Part a)

```
# question 1 - part a
# read the dataset as a set of arrays.

import numpy as np
from typing import Tuple

def load_dataset(src_dir: str) -> Tuple[np.ndarray, np.ndarray,
np.ndarray, np.ndarray, np.ndarray]:

    x_train = np.loadtxt(src_dir + 'x_train.txt')
    y_train = np.loadtxt(src_dir + 'y_train.txt')
    x_val = np.loadtxt(src_dir + 'x_val.txt')
    y_val = np.loadtxt(src_dir + 'y_val.txt')
    x_test = np.loadtxt(src_dir + 'x_test.txt')

    return x_train, y_train, x_val, y_val, x_test
```

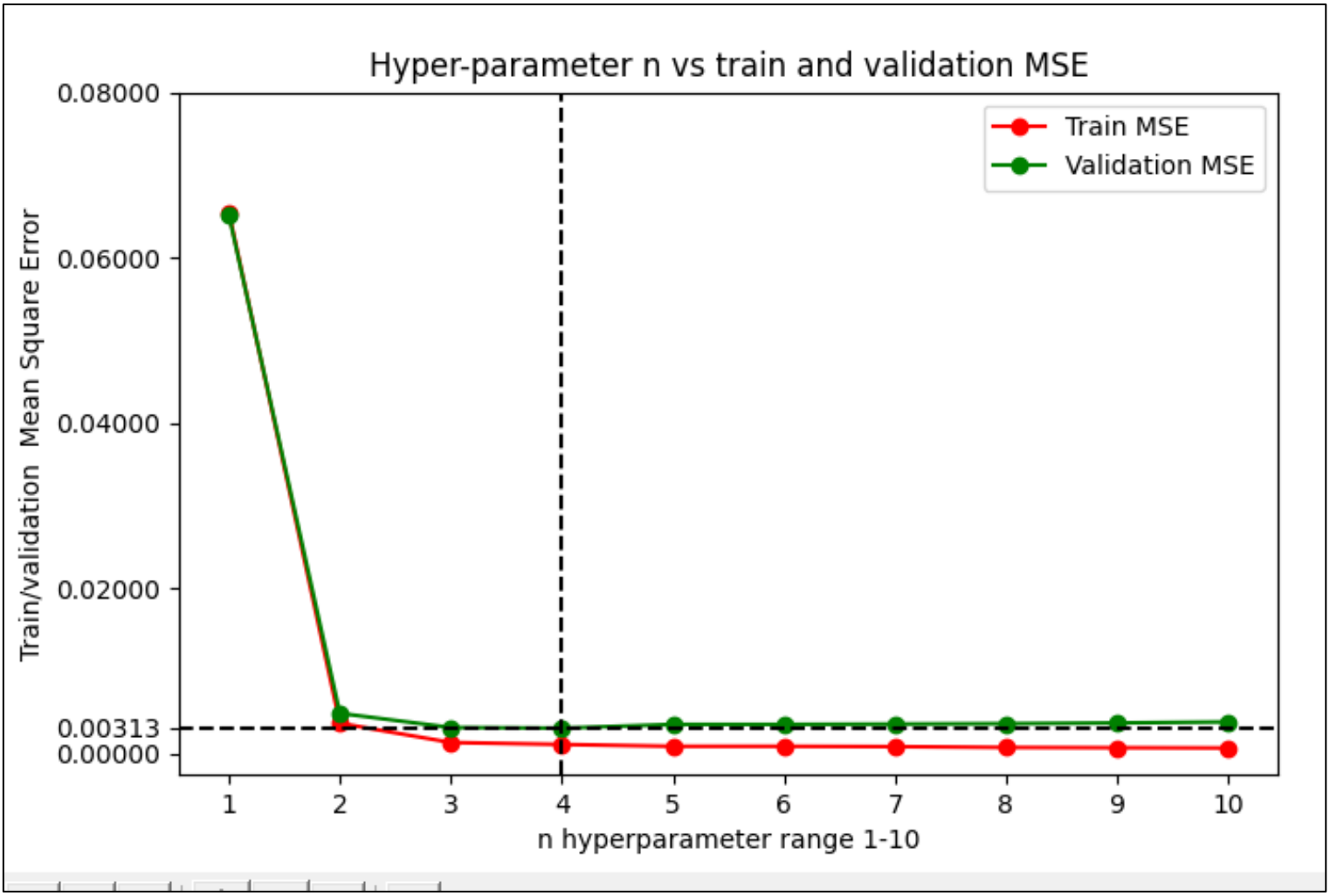
Part b) i.

```
# n - hyper parameter
# each w is a weight
# question 1 part b sub part a
# make input features for the above linear regression model.

def get_features(x: np.ndarray, n: int) -> np.ndarray:
    features = []
    for i in range(1,n+1):
        features.append(np.power(x,i))

    features_nparray = np.array(features)
    features_output = np.transpose(features_nparray)

    return features_output
```

Part b) ii.

```
# question 1 part b sub part 2
# fit and evaluate function

from sklearn.linear_model import LinearRegression
# from sklearn.metrics import mean_squared_error # this can be
achieved using np.mean function

def fit_and_evaluate(x_train:np.ndarray, y_train:np.ndarray,
x_val:np.ndarray, y_val:np.ndarray, n:int) -> Tuple[float,
float]:
    regressor = LinearRegression(fit_intercept = False)

    x_train_features = get_features(x_train, n)
    x_val_features = get_features(x_val, n)
    regressor.fit(x_train_features,y_train)

    # W = regressor.coef_

    y_predict_train = regressor.predict(x_train_features)
    y_predict_val = regressor.predict(x_val_features)

    train_mse = np.mean((y_predict_train - y_train)**2)
    val_mse = np.mean((y_predict_val - y_val)**2)

    return train_mse, val_mse
```

Part c)



Part d)

Since the validation MSE is least at n=4 we chose it for prediction on test set.