In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
np.set_printoptions(suppress=True)
import seaborn as sns
from math import sin
import time
from sklearn.metrics import mean_squared_error as mse
import pickle as pkl
```

In [2]:
```
trainData=pd.read_csv('Dataset/problem4b_train.csv',header=None)
trainData.head()
```

Out[2]:

	0	1	2	3	4
0	28.46	49.16	1004.6	38.10	438.02
1	27.20	49.16	1005.3	46.73	440.57
2	27.36	66.54	1011.3	45.30	436.69
3	10.77	41.46	1021.4	87.18	479.22
4	14.79	47.83	1007.3	92.04	463.22

In [3]:
```
x_train=trainData.iloc[:,:4].values
x_train.shape
```

Out[3]: (8611, 4)

In [4]:
```
y_train=trainData.iloc[:,-1].values
y_train.shape
```

Out[4]: (8611,)

In [5]:
```
testData=pd.read_csv('Dataset/problem4b_test.csv',header=None)
testData.head()
```

Out[5]:

	0	1	2	3
0	12.71	43.80	1023.1	71.16
1	25.69	57.32	1012.3	44.18
2	14.40	42.18	1015.5	77.70
3	11.45	40.80	1027.7	78.60
4	25.36	61.41	1011.9	55.64

In [6]:
```
x_test=testData.iloc[:,:].values
x_test.shape
```

Out[6]: (957, 4)

In [7]:
```
def get_y(x_test):

    y_test=np.genfromtxt('Dataset/problem4b_sol.csv',delimiter=',')
    return y_test
y_test=get_y(x_test)
y_test=y_test.reshape(-1,1)
print(y_test.shape)
```

```
(957, 1)
```

In [8]:
```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
from sklearn.gaussian_process.kernels import RationalQuadratic as RQ
from sklearn.gaussian_process.kernels import ExpSineSquared as ESS
```

In [9]:
```
C1 = C(1.0, (1e-3, 1e3))
C2 = C(0.5, (1e-3, 1e3))
RBF1 = RBF(10, (1e-2, 1e2))
RBF2 = RBF(0.5, (1e-2, 1e2))
RQ1 = RQ(10, 0.5 ,(1e-2, 1e2))
#ESS1 = ESS(1.0, 1.0, (1e-05, 100000.0), (1e-05, 100000.0))
kernel1 = C1 * RBF1 + C2
kernel2 = C1 * RBF1 + RBF2
kernel3 = C1 * RQ1 + RBF2
#kernel4 = C1 * ESS1 + RBF2
```

In [ ]:
```
GP = []
for ndx, kernel in zip([1,2,3], [kernel1,kernel2, kernel3]):
    t = time.time()
    print('------------------------------------------------------------
    print(f'time - {t} :: Fitting GP for kernel - {ndx}')
    gp = GaussianProcessRegressor(kernel=kernel, alpha=0.5 ** 5,
                                   n_restarts_optimizer=1)
    gp.fit(x_train, y_train)
    with open( f"Dataset/GP_for_Kernel_CCPP_{ndx}.pkl", "wb" ) as f:
        pkl.dump(gp, f)
    GP.append(gp)
    print(f'GP for Kernel - {ndx} Finished :: Elapsed Time - {time.time
    print('------------------------------------------------------------
```

In [ ]:
```
y_pred=[]
sigma=[]
for gp in (GP):
    pred1, sigma1 = gp.predict(x_test, return_std=True)
    print(gp.kernel_)
    y_pred.append(pred1)
    sigma.append(sigma1)
y_pred1=np.array(y_pred)
y_pred1=y_pred1.T
for i in range(0,y_pred1.shape[1]):
    print(f'MSE for kernel {i} is {mse(y_pred1[:,i],y_test)}')
```

In [ ]:
```
for i in range(0,len(y_pred)):
    plt.plot(x_train, y_train, 'r:', label=r'$f(x) = sin(3x)$')
    plt.errorbar(x_train.ravel(), y_train, 0, fmt='r.', markersize=10, l
    plt.plot(x_test, y_pred[i], 'b-', label='Prediction')
    plt.fill(np.concatenate([x_test, x_test[::-1]]),
                 np.concatenate([y_pred[i] - 1.95 * sigma[i],
                               (y_pred[i] + 1.95 * sigma[i])[::-1]]),
                 alpha=.5, fc='b', ec='None', label='95% confidence inte
    plt.legend(loc='upper left')
    plt.show()
```

In [ ]:
```
f, axs = plt.subplots(2,2)
for ndx, gp, ax in zip([1,2,3,4], GP, [[0,0],[0,1],[1,0],[1,1]]):
    y_pred, sigma = gp.predict(x_test, return_std=True)
    axs[ax[0],ax[1]].plot(x_train, y_train, 'r:', label=r'$f(x) = sin(3x
    axs[ax[0],ax[1]].errorbar(x_train.ravel(), y_train,0, fmt='r.', mark
    axs[ax[0],ax[1]].plot(x_test, y_pred, 'b-', label='Prediction')
    axs[ax[0],ax[1]].fill(np.concatenate([x_test, x_test[::-1]]),
             np.concatenate([y_pred - 1.95 * sigma,
                           (y_pred + 1.95 * sigma)[::-1]]),
             alpha=.5, fc='b', ec='None', label='95% confidence interva
    axs[ax[0],ax[1]].set_xlabel('$x$')
    axs[ax[0],ax[1]].set_ylabel('$f(x)$')
    axs[ax[0],ax[1]].set_ylim(-10, 20)
    #axs[ax[0],ax[1]].legend(loc='upper left')
```