~~~
==== README.md ====


# Diabetes Prediction - AI for Healthcare (Beginner Project)

## Overview
This repository contains code to build a Diabetes Prediction model using the Pima Indians Diabetes Dataset
The project includes data preprocessing, model training, evaluation, and a simple Streamlit app to demo pr

**Note:** This bundle does NOT include the dataset (`diabetes.csv`). Download the **Pima Indians Diabetes
https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database
Place the file in the project root and name it `diabetes.csv`.

## Files
- data_preprocessing.py  -> Load & preprocess the dataset
- model_training.py      -> Train models and save best model (Pickle)
- app_streamlit.py       -> Streamlit app to interact with the model
- requirements.txt       -> Python libraries required
- full_code.pdf          -> PDF containing all code (for easy reading/printing)
- diabetes_model.pkl     -> (created after running model_training.py)

## Quick steps
1. Create a virtualenv and install dependencies:
   ```bash
   python -m venv venv
   source venv/bin/activate  # or venv\Scripts\activate on Windows
   pip install -r requirements.txt
   ```
2. Put `diabetes.csv` in the project root.
3. Train and save model:
   ```bash
   python model_training.py
   ```
   This will create `diabetes_model.pkl`.
4. Run the app:
   ```bash
   streamlit run app_streamlit.py
   ```

## License
MIT
~~~

````
==== data_preprocessing.py ====


"""data_preprocessing.py
Load the Pima Indians Diabetes Dataset, do basic cleaning and feature preparation.
Expects `diabetes.csv` in the same folder.
"""
import pandas as pd
import numpy as np

def load_data(path='diabetes.csv'):
    df = pd.read_csv(path)
    return df
````

```
def clean_data(df):
    # Replace zeros in certain columns with NaN (they indicate missing measurements)
    cols_with_zero = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
    for col in cols_with_zero:
        df[col] = df[col].replace(0, pd.NA)
    # Fill missing values with median
    df = df.fillna(df.median())
    return df

def feature_target_split(df):
    X = df.drop('Outcome', axis=1)
    y = df['Outcome']
    return X, y

if __name__ == '__main__':
    df = load_data()
    print('Raw shape:', df.shape)
    df = clean_data(df)
    X, y = feature_target_split(df)
    print('Features shape:', X.shape, 'Target shape:', y.shape)
```

==== model_training.py ====

```
"""model_training.py
Train multiple models and save the best performing one (by F1-score).
Produces a simple evaluation printout.
"""
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
import joblib
from data_preprocessing import load_data, clean_data, feature_target_split

def train_and_evaluate(path='diabetes.csv', random_state=42):
    df = load_data(path)
    df = clean_data(df)
    X, y = feature_target_split(df)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=random_state, st

    scaler = StandardScaler()
    X_train_sc = scaler.fit_transform(X_train)
    X_test_sc = scaler.transform(X_test)

    # Models to try
    models = {
        'logreg': LogisticRegression(max_iter=1000, random_state=random_state),
        'rf': RandomForestClassifier(n_estimators=100, random_state=random_state)
    }

    results = {}
    for name, model in models.items():
        model.fit(X_train_sc, y_train)
        preds = model.predict(X_test_sc)
        results[name] = {
            'accuracy': accuracy_score(y_test, preds),
            'precision': precision_score(y_test, preds, zero_division=0),
            'recall': recall_score(y_test, preds, zero_division=0),
            'f1': f1_score(y_test, preds, zero_division=0),
            'model': model
        }

    # Pick best by F1
    best_name = max(results, key=lambda n: results[n]['f1'])
    best = results[best_name]
    print('Evaluation results:')
    for k, v in results.items():
        print(f"Model: {k} -> Accuracy: {v['accuracy']:.4f}, Precision: {v['precision']:.4f}, Recall: {v['

    # Save scaler + model using joblib
    pipeline = {'scaler': scaler, 'model': best['model']}
    joblib.dump(pipeline, 'diabetes_model.pkl')
    print(f"Best model: {best_name} saved to diabetes_model.pkl")

    # Detailed report for best model
    y_pred = best['model'].predict(X_test_sc)
    print('\nClassification report for best model:')
    print(classification_report(y_test, y_pred))

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

==== app_streamlit.py ====

```
"""app_streamlit.py
Simple Streamlit interface to load the trained model and make predictions.
Run with: streamlit run app_streamlit.py
"""
import streamlit as st
import pandas as pd
import joblib
import numpy as np

st.title('Diabetes Prediction Demo')
st.write('Enter patient data below to get a prediction (0 = no diabetes, 1 = diabetes).')

def user_input_form():
    pregnancies = st.number_input('Pregnancies', min_value=0, value=1)
    glucose = st.number_input('Glucose', min_value=0.0, value=120.0)
    bp = st.number_input('BloodPressure', min_value=0.0, value=70.0)
    skin = st.number_input('SkinThickness', min_value=0.0, value=20.0)
    insulin = st.number_input('Insulin', min_value=0.0, value=79.0)
    bmi = st.number_input('BMI', min_value=0.0, value=32.0)
    dpf = st.number_input('DiabetesPedigreeFunction', min_value=0.0, value=0.471)
    age = st.number_input('Age', min_value=0, value=33)
    data = {'Pregnancies': pregnancies, 'Glucose': glucose, 'BloodPressure': bp,
            'SkinThickness': skin, 'Insulin': insulin, 'BMI': bmi,
            'DiabetesPedigreeFunction': dpf, 'Age': age}
    features = pd.DataFrame([data])
    return features

uploaded = st.file_uploader('Upload diabetes.csv (optional)', type='csv')
if uploaded is not None:
    st.write('Sample of uploaded data:')
    st.dataframe(pd.read_csv(uploaded).head())

input_df = user_input_form()
st.write('Input features:')
st.dataframe(input_df)

if st.button('Predict'):
    try:
        pipeline = joblib.load('diabetes_model.pkl')
    except Exception as e:
        st.error('Model not found. Please run model_training.py first to create diabetes_model.pkl')
        raise e
    scaler = pipeline['scaler']
    model = pipeline['model']
    X_sc = scaler.transform(input_df)
    pred = model.predict(X_sc)[0]
    prob = model.predict_proba(X_sc)[0][1] if hasattr(model, 'predict_proba') else None
    st.success(f'Prediction: {int(pred)}')
    if prob is not None:
        st.info(f'Probability of diabetes: {prob:.2f}')
```

==== requirements.txt ====

```
pandas
numpy
scikit-learn
matplotlib
seaborn
streamlit
joblib
fpdf
```

==== LICENSE ====

copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.