import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

    # number of observations/points

    n = np.size(x)

    # mean of x and y vector

    m\_x = np.mean(x)

    m\_y = np.mean(y)

    # calculating cross-deviation and deviation about x

    SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

    SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

    # calculating regression coefficients

    b\_1 = SS\_xy / SS\_xx

    b\_0 = m\_y - b\_1\*m\_x

    return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

    # plotting the actual points as scatter plot

    plt.scatter(x, y, color = "m",

            marker = "o", s = 30)

    # predicted response vector

    y\_pred = b[0] + b[1]\*x

    # plotting the regression line

    plt.plot(x, y\_pred, color = "g")

    # putting labels

    plt.xlabel('x')

    plt.ylabel('y')

    # function to show plot

    plt.show()

def main():

    # observations / data

    dataset = pd.read\_csv("Salary\_Data.csv")

    years = dataset.iloc[:,0:1].values

    x = years

    salary= data = dataset.iloc[:,1:2].values

    y = salary

    # estimating coefficients

    b = estimate\_coef(x, y)

    print("Estimated coefficients:\nb\_0 = {} \

        \nb\_1 = {}".format(b[0], b[1]))

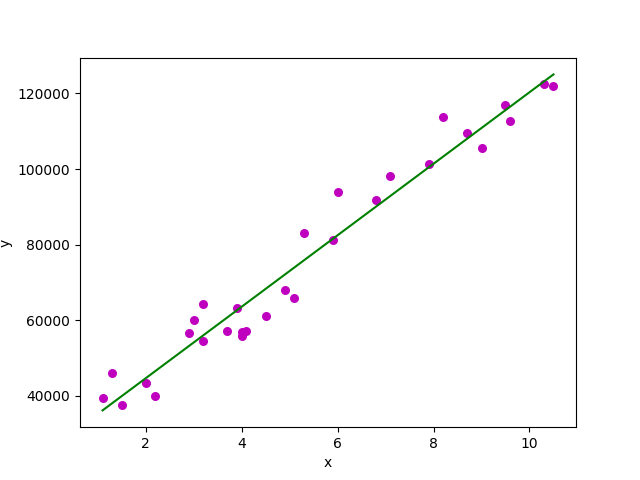
    # plotting regression line

    plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output



import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear\_model, metrics

# load the boston dataset

boston = datasets.load\_boston(return\_X\_y=False)

# defining feature matrix(X) and response vector(y)

X = boston.data

y = boston.target

# splitting X and y into training and testing sets

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4,

                                                    random\_state=1)

# create linear regression object

reg = linear\_model.LinearRegression()

# train the model using the training sets

reg.fit(X\_train, y\_train)

# regression coefficients

print('Coefficients: ', reg.coef\_)

# variance score: 1 means perfect prediction

print('Variance score: {}'.format(reg.score(X\_test, y\_test)))

# plot for residual error

## setting plot style

plt.style.use('fivethirtyeight')

## plotting residual errors in training data

plt.scatter(reg.predict(X\_train), reg.predict(X\_train) - y\_train,

            color = "green", s = 10, label = 'Train data')

## plotting residual errors in test data

plt.scatter(reg.predict(X\_test), reg.predict(X\_test) - y\_test,

            color = "blue", s = 10, label = 'Test data')

## plotting line for zero residual error

plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)

## plotting legend

plt.legend(loc = 'upper right')

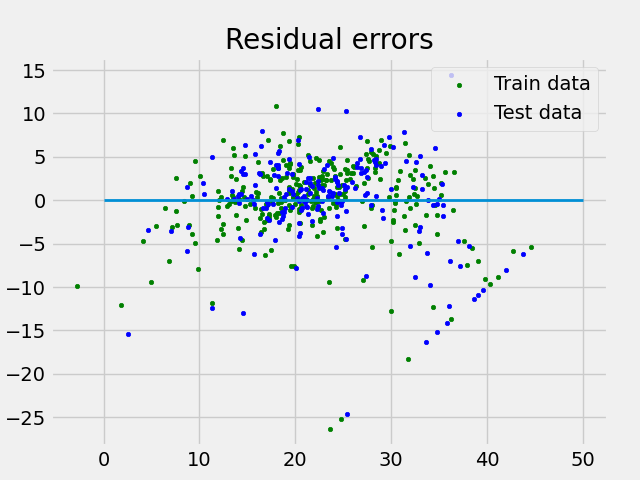
## plot title

plt.title("Residual errors")

## method call for showing the plot

plt.show()

output



import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

dataset = pd.read\_csv("Social\_Network\_Ads")

# input

x = dataset.iloc[:, [2, 3]].values

# output

y = dataset.iloc[:, 4].values

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc\_x = StandardScaler()

xtrain = sc\_x.fit\_transform(xtrain)

xtest = sc\_x.transform(xtest)

print (xtrain[0:10, :])

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

classifier.fit(xtrain, ytrain)

y\_pred = classifier.predict(xtest)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(ytest, y\_pred)

print ("Confusion Matrix : \n", cm)

from sklearn.metrics import accuracy\_score

print ("Accuracy : ", accuracy\_score(ytest, y\_pred))

from matplotlib.colors import ListedColormap

X\_set, y\_set = xtest, ytest

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1,

                            stop = X\_set[:, 0].max() + 1, step = 0.01),

                    np.arange(start = X\_set[:, 1].min() - 1,

                            stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(

            np.array([X1.ravel(), X2.ravel()]).T).reshape(

            X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Classifier (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

