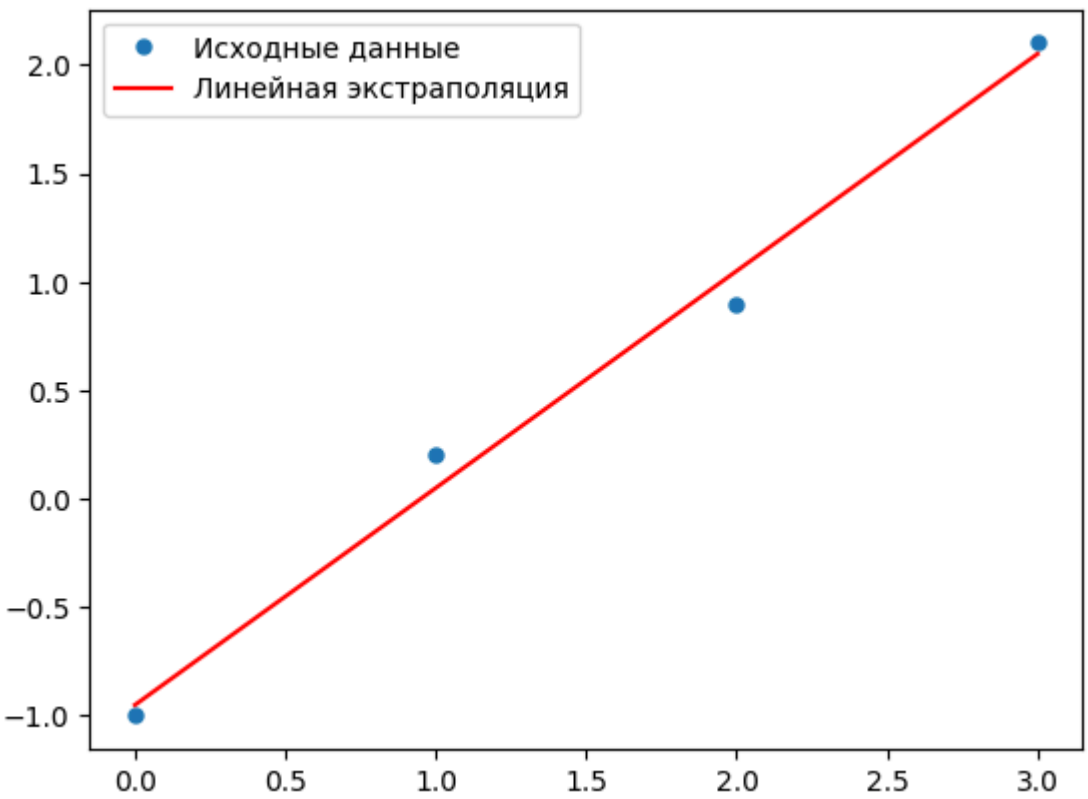
1.1.1 - Проведем y = kx + b через экспериментальные точки.

In [48]:
```
import numpy as np
x = np.array([0, 1, 2, 3])
y = np.array([-1, 0.2, 0.9, 2.1])

# Перепишем линейное уравнение y = kx + b как y = Ap, где A = [[x1]] и  p = [[k]
# Построим A по x:

A = np.vstack([x, np.ones(len(x))]).T
print(A)

# Используем метод lstsq для решения его относительно вектора p
k, b = np.linalg.lstsq(A, y, rcond = None)[0]
print(k, b)

# Построим график полученной прямой и укажем на нем точки
import matplotlib.pyplot as plt
plt.plot(x, y, 'o', label='Исходные данные', markersize = 5)
plt.plot(x, k*x + b, 'r', label='Линейная экстраполяция')
plt.legend()
plt.show()
```

```
[[0. 1.]
 [1. 1.]
 [2. 1.]
 [3. 1.]]
0.9999999999999999 -0.9499999999999997
```

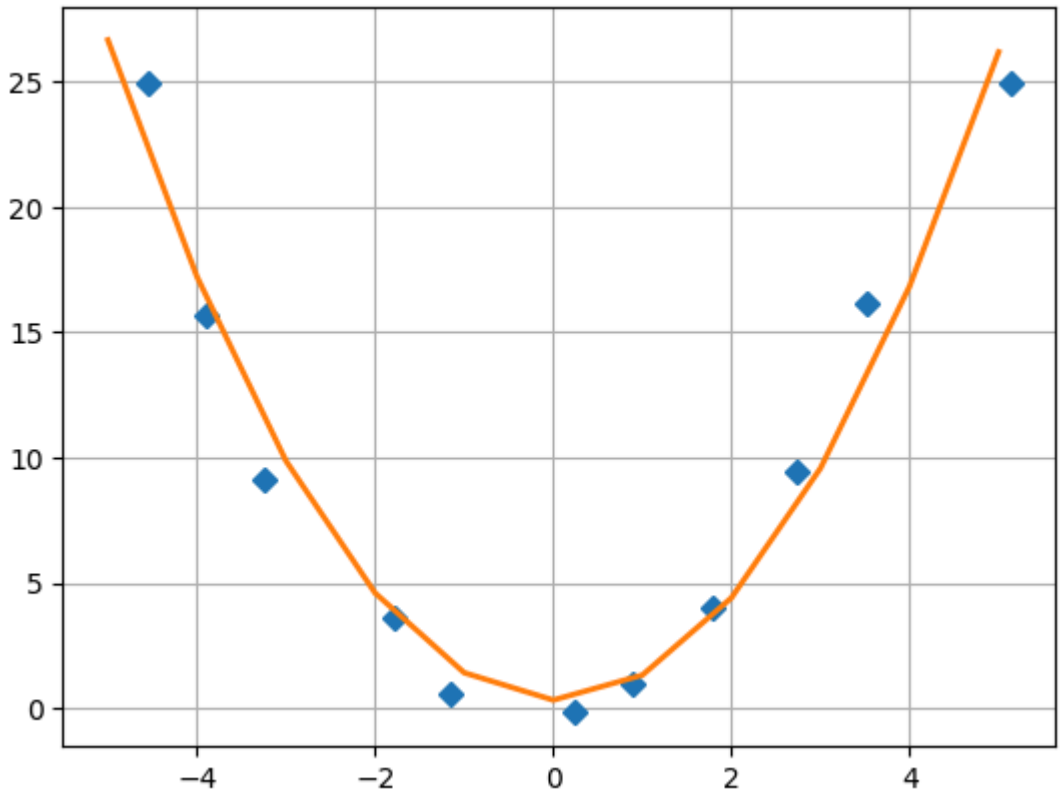


1.1.2 - Пусть x,y – вектора длиной n > 3 (точек > 3). Задача заключается в построении эстраполяционного полинома второго порядка (параболы). Таким образом, необходимо найти такие коэффициенты полинома a,b,c по методу

наименьших квадратов. Данные мтогут быть получены в результате измерений. Покажем пример генерации данных случайным образом и загрузки их из файла.

In [49]:
```
from numpy import *
from numpy.random import *
# генерируем случайные x, y
delta = 1.0
x = linspace(-5, 5, 11)
y = x**2 + delta*(rand(11)- 0.5)
x += delta*(rand(11) - 0.5)

#Записываем данные в файл
x.tofile('x_data.txt', '\n')
y.tofile('y_data.txt', '\n')

# читаем данные из файлов
x = fromfile('x_data.txt', float, sep='\n')
y = fromfile('y_data.txt', float, sep='\n')
print(x)
print(y)

# Нахождение коэффициентов функции вида y = ax^2 + bx + c методом наименьших ква
# Задаем вектор m = [x**2, x, E]
m = vstack((x**2, x, ones(11))).T
# Находим коэффициенты при составляющих вектора m
s = np.linalg.lstsq(m, y, rcond = None)[0]

# На отрезке [-5,5]
x_prec = linspace(-5, 5, 11)
# рисуем точки
plt.plot(x, y, 'D')
# рисуем кривую вида y = ax^2 + bx + c, подставляя из решения коэффициенты s[0],
plt.plot(x_prec, s[0] * x_prec**2 + s[1] * x_prec + s[2], '-', lw=2)
plt.grid()
plt.savefig('парабола.png')
```

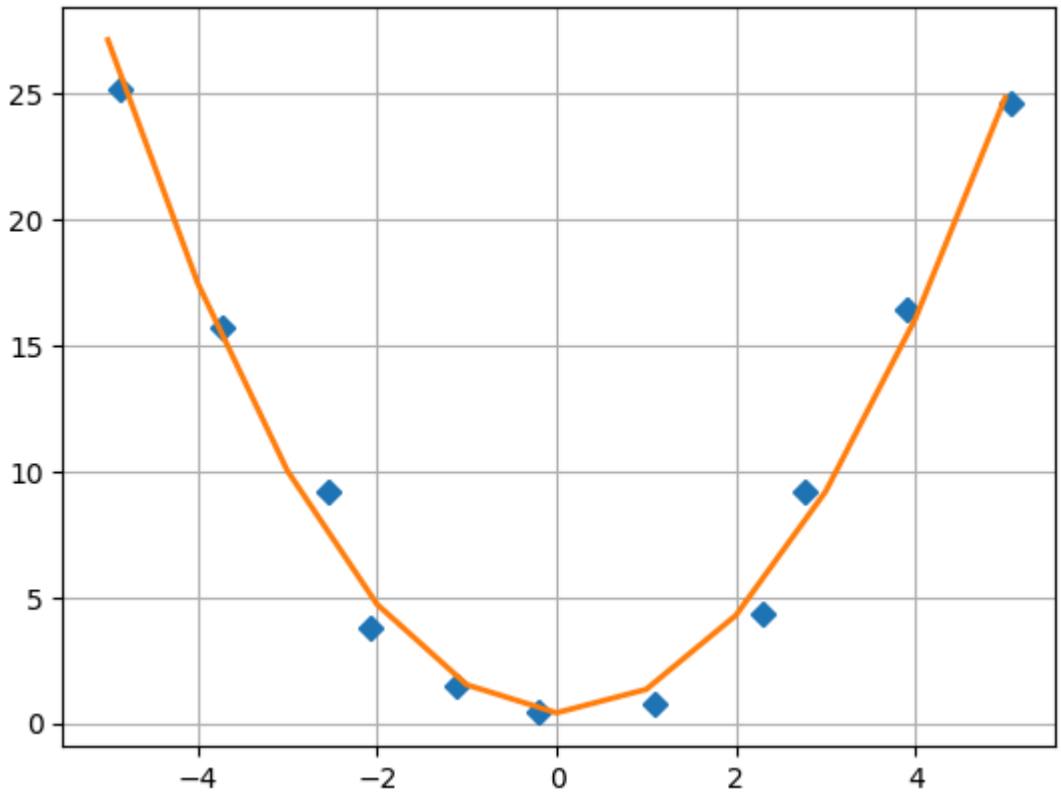
```
[-4.53916768 -3.90688716 -3.24114104 -1.78823749 -1.14666794  0.24026383
  0.8963978   1.79730338  2.74103227  3.52917754  5.13341142]
[24.90301926 15.71100943  9.1565589   3.60927742  0.60629349 -0.18938983
  0.9945613   3.982578    9.46942743 16.14168995 24.90556664]
```



1.1.3 - По данным предыдущего примера постройте эстраполяционного полинома третьего порядка

In [50]:
```
delta = 1.0
x = linspace(-5, 5, 11)
y = x**2 + delta*(rand(11)- 0.5)
x += delta*(rand(11) - 0.5)

x.tofile('x_data.txt', '\n')
y.tofile('y_data.txt', '\n')

x = fromfile('x_data.txt', float, sep='\n')
y = fromfile('y_data.txt', float, sep='\n')
print(x)
print(y)

m = vstack((x**3 ,x**2, x, ones(11))).T
s = np.linalg.lstsq(m, y, rcond = None)[0]

x_prec = linspace(-5, 5, 11)
plt.plot(x, y, 'D')
plt.plot(x_prec, s[0] * x_prec**3 + s[1] * x_prec**2 + s[2] * x_prec + s[3], '-'
plt.grid()
plt.savefig('полином 3-ей степени.png')
```

```
[-4.87063808 -3.72390859 -2.54563735 -2.08593273 -1.10807169 -0.20703298
  1.09835035  2.29823849  2.75710882  3.89266957  5.05944894]
[25.17668107 15.70894612  9.18790811  3.79617074  1.44560573  0.47123205
  0.79885766  4.35346139  9.21234694 16.40632147 24.60760131]
```

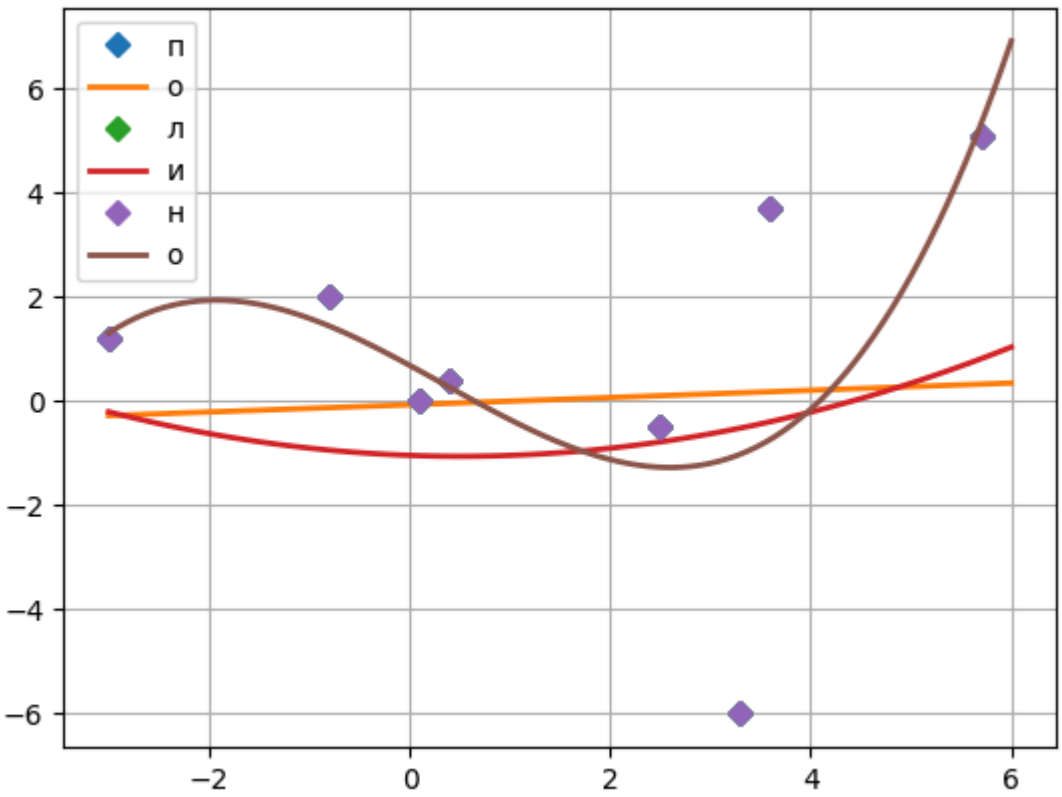


Задание - Представьте собственные данные и постройте экстраполяцию полиномами первой, второй и третьей степени

In [72]:
```
x = array([-3, -0.8, 0.1, 0.4, 2.5, 3.3, 3.6, 5.7])
y = array([1.2, 2, 0, 0.4, -0.5, -6, 3.7, 5.1])

x.tofile('x_data.txt', '\n')
y.tofile('y_data.txt', '\n')

x = fromfile('x_data.txt', float, sep='\n')
y = fromfile('y_data.txt', float, sep='\n')
print(x)
print(y)

m = vstack((x**3 ,x**2, x, ones(8))).T
s = np.linalg.lstsq(m, y, rcond = None)[0]

x_prec = linspace(-3, 6, 100)
plt.plot(x, y, 'D')
plt.plot(x_prec, s[0] * x_prec + s[1], '-', lw=2)
plt.legend('линейная экстраполяция')
plt.grid()
plt.savefig('линейная экстраполяция.png')

plt.plot(x, y, 'D')
plt.plot(x_prec, s[0] * x_prec**2 + s[1] * x_prec + s[2], '-', lw=2)
plt.legend('параболическая экстраполяция')
plt.grid()
plt.savefig('параболическая экстраполяция.png')

plt.plot(x, y, 'D')
plt.plot(x_prec, s[0] * x_prec**3 + s[1] * x_prec**2 + s[2] * x_prec + s[3], '-'
plt.legend('полином 3-ей степени')
```

```
plt.grid()
plt.savefig('полином 3-ей степени.png')
```

```
[-3.  -0.8  0.1  0.4  2.5  3.3  3.6  5.7]
[ 1.2  2.   0.   0.4 -0.5 -6.   3.7  5.1]
```

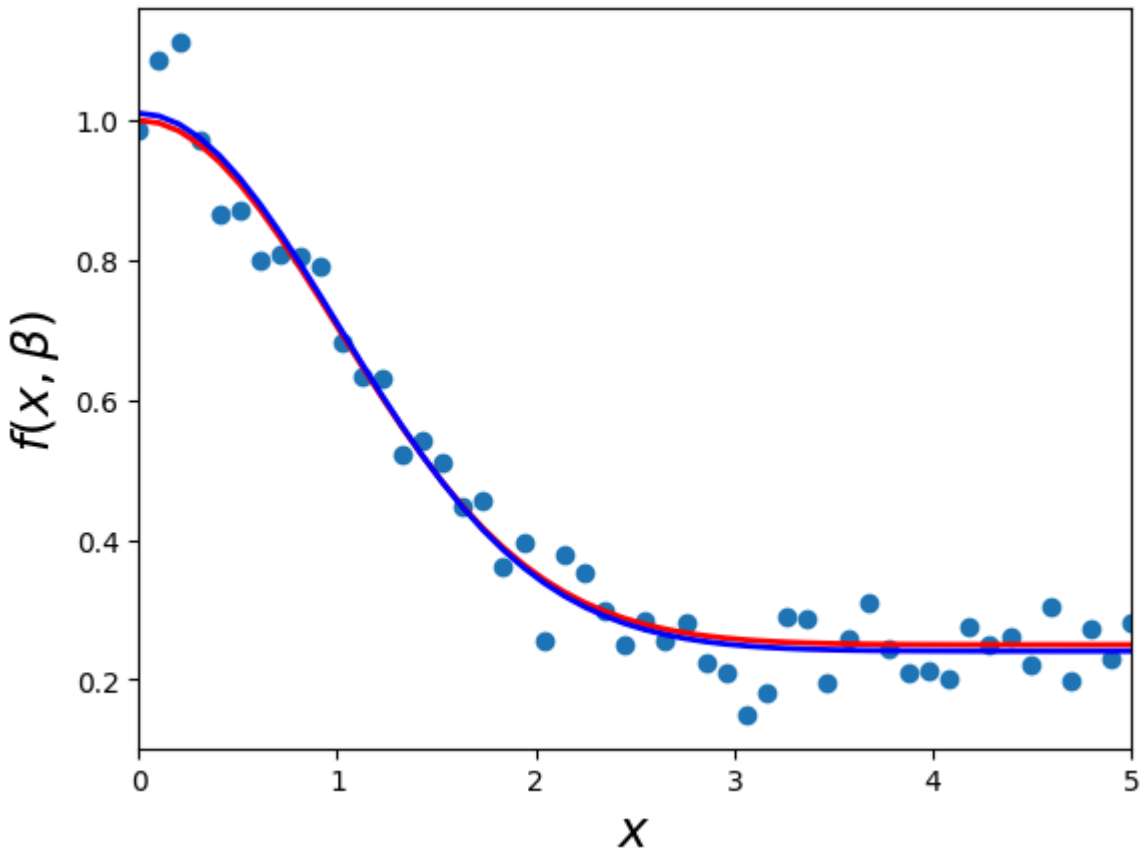


1.1.4 - Необходимо проверить гипотезу, что наши точечно заданная функция ложится на кривую вида f(x,b) = b_0 + b_1 exp(-b2x^2)

In [52]:
```
# Добавим шума в данные, сделанные по функции f(x, b) с коэффициентами b = (0.25
beta = (0.25, 0.75, 0.5)
def f(x, b0, b1, b2):
    return b0 + b1 * np.exp(-b2 * x**2)
# Зададим массив точек xi
xdata = np.linspace(0, 5, 50)
# Создаем теоретически правильные значения точек yi (без шума)
y = f(xdata, *beta)
# зашумляем эти данные
ydata = y + 0.05 * np.random.randn(len(xdata))

# Используем функцию для получения решения в виде коэффициентов функции f(x) для
from scipy.optimize import curve_fit
beta_opt, beta_cov = curve_fit(f, xdata, ydata)
print(beta_opt)

# Вычислим линейное отклонение
lin_dev = sum(beta_cov[0])
print(lin_dev)

# Вычислим квадратичное отклонение
residuals = ydata - f(xdata, *beta_opt)
fres = sum(residuals**2)
print(fres)

fig, ax = plt.subplots()
```

```
ax.scatter(xdata, ydata)
ax.plot(xdata, y, 'r', lw=2)
ax.plot(xdata, f(xdata, *beta_opt), 'b', lw=2)
ax.set_xlim(0, 5)
ax.set_xlabel(r"$x$", fontsize=18)
ax.set_ylabel(r"$f(x, \beta)$", fontsize=18)
plt.show()
```

```
[0.24093083 0.76918591 0.4955941 ]
0.0001944716748439533
0.10308325868635024
```



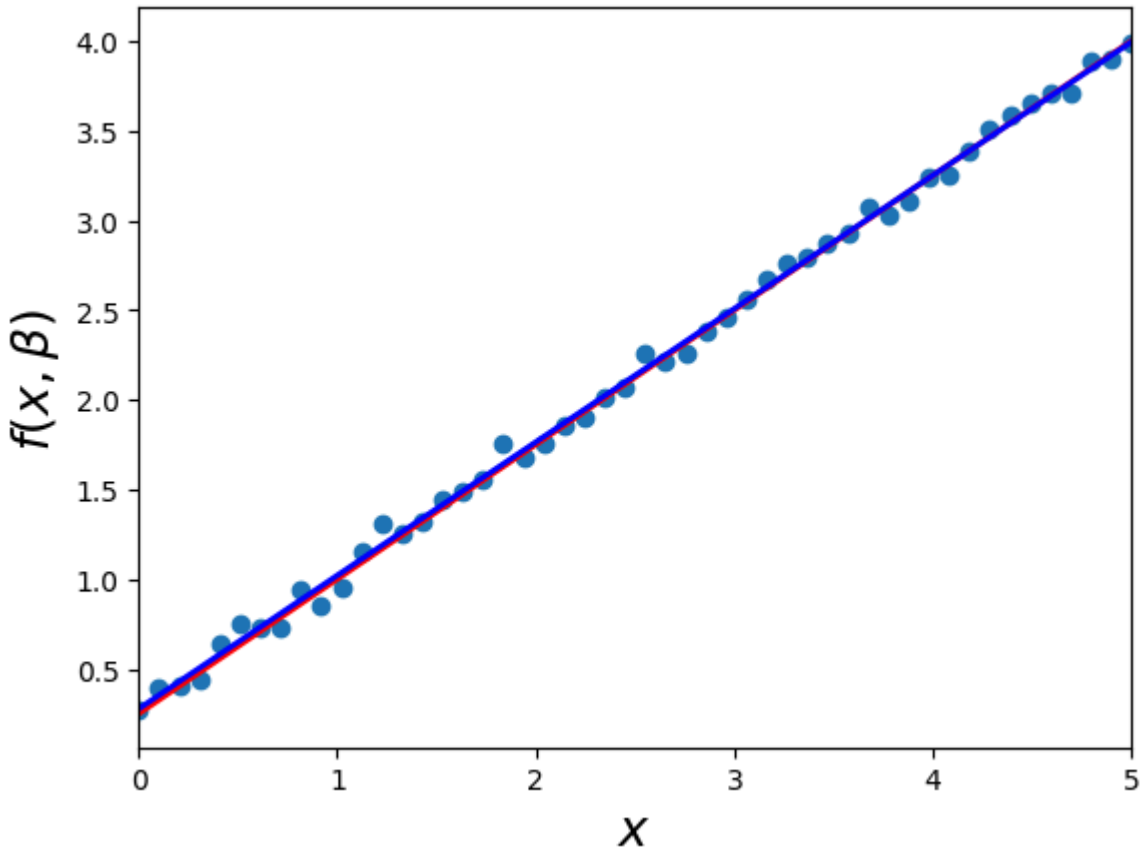
1.1.5 - Необходимо проверить гипотезу, что наши точечно заданная функция ложится на кривые вида:

1. f(x,b) = b_0 + b_1 x
2. f(x,b) = b_0 + b_1 x + b_2 x^2
3. f(x,b) = b_0 + b_1 ln(x)
4. f(x,b) = b_0 x^(b_1)

In [53]:
```
#
#
#
beta = (0.25, 0.75)
def f(x, b0, b1):
    return b0 + b1 * x
#
xdata = np.linspace(0, 5, 50)
#
y = f(xdata, *beta)
#
ydata = y + 0.05 * np.random.randn(len(xdata))
```

```
beta_opt, beta_cov = curve_fit(f, xdata, ydata)
print(beta_opt)
print(lin_dev)

# Вычислим квадратичное отклонение
residuals = ydata - f(xdata, *beta_opt)
fres = sum(residuals**2)
print(fres)

fig, ax = plt.subplots()
ax.scatter(xdata, ydata)
ax.plot(xdata, y, 'r', lw=2)
ax.plot(xdata, f(xdata, *beta_opt), 'b', lw=2)
ax.set_xlim(0, 5)
ax.set_xlabel(r"$x$", fontsize = 18)
ax.set_ylabel(r"$f(x, \beta)$", fontsize = 18)
plt.show()
```

```
[0.2768906  0.74355186]
0.0001944716748439533
0.12800181027254826
```

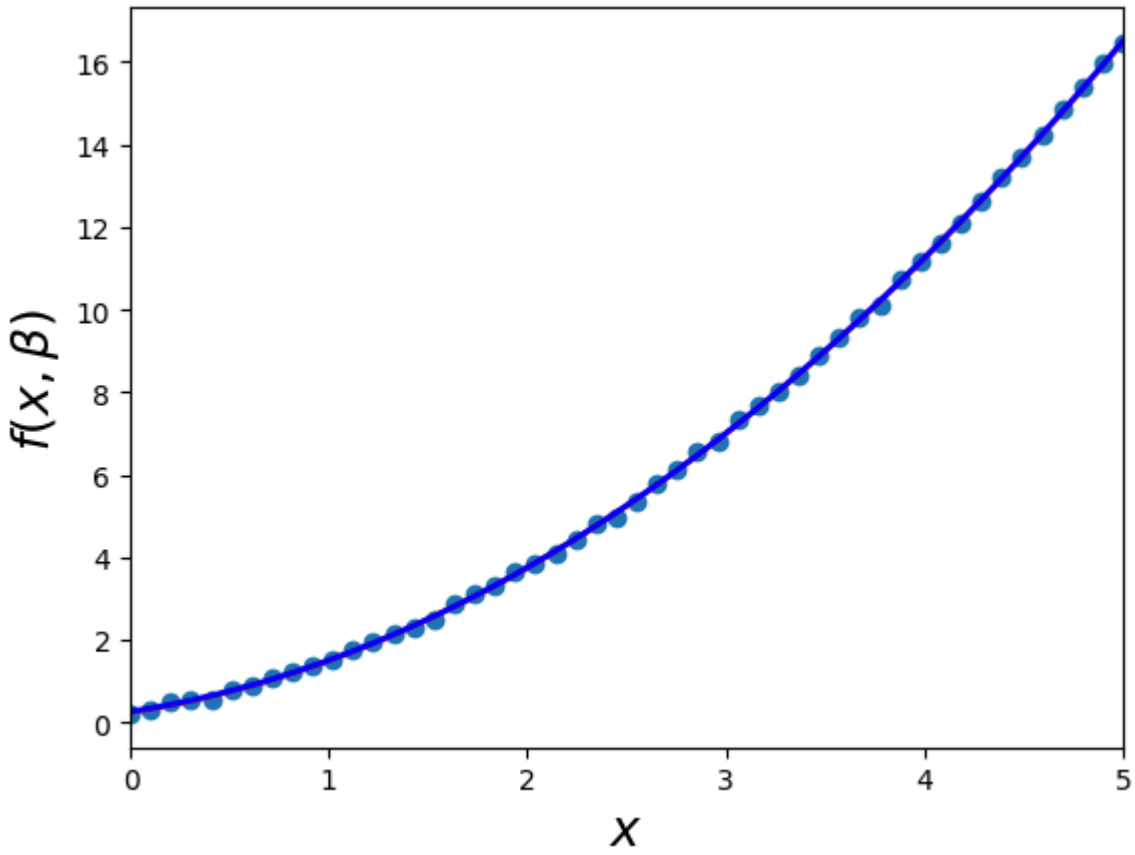


In [54]:

```
#
#
#
beta = (0.25, 0.75, 0.5)
def f(x, b0, b1, b2):
    return b0 + b1 * x + b2 * x**2
#
xdata = np.linspace(0, 5, 50)
#
y = f(xdata, *beta)
#
ydata = y + 0.05 * np.random.randn(len(xdata))
```

```
beta_opt, beta_cov = curve_fit(f, xdata, ydata)
print(beta_opt)
#
lin_dev = sum(beta_cov[0])
print(lin_dev)

#
residuals = ydata - f(xdata, *beta_opt)
fres = sum(residuals**2)
print(fres)

fig, ax = plt.subplots()
ax.scatter(xdata, ydata)
ax.plot(xdata, y, 'r', lw=2)
ax.plot(xdata, f(xdata, *beta_opt), 'b', lw=2)
ax.set_xlim(0, 5)
ax.set_xlabel(r"$x$", fontsize=18)
ax.set_ylabel(r"$f(x, \beta)$", fontsize=18)
plt.show()
```

```
[0.26250117 0.74332269 0.50124073]
0.0001574117040334852
0.1312648850255865
```

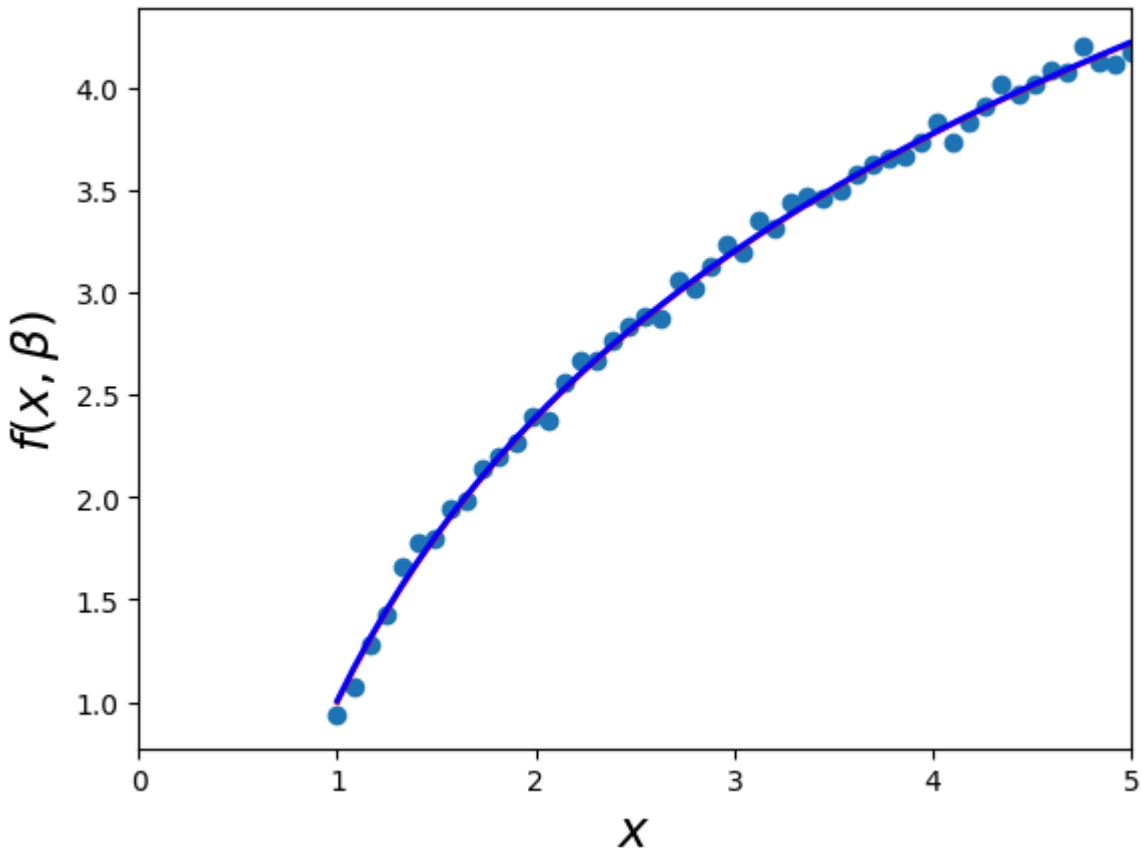


In [55]:

```
#
#
#
beta = (1, 2)
def f(x, b0, b1):
    return b0 + b1 * np.log(x)
#
xdata = np.linspace(1, 5, 50)
#
y = f(xdata, *beta)
```

```
#
ydata = y + 0.05 * np.random.randn(len(xdata))
beta_opt, beta_cov = curve_fit(f, xdata, ydata)
print(beta_opt)
#
lin_dev = sum(beta_cov[0])
print(lin_dev)

#
residuals = ydata - f(xdata, *beta_opt)
fres = sum(residuals**2)
print(fres)

fig, ax = plt.subplots()
ax.scatter(xdata, ydata)
ax.plot(xdata, y, 'r', lw=2)
ax.plot(xdata, f(xdata, *beta_opt), 'b', lw=2)
ax.set_xlim(0, 5)
ax.set_xlabel(r"$x$", fontsize=18)
ax.set_ylabel(r"$f(x, \beta)$", fontsize=18)
plt.show()
```

```
[1.00666998 1.99787972]
4.7166263521045604e-05
0.1090692207792277
```

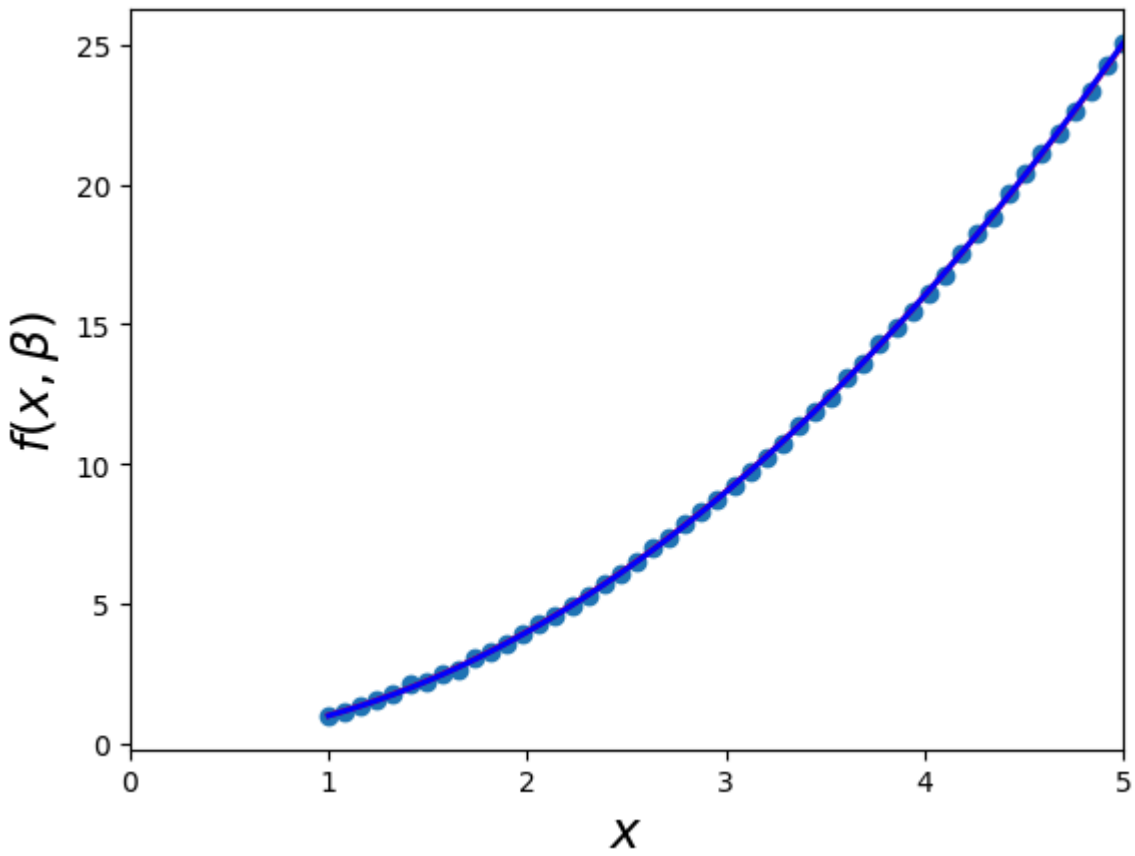


In [56]:

```
#
#
#
beta = (1, 2)
def f(x, b0, b1):
    return b0 * x**b1
#
xdata = np.linspace(1, 5, 50)
```

```
#
y = f(xdata, *beta)
#
ydata = y + 0.05 * np.random.randn(len(xdata))
beta_opt, beta_cov = curve_fit(f, xdata, ydata)
print(beta_opt)
lin_dev = sum(beta_cov[0])
print(lin_dev)

#
residuals = ydata - f(xdata, *beta_opt)
fres = sum(residuals**2)
print(fres)

fig, ax = plt.subplots()
ax.scatter(xdata, ydata)
ax.plot(xdata, y, 'r', lw=2)
ax.plot(xdata, f(xdata, *beta_opt), 'b', lw=2)
ax.set_xlim(0, 5)
ax.set_xlabel(r"$x$", fontsize=18)
ax.set_ylabel(r"$f(x, \beta)$", fontsize=18)
plt.show()
```

```
[0.99380372 2.00454366]
5.0569284181965635e-06
0.12308202267285284
```

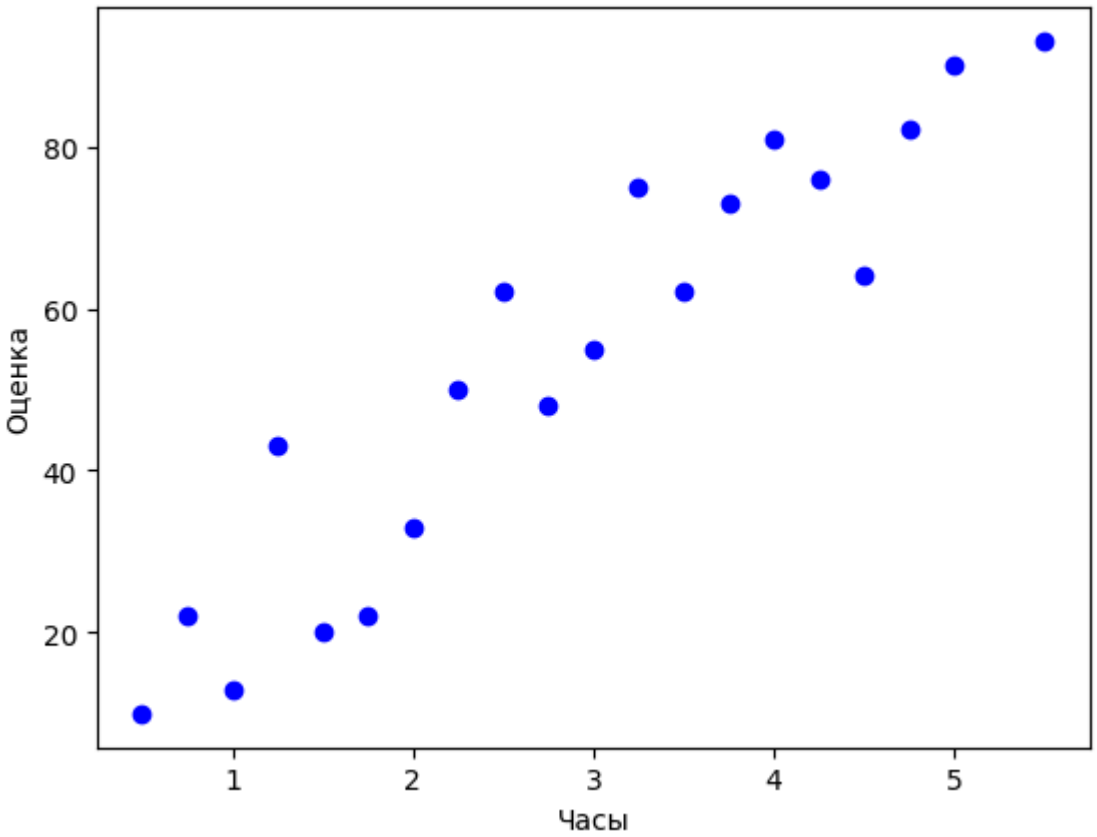


1.2.1 - Построим простую линейную регрессию в Python с использованием библиотеки scikit-learn

In [58]:
```
#
#
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from pandas import DataFrame,Series
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

#
my_dict = {'Учебное время': [0.50,0.75,1.00,1.25,1.50,1.75,2.00,2.25,2.50,2.75,3
           'Оценка': [10, 22, 13, 43, 20, 22, 33, 50, 62, 48, 55, 75, 62, 73, 81
dataset = pd.DataFrame(my_dict)
print(dataset.head())

#
print(dataset.shape)
print(dataset.describe())

#
plt.scatter (dataset['Учебное время'], dataset['Оценка'], color = 'b', label = "
plt.xlabel("Часы")
plt.ylabel("Оценка")
plt.show()
```

```
   Учебное время  Оценка
0           0.50      10
1           0.75      22
2           1.00      13
3           1.25      43
4           1.50      20
(20, 2)
       Учебное время     Оценка
count      20.000000  20.000000
mean        2.887500  53.700000
std         1.501041  26.435821
min         0.500000  10.000000
25%         1.687500  30.250000
50%         2.875000  58.500000
75%         4.062500  75.250000
max         5.500000  93.000000
```



После того как мы получили представление о данных, разделим информацию на «атрибуты» и «метки». Атрибуты – это независимые переменные, а метки – это зависимые переменные, значения которых должны быть предсказаны. В нашем наборе всего два столбца и необходимо предсказать оценку в зависимости от количества часов. Чтобы извлечь атрибуты и метки, выполните следующий скрипт:

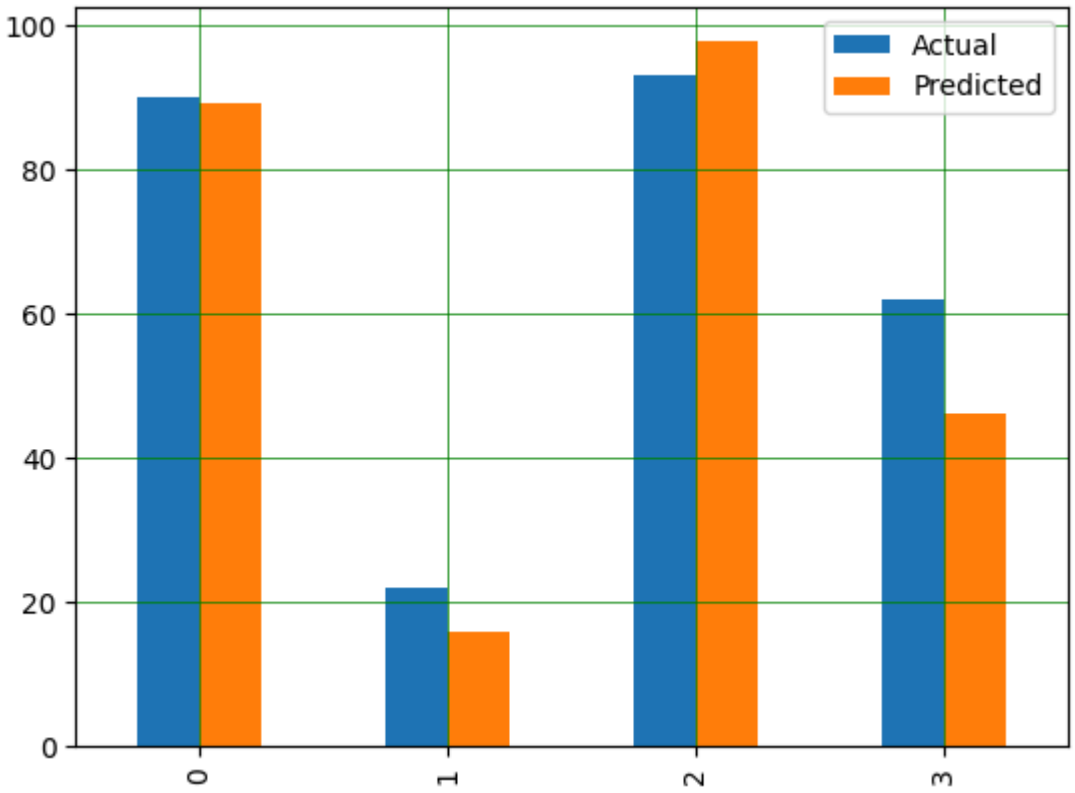
In [64]:
```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
print(X)
print(y)

#
#
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rando

#
#
regressor = LinearRegression()
regressor.fit(X_train, y_train)

#
print(regressor.intercept_)
print(regressor.coef_)
```

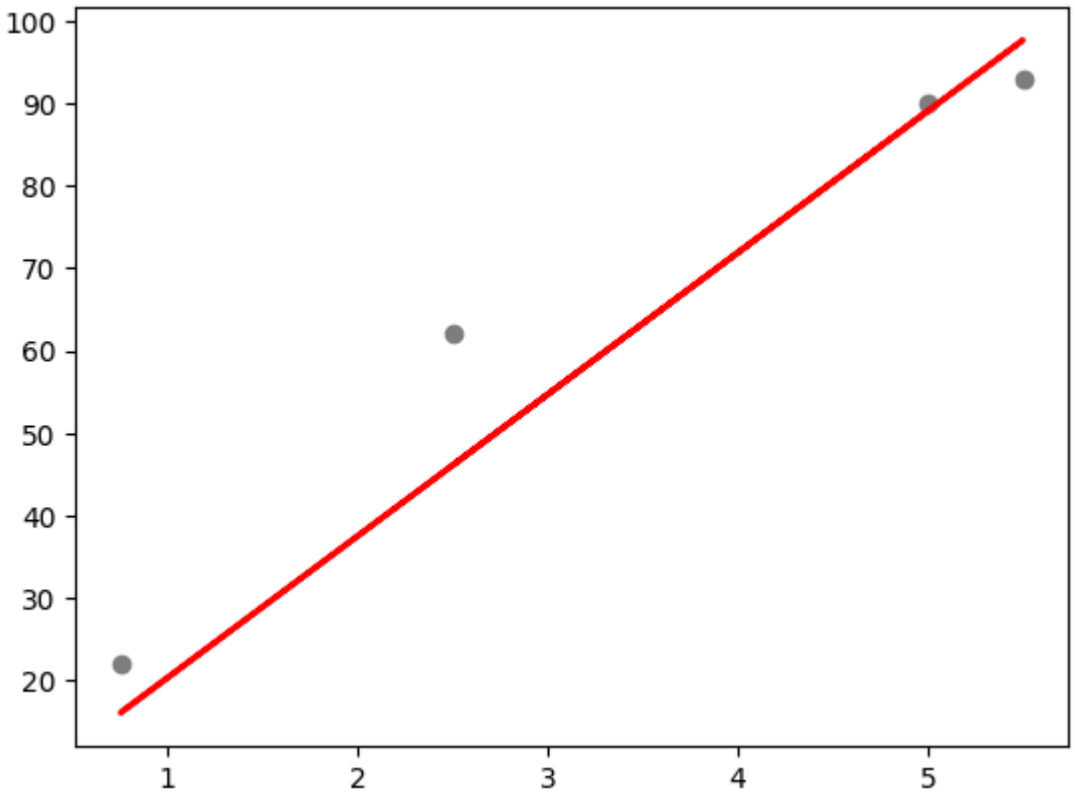
```
[[0.5 ]
 [0.75]
 [1.  ]
 [1.25]
 [1.5 ]
 [1.75]
 [2.  ]
 [2.25]
 [2.5 ]
 [2.75]
 [3.  ]
 [3.25]
 [3.5 ]
 [3.75]
 [4.  ]
 [4.25]
 [4.5 ]
 [4.75]
 [5.  ]
 [5.5 ]]
[10 22 13 43 20 22 33 50 62 48 55 75 62 73 81 76 64 82 90 93]
3.168632075471699
[17.18867925]
```

Получившийся результат можно интерпретировать следующим образом: с каждым затраченным часом на обучение результат экзамена повышается приблизительно на 17 баллов. Далее можно построить прогнозы. Для этого мы будем использовать наши тестовые данные и посмотрим, насколько точно наш алгоритм предсказывает процентную оценку. Чтобы сделать прогноз на тестовых данных необходимо выполнить следующий код:

In [67]:
```
y_pred = regressor.predict(X_test)
#
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(df)

#
df.plot(kind='bar')
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()

#
plt.scatter(X_test, y_test, color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```

```
   Actual  Predicted
0      90  89.112028
1      22  16.060142
2      93  97.706368
3      62  46.140330
```





Задание - Постройте модель линейной регрессии для произвольных данных из двух столбцов. Для примера можно взять точечную зависимость заработной платы от опыта работы: (https://raw.githubusercontent.com/AnnaShestova/salary-years-simple-linear-regression/master/Salary_Data.csv). Найдите коэффициенты линии регрессии. Постройте прогноз.

In [ ]:

1.3.1 - Для решения задачи множественной регрессии можно задействовать уже известный метод numpy.linalg.lstsq.

In [68]:
```
y = [1,2,3,4,3,4,5,3,5,5,4,5,4,5,4,5,6,0,6,3,1,3,1]
X = [[0,2,4,1,5,4,5,9,9,9,3,7,8,8,6,6,5,5,5,6,6,5,5],
     [4,1,2,5,6,7,8,9,7,8,7,8,7,8,7,8,6,8,9,2,1,5,6],
     [4,1,2,5,6,7,8,9,7,8,7,8,7,4,3,1,2,3,4,1,3,9,7]]
X = np.transpose(X) #
X = np.c_[X, np.ones(X.shape[0])] #
linreg = np.linalg.lstsq(X, y, rcond=None)[0]
print(linreg)
```

```
[ 0.11907344  0.34594584 -0.09972351  1.34067054]
```

Для данных из предыдущей задачи построить модель множественной линейной регрессии с использованием средств библиотеки scikit-learn.

In [71]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

y = [1,2,3,4,3,4,5,3,5,5,4,5,4,5,4,5,6,0,6,3,1,3,1]
X = [[0,2,4,1,5,4,5,9,9,9,3,7,8,8,6,6,5,5,5,6,6,5,5],
     [4,1,2,5,6,7,8,9,7,8,7,8,7,8,7,8,6,8,9,2,1,5,6],
     [4,1,2,5,6,7,8,9,7,8,7,8,7,4,3,1,2,3,4,1,3,9,7]]

#
new_y = np.array(y)
new_y = new_y.transpose()
df1 = pd.DataFrame(new_y)
new_X = np.array(X)
new_X = new_X.transpose()
df2 = pd.DataFrame(new_X)
df1 = df1.rename(columns = {0: 'y'}, inplace=False)
df2 = df2.rename(columns = {0: 'x1', 1: 'x2', 2: 'x3'}, inplace = False)

frames = [df1, df2]
dataset = pd.concat([df1, df2], axis=1, join="inner")
print(dataset.head())

#
print(dataset.shape)
print(dataset.describe())

#
X = dataset[['x1', 'x2', 'x3']]
y = dataset['y']

#
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

#
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```python
#
coeff_df = pd.DataFrame(regressor.coef_, X.columns, columns=['Coefficient'])
print(coeff_df)

#
y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(df)

#
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
```

```
   y  x1  x2  x3
0  1   0   4   4
1  2   2   1   1
2  3   4   2   2
3  4   1   5   5
4  3   5   6   6
(23, 4)
                y         x1         x2         x3
count  23.000000  23.000000  23.000000  23.000000
mean    3.565217   5.347826   6.043478   5.043478
std     1.674029   2.404706   2.476770   2.704849
min     0.000000   0.000000   1.000000   1.000000
25%     3.000000   4.500000   5.000000   3.000000
50%     4.000000   5.000000   7.000000   5.000000
75%     5.000000   6.500000   8.000000   7.000000
max     6.000000   9.000000   9.000000   9.000000
    Coefficient
x1     0.191050
x2     0.257673
x3    -0.142772
    Actual  Predicted
11       5   3.941882
10       4   3.062784
21       3   2.643993
14       4   4.207022
20       1   2.660987
Mean Squared Error: 0.985292610647285
```