```python
import numpy as np

def fun(x):
    """
        Compute a simple quadratic function f:R^5 -> R.

    """

    return 1.3 + 2.*x[0] - 1.1*x[1] + 0.7*x[2]  + 1.2*x[3] + \
            0.4* (x[0]**2) -1.5 *x[1]*x[3] - 0.7 * (x[4]**2)

def postfix(N,d,sigma):
    """Converts parameters into a handy string, to be appended to file names."""
    return "_N_%d_d_%d" % (N,d) \
         + "_sig_%s" % str(sigma).replace(".","_")


if __name__ == "__main__":
    # Set random seed for reproducibility
    np.random.seed(42)


    # Number of samples
    N = 1000

    # Noise variance
    sigma = 0.01


    # Feature dimension
    d = 40

    print("Generating dataset with N = %d, σ = %f, d = %d..." % (N,sigma,d), end="")

    # Generate random features
    X = np.random.randn(N, d)

    # Generate pure labels
    y = []

    for i in range(N):
        y.append(
                    fun(X[i,:])
```

```
                       )
    y = np.array(y)

    # Add noise to labels
    err = np.random.normal(scale = sigma, size = N)
    y = y + err

    print(" done")

    psfx = postfix(N,d,sigma)

    print("Saving X and y... ",end="")
    np.save("X" + psfx,X) #saves X with name, X N d sigma
    np.save("y" + psfx,y) #saves y with name, y N d sigma
    print(" done")



```

```python
import numpy as np
from sklearn.model_selection import KFold,train_test_split,
cross_val_score
from sklearn.linear_model import Lasso
from sklearn.metrics import root_mean_squared_error as rmse
import matplotlib.pyplot as plt
from data_generator import postfix


# Number of samples
N = 1000

# Noise variance
sigma = 0.01


# Feature dimension
d = 40

psfx = postfix(N,d,sigma)

# function lift = liftDataset
def lift(x_initial):
    x_prime = []

    xi = len(x_initial)
    for row in range(xi):
        x_expand = []
        x_0 = x_initial[row]

        l = len(x_0)
        for i in range(l):
            x_expand.append(x_0[i])
            for j in range(i,l):
                x_expand.append(x_0[j]*x_0[i])

        x_prime.append(x_expand)

    return np.array(x_prime)


X = np.load("X"+psfx+".npy")
y = np.load("y"+psfx+".npy")
X = lift(X)

```

```
print("Dataset has n=%d samples, each with d=%d features," % X.shape,"as well as %d labels." % y.shape[0])

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=42)

print("Randomly split dataset to %d training and %d test samples" % (X_train.shape[0],X_test.shape[0]))


# alpha = 0.1
#
# model = Lasso(alpha = alpha)
#
# cv = KFold(
#         n_splits=5,
#         random_state=42,
#         shuffle=True
#         )
#
#
#
# scores = cross_val_score(
#         model, X_train, y_train, cv=cv,scoring="neg_root_mean_squared_error")
#
#
# print("Cross-validation RMSE for α=%f : %f ± %f" % (alpha,-np.mean(scores),np.std(scores)) )

# === New code: scan alpha values and pick the best one ===

alphas = 2.0 ** np.arange(-10, 11)  # from 2^-10 to 2^10
cv = KFold(n_splits=5, random_state=42, shuffle=True)

mean_rmse = []
std_rmse = []

print("Performing 5-fold CV for multiple α values...")

for alpha in alphas:
    model = Lasso(alpha=alpha, max_iter=10000, random_state=42)
    scores = cross_val_score(
        model, X_train, y_train,
        cv=cv,
        scoring="neg_root_mean_squared_error"
    )

    mean_rmse.append(-np.mean(scores))
    std_rmse.append(np.std(scores))
    print("α = %8.5f -> CV RMSE = %.6f ± %.6f" % (alpha, -np.mean(scores), np.std(scores)))

# Find best alpha (lowest mean RMSE)
best_idx = np.argmin(mean_rmse)
best_alpha = alphas[best_idx]

print("\nBest α found: %f (CV RMSE = %.6f)" % (best_alpha, mean_rmse[best_idx]))

model = Lasso(alpha=best_alpha, max_iter=10000, random_state=42)
print("Fitting linear model over entire training set...",end="")
model.fit(X_train, y_train)
print(" done")


# Compute RMSE
rmse_train = rmse(y_train,model.predict(X_train))
rmse_test = rmse(y_test,model.predict(X_test))

print("Train RMSE = %f, Test RMSE = %f" % (rmse_train,rmse_test))



# Plot CV mean RMSE as a function of alpha with error bars
plt.figure(figsize=(8, 5))
plt.errorbar(alphas, mean_rmse, yerr=std_rmse, fmt='-o', capsize=4, label='CV RMSE')
plt.xlabel('Alpha')
plt.ylabel('Mean Cross-Validation RMSE')
plt.title('RMSE vs Alpha')
plt.grid(True, which='both', linestyle='--', alpha=0.6)
plt.legend()
plt.show()



print("Model parameters:")
print("\t Intercept: %3.5f" % model.intercept_,end="")
```

```python
for i,val in enumerate(model.coef_):
    print(", β%d: %3.5f" % (i,val), end="")
print("\n")



```

```python
import numpy as np
from mpmath.matrices.eigen_symmetric import c_he_tridiag_0
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.metrics import root_mean_squared_error as rmse
from data_generator import postfix



# Number of samples
N = 1000

# Noise variance
sigma = 0.01

# Feature dimension
d = 40


psfx = postfix(N,d,sigma)

# function lift = liftDataset
def lift(x_initial):
    x_prime = []

    xi = len(x_initial)
    for row in range(xi):
        x_expand = []
        x_0 = x_initial[row]

        l = len(x_0)
        for i in range(l):
            x_expand.append(x_0[i])
            for j in range(i,l):
                x_expand.append(x_0[j]*x_0[i])

        x_prime.append(x_expand)

    return np.array(x_prime)
# test lift function
# x_initials = [[1,2,3,4,5],
#               [2,4,6,8,10]]
# new_x_i =lift(x_initials)
# print(new_x_i)
```

```python



X = np.load("X"+psfx+".npy")
y = np.load("y"+psfx+".npy")



print("Dataset has n=%d samples, each with d=%d features," % X.shape,"as well as %d labels." % y.shape[0])
frac = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
rmse_train_array = []
rmse_test_array = []
training_length_array = []
X = lift(X)

for i, fr in enumerate(frac):
    # ... use both i and frac
    print(f"Training with {fr*100}% of the training data")
    #Split data: test set made up of fraction of N data (ex 0.30*1000 = 300)
    testSize = 0.30
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=testSize, random_state=42)

    print("Randomly split dataset to %d training and %d test samples" % (X_train.shape[0],X_test.shape[0]))
    print("Training with number of n samples = %d" % int(len(X_train) * fr))

    ### select only a fraction (ex:10%) of the test data (X_train y_train)
    #fr = 0.1 # fraction of test set (ex 10%)
    # Calculate the number of elements to select (ex:10% of 1000)
    num_elements_to_select = int(len(X_train) * fr)

    # Generate random indices
    random_indices = np.random.choice(len(X_train), num_elements_to_select, replace=False)

    #print("Original X array:", X_train)
    #print("Original y array:", y_train)

    # Select the elements using the random indices
    X_train = X_train[random_indices]
    y_train = y_train[random_indices]

    #print(f"Selected {fr} of X data:", X_train)
    #print(f"Selected {fr} of y data:", y_train)

    ###


    model = LinearRegression()

    print("Fitting linear model...",end="")
    model.fit(X_train, y_train)
    print(" done")


    # Compute RMSE on train and test sets
    rmse_train = rmse(y_train,model.predict(X_train))
    rmse_test = rmse(y_test,model.predict(X_test))
    rmse_train_array.append(rmse_train)
    rmse_test_array.append(rmse_test)
    training_length_array.append(len(y_train))

    print("Train RMSE = %f, Test RMSE = %f" % (rmse_train, rmse_test))


    print("Model parameters:")
    print("\t Intercept: %3.5f" % model.intercept_,end="")
    for i,val in enumerate(model.coef_):
        print(", β%d: %3.5f" % (i,val), end="")
    print("\n")
    #print(f'x1',X_test)

#Plot the train and test RMSE as a function of the number of training samples given to model

plt.figure(figsize=(8, 6))
plt.plot(training_length_array, rmse_train_array, marker='o', label='Train RMSE')
#plt.plot(training_length_array, rmse_test_array, marker='s', label='Test RMSE')
plt.xlabel("Training Set Size: number of training samples")
plt.ylabel("RMSE")
plt.title(f"Training and Test RMSE vs Training Set Size (Data from {psfx})")
plt.legend()
plt.grid(True)
plt.show()



```