Explanation of code

# Utils.py

'''

 Example Working 1

# Step 1: Initialize empty report

report = {}  # → {}

# Step 2: First iteration (Linear Regression)

model\_name = "Linear Regression"

model = LinearRegression()

model.fit(X\_train, y\_train)                    # Train

predictions = model.predict(X\_test)            # Predict

score = r2\_score(y\_test, predictions)          # Evaluate (e.g., 0.72)

report["Linear Regression"] = 0.72            # Store

# report = {"Linear Regression": 0.72}

# Step 3: Second iteration (Random Forest)

model\_name = "Random Forest"

# ... training, prediction, evaluation ...

report["Random Forest"] = 0.89

# report = {"Linear Regression": 0.72, "Random Forest": 0.89}

# Step 4: Continue for all models...

# Finally return the completed dictionary

'''

'''

Example Working 2

What params.get(model\_name, {}) Does:

params = {

    "Random Forest": {

        'n\_estimators': [50, 100, 200],

        'max\_depth': [10, 20, None]

    },

    "XGBoost": {

        'n\_estimators': [100, 200],

        'learning\_rate': [0.01, 0.1]

    }

    # Note: No "Linear Regression" or "Decision Tree" in params!

}

The Problem:

What happens when we try to access a key that doesn't exist in the dictionary?

# This would CRASH with KeyError:

model\_params = params["Linear Regression"]  # Key doesn't exist!

The Solution: dict.get()

model\_params = params.get(model\_name, {})

How .get() Works:

dict.get(key, default\_value)

Example 1: Model WITH hyperparameters

model\_name = "Random Forest"

model\_params = params.get(model\_name, {})

print(model\_params)

# Output: {'n\_estimators': [50, 100, 200], 'max\_depth': [10, 20, None]}

# Key exists, returns the hyperparameter grid

Example 2: Model WITHOUT hyperparameters

model\_name = "Linear Regression"

model\_params = params.get(model\_name, {})

print(model\_params)

# Output: {}

#  Key doesn't exist, returns empty dict {} (default value)

model.set\_params(\*\*gs.best\_params\_)

model.fit(X\_train, y\_train)

gs.best\_params\_ contains the best parameter combination

\*\*gs.best\_params\_ unpacks the dictionary into parameters

model.set\_params() applies these best parameters to the model

model.fit() retrains the model with optimal parameters

-------------------------------------------------------

model = RandomForestRegressor()

model\_params = {

    'n\_estimators': [50, 100, 200],

    'max\_depth': [10, 20, None]

}

# GridSearchCV tests:

# 1. n\_estimators=50, max\_depth=10

# 2. n\_estimators=50, max\_depth=20

# 3. n\_estimators=50, max\_depth=None

# 4. n\_estimators=100, max\_depth=10

# ... and so on (9 combinations total)

# Suppose best combination is: n\_estimators=200, max\_depth=20

gs.best\_params\_ = {'n\_estimators': 200, 'max\_depth': 20}

# Apply to model:

model.set\_params(n\_estimators=200, max\_depth=20)

model.fit(X\_train, y\_train)  # Train with best parameters

'''

# exception.py

"""

# When this executes:

self.error\_message = error\_message\_details(error\_message, error\_detail)

# What happens:

# 1. The constructor's 'error\_message' parameter value is passed

# 2. It becomes the 'error' parameter in error\_message\_details()

# 3. Inside the function, a NEW local variable 'error\_message' is created

# 4. This local variable is returned and stored in self.error\_message

"""

# data\_transformation.py

'''

 # Example Working 1

# preprocessing\_obj.fit\_transform() does:

# 1. SPLITS data by column names

# 2. Applies num\_pipeline to ["writing\_score", "reading\_score"]

# 3. Applies cat\_pipeline to ["gender", "race\_ethnicity", ...]

# 4. COMBINES results into final array

# Output (input\_feature\_train\_arr):

# [[0.85, -1.23, 1.0, 0.0, 0.0, ...],  # ← Scaled nums + One-hot encoded cats

#  [-0.92, 0.45, 0.0, 1.0, 0.0, ...],

#  ...]

'''

'''

# Example Working 2

# Transformed features (without target)

input\_feature\_train\_arr = [

    [0.85, -1.23, 1.0, 0.0, 0.0],  # ← Features only

    [-0.92, 0.45, 0.0, 1.0, 0.0],

    [1.15, 0.89, 0.0, 0.0, 1.0]

]

# Target values (separate)

target\_feature\_train\_df = [85, 72, 90]  # ← Math scores

After np.c\_[]:

train\_arr = np.c\_[input\_feature\_train\_arr, target\_feature\_train\_df]

# Result:

# [

#   [0.85, -1.23, 1.0, 0.0, 0.0, 85],  # ← Features + Target together!

#   [-0.92, 0.45, 0.0, 1.0, 0.0, 72],

#   [1.15, 0.89, 0.0, 0.0, 1.0, 90]

# ] '''

# model\_trainer.py

'''

Example Working 1

best\_model\_score = max(sorted(model\_report.values()))

What happens:

model\_report.values() → [0.72, 0.89, 0.91, 0.65, 0.78]

sorted([0.72, 0.89, 0.91, 0.65, 0.78]) → [0.65, 0.72, 0.78, 0.89, 0.91]

max([0.65, 0.72, 0.78, 0.89, 0.91]) → 0.91

Result: best\_model\_score = 0.91

Example Working 2

What happens:

model\_report.values() → [0.72, 0.89, 0.91, 0.65, 0.78]

.index(0.91) → Finds position of 0.91 in the list → 2 (3rd position)

list(model\_report.keys()) → ["Linear Regression", "Random Forest", "XGBoost", "Decision Tree", "K-Neighbors"]

["Linear Regression", "Random Forest", "XGBoost", "Decision Tree", "K-Neighbors"][2] → "XGBoost"

Result: best\_model\_name = "XGBoost"

best\_model = models[best\_model\_name]

# best\_model = models["XGBoost"] → XGBRegressor() object

Example Working 3

Step 1: Understand model\_report

model\_report = {

    "Linear Regression": 0.72,

    "Random Forest": 0.89,

    "XGBoost": 0.91,      # ← This is the best!

    "Decision Tree": 0.65

}

Step 2: max(model\_report, key=model\_report.get) Breakdown

What max() normally does:

max(["apple", "banana", "cherry"])  # → "cherry" (alphabetical)

But with key parameter:

max(model\_report, key=model\_report.get)

What happens internally:

Iterates through keys: "Linear Regression", "Random Forest", "XGBoost", "Decision Tree"

For each key, gets the value using model\_report.get(key):

model\_report.get("Linear Regression") → 0.72

model\_report.get("Random Forest") → 0.89

model\_report.get("XGBoost") → 0.91

model\_report.get("Decision Tree") → 0.65

Finds the maximum among these values: 0.91

Returns the corresponding key: "XGBoost"

Step 3: Get the Score

best\_model\_score = model\_report[best\_model\_name]

# best\_model\_score = model\_report["XGBoost"] → 0.91

Step 4: Get the Model Object

best\_model = models[best\_model\_name]

# best\_model = models["XGBoost"] → The actual XGBRegressor() object

'''