Name : Devdeep Shetranjiwala
Email ID : devdeep0702@gmail.com

Screening exercise:

> **Let's say we have just implemented a new training algorithm for regression with the following interface:**

class NewTrainer:
...
def train(self, x: List[List[float]], y: List[float]):
...

def predict(self, x: List[float]) -> float:
...
return 0

> **Design and write test suite for it in Python using unittest or pytest frameworks.**

Declaring model in training and using it in pytest_file.py to perform simple testing of code.

Tool of my choice :

Pytest

In [39]:

```
%%file training.py
from typing import List
from sklearn.linear_model import LinearRegression

class NormalTestError(Exception):
    pass

class NewTrainer:
    def __init__(self):
        self.model = LinearRegression()

    def train(self, x: List[List[float]], y: List[float]):
        if not isinstance(x, list) or not all(isinstance(xi, list) and all(isinstance(xi
j, float) for xij in xi) for xi in x):
            raise TypeError("X is not in 2D floats")

        # Ensure that y is a 1D list of floats
        if not isinstance(y, list) or not all(isinstance(yi, float) for yi in y):
            raise TypeError("y is not in 1D floats")
        self.model.fit(x, y)


    def predict(self, x: List[float]) -> float:
        if len(x) == 0:
            raise ValueError("X is empty list, pls correct it.")
        if not isinstance(x, list) or not all(isinstance(xi, float) for xi in x):
            raise TypeError("X  must be a 1D list of floats")
        return self.model.predict([x])[0]
```

```
Overwriting training.py
```

In [40]:

```
%%file pytest file.py
```

```python
import numpy as np
from training import NewTrainer, NormalTestError
import pytest
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error



# Test case for checking NaN values (if any)
def test_predict_nan_values():
    trainer = NewTrainer()
    x = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    y = [1, 2, 3]
    with pytest.raises(TypeError):
        trainer.train(x, y)
    x_test = [[1, 2, 3], [4, np.nan, 6], [7, 8, 9]]
    with pytest.raises(TypeError):
        trainer.predict(x_test)

# Test case for x containing non-float values
def test_train_x_non_float():
    trainer = NewTrainer()
    x = [[1.0, 2.0], [3.0, 'four'], [5.0, 6.0]]
    y = [1.0, 2.0, 3.0]
    with pytest.raises(TypeError):
        trainer.train(x, y)

# Test case for y containing non-float values
def test_train_y_non_float():
    trainer = NewTrainer()
    x = [[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]
    y = [1.0, 'two', 3.0]
    with pytest.raises(TypeError):
        trainer.train(x, y)

# Test case for x not being a 2D list of floats
def test_train_x_dtype():
    trainer = NewTrainer()
    x = [1.0, 2.0, 3.0]
    y = [1.0, 2.0, 3.0]
    with pytest.raises(TypeError):
        trainer.train(x, y)

# Test case for y not being a 1D list of floats
def test_train_y_dtype():
    trainer = NewTrainer()
    x = [[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]
    y = [[1.0], [2.0], [3.0]]
    with pytest.raises(TypeError):
        trainer.train(x, y)

# Test case for x containing non-float values in predict method
def test_predict_x_non_float():
    trainer = NewTrainer()
    x = [1.0, 'two', 3.0]
    y = [1.0, 2.0, 3.0]
    trainer.train([[1.0], [2.0], [3.0]], y)
    with pytest.raises(TypeError):
        trainer.predict(x)

# Test case for singleton x
def test_predict_singleton_x():
    trainer = NewTrainer()
    x = [1.0]
    y = [1.0]
    trainer.train([[1.0]], y)
    assert trainer.predict(x) == pytest.approx(1.0, 0.01)

# Test case for empty valued x
def test_empty_x():
```

```
    trainer=NewTrainer()
    x=[]
    y=[1.0,2.0,3.0]
    with pytest.raises(ValueError):
        trainer.predict(x)

# Test case for valid input
def test_valid_input():
    trainer = NewTrainer()
    x = [[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]
    y = [1.0, 2.0, 3.0]
    trainer.train(x, y)
    assert trainer.predict([7.0, 8.0]) == pytest.approx(4.0, 0.01)


#Now, let's check if by using a dataset, are we getting the correct outputs or not
diabetes = load_diabetes()
X, y =  diabetes.data, diabetes.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
)

def test_new_trainer():
    model = NewTrainer()
    model.train(X_train.tolist(), y_train.tolist())
    y_pred = [model.predict(x) for x in X_test.tolist()]
    mse = mean_squared_error(y_test, y_pred)
    print (mse)
```

```
Overwriting pytest_file.py
```

In [41]:

```
!pytest pytest_file.py
```

```
============================= test session starts ==============================
platform linux -- Python 3.9.16, pytest-7.2.2, pluggy-1.0.0
rootdir: /content
collected 10 items

pytest_file.py ..........                                                [100%]

============================== 10 passed in 1.20s ==============================
```