALTHCHECK CMD curl --fail http://localhost:8501/_stcore/health

# Run the application
ENTRYPOINT ["streamlit", "run", "ui/app.py", "--server.port=8501", "--server.address=0.0.0.0"]
```

Build and Run Docker Container

bash

```
# Build the image
docker build -t heart-disease-app .

# Run the container
docker run -p 8501:8501 heart-disease-app

# Run in background
docker run -d -p 8501:8501 --name heart-app heart-disease-app
```

☁️ Cloud Deployment Options

1. Streamlit Cloud

1. Push code to GitHub repository
2. Visit share.streamlit.io
3. Connect your GitHub account
4. Select repository and branch
5. Set main file path: `ui/app.py`
6. Deploy!

2. Heroku Deployment

```bash
# Install Heroku CLI
# Create Procfile
echo "web: streamlit run ui/app.py --server.port=$PORT --server.address=0.0.0.0" > Procfile

# Create runtime.txt
echo "python-3.9.16" > runtime.txt

# Deploy to Heroku
heroku create your-app-name
git add .
git commit -m "Deploy to Heroku"
git push heroku main
```

3. AWS EC2 Deployment

```bash
```

```
# Launch EC2 instance with Ubuntu
# SSH into instance
ssh -i your-key.pem ubuntu@your-ec2-ip

# Install dependencies
sudo apt update
sudo apt install python3-pip
pip3 install -r requirements.txt

# Install and configure nginx (optional)
sudo apt install nginx

# Create systemd service
sudo nano /etc/systemd/system/heart-disease-app.service

# Add the following content:
[Unit]
Description=Heart Disease ML App
After=network.target

[Service]
User=ubuntu
WorkingDirectory=/home/ubuntu/heart-disease-ml-pipeline
Environment="PATH=/home/ubuntu/.local/bin"
ExecStart=/home/ubuntu/.local/bin/streamlit run ui/app.py --server.port=8501
Restart=always

[Install]
WantedBy=multi-user.target

# Enable and start service
sudo systemctl daemon-reload
sudo systemctl enable heart-disease-app
sudo systemctl start heart-disease-app
```

4. Google Cloud Platform

bash

```
# Create app.yaml for Google App Engine
runtime: python39

service: heart-disease-app

env_variables:
  STREAMLIT_SERVER_PORT: 8080
  STREAMLIT_SERVER_ADDRESS: 0.0.0.0

# Deploy
gcloud app deploy
```

🔧 Configuration Files

Streamlit Config (.streamlit/config.toml)

```toml
[global]
developmentMode = false

[server]
port = 8501
enableCORS = false
enableXsrfProtection = false

[browser]
gatherUsageStats = false

[theme]
primaryColor = "#FF6B6B"
backgroundColor = "#FFFFFF"
secondaryBackgroundColor = "#F0F2F6"
textColor = "#262730"
font = "sans serif"
```

Environment Variables (.env)

```bash
```

```
# Application Settings
STREAMLIT_SERVER_PORT=8501
STREAMLIT_SERVER_ADDRESS=0.0.0.0

# Model Settings
MODEL_PATH=models/final_model.pkl
MODEL_METADATA_PATH=models/final_model_metadata.pkl

# Logging
LOG_LEVEL=INFO
LOG_FILE=logs/app.log

# Security
SECRET_KEY=your-secret-key-here
ALLOWED_HOSTS=localhost,127.0.0.1,your-domain.com
```

🔍 Monitoring & Logging

Application Logging

python

```python
import logging
import streamlit as st

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('logs/app.log'),
        logging.StreamHandler()
    ]
)

logger = logging.getLogger(__name__)

# Usage in Streamlit app
@st.cache_data
def log_prediction(user_input, prediction, probability):
    logger.info(f"Prediction made: Input={user_input}, Output={prediction}, Probability={probability}")
```

Health Checks

```python
# health_check.py
import requests
import sys

def check_app_health():
    try:
        response = requests.get('http://localhost:8501/_stcore/health', timeout=5)
        if response.status_code == 200:
            print("✅ Application is healthy")
            return True
        else:
            print("❌ Application health check failed")
            return False
    except Exception as e:
        print(f"❌ Health check error: {e}")
        return False

if __name__ == "__main__":
    if not check_app_health():
        sys.exit(1)
```

🛡️ Security Best Practices

1. Input Validation

```python
```

```python
def validate_user_input(age, sex, cp, trestbps, chol):
    """Validate user inputs to prevent attacks"""

    # Age validation
    if not isinstance(age, (int, float)) or not (1 <= age <= 120):
        raise ValueError("Invalid age value")

    # Sex validation
    if sex not in [0, 1]:
        raise ValueError("Invalid sex value")

    # Chest pain validation
    if cp not in [0, 1, 2, 3]:
        raise ValueError("Invalid chest pain type")

    # Blood pressure validation
    if not isinstance(trestbps, (int, float)) or not (50 <= trestbps <= 300):
        raise ValueError("Invalid blood pressure value")

    # Cholesterol validation
    if not isinstance(chol, (int, float)) or not (50 <= chol <= 1000):
        raise ValueError("Invalid cholesterol value")

    return True
```

2. Rate Limiting

python

```python
import time
from functools import wraps

def rate_limit(max_calls=10, time_window=60):
    """Rate limiting decorator"""
    calls = {}

    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            now = time.time()
            client_id = st.session_state.get('client_id', 'anonymous')

            if client_id not in calls:
                calls[client_id] = []

            # Clean old calls
            calls[client_id] = [call_time for call_time in calls[client_id]
                                if now - call_time < time_window]

            if len(calls[client_id]) >= max_calls:
                st.error("Rate limit exceeded. Please wait before making another request.")
                return None

            calls[client_id].append(now)
            return func(*args, **kwargs)

        return wrapper
    return decorator

# Usage
@rate_limit(max_calls=5, time_window=60)
def make_prediction(features):
    return model.predict(features)
```

3. HTTPS Configuration

python

```python
# For production deployment
import ssl
import streamlit as st

# SSL context for HTTPS
ssl_context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
ssl_context.load_cert_chain('path/to/cert.pem', 'path/to/key.pem')

# Run with HTTPS
if __name__ == "__main__":
    st.run(ssl_context=ssl_context, port=443)
```

📊 Performance Optimization

1. Caching Strategies

python

```python
import streamlit as st

# Cache model loading
@st.cache_resource
def load_model():
    return joblib.load('models/final_model.pkl')

# Cache data processing
@st.cache_data
def preprocess_data(data):
    return scaler.transform(data)

# Cache expensive computations
@st.cache_data(ttl=3600)  # Cache for 1 hour
def generate_insights():
    return expensive_analysis()
```

2. Memory Management

python

```python
import gc
import psutil

def monitor_memory_usage():
    """Monitor application memory usage"""
    process = psutil.Process()
    memory_info = process.memory_info()

    st.sidebar.metric(
        "Memory Usage",
        f"{memory_info.rss / 1024 / 1024:.1f} MB"
    )

    # Garbage collection if memory usage is high
    if memory_info.rss > 500 * 1024 * 1024:  # 500 MB
        gc.collect()
```

3. Database Integration (Optional)

```python
```

```python
import sqlite3
import pandas as pd

class PredictionLogger:
    def __init__(self, db_path="predictions.db"):
        self.db_path = db_path
        self.init_db()

    def init_db(self):
        """Initialize database"""
        conn = sqlite3.connect(self.db_path)
        conn.execute("""
            CREATE TABLE IF NOT EXISTS predictions (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
                input_features TEXT,
                prediction INTEGER,
                probability REAL
            )
        """)
        conn.close()

    def log_prediction(self, features, prediction, probability):
        """Log prediction to database"""
        conn = sqlite3.connect(self.db_path)
        conn.execute(
            "INSERT INTO predictions (input_features, prediction, probability) VALUES (?, ?, ?)",
            (str(features), prediction, probability)
        )
        conn.commit()
        conn.close()
```

🧪 Testing Deployment

Automated Testing Script

python

```python
#!/usr/bin/env python3
import requests
import json
import time

def test_deployment(base_url):
    """Test deployed application"""

    print(f"Testing deployment at: {base_url}")

    # Test 1: Health check
    try:
        response = requests.get(f"{base_url}/_stcore/health", timeout=10)
        assert response.status_code == 200
        print("✅ Health check passed")
    except Exception as e:
        print(f"❌ Health check failed: {e}")
        return False

    # Test 2: Main page load
    try:
        response = requests.get(base_url, timeout=10)
        assert response.status_code == 200
        assert "Heart Disease" in response.text
        print("✅ Main page loads correctly")
    except Exception as e:
        print(f"❌ Main page test failed: {e}")
        return False

    print("✅ All tests passed!")
    return True

if __name__ == "__main__":
    # Test local deployment
    test_deployment("http://localhost:8501")

    # Test production deployment
    # test_deployment("https://your-app.ngrok.io")
```

Load Testing

python

```python
import concurrent.futures
import requests
import time

def load_test(url, num_requests=100, concurrent_users=10):
    """Simple load testing"""

    def make_request():
        try:
            start_time = time.time()
            response = requests.get(url, timeout=10)
            end_time = time.time()

            return {
                'status_code': response.status_code,
                'response_time': end_time - start_time,
                'success': response.status_code == 200
            }
        except Exception as e:
            return {
                'status_code': 0,
                'response_time': 0,
                'success': False,
                'error': str(e)
            }

    # Run concurrent requests
    with concurrent.futures.ThreadPoolExecutor(max_workers=concurrent_users) as executor:
        futures = [executor.submit(make_request) for _ in range(num_requests)]
        results = [future.result() for future in concurrent.futures.as_completed(futures)]

    # Analyze results
    successful = sum(1 for r in results if r['success'])
    avg_response_time = sum(r['response_time'] for r in results if r['success']) / max(successful, 1)

    print(f"Load Test Results:")
    print(f"  Total Requests: {num_requests}")
    print(f"  Successful: {successful}")
    print(f"  Success Rate: {successful/num_requests*100:.1f}%")
    print(f"  Average Response Time: {avg_response_time:.3f}s")
```

```
if __name__ == "__main__":
    load_test("http://localhost:8501")
```

🔄 Continuous Deployment

GitHub Actions Workflow (.github/workflows/deploy.yml)

yaml

```yaml
name: Deploy Heart Disease App

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

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: 3.9

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run tests
      run: |
        python -m pytest tests/

    - name: Test model pipeline
      run: |
        python src/heart_disease_pipeline.py

  deploy:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
    - uses: actions/checkout@v2

    - name: Deploy to Streamlit Cloud
      run: |
```

```
    # Trigger Streamlit Cloud deployment
    curl -X POST ${{ secrets.STREAMLIT_WEBHOOK_URL }}
```

This comprehensive deployment guide covers all the major deployment options and best practices for the Heart Disease ML Pipeline project!