Project: Concrete Compressive Strength Prediction

1: Data Cleansing Function

I created a function called `cleanse_data` that takes a Pandas `DataFrame` as input and returns another `DataFrame` that is identical to the input except any invalid rows have been removed. For this dataset, an invalid row is any row with:

- A NaN appearing in any column
- A blank entry for any column
- A negative entry** for any column

2: Model Evaluation Function

I also created a function called `eval_model` that evaluates a fitted regression model. The function takes a trained Pipeline object as input along with training and test set data. It evaluates training and test statistics as follows:

- RMSE for the training set
- RMSE for the test set
- r^2 for the test set

Inputs	`pipe`	A fitted `Pipeline` object	Required
	`X_train`	Pandas `DataFrame` object containing training set feature data	Required
	`y_train`	Pandas `DataFrame` or `Series` object of response values corresponding to examples in `X_train`	Required
	`X_test`	Pandas `DataFrame` object containing test set feature data	Required
	`y_test`	Pandas `DataFrame` or `Series` object of response values corresponding to examples in `X_test`	Required
	`name`	String with descriptive name of analysis	Optional. Default set to empty string.

Output	`results`	Python `dictionary` with the following entries in this order: 1. 'name' - the name passed as input to this function 2. 'rmse train' - the RMSE error on the training set 3. 'rmse test' - the RMSE error on the test set 4. 'r2 score' - the r2 score on test set data	

Results from these evaluations are stored in a python `dictionary` and returned to the caller.

3: Model Comparison

I created a script called `mp0task3.py`. It places my `cleanse_data` and `eval_model` functions in it. In the remainder of the script, it trains and compares the following regression techniques generates a comparison table and saves the results as indicated below.

For each technique:

- If needed, it creates a scaler using `StandardScaler`
- Creates a `pipeline` (even if a scaler or other preprocessing is not required)
- Trains the model
- Determines the training RMSE, test RMSE and r^2. Uses the 20% holdout testing set.
- Uses `pickle` to save a single tuple with the following items (in the following order):
    - the trained pipeline
    - results dictionary returned by your `eval_model` function

Technique	**Hyperparameters**	**Pickle File Name**
Linear Regression	Generate 3rd degree polynomial features. Set `fit_intercept=False`.	poly3.dat
ANN	One hidden layer of 100 nodes, relu activation function and sequential gradient descent solver, 2500 max iterations	ann1.dat
ANN	One hidden layer of 100 nodes, tanh activation function and sequential gradient descent solver, 2500 max iterations	ann2.dat
SVR	Radial basis function kernel, epsilon of 5, C=10, gamma = 0.1	svr.dat
Random Forest	100 trees	rfr.dat

ADABOOST	100 tree stumps, loss function = 'square'	ada.dat

4: Hyperparameter Tuning

I created a script called `mp0task4.py` in which you use `GridSearchCV` to perform hyperparameter tuning on the support vector regressor. My script operates as follows:

- Uses 80% of the cleansed dataset for `GridSearchCV`, reserving 20% for a final test
- Creates a `Pipeline` object with a `StandardScaler` and `SVR`.
- Defines the grid search parameters according to the table given below.
- Constructs a GridSearchCV with cv=8.
- Conducts the grid search
- Reports results by printing to the screen:
    - Root mean squared error for the best combination of parameters
    - Best combination of hyperparameter values (C, gamma and epsilon, each clearly labeled on its own line and all limited to four decimal places)

Hyperparameter	**Search Range**
C	Powers of 10 from 10^{-2} through 10^{2}
gamma	Five samples along a log scale from 10^{-3} to 10
epsilon	Five samples along a log scale from 10^{-1} to 10