Files for AI Tools Assignment

This document contains two files you requested: `ai_tools_assignment.py` (code) and `README.md` (theoretical answers + instructions). Open each section to copy or download the file contents.

File: ai_tools_assignment.py

```
"""
ai_tools_assignment.py

Practical Implementation — Classical ML with Scikit-learn

This script demonstrates a complete, self-contained classical ML workflow using
scikit-learn.

Features:
- Loads the Breast Cancer dataset from scikit-learn (tabular, classification).
- Preprocesses data (train/test split, optional scaling).
- Trains multiple classical ML models (Logistic Regression, Random Forest, SVM).
- Evaluates models with accuracy, classification report, confusion matrix.
- Provides functions to train, evaluate, export, and predict.
- CLI-friendly: run with `python ai_tools_assignment.py --help`.

Requirements:
- Python 3.8+
- scikit-learn
- pandas
- joblib
- matplotlib (optional for plotting)

Example:
    python ai_tools_assignment.py --train --model rf
    python ai_tools_assignment.py --predict --input 14.34,20.5,92.44,....

"""

import argparse
import json
import os
from typing import Any, Dict, Tuple

import joblib
import matplotlib.pyplot as plt
```

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (ConfusionMatrixDisplay, accuracy_score,
                             classification_report)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC


MODEL_DIR = "models"
os.makedirs(MODEL_DIR, exist_ok=True)


def load_data() -> Tuple[pd.DataFrame, pd.Series]:
    """Load the Breast Cancer dataset and return X (DataFrame) and y
(Series)."""
    data = load_breast_cancer(as_frame=True)
    X = data.frame.drop(columns=[data.target_names[0]]) if False else data.data
    y = data.target
    feature_names = data.feature_names.tolist()
    X = pd.DataFrame(X, columns=feature_names)
    y = pd.Series(y, name='target')
    return X, y


def preprocess(X: pd.DataFrame, scaler: StandardScaler = None) ->
Tuple[np.ndarray, StandardScaler]:
    """Scale features with StandardScaler. Returns scaled array and scaler
used."""
    if scaler is None:
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)
    else:
        X_scaled = scaler.transform(X)
    return X_scaled, scaler


def train_models(X_train: np.ndarray, y_train: np.ndarray) -> Dict[str, Any]:
    """Train multiple classical ML models and return them in a dict."""
    models = {
        'logreg': LogisticRegression(max_iter=1000),
        'rf': RandomForestClassifier(n_estimators=100, random_state=42),
        'svm': SVC(probability=True),
    }
    trained = {}
```

```
    for name, model in models.items():
        print(f"Training {name}...")
        model.fit(X_train, y_train)
        trained[name] = model
    return trained


def evaluate_model(model, X_test: np.ndarray, y_test: np.ndarray, model_name:
str = '') -> Dict[str, Any]:
    """Evaluate a single model and print results. Returns a results dict."""
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)

    print(f"\nEvaluation for {model_name}")
    print(f"Accuracy: {acc:.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

    # Plot confusion matrix
    try:
        disp = ConfusionMatrixDisplay.from_estimator(model, X_test, y_test)
        disp.ax_.set_title(f"Confusion Matrix - {model_name}")
        plt.tight_layout()
        fig_path = os.path.join(MODEL_DIR, f"confusion_{model_name}.png")
        plt.savefig(fig_path)
        print(f"Saved confusion matrix to: {fig_path}")
        plt.clf()
    except Exception as e:
        print(f"Could not plot confusion matrix for {model_name}: {e}")

    return {'accuracy': acc, 'report': report}


def save_model(obj: Any, name: str):
    path = os.path.join(MODEL_DIR, f"{name}.joblib")
    joblib.dump(obj, path)
    print(f"Saved {name} to {path}")
    return path


def load_model(name: str):
    path = os.path.join(MODEL_DIR, f"{name}.joblib")
    if not os.path.exists(path):
        raise FileNotFoundError(f"Model file not found: {path}")
    return joblib.load(path)
```

```
def build_and_evaluate(save: bool = True):
    X, y = load_data()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

    X_train_scaled, scaler = preprocess(X_train)
    X_test_scaled, _ = preprocess(X_test, scaler)

    trained = train_models(X_train_scaled, y_train.values)

    results = {}
    for name, model in trained.items():
        results[name] = evaluate_model(model, X_test_scaled, y_test.values,
model_name=name)
        if save:
            save_model(model, f"model_{name}")

    # Save scaler and metadata
    if save:
        save_model(scaler, "scaler")
        metadata = {
            'features': X.columns.tolist(),
            'dataset': 'breast_cancer',
        }
        with open(os.path.join(MODEL_DIR, 'metadata.json'), 'w') as f:
            json.dump(metadata, f, indent=2)
        print(f"Saved metadata to {os.path.join(MODEL_DIR, 'metadata.json')}")

    return results


def predict(model_name: str, input_vector: list):
    """Load scaler and model, run a prediction on the provided input vector
(list of floats)."""
    scaler = load_model('scaler')
    model = load_model(f"model_{model_name}")
    input_arr = np.array(input_vector, dtype=float).reshape(1, -1)
    input_scaled = scaler.transform(input_arr)
    prob = model.predict_proba(input_scaled)[0].tolist() if hasattr(model,
'predict_proba') else None
    pred = int(model.predict(input_scaled)[0])
    return {'prediction': pred, 'probabilities': prob}


def parse_args():
    parser = argparse.ArgumentParser(description='Classical ML demo with scikit-
learn')
    parser.add_argument('--train', action='store_true', help='Train models and
```

```
save them')
    parser.add_argument('--model', type=str, default='rf', choices=['logreg',
'rf', 'svm'], help='Model to use for predict or quick train')
    parser.add_argument('--predict', action='store_true',
help='Run a prediction using a saved model')
    parser.add_argument('--input', type=str, help='Comma-separated input vector
for prediction')
    return parser.parse_args()


def main():
    args = parse_args()
    if args.train:
        print("Starting training pipeline...")
        results = build_and_evaluate(save=True)
        print("Training complete. Results summary:")
        print(json.dumps({k: v['accuracy'] for k, v in results.items()},
indent=2))

    if args.predict:
        if not args.input:
            raise ValueError('Provide --input with comma-separated feature
values ({} features expected).')
        # Load feature count from metadata
        with open(os.path.join(MODEL_DIR, 'metadata.json')) as f:
            metadata = json.load(f)
        n_features = len(metadata['features'])
        raw = [float(x) for x in args.input.split(',')]
        if len(raw) != n_features:
            raise ValueError(f'Input length {len(raw)} does not match expected
{n_features}')
        out = predict(args.model, raw)
        print('Prediction result:')
        print(json.dumps(out, indent=2))


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

File: README.md

```
# AI Tools Assignment — Part 1 (Theoretical) & Part 2 (Practical)

## Part 1: Theoretical Understanding (40%)
```

```
### 1. Short Answer Questions

**Q1: Explain the primary differences between TensorFlow and PyTorch. When would you choose one over the other?**

**Answer:**
- **Computation graph model:** PyTorch uses dynamic (eager) computation graphs by default, which makes debugging and prototyping intuitive — operations are executed immediately. TensorFlow originally used static graphs (define-and-run) via `tf.Graph`, making deployment on production or optimized graph transforms easier; however, TensorFlow 2.x introduced eager execution to align better with PyTorch.
- **Ecosystem & deployment:** TensorFlow has a larger edge and production deployment ecosystem (TensorFlow Serving, TensorFlow Lite, TensorFlow.js, TPU support). PyTorch has improved deployment (TorchScript, TorchServe) but TensorFlow historically had an edge for production pipelines.
- **Research vs Production:** PyTorch is often preferred in research due to its Pythonic API and dynamic graphs. TensorFlow is favored in some production contexts and in organizations with existing TF infrastructure.
- **APIs & usability:** PyTorch code feels more like standard Python; TensorFlow 2 with `tf.keras` narrowed the gap and improved usability.

**When to choose:**
- Choose **PyTorch** for fast prototyping, research experiments, and when you value an intuitive debugging experience.
- Choose **TensorFlow** if you need robust deployment options across mobile/web/TPU or want the mature production tooling.


**Q2: Describe two use cases for Jupyter Notebooks in AI development.**

**Answer:**
1. **Exploratory Data Analysis (EDA):** Interactive charts, step-by-step transformations, data sampling, and quick visual checks are ideal in notebooks.
2. **Prototyping models & experiments:** Running small experiments, visualizing training curves, and documenting hyperparameter changes inline are practical uses.


**Q3: How does spaCy enhance NLP tasks compared to basic Python string operations?**

**Answer:**
- **Robust tokenization:** spaCy tokenizes text using linguistic rules (handling punctuation, contractions, multi-word tokens), whereas basic string split is naive.
- **Pretrained models:** spaCy provides pretrained pipelines for POS tagging,
```

dependency parsing, and NER, giving structured linguistic annotations.
- **Efficiency & production-readiness:** spaCy is optimized for speed and memory and offers easy serialization, making it suitable for production NLP pipelines.

2. Comparative Analysis

Scikit-learn vs TensorFlow

- **Target applications:**
  - **Scikit-learn:** Classical ML – linear models, SVMs, decision trees, ensemble methods, clustering, dimensionality reduction.
  - **TensorFlow:** Deep learning and neural networks (CNNs, RNNs, Transformers), large-scale model training on GPUs/TPUs.

- **Tools & Resources:**
  - **Scikit-learn:** Simple API for model selection, pipelines, and evaluation. Great for small to medium datasets.
  - **TensorFlow:** Rich set of tools for deep learning (Keras API, TF Datasets, TF Hub), deployment (TF Lite, TF Serving, TF.js), and hardware acceleration.

- **Ease of use for beginners:** Scikit-learn is often easier for beginners due to its consistent API and smaller conceptual overhead. TensorFlow (especially TF2 with Keras) has become much more beginner-friendly.

- **Community support:** Both have strong communities; TensorFlow has broader adoption in large-scale DL applications, while scikit-learn dominates teaching and classical ML tasks.

Frameworks & Platforms

- **Frameworks:** TensorFlow, PyTorch, Scikit-learn, spaCy – each covers different needs (deep learning, research, classical ML, and NLP respectively).
- **Platforms:** Google Colab provides free GPU/TPU access for prototyping; Jupyter Notebook is excellent for EDA and experiment logs.
- **Datasets:** Kaggle hosts community datasets and competitions; TensorFlow Datasets provides easy programmatic access to standard datasets.

Why This Matters

- **Real-World Impact:** These tools power solutions across industries – from medical imaging to fraud detection.
- **Skill Validation:** Employers value hands-on ability with TensorFlow, PyTorch, and scikit-learn.

```
### Practical Tips & Resources

- Use official docs: TensorFlow, PyTorch, spaCy, scikit-learn.
- Post questions with #AIToolsAssignment on the LMS community.
- Pro tip: test small pieces independently and keep notebooks clean with clear
narrative cells.


## Part 2: Practical Implementation (50%)

### Task 1: Classical ML with Scikit-learn

The provided script `ai_tools_assignment.py` (in this submission) performs a
classical ML workflow using scikit-learn on the Breast Cancer dataset. It trains
multiple models, evaluates them, saves models and scaler, and supports
prediction via CLI.

### How to run

1. Create a virtual environment and install requirements:

```bash
python -m venv venv
source venv/bin/activate  # or venv\Scripts\activate on Windows
pip install -U pip
pip install scikit-learn pandas joblib matplotlib
```

1. Train models:

```
python ai_tools_assignment.py --train
```

1. Predict (example):

2. After training, open `models/metadata.json` to see the list of features and prepare a comma-separated input with the correct number of features.

3. Run:

```
python ai_tools_assignment.py --predict --model rf --input 12.45,17.23,78.32,...
```

Deliverables included

- `ai_tools_assignment.py` — single-file implementation (training, evaluation, save/load, predict)
- `README.md` — theoretical answers (Part 1), instructions and run guide
```

Deadline

- 7 days from assignment posting. Showcase your AI toolkit mastery and include small experiments (e.g., hyperparameter changes) in your submission.

Good luck and 🚀! ```

End of files.