Detailed Report on the `experiment_runner.py` Module

Overall Purpose: The `experiment_runner.py` module is the engine of the entire project. Its single responsibility is to conduct a complete, fair, and statistically valid experiment comparing your from-scratch KNN implementation against the industry-standard Scikit-Learn implementation for a given dataset. It takes a dataset configuration as input and produces two final, human-readable report files as output.

Breakdown of the Code

1. Imports

The script begins by importing all the necessary tools for the experiment:

- **Standard Libraries:** `pandas`, `os`, `datetime`, `numpy` for data handling and file operations.
- **Scikit-Learn (`sklearn`):** This is the industry-standard machine learning library in Python.
    - `KNeighborsClassifier`: The actual KNN algorithm from Scikit-Learn that we will compare against.
    - `KFold`: A tool to create the 10 data splits (folds) for cross-validation.
    - `accuracy_score`: Scikit-Learn's function for calculating accuracy.
- **SciPy (`scipy`):** A library for scientific and technical computing.
    - `ttest_rel`: The function that performs a **paired t-test**, which is the specific hypothesis test required for comparing two models on the same cross-validation folds.
- **Our Custom Modules:**
    - `knn_from_scratch.knn.predict_classification`: Your core from-scratch prediction function (the "engine").
    - `knn_from_scratch.cross_validation.accuracy_metric`: Your custom function for calculating accuracy.

2. The `run_experiment` Function

This is the main function in the module. It takes a `config` dictionary (which contains all the paths and parameters for one specific dataset) as its primary input.

Step 1: Data Loading

- The function first reads the `config` object to find the path to the preprocessed data.
- It loads the `X_..._processed.csv` and `y_..._processed.csv` files.
- If the files aren't found, it prints a helpful error message telling you to run the preprocessing step first.
- It then prepares the data in two formats, because our models expect different inputs:
    - **NumPy Arrays (`X_np`, `y_np`):** This is the standard format required by Scikit-Learn.
    - **List of Lists (`dataset_list`):** This is the format our from-scratch functions were designed to use.

Step 2: Setting up a Fair Cross-Validation

- This is one of the most important steps for a fair scientific comparison.
- It creates a `KFold` object with `n_splits=10`.
- Crucially, it sets `shuffle=True` and `random_state=42`.
    - `shuffle=True` ensures the data is mixed before splitting, which is important if the original data has any order.
    - `random_state=42` is a seed for the shuffling. This **guarantees** that the 10 folds will be created in the exact same way every single time the script is run. This ensures that your KNN and Scikit-Learn's KNN are trained and tested on the **exact same splits of data**, making the comparison perfectly fair.

Step 3: The Main Experiment Loop

- The code iterates through the 10 folds created by the `KFold` object. For each fold:
    1. **Run From-Scratch KNN:**

- It prepares the training data in the `list of lists` format.
- It then loops through each row of the test set and calls your `predict_classification` function to get a prediction.
- Finally, it uses your `accuracy_metric` to calculate the accuracy for this fold and stores it in the `custom_knn_scores` list.

2. **Run Scikit-Learn KNN:**
   - It initializes Scikit-Learn's `KNeighborsClassifier` with the same `k` value from our config.
   - It cleverly maps our distance metric name (e.g., `'manhattan_distance'`) to the name Scikit-Learn expects (`'manhattan'`).
   - It trains the model using the `.fit(X_train, y_train)` method.
   - It gets all predictions at once using the `.predict(X_test)` method.
   - It calculates the accuracy and stores it in the `sklearn_knn_scores` list.

Step 4: Hypothesis Testing

- After the loop finishes, we have two lists of 10 scores each.
- The script calls `ttest_rel(custom_knn_scores, sklearn_knn_scores)`.
- A **paired t-test** (`ttest_rel`) is the correct statistical test here because the scores are paired: the first score from your KNN and the first score from Scikit-Learn both came from the exact same fold of data, the second scores came from the second fold, and so on.
- This test returns two values:
  - **T-statistic:** A measure of how different the two sets of scores are.
  - **P-value:** The probability of observing this difference if there was actually no difference between the models. A small p-value (typically < 0.05) suggests the difference is real and not due to random chance.

Step 5 & 6: Generating the Reports

The script's final job is to present the results in the two output files you requested.

1. **The Comparison Report (`cross_validation_report` folder):**
   - This report is designed for easy comparison.

- It creates a clear table showing the accuracy of both your model and Scikit-Learn's for each of the 10 folds, side-by-side.
- It also includes summary statistics (mean and standard deviation) so you can see the average performance and the stability of each model at a glance.

2. **The Hypothesis Test Report (`evaluation_test_report` folder):**
    - This report formally presents the results of the statistical test.
    - It clearly states the **Null Hypothesis** (that there is no difference between the models).
    - It reports the calculated **p-value**.
    - It provides a plain-English **Conclusion**: Based on whether the p-value is less than the significance level (alpha = 0.05), it tells you whether you should **"reject the null hypothesis"** (meaning there *is* a statistically significant difference) or **"fail to reject the null hypothesis"** (meaning there is *no* statistically significant difference).