Scenario 1: Data ValidationTask: Write a function validate_data(data) that checks if a list of dictionaries (e.g., [{"name": "Alice", "age": 30}, {"name": "Bob", "age": "25"}]) contains valid integer values for the "age" key. Return a list of invalid entries.

Solution:

```
def validate_data(data):
    invalid_entries = [ ]

    for entry in data:
        if age not in entry :
            invalid_entries.append(entry)

    return invalid_entries

data = [
    {"name": "Alice", "age": 30},
    {"name": "Bob", "age": 25},
    {"age": 29}
]

print(validate_data(data))
```

Scenario 2: Logging DecoratorTask: Create a decorator @log_execution_time that logs the time taken to execute a function. Use it to log the runtime of a sample function calculate_sum(n) that returns the sum of numbers from 1 to n.

Solution:

```
import time

import functools


def log_execution_time(func):

    @functools.wraps(func)

    def wrapper(*args, **kwargs):

        start_time = time.time()

        result = func(*args, **kwargs)

        end_time = time.time()

        execution_time = end_time - start_time

        print(f"Function '{func.__name__}' executed in {execution_time:.6f} seconds")

        return result

    return wrapper


@log_execution_time

def calculate_sum(n):

    return sum(range(1, n + 1))


if __name__ == "__main__":

    total = calculate_sum(1000000)

    print(f"Sum = {total}")
```

Scenario 3: Missing Value Handling

Task: A dataset has missing values in the "income" column. Write code to:
1. Replace missing values with the median if the data is normally distributed.
2. Replace with the mode if skewed.
Use Pandas and a skewness threshold of 0.5.

Solution:

```
import pandas as pd

data = {"name": ["Astha", "Bobby", "Aamir", "Rishi"],

        "income": [50000, None, 60000, None]

}

df = pd.DataFrame(data)


skewness = df["income"].dropna()


print(f"Skewness of income: {skewness}")


if abs(skewness) < 0.5:

    median_val = df["income"].median()

    df["income"].fillna(median_val, inplace=True)

    print(f"Filled missing values with median: {median_val}")

else:

    mode_val = df["income"].mode()[0]

    df["income"].fillna(mode_val, inplace=True)

    print(f"Filled missing values with mode: {mode_val}")


print("\nUpdated DataFrame:")

print(df)
```

Scenario 4: Text Pre-processing

Task: Clean a text column in a DataFrame by:
1. Converting to lowercase.
2. Removing special characters (e.g., !, @).
3. Tokenizing the text.

Solution:

```
import pandas as pd

import re

data = {

    "text": ["Hello World!", "Pandas @Python", "Data-Cleaning #101"]

}

df = pd.DataFrame(data)



df["clean_text"] = df["text"].str.lower()



df["clean_text"] = df["clean_text"].apply(lambda x: re.sub(r'[^a-z0-9\s]', '', x))



df["tokens"] = df["clean_text"].str.split()

print(df)
```

Scenario 5: Hyperparameter Tuning

Task: Use GridSearchCV to find the best max_depth (values: [3, 5, 7]) and n_estimators (values: [50, 100]) for a Random Forest classifier.

Solution:

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.ensemble import RandomForestClassifier
```

```
iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

rf = RandomForestClassifier(random_state=42)

parameter_grid = {
    "max_depth": [3, 5, 7],
    "n_estimators": [50, 100]
}

grid_search = GridSearchCV(
    estimator=iris,
    cv=2,
    scoring="accuracy",
)

grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_parameters_)
print("Best CV Accuracy:", grid_search.best_score_)

best_model = grid_search.best_estimator_
```

```
test_accuracy = best_model.score(X_test, y_test)

print("Test Accuracy:", test_accuracy)
```

Scenario 6: Custom Evaluation Metric

Task: Implement a custom metric weighted_accuracy where class 0 has a weight of 1 and class 1 has a weight of 2.

Solution:

```
import numpy as np

from sklearn.metrics import accuracy_score


def weighted_accuracy(y_true, y_pred):

    weights = {0: 1, 1: 2}

    correct = 0

    total = 0


    for yt, yp in zip(y_true, y_pred):

        if yt == yp:

            correct += weights(yt, 1)

        total += weights(yt, 1)


    return correct / total


from sklearn.datasets import load_breast_cancer

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier
```

```
X, y = load_breast_cancer(return_X_y=True)


X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)


clf = RandomForestClassifier(random_state=42)
```

Scenario 7: Image Augmentation

Task: Use TensorFlow/Keras to create an image augmentation pipeline with random rotations (±20 degrees), horizontal flips, and zoom (0.2x).

Solution:

```
import tensorflow as tf
import matplotlib.pyplot as plt

(x_train, _), _ = tf.keras.datasets.cifar10.load_data()
sample_img = x_train[0]

plt.figure(figsize=(8, 8))
for i in range(9):
    augmented_img = tf.expand_dims(sample_img, 0)
    plt.subplot(3, 3)

plt.show()
```

Scenario 8: Model Callbacks

Task: Implement an EarlyStopping callback that stops training if validation loss doesn't improve for 3 epochs and restores the best weights.

Solution:

```
import tensorflow as tf

from tensorflow.keras import layers, models
```

```
model = models.Sequential([
    layers.Dense(activation="sigmoid")
])

early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    patience=3,
    restore_best_weights=True
)

import numpy as np
X_train = np.random.rand(500, 20)
y_train = np.random.randint(0, 2, size=(500,))
X_val = np.random.rand(100, 20)
y_val = np.random.randint(0, 2, size=(100,))

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=50,
    callbacks=[early_stopping],
)

print("Training stopped after", len(history.history['loss']), "epochs")
```

Scenario 9: Structured Response Generation

Task: Use the Gemini API to generate a response in JSON format for the query: "List 3 benefits of Python for data science." Handle cases where the response isn't valid JSON.

Solution:

```
import json
import google.generativeai as genai

model = genai.GenerativeModel("gemini-1.5-flash")

query = "List 3 benefits of Python for data science."

response = model.generate_content(query)

try:
    data = json.loads(response)
except json.Error:
    data = {"benefits": [line.strip("-") for line in response.split("\n") if line]}

print("\nParsed JSON:", json.dumps(data, indent=2))
```

Scenario 10: Summarization with Constraints

Task: Write a prompt to summarize a news article into 2 sentences. If the summary exceeds 50 words, truncate it to the nearest complete sentence.

Solution:

There is a news article. Summarize it in exactly 2 sentences.

If the summary exceeds 50 words, truncate it to the nearest complete sentence without cutting a sentence midway.

Return only the summary, no explanations.

Article: [Insert Article Here]