```
 _                _                          _
| |__  __      __/ |         ___   ___    __| |  ___
| '_ \ \ \ /\ / /| |        / __| / _ \  / _` | / _ \
| | | | \ V  V / | |       | (__ | (_) || (_| ||  __/
|_| |_|  \_/\_/  |_| _____  \___| \___/  \__,_| \___|
                    |_____|
```

# .~lock.USArrests.csv#

```
,adam,ahlan,07.07.2023 02:20,file:///home/adam/.config/libreoffice/4;
```

# ch12prob9.py

```python
from numpy import *
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import stats
from scipy.interpolate import CubicSpline
from scipy.interpolate import splrep, BSpline


#np.genfromtxt("Auto.csv", delimiter=",")
#with open("Auto.csv", "r") as f:
#    print(f.read())

#data_a = loadtxt('Auto.csv',skiprows=1, usecols=(0,3), delimiter=',')
#print(data_a)

#CHAPTER 3 QUESTION 9

def data_loader(fname):
    data_a = loadtxt(fname,skiprows=1, usecols=(0,1,2,3,4,5,6,7), delimiter=',')

    #mpg        cylinders       displacement    horsepower      weight  acceleration    year    origin  name


    #print(data_a.shape)
    #print(data_a[0:,1].shape)
    # We want it in the form  of Y = XB
    # Where Y is the response variable
    # Where X is an array with size nx2 where n is the predictor variable and the other column is a 1
    # B is the coeeficents (slope and intercept) that we are trying to solve for    year_v = data_a[:,1]

    n, m = data_a.shape
    pred_a = data_a[:, 1:]

    x_a = empty([n,pred_a.shape[1]+1], dtype=float64)
    x_a[:,:-1] =  pred_a
    x_a[:,-1] = 1
    name_l = ['mpg',    'cylinders',    'displacement', 'horsepower',   'weight',       'acceleration', 'year', 'origin']

    #print(x_v.shape)

    y_v = data_a[:,0]

    return x_a, y_v, data_a, name_l

def scatter_matrix(data_a, name_l):
    n, p = data_a.shape
    fig, axs = plt.subplots(4, 7)
    ax_l = list(axs.flat)
    mpl.rcParams['figure.autolayout'] = True
    font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 10}

    #mpl.rc('font', **font)
    for i in range(p):
        for j in range(i+1,p):
            print(i, j, name_l[i], name_l[j])
            x_v = data_a[:,i]
            y_v = data_a[:,j]
            ax = ax_l.pop(0)
            ax.scatter(x_v, y_v, s=2**2)
            title = f'{name_l[i]} vs {name_l[j]}'
            ax.set_title(title[:25])
    plt.tight_layout()
    fig.savefig('ch12_9.pdf')
    plt.show()

def diag_plot(y_v, yfit_v):
    res_v = y_v - yfit_v
    i_v = yfit_v.argsort()
    yfit1_v = yfit_v[i_v]
    res1_v = res_v[i_v]
    fig, ax = plt.subplots(figsize=(15, 15))
    ax.scatter(yfit1_v,res1_v)
    #spline = CubicSpline(yfit1_v, res1_v)
    #res2_v = spline(yfit1_v)
    tck = splrep(yfit1_v, res1_v, s=3500)
    res2_v = BSpline(*tck)(yfit1_v)
    #ax.plot(yfit1_v, res1_v)
    ax.plot(yfit1_v, res2_v)

    plt.show()

def transform1_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = (p_a)**2

    return hstack((p_a,x_a))

def transform2_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = (p_a)**0.5

    return hstack((p_a,x_a))

def transform3_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = log((p_a))

    return hstack((p_a,x_a))

def transform4_reg(x_a):
    p_a = x_a[:,2:4]
    p_a = product(p_a,axis=1,keepdims=True)

    return hstack((p_a,x_a))

def transform5_reg(x_a):
    p_a = x_a[:,2:5]
    p_a = product(p_a,axis=1,keepdims=True)

    return hstack((p_a,x_a))


def interaction_study(x_a, y_v, name):
    b_v = lin_regression(x_a,y_v)
    yfit_v = fitted_func(x_a, b_v)
    print(f'{name}: Coefficients {b_v=}')
    print(f'{name}: r squared = ', r_square(y_v,yfit_v))




def lin_regression(x_a,y):
    #y_v = X@B
    b_v = linalg.pinv(x_a)@y
    #print(b)
    return b_v

def fitted_func(x_a,b_v):
    yfit_v = x_a@b_v
    # with np.printoptions(precision=2):
    #     print(f'predicted mpg of cars {yfit_v=}')
    return yfit_v

def r_square(y_v,yfit_v):
    # This function is to find the r squared value
    # This will be calcualted by doing 1 - variance of (actual - predicited)/variance of actual
    rsq = 1 - (var(y_v-yfit_v))/(var(y_v))

    return rsq

def main ():
    x_a, y_v, data_a, name_l  = data_loader('Auto.csv')
    cor_a = corrcoef(data_a, rowvar=False)
    print(cor_a.shape)
    with np.printoptions(precision=4):
        print(cor_a)

    #print(x_a,y_v)
    #scatter_matrix(data_a, name_l)

    #print(x_a,y_v)
    b_v = lin_regression(x_a,y_v)
    yfit_v = fitted_func(x_a, b_v)
    i_v = abs(b_v).argsort()[::-1]
    print(f'Coefficients {b_v=}')
    print('Coefficients:', [(name, b) for name, b in zip(name_l, b_v)])
    print("pred_influence",[name_l[i] for i in i_v])
    print('r squared = ', r_square(y_v,yfit_v))

    #diag_plot(y_v, yfit_v)
    interaction_study(x_a, y_v, 'control')
    interaction_study(transform1_reg(x_a),y_v, 'Horse Power Squared')
    interaction_study(transform2_reg(x_a),y_v, 'Horse Power Square Root')
    interaction_study(transform3_reg(x_a),y_v, 'Horse Power Log')
    interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight')
    interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight * Acceleration')

    raise SystemExit

    #print(yhat_v)
if __name__ == '__main__':
        main()


# zeros, full, empty, array, arange, indexing, transpose

#URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing_splines.html
#
```

# ch3prob9.py

```python
from numpy import *
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import stats
from scipy.interpolate import CubicSpline
from scipy.interpolate import splrep, BSpline
import statsmodels.api as sm


#Adam Kainikara
#This code is for
#CHAPTER 3 QUESTION 9

def data_loader(fname):
    data_a = loadtxt(fname,skiprows=1, usecols=(0,1,2,3,4,5,6,7), delimiter=',')

    #mpg        cylinders       displacement    horsepower      weight  acceleration    year    origin  name


    #print(data_a.shape)
    #print(data_a[0:,1].shape)
    # We want it in the form  of Y = XB
    # Where Y is the response variable
    # Where X is an array with size nx2 where n is the predictor variable and the other column is a 1
    # B is the coeeficents (slope and intercept) that we are trying to solve for    year_v = data_a[:,1]

    n, m = data_a.shape
    pred_a = data_a[:, 1:]

    x_a = empty([n,pred_a.shape[1]+1], dtype=float64)
    x_a[:,:-1] =  pred_a
    x_a[:,-1] = 1
    name_l = ['mpg',    'cylinders',    'displacement', 'horsepower',   'weight',       'acceleration', 'year', 'origin']

    #print(x_v.shape)

    y_v = data_a[:,0]

    return x_a, y_v, data_a, name_l

def scatter_matrix(data_a, name_l):
    n, p = data_a.shape
    fig, axs = plt.subplots(4, 7)
    ax_l = list(axs.flat)
    mpl.rcParams['figure.autolayout'] = True
    font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 10}

    #mpl.rc('font', **font)
    for i in range(p):
        for j in range(i+1,p):
            print(i, j, name_l[i], name_l[j])
            x_v = data_a[:,i]
            y_v = data_a[:,j]
            ax = ax_l.pop(0)
            ax.scatter(x_v, y_v, s=2**2)
            title = f'{name_l[i]} vs {name_l[j]}'
            ax.set_title(title[:25])
    plt.tight_layout()
    fig.savefig('ch12_9.pdf')
    plt.show()

def diag_plot(y_v, yfit_v):
    res_v = y_v - yfit_v
    i_v = yfit_v.argsort()
    yfit1_v = yfit_v[i_v]
    res1_v = res_v[i_v]
    fig, ax = plt.subplots(figsize=(15, 15))
    ax.scatter(yfit1_v,res1_v)
    #spline = CubicSpline(yfit1_v, res1_v)
    #res2_v = spline(yfit1_v)
    tck = splrep(yfit1_v, res1_v, s=3500)
    res2_v = BSpline(*tck)(yfit1_v)
    #ax.plot(yfit1_v, res1_v)
    ax.plot(yfit1_v, res2_v)

    plt.show()

def transform1_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = (p_a)**2

    return hstack((p_a,x_a))

def transform2_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = (p_a)**0.5

    return hstack((p_a,x_a))

def transform3_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = log((p_a))

    return hstack((p_a,x_a))

def transform4_reg(x_a):
    p_a = x_a[:,2:4]
    p_a = product(p_a,axis=1,keepdims=True)

    return hstack((p_a,x_a))

def transform5_reg(x_a):
    p_a = x_a[:,2:5]
    p_a = product(p_a,axis=1,keepdims=True)

    return hstack((p_a,x_a))


def interaction_study(x_a, y_v, name):
    b_v = lin_regression(x_a,y_v,name=name)
    yfit_v = fitted_func(x_a, b_v)
    print(f'{name}: Coefficients {b_v=}')
    print(f'{name}: r squared = ', r_square(y_v,yfit_v))




def lin_regression(x_a,y_v,name=''):
    if name:
        print(f'\n\n----------------------- {name} --------------------')
        model = sm.OLS(y_v, x_a)
        results = model.fit()
        print(results.summary())
    #Using stats models to get p value even though I did my own regression
    # y_v = X@B
    b_v = linalg.pinv(x_a)@y_v
    #print(b)
    return b_v

def fitted_func(x_a,b_v):
    yfit_v = x_a@b_v
    # with np.printoptions(precision=2):
    #     print(f'predicted mpg of cars {yfit_v=}')
    return yfit_v

def r_square(y_v,yfit_v):
    # This function is to find the r squared value
    # This will be calcualted by doing 1 - variance of (actual - predicited)/variance of actual
    rsq = 1 - (var(y_v-yfit_v))/(var(y_v))

    return rsq

def main ():
    x_a, y_v, data_a, name_l  = data_loader('Auto.csv')
    cor_a = corrcoef(data_a, rowvar=False)
    print(cor_a.shape)
    with np.printoptions(precision=4):
        print(cor_a)

    #print(x_a,y_v)
    scatter_matrix(data_a, name_l)

    #print(x_a,y_v)
    b_v = lin_regression(x_a,y_v, name='Main Regression')
    yfit_v = fitted_func(x_a, b_v)
    i_v = abs(b_v).argsort()[::-1]
    print(f'Coefficients {b_v=}')
    print('Coefficients:', [(name, b) for name, b in zip(name_l, b_v)])
    print("pred_influence",[name_l[i] for i in i_v])
    print('r squared = ', r_square(y_v,yfit_v))
    diag_plot(y_v, yfit_v)

    with np.printoptions(precision=2):
        interaction_study(x_a, y_v, 'control')
        interaction_study(transform1_reg(x_a),y_v, 'Horse Power Squared')
        interaction_study(transform2_reg(x_a),y_v, 'Horse Power Square Root')
```

```python
        interaction_study(transform3_reg(x_a),y_v, 'Horse Power Log')
        interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight')
        interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight * Acceleration')

    raise SystemExit

    #print(yhat_v)
if __name__ == '__main__':
        main()


# zeros, full, empty, array, arange, indexing, transpose

#URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing_splines.html
#
```

ch3q14.py

```python
from numpy import *
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import stats
from scipy.interpolate import CubicSpline
from scipy.interpolate import splrep, BSpline
import statsmodels.api as sm


#Adam Kainikara
#This code is for
#CHAPTER 3 QUESTION 14

def data_loader():
    random.seed(1)
    x1_v = random.uniform(size=100)
    x2_v = 0.5*x1_v + random.normal(size=100)/10
    y_v = 2+2*x1_v+0.3*x2_v+random.normal(size=100)
    x_a = empty((100,3), dtype=float64)
    x_a[:,0] = 1
    x_a[:,1] = x1_v
    x_a[:,2] = x2_v
    return x_a, y_v

def scatter_plot(x1_v,x2_v):
    plt.scatter(x1_v,x2_v)
    plt.show()

def rand_var_study(x_a, y_v, name):
    b_v = lin_regression(x_a,y_v,name=name)
    yfit_v = fitted_func(x_a, b_v)
    print(f'{name}: Coefficients {b_v=}')
    print(f'{name}: r squared = ', r_square(y_v,yfit_v))

def transform1_x1_only(x_a):
    #X1 and constant only
    return hstack((x_a[:,0:1],x_a[:,1:2]))


def transform1_x2_only(x_a):
    #X2 and constant only
    return hstack((x_a[:,0:1],x_a[:,2:]))

def transform3_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = log((p_a))

    return hstack((p_a,x_a))

def transform4_reg(x_a):
    p_a = x_a[:,2:4]
    p_a = product(p_a,axis=1,keepdims=True)

    return hstack((p_a,x_a))

def transform5_reg(x_a):
    p_a = x_a[:,2:5]
    p_a = product(p_a,axis=1,keepdims=True)

    return hstack((p_a,x_a))


def interaction_study(x_a, y_v, name):
    b_v = lin_regression(x_a,y_v, name=name)
    yfit_v = fitted_func(x_a, b_v)
    print(f'{name}: Coefficients {b_v=}')
    print(f'{name}: r squared = ', r_square(y_v,yfit_v))




def lin_regression(x_a,y_v,name=''):
    if name:
        print(f'\n\n----------------------- {name} --------------------')
        model = sm.OLS(y_v, x_a)
        results = model.fit()
        print(results.summary())
    #Using stats models to get p value even though I did my own regression
    # y_v = X@B
    b_v = linalg.pinv(x_a)@y_v
    #print(b)
    return b_v

def fitted_func(x_a,b_v):
    yfit_v = x_a@b_v
    # with np.printoptions(precision=2):
    #     print(f'predicted mpg of cars {yfit_v=}')
    return yfit_v

def r_square(y_v,yfit_v):
    # This function is to find the r squared value
    # This will be calcualted by doing 1 - variance of (actual - predicited)/variance of actual
    rsq = 1 - (var(y_v-yfit_v))/(var(y_v))

    return rsq

def main ():
    x_a, y_v = data_loader()
    b_v = lin_regression(x_a,y_v,name='Main Regression')
    print('beta values', b_v)
    scatter_plot(x_a[:,1],x_a[:,2])
    cor_a = corrcoef(x_a[:,1:], rowvar=False)
    print(cor_a.shape)
    with np.printoptions(precision=4):
        print(cor_a)
    rand_var_study(x_a,y_v,'Control')
    rand_var_study(transform1_x1_only(x_a),y_v,'X1 Only')
    rand_var_study(transform1_x2_only(x_a),y_v,'X2 Only')

    x1_a = vstack((x_a,array([[1,0.1,0.8]])))
    y1_v = hstack((y_v,array([6])))
    rand_var_study(x_a,y_v,'New Control')
    rand_var_study(transform1_x1_only(x1_a),y1_v,'New X1 Only')
    rand_var_study(transform1_x2_only(x1_a),y1_v,'New X2 Only')

    raise SystemExit

    #print(x_a,y_v)
    #scatter_matrix(data_a, name_l)

    #print(x_a,y_v)
    yfit_v = fitted_func(x_a, b_v)
    i_v = abs(b_v).argsort()[::-1]
    print(f'Coefficients {b_v=}')
    print('Coefficients:', [(name, b) for name, b in zip(name_l, b_v)])
    print("pred_influence",[name_l[i] for i in i_v])
    print('r squared = ', r_square(y_v,yfit_v))

    #diag_plot(y_v, yfit_v)
    interaction_study(x_a, y_v, 'control')
    interaction_study(transform1_reg(x_a),y_v, 'Horse Power Squared')
    interaction_study(transform2_reg(x_a),y_v, 'Horse Power Square Root')
    interaction_study(transform3_reg(x_a),y_v, 'Horse Power Log')
    interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight')
    interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight * Acceleration')


    #print(yhat_v)
if __name__ == '__main__':
        main()


# zeros, full, empty, array, arange, indexing, transpose

#URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing_splines.html
#
```

dendogram.py

```python
from numpy import *
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy.interpolate import splrep, BSpline
from scipy.cluster.hierarchy import complete, fcluster
from scipy.cluster import hierarchy
from scipy.spatial.distance import pdist

#Chapter 12 question 9





def data_loader(fname):
    x_a = loadtxt(fname,skiprows=1, usecols=(1,2,3,4), delimiter=',')


    state_v = loadtxt(fname, skiprows=1, usecols=(0), delimiter=',', dtype=str)



    print(state_v)
    #print(data_a[0:,1].shape)
    # We want it in the form  of Y = XB
    # Where Y is the response variable
    # Where X is an array with size nx2 where n is the predictor variable and the other column is a 1
    # B is the coeeficents (slope and intercept) that we are trying to solve for    year_v = data_a[:,1]



    return x_a, state_v

def dendo_construct(x_a, state_v):
    n = state_v.shape[0]

    def label_func(id):
        if id < n:
            return state_v[id]
        return f'{id}'

    d_a = pdist(x_a)
    print(d_a.shape)
    print(d_a)
    z_a = complete(d_a)
    print(z_a.shape)
    print(z_a)
    plt.figure()
    dn = hierarchy.dendrogram(z_a, leaf_label_func=label_func)
    plt.show()

def z_score(x_a):
    m_v = x_a.mean(axis=0)
    std_v = x_a.std(axis=0)
    z_a = (x_a - m_v)/std_v
    return z_a

def scatter_matrix(data_a, name_l):
    n, p = data_a.shape
    fig, axs = plt.subplots(4, 7)
    ax_l = list(axs.flat)
    mpl.rcParams['figure.autolayout'] = True
    font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 10}

    #mpl.rc('font', **font)
    for i in range(p):
        for j in range(i+1,p):
            print(i, j, name_l[i], name_l[j])
            x_v = data_a[:,i]
            y_v = data_a[:,j]
            ax = ax_l.pop(0)
            ax.scatter(x_v, y_v, s=2**2)
            title = f'{name_l[i]} vs {name_l[j]}'
            ax.set_title(title[:25])
    plt.tight_layout()
    fig.savefig('ch12_9.pdf')
    plt.show()

def main ():
    x_a, state_v  = data_loader('USArrests.csv')
    dendo_construct(x_a, state_v)
    dendo_construct(z_score(x_a), state_v)
    raise SystemExit
    raise SystemExit

    #print(yhat_v)
if __name__ == '__main__':
        main()


# zeros, full, empty, array, arange, indexing, transpose

#URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing_splines.html
#https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendrogram.html#scipy.cluster.hierarchy.dendrogram
```

hw1graphs.py

```python
from numpy import *
import matplotlib.pyplot as plt
from matplotlib import patheffects
from statistics import linear_regression
from scipy.interpolate import CubicSpline


#THIS CODE IS FOR THE 5 THINGS GRAPH


fig, ax = plt.subplots(figsize=(15, 15))

#x x^2 constant
def powxy(x, y):
    return [x**2, x, 1, y]
def powxy2(x, y):
    return [x**-3, x**-2, x**-1, 1, y]
def startdata():
    pass
    data_a = array([
        powxy(0,4),
        #powxy(11,90),
        powxy(15,170),
        powxy(20,400)
        ])
   # yvalue_start = array([
    #    powxy[4]
   # ])
    X_a = data_a[:,:-1]
    Y_a = data_a[:,-1]
    print(X_a.shape)
    print(Y_a.shape)
    Xinv_a = linalg.pinv(X_a)
    coefs = Xinv_a @ Y_a
    print(coefs)

    x_v = linspace(0.5,20,400)
    y_v = coefs[0] * (x_v **2) + coefs[1] * (x_v ** 1) + coefs[2]

    #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
    print(y_v)
    fig[0, 0].plot(x_v, y_v, label='var')
    ax.legend()

    plt.show()



def startdata_cubic():

    data_a = array([
        (0,4),
        (5,20),
        (15,140),
        (20,400)
        ])
   # yvalue_start = array([
    #    powxy[4]
   # ])
    X_v = data_a[:,0]
    Y_v = data_a[:,1]
    spline = CubicSpline(X_v, Y_v)


    x1_v = linspace(0.5,20,400)
    y1_v = spline(x1_v)

    #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
    print(y1_v)
    ax.plot(x1_v, y1_v, label='variance')
    ax.legend()


def bias_cubic():

    data_a = array([
        (0,400),
        (5,140),
        (15,20),
        (20,4)
        ])
   # yvalue_start = array([
    #    powxy[4]
   # ])
    X_v = data_a[:,0]
    Y_v = data_a[:,1]
    spline = CubicSpline(X_v, Y_v)


    x1_v = linspace(0.5,20,400)
    y1_v = spline(x1_v)

    #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
    print(y1_v)
    ax.plot(x1_v, y1_v, label='squared bias')

    ax.legend()

def trainerror_cubic():

    data_a = array([
        (0,340),
        (5,210),
        (10,110),
        (15,60),
        (20,15)
        ])
   # yvalue_start = array([
    #    powxy[4]
   # ])
    X_v = data_a[:,0]
    Y_v = data_a[:,1]
    spline = CubicSpline(X_v, Y_v)


    x1_v = linspace(0.5,20,400)
    y1_v = spline(x1_v)

    #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
    print(y1_v)
    ax.plot(x1_v, y1_v, label='training error')

    ax.legend()

def testerror_cubic():

    data_a = array([
        (0,350),
        (5,250),
        (10,140),
        (15,210),
        (20,340)
        ])
   # yvalue_start = array([
    #    powxy[4]
   # ])
    X_v = data_a[:,0]
    Y_v = data_a[:,1]
    spline = CubicSpline(X_v, Y_v)


    x1_v = linspace(0.5,20,400)
    y1_v = spline(x1_v)

    #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
    print(y1_v)
    ax.plot(x1_v, y1_v, label='test error')

    ax.legend()

def irrerror():
    ax.axhline(y=200, xmin = 0.05, xmax = 0.95, label='irreducible error')
    ax.legend()
def bias():
    data_a = array([
        powxy2(2,25),
        powxy2(3,16),
        powxy2(4,9),
        powxy2(5,4)
        ])
   # yvalue_start = array([
    #    powxy[4]
```

```python
   # ])
    X_a = data_a[:,:-1]
    Y_a = data_a[:,-1]
    print(X_a.shape)
    print(Y_a.shape)
    Xinv_a = linalg.pinv(X_a)
    coefs = Xinv_a @ Y_a
    print(coefs)

    x_v = linspace(1,20,400)
    y_v = coefs[0] * (x_v **3) + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
    print(y_v)
    ax.plot(x_v, y_v, label='bias')
    ax.legend()

    plt.show()
def main():
    plt.xlabel("flexibility")
    plt.ylabel("value")
    startdata_cubic()
    bias_cubic()
    trainerror_cubic()
    testerror_cubic()
    irrerror()
    plt.show()
    #startdata()
    #bias()
if __name__ == '__main__':
        main()

#y_v = (coefs[0] * x_v) + ((coefs[1]) * 2) + coefs[3]

#https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.axhline.html
```