Lab Course: Distributed Data Analytics

Exercise Sheet 0

Syed Wasif Murtaza Jafri-311226

Exercise 1: Pandas and Numpy

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (10,6)
np.random.seed(0)
import pandas as pd
```

Matrix Multipication

In [2]:
```python
def multiplyMatrix(X, Y):

    rowsX,colX = X.shape
    rowsY,colY = Y.shape

    # checking the dimensions of matrices
    if(colX!=rowsY):
        return 'Dimensions not matched'

    else:
        output =np.zeros((rowsX,colY))

        for i in range(rowsX): # for selecting rows of first matrix

            for j in range(colY): # for selecting columns of second matrix

                for k in range(colX): # for multiplying row to column

                    output[i][j] += X[i][k]*X[j][k]

    return output
```

In [3]:
```python
A = np.random.normal(2,0.01,size=(100,20)) # random vector with normal distribution
v = np.random.normal(2,0.01,size=(20,1)) # random vector with normal distribution
c = multiplyMatrix(A,v)
print(' Output vector c:',c)
print(' Dimensions of vector c:',c.shape)
```

```
 Output vector c: [[80.45756158]
 [80.25027396]
 [80.06957175]
 [80.08615117]
 [80.39618474]
 [80.42657748]
 [80.25591433]
 [80.21378879]
 [80.23160787]
 [80.17520718]
 [80.2755339 ]
 [80.12540249]
 [80.11896468]
 [80.13960591]
 [80.35990258]
 [80.14571153]
 [80.13624898]
 [80.10360547]
 [80.2340728 ]
 [80.12280755]
 [80.28468015]
 [80.18222662]
 [80.19330831]
 [80.17976747]
 [80.27409155]
 [80.21300872]
 [80.09398319]
 [80.14033667]
 [80.22892591]
 [80.05546609]
 [80.09107251]
 [80.24192986]
 [80.19698176]
 [80.1051417 ]
 [80.2325691 ]
 [80.26381545]
 [80.10094616]
 [80.20358419]
 [80.15367202]
 [80.16883944]
 [80.32060608]
 [80.29652524]
 [80.09240923]
 [80.15798792]
 [80.21580834]
 [80.15410049]
 [80.30043781]
 [80.41653672]
 [80.34874881]
 [80.24700486]
 [80.22884963]
 [80.23402913]
 [80.35155015]
 [80.15685334]
 [80.16285662]
 [80.37583985]
 [80.26405229]
 [80.2290419 ]
 [80.40033819]
 [80.25995209]
 [80.23180299]
 [80.23893369]
 [80.21708772]
 [80.23702074]
 [80.21094788]
 [80.06394378]
 [80.21611208]
 [80.27524569]
 [80.28269688]
 [80.30682171]
 [80.13520323]
 [80.19487135]
 [80.20261423]
 [80.28185268]
 [80.22969837]
 [80.19235776]
 [80.24428831]
 [80.30717549]
 [80.21574763]
 [80.29432315]
 [80.20980874]
 [80.176725  ]
 [80.29940062]
 [80.21579943]
 [80.36696105]
 [80.12371939]
 [80.33261055]
 [80.14837588]
 [80.2968387 ]
 [80.10300352]
 [80.16859421]
 [80.10865061]
 [80.24894437]
 [80.12997645]
 [80.28617415]
 [80.1985361 ]
 [80.31613118]
 [80.22647659]
 [80.3089705 ]
 [80.15270138]]
 Dimensions of vector c: (100, 1)
```

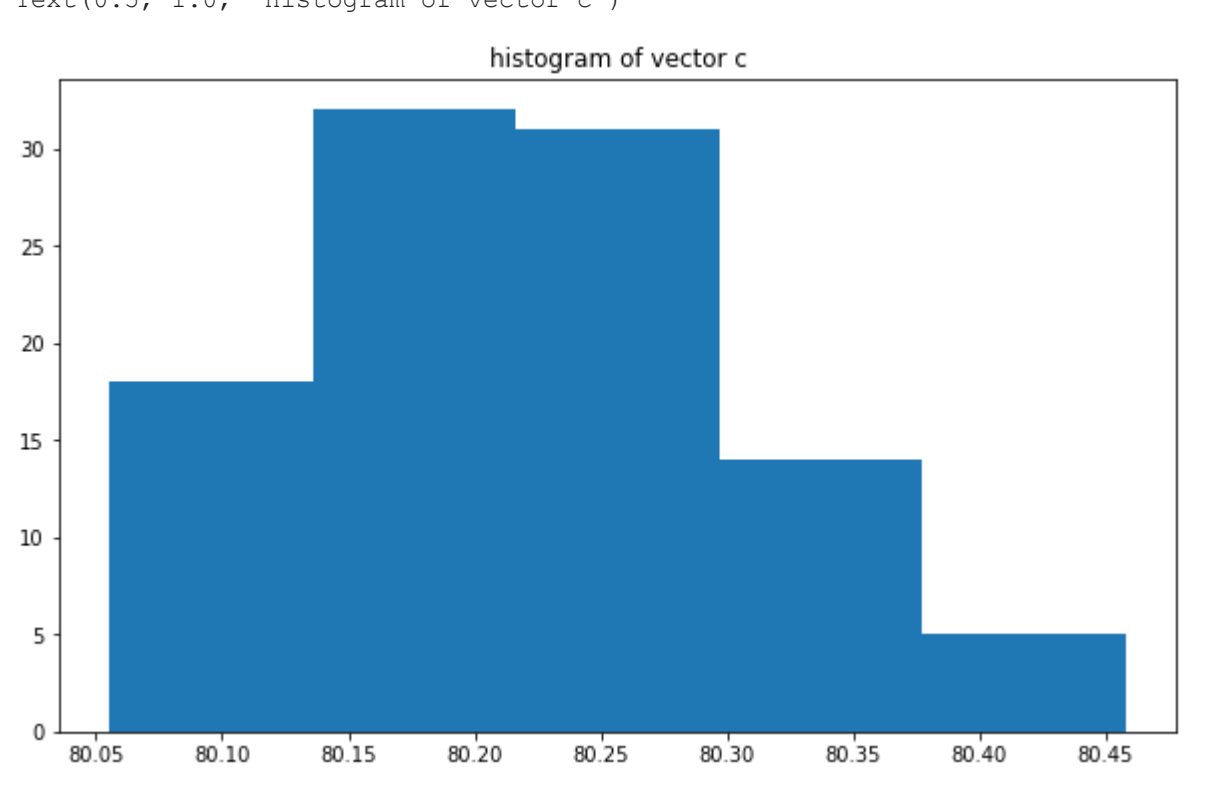
In [4]:
```python
# function for calculation in mean and standard deviation
def SdMeanCal(c):
    m=0
    s=0

    for i in range(len(c)): # summing the values of vector
        m += c[i]
    mean = m/len(c) # divinding the sum with total size of vector

    for i in range(len(c)):
        s += (c[i]-mean)**2 # subracting mean from each value and then squaring and sum
    sd = (s/len(c))**0.5 # dividing with total size and then taking square root

    return np.squeeze(mean),np.squeeze(sd) # changing the shape to scalar.
```

In [5]:
```python
mean,sd = SdMeanCal(c)
print('Mean :' , mean)
print('Stamdard Deviation :' , sd)
```

```
Mean : 80.22139714094364
Stamdard Deviation : 0.08689637676505708
```

In [6]:
```python
# plotting the histogram with 5 bins with mean value in centre
n_bins = 5
fig, axs = plt.subplots(1,1)
axs.hist(c, bins=n_bins)
plt.title('histogram of vector c')
```

Out[6]: `Text(0.5, 1.0, 'histogram of vector c')`



Grading Program:

In [7]:
```python
# reading data from csv
df = pd.read_csv('Grades.csv')
df. head(10)
```

Out[7]:

	First Name	Last Name	English	Maths	Science	German	Sports	Final Grade
0	Robyn	Hobgood	60.95	24.77	20.60	69.32	8.36	184.00
1	Eddy	Swearngin	100.00	12.99	100.00	52.24	100.00	365.23
2	Leoma	Bridgman	83.37	100.00	78.69	100.00	19.50	381.56
3	Arnetta	Peart	87.75	100.00	86.93	87.90	41.73	404.31
4	Maryland	Colby	100.00	100.00	100.00	18.87	88.72	407.59
5	Sherron	Sherron	92.06	55.91	93.93	-56.74	77.71	262.87
6	Glendora	Christopher	78.26	100.00	25.60	100.00	100.00	403.86
7	Darlena	Gunn	100.00	64.53	100.00	23.21	79.01	366.75
8	Aldo	Armas	100.00	83.49	100.00	100.00	92.32	475.81
9	Tiny	Jack	94.35	33.09	82.57	31.13	100.00	341.14

In [8]:
```python
# computing sum of all subjects
df['totalMarks'] = df["English"] + df[" Maths"]+ df["Science"] + df["German"] +df["Sports"]
df.head(10)
```

Out[8]:

	First Name	Last Name	English	Maths	Science	German	Sports	Final Grade	totalMarks
0	Robyn	Hobgood	60.95	24.77	20.60	69.32	8.36	184.00	184.00
1	Eddy	Swearngin	100.00	12.99	100.00	52.24	100.00	365.23	365.23
2	Leoma	Bridgman	83.37	100.00	78.69	100.00	19.50	381.56	381.56
3	Arnetta	Peart	87.75	100.00	86.93	87.90	41.73	404.31	404.31
4	Maryland	Colby	100.00	100.00	100.00	18.87	88.72	407.59	407.59
5	Sherron	Sherron	92.06	55.91	93.93	-56.74	77.71	262.87	262.87
6	Glendora	Christopher	78.26	100.00	25.60	100.00	100.00	403.86	403.86
7	Darlena	Gunn	100.00	64.53	100.00	23.21	79.01	366.75	366.75
8	Aldo	Armas	100.00	83.49	100.00	100.00	92.32	475.81	475.81
9	Tiny	Jack	94.35	33.09	82.57	31.13	100.00	341.14	341.14

In [9]:
```python
# calculating average points for each student by dividing with total points 500.
df['AveragePoint'] = (df['totalMarks']/500)*100
df.head(10)
```

Out[9]:

	First Name	Last Name	English	Maths	Science	German	Sports	Final Grade	totalMarks	AveragePoint
0	Robyn	Hobgood	60.95	24.77	20.60	69.32	8.36	184.00	184.00	36.800
1	Eddy	Swearngin	100.00	12.99	100.00	52.24	100.00	365.23	365.23	73.046
2	Leoma	Bridgman	83.37	100.00	78.69	100.00	19.50	381.56	381.56	76.312
3	Arnetta	Peart	87.75	100.00	86.93	87.90	41.73	404.31	404.31	80.862
4	Maryland	Colby	100.00	100.00	100.00	18.87	88.72	407.59	407.59	81.518
5	Sherron	Sherron	92.06	55.91	93.93	-56.74	77.71	262.87	262.87	52.574
6	Glendora	Christopher	78.26	100.00	25.60	100.00	100.00	403.86	403.86	80.772
7	Darlena	Gunn	100.00	64.53	100.00	23.21	79.01	366.75	366.75	73.350
8	Aldo	Armas	100.00	83.49	100.00	100.00	92.32	475.81	475.81	95.162
9	Tiny	Jack	94.35	33.09	82.57	31.13	100.00	341.14	341.14	68.228

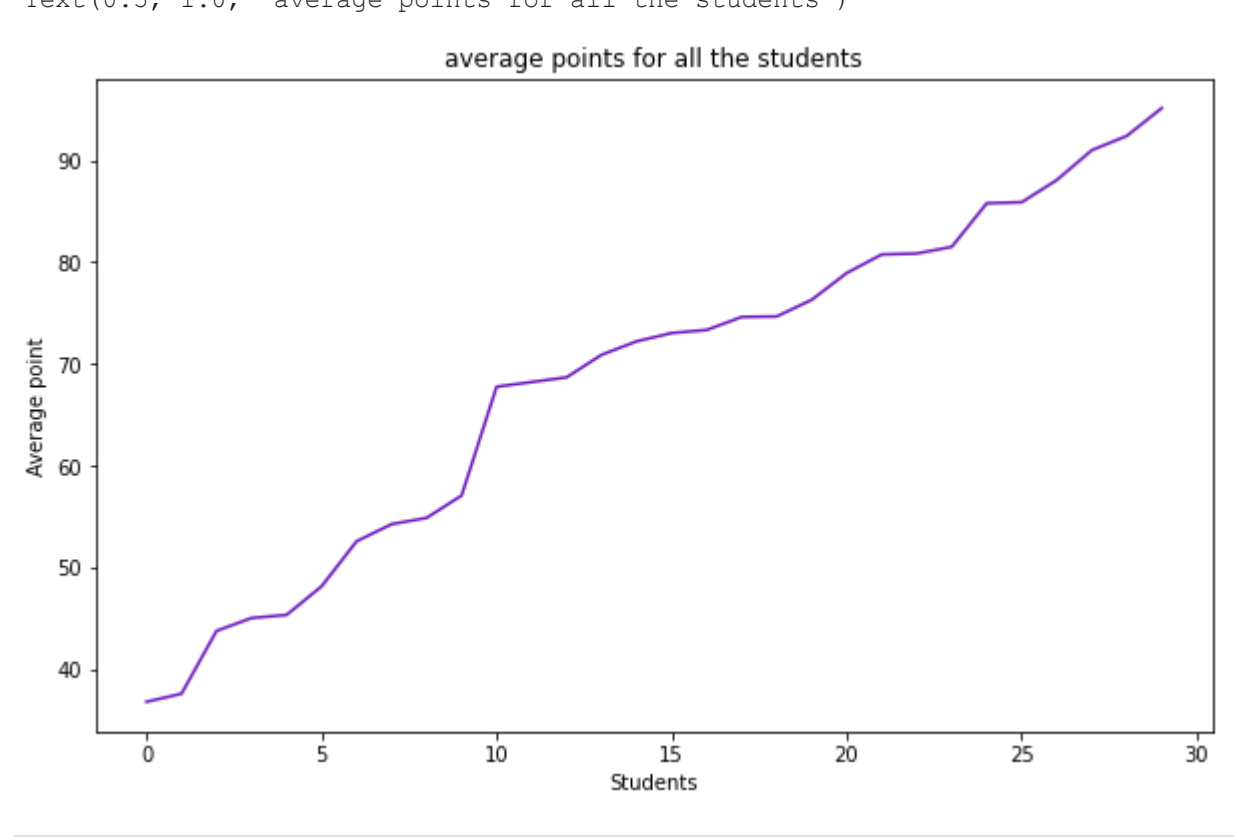
In [10]:
```python
# calculating standard deviation over average point column by subracting mean of this column from each average
df['SD'] = (((df['AveragePoint']-df['AveragePoint'].mean())**2)* (1/len(df)))**0.5
df.head(10)
```

Out[10]:

	First Name	Last Name	English	Maths	Science	German	Sports	Final Grade	totalMarks	AveragePoint	SD
0	Robyn	Hobgood	60.95	24.77	20.60	69.32	8.36	184.00	184.00	36.800	5.792507
1	Eddy	Swearngin	100.00	12.99	100.00	52.24	100.00	365.23	365.23	73.046	0.825077
2	Leoma	Bridgman	83.37	100.00	78.69	100.00	19.50	381.56	381.56	76.312	1.421364
3	Arnetta	Peart	87.75	100.00	86.93	87.90	41.73	404.31	404.31	80.862	2.252077
4	Maryland	Colby	100.00	100.00	100.00	18.87	88.72	407.59	407.59	81.518	2.371846
5	Sherron	Sherron	92.06	55.91	93.93	-56.74	77.71	262.87	262.87	52.574	2.912582
6	Glendora	Christopher	78.26	100.00	25.60	100.00	100.00	403.86	403.86	80.772	2.235645
7	Darlena	Gunn	100.00	64.53	100.00	23.21	79.01	366.75	366.75	73.350	0.880580
8	Aldo	Armas	100.00	83.49	100.00	100.00	92.32	475.81	475.81	95.162	4.862888
9	Tiny	Jack	94.35	33.09	82.57	31.13	100.00	341.14	341.14	68.228	0.054565

In [11]:
```python
# plotting the sorted average points of each students
plt.plot(df.sort_values('AveragePoint',ignore_index=True)['AveragePoint'],color ='#6c1abd')
plt.xlabel('Students')
plt.ylabel('Average point')
plt.title('average points for all the students')
```

Out[11]: `Text(0.5, 1.0, 'average points for all the students')`



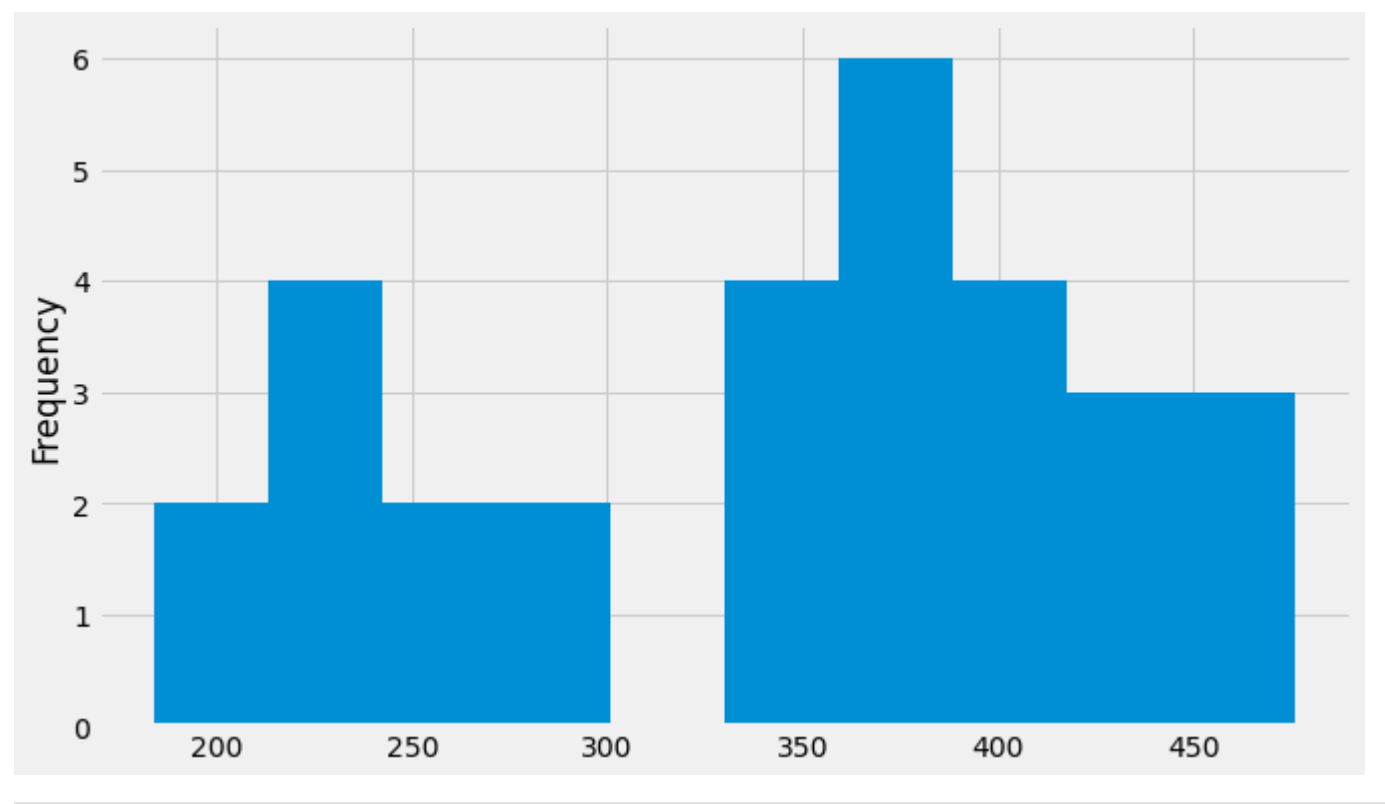
In [12]:
```python
# defining a diction for grading systems
grades = {
                (0, 55) : 'F',
                (56, 59) : 'D',
                (60, 65) : 'C',
                (66, 69) : 'C+',
                (70, 75) : 'B-',
                (76, 79) : 'B',
                (80, 85) : 'B+',
                (86, 89) : 'A-',
                (90, 95) : 'A',
                (96, 100) : 'A+',
                }

# defining a function for getting a grade from dictionary for a given point
def getGrade(point):
    result = 'None'

    # iterating through grades dictionary
    for (k1, k2) in grades:

        # checking if point lies between the dictionary key
        if (k1 <= int(point+0.5) and k2 >= int(point+0.5)): # adding 0.5 for rounding off values above .5
            result =  grades[(k1,k2)] # assigning grade if value is in between
            break

    return result

# calculating grade column from Average Point Column and for each row of AveragePoint columnn mapping getGrade
df['grades']= df['AveragePoint'].map(lambda x: getGrade(x))

df.head(10)
```

Out[12]:

	First Name	Last Name	English	Maths	Science	German	Sports	Final Grade	totalMarks	AveragePoint	SD	grades
0	Robyn	Hobgood	60.95	24.77	20.60	69.32	8.36	184.00	184.00	36.800	5.792507	F
1	Eddy	Swearngin	100.00	12.99	100.00	52.24	100.00	365.23	365.23	73.046	0.825077	B-
2	Leoma	Bridgman	83.37	100.00	78.69	100.00	19.50	381.56	381.56	76.312	1.421364	B
3	Arnetta	Peart	87.75	100.00	86.93	87.90	41.73	404.31	404.31	80.862	2.252077	B+
4	Maryland	Colby	100.00	100.00	100.00	18.87	88.72	407.59	407.59	81.518	2.371846	B+
5	Sherron	Sherron	92.06	55.91	93.93	-56.74	77.71	262.87	262.87	52.574	2.912582	F
6	Glendora	Christopher	78.26	100.00	25.60	100.00	100.00	403.86	403.86	80.772	2.235645	B+
7	Darlena	Gunn	100.00	64.53	100.00	23.21	79.01	366.75	366.75	73.350	0.880580	B-
8	Aldo	Armas	100.00	83.49	100.00	100.00	92.32	475.81	475.81	95.162	4.862888	A
9	Tiny	Jack	94.35	33.09	82.57	31.13	100.00	341.14	341.14	68.228	0.054565	C+

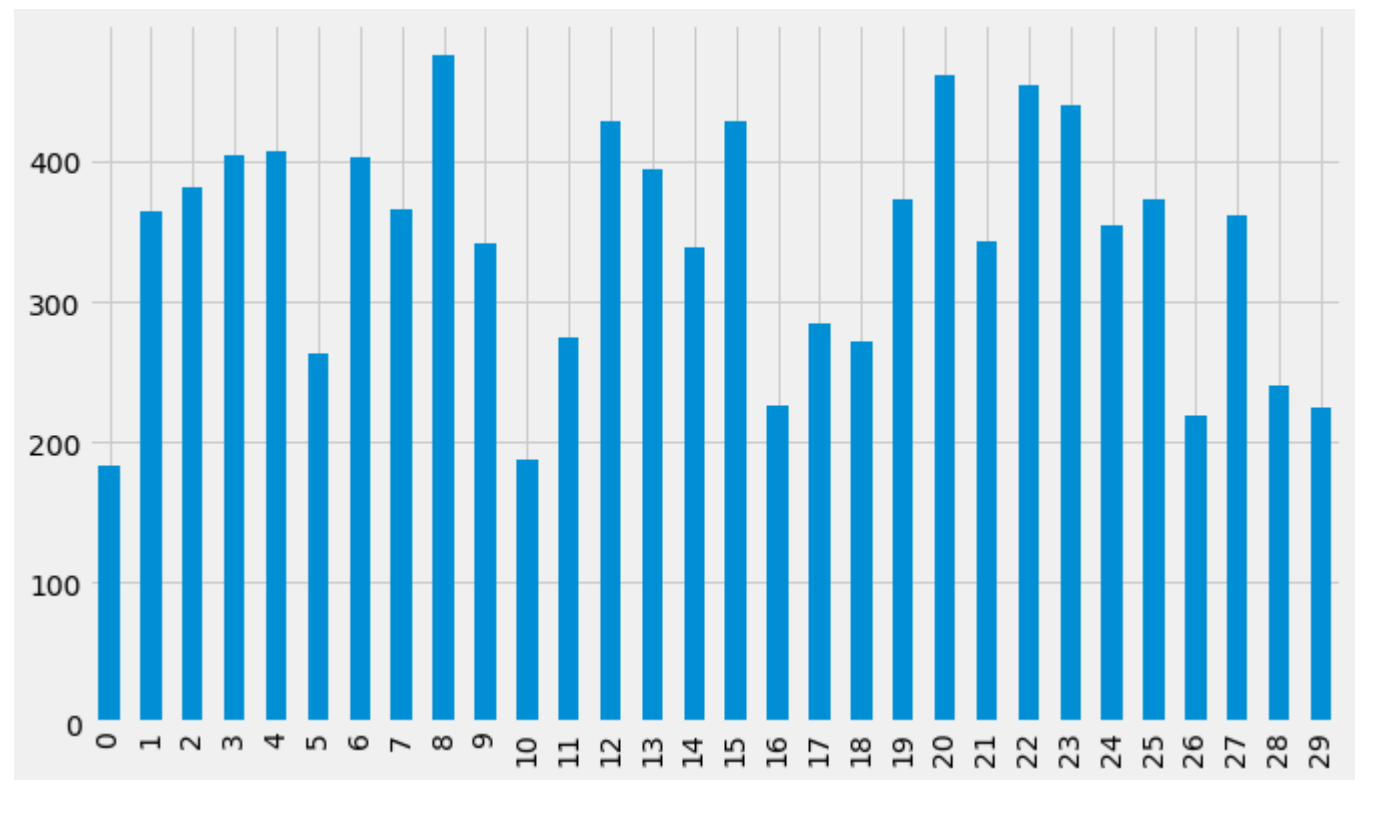
In [42]:
```python
df['Final Grade'].plot(kind='hist')
```

Out[42]: `<AxesSubplot:ylabel='Frequency'>`



In [43]:
```python
df['Final Grade'].plot(kind='bar')
```

Out[43]: `<AxesSubplot:>`



Exercise 2: Linear Regression through exact form.

- Generate 3 sets of simple data; a matrix A with dimensions 100 × 2. Initialize it with normal distribution $\mu = 2$ and $\sigma = [0.01, 0.1, 1]$

In [13]:
```python
np.random.seed(0)
rows = 100
col = 2
mean, sd = 2, 0.01
A = np.random.normal(mean, sd, (rows,col))
```

- Implement LEARN-SIMPLE-LINREG algorithm and train it using matrix A to learn values of $\beta 0$ and $\beta 1$

In [14]:
```python
def LEARN_SIMPLE_LINREG(A,n):

    numerator=0
    denominator=0
    X= A[:, 0] # first column as X
    Y= A[:, 1] # second column as target Y
    Xmean= np.mean(A[:, 0])
    Ymean= np.mean(A[:, 1])

    for i in range (n): # for calculating B1
        numerator += (X[i] - Xmean) * (Y[i] - Ymean)
        denominator += (X[i] - Xmean) ** 2

    B1 =numerator/denominator
    B0= Ymean - (B1 * Xmean)

    return B0,B1,X,Y
```

- Implement PREDICT-SIMPLE-LINREG and calculate the points for each training example in matrix A.

In [15]:
```python
def predict_simple_linreg(X,B0,B1):

    y_hat= B0 + (B1 * X) # output of simple linear model

    return y_hat
```

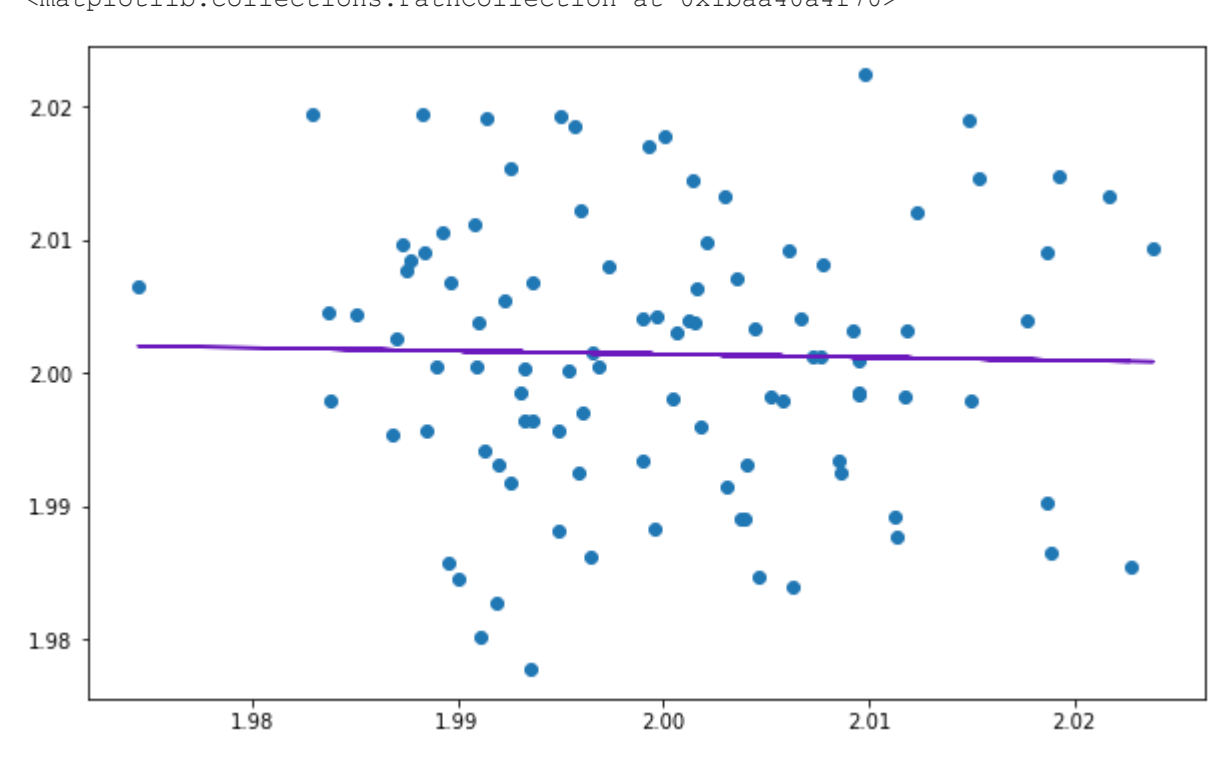
In [16]:
```python
B0,B1,X,Y=LEARN_SIMPLE_LINREG(A,rows) # learning values of B0,B1
Y_hat = predict_simple_linreg(X,B0,B1) # predicting output for the model
print("b0 =", B0)
print("b1 =", B1)
```

```
b0 = 2.0491493939501475
b1 = -0.023860917890473884
```

- Plot the training data (use plt.scatter) and your predicted line (use plt.plot).

In [17]:
```python
plt.plot(X,Y_hat,color ='#6c1abd') # regression line for targets
plt.scatter(X,Y) # for training points
```

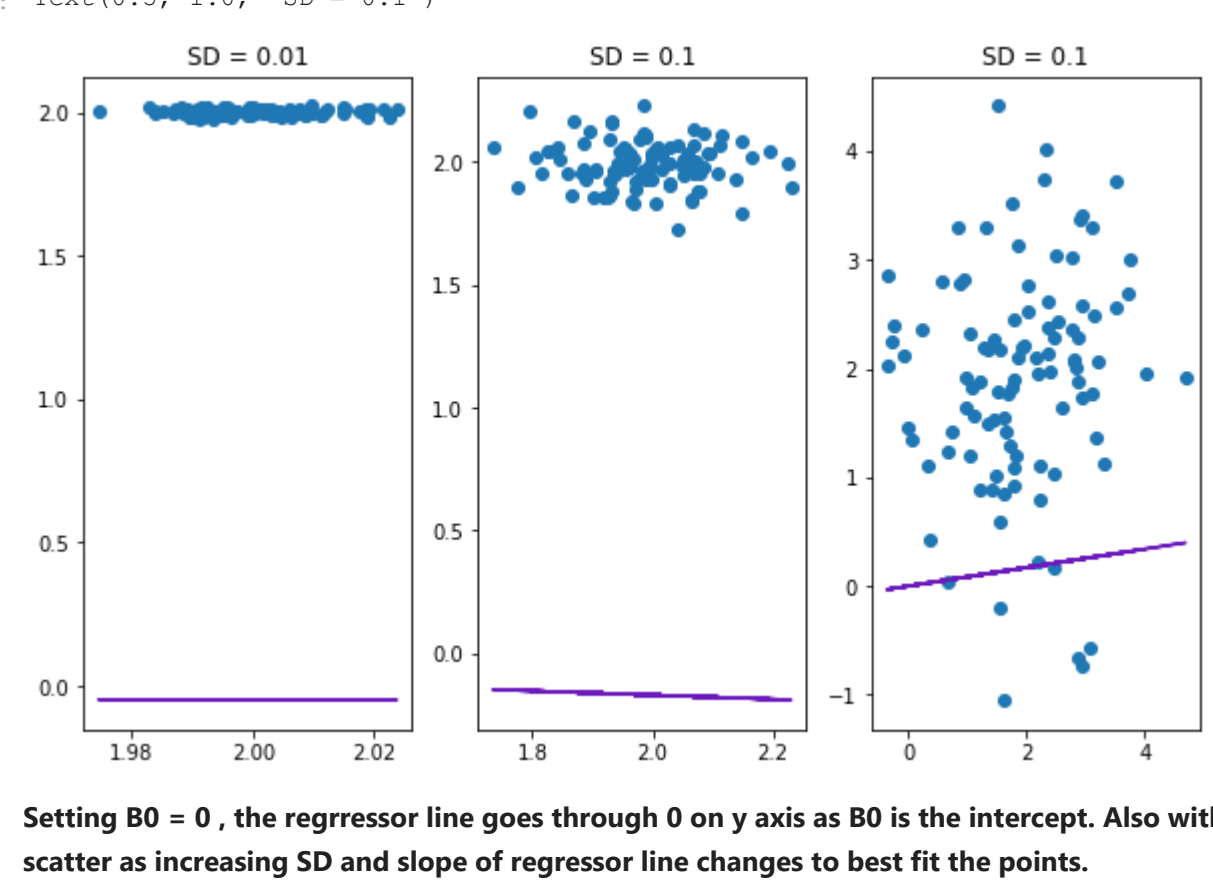
Out[17]: `<matplotlib.collections.PathCollection at 0x1baa40a4f70>`



- Put $\beta 0$ to zero and rerun the program to generate the predicted line. Comment on the change you see for the varying values of σ

In [18]:
```python
# For Standard Deviation = 0.01
np.random.seed(0)
mean, sd = 2, 0.01
A = np.random.normal(mean, sd, (rows,col))
B0,B1,X,Y=LEARN_SIMPLE_LINREG(A,rows)
Y_hat = predict_simple_linreg(X,0,B1)
plt.subplot(1, 3, 1)
plt.plot(X,Y_hat,color ='#6c1abd')
plt.scatter(X,Y)
plt.title("SD = 0.01")

# For Standard Deviation = 0.1
mean, sd = 2, 0.1
A = np.random.normal(mean, sd, (rows,col))
B0,B1,X,Y=LEARN_SIMPLE_LINREG(A,rows)
Y_hat = predict_simple_linreg(X,0,B1)
plt.subplot(1, 3, 2)
plt.plot(X,Y_hat,color ='#6c1abd')
plt.scatter(X,Y)
plt.title("SD = 0.1")

# For Standard Deviation = 1
mean, sd = 2, 1
A = np.random.normal(mean, sd, (rows,col))
B0,B1,X,Y=LEARN_SIMPLE_LINREG(A,rows)
Y_hat = predict_simple_linreg(X,0,B1)
plt.subplot(1, 3, 3)
plt.plot(X,Y_hat,color ='#6c1abd')
plt.scatter(X,Y)
plt.title("SD = 0.1")
```

Out[18]: `Text(0.5, 1.0, 'SD = 0.1')`

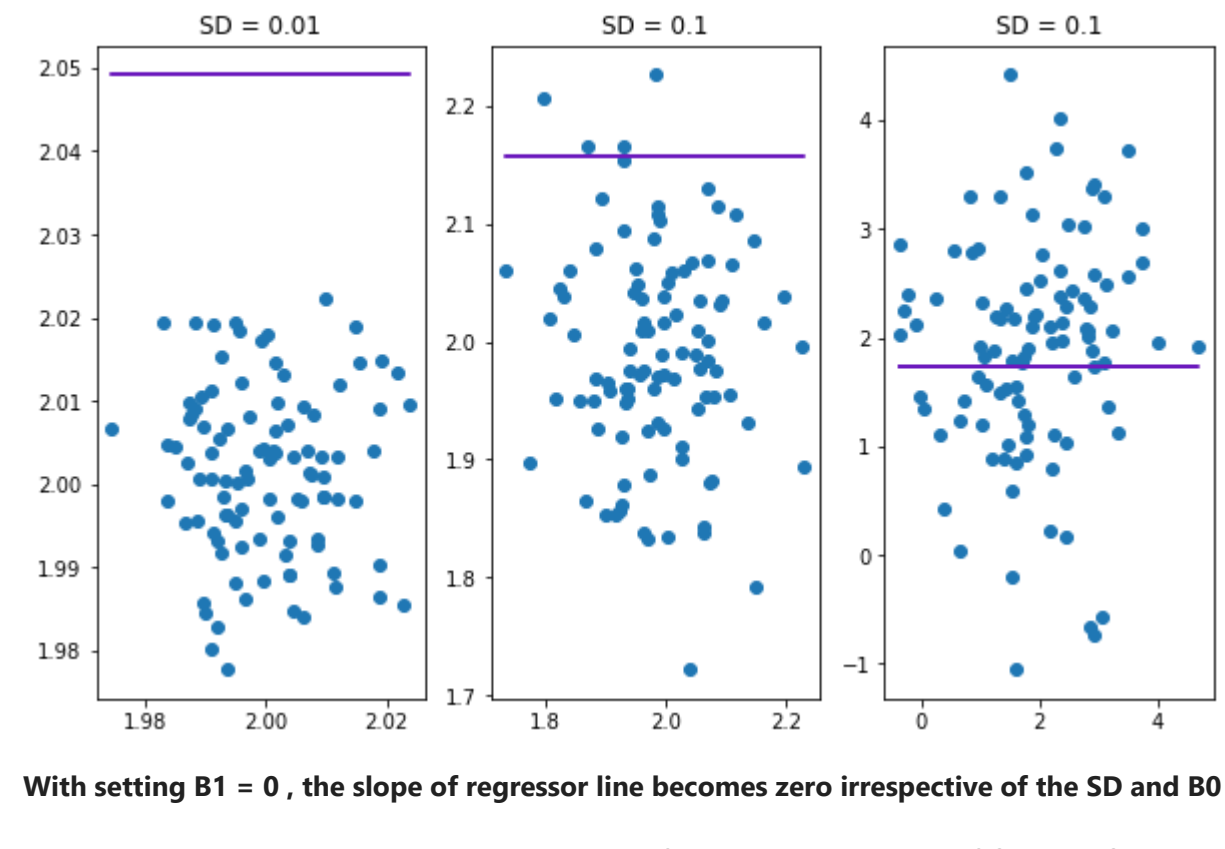


Setting B0 = 0 , the regrressor line goes through 0 on y axis as B0 is the intercept. Also with variation of SD, points becomes more scatter as increasing SD and slope of regressor line changes to best fit the points.

- Put $\beta 1$ to zero and rerun the program to generate the predicted line. Comment on the change you see for the varying values of σ

In [19]:
```python
np.random.seed(0)
mean, sd = 2, 0.01
A = np.random.normal(mean, sd, (rows,col))
B0,B1,X,Y=LEARN_SIMPLE_LINREG(A,rows)
Y_hat = predict_simple_linreg(X,B0,0)
plt.subplot(1, 3, 1)
plt.plot(X,Y_hat,color ='#6c1abd')
plt.scatter(X,Y)
plt.title("SD = 0.01")

mean, sd = 2, 0.1
A = np.random.normal(mean, sd, (rows,col))
B0,B1,X,Y=LEARN_SIMPLE_LINREG(A,rows)
Y_hat = predict_simple_linreg(X,B0,0)
plt.subplot(1, 3, 2)
plt.plot(X,Y_hat,color ='#6c1abd')
plt.scatter(X,Y)
plt.title("SD = 0.1")

mean, sd = 2, 1
A = np.random.normal(mean, sd, (rows,col))
B0,B1,X,Y=LEARN_SIMPLE_LINREG(A,rows)
Y_hat = predict_simple_linreg(X,B0,0)
plt.subplot(1, 3, 3)
plt.plot(X,Y_hat,color ='#6c1abd')
plt.scatter(X,Y)
plt.title("SD = 0.1")
```

Out[19]: `Text(0.5, 1.0, 'SD = 0.1')`

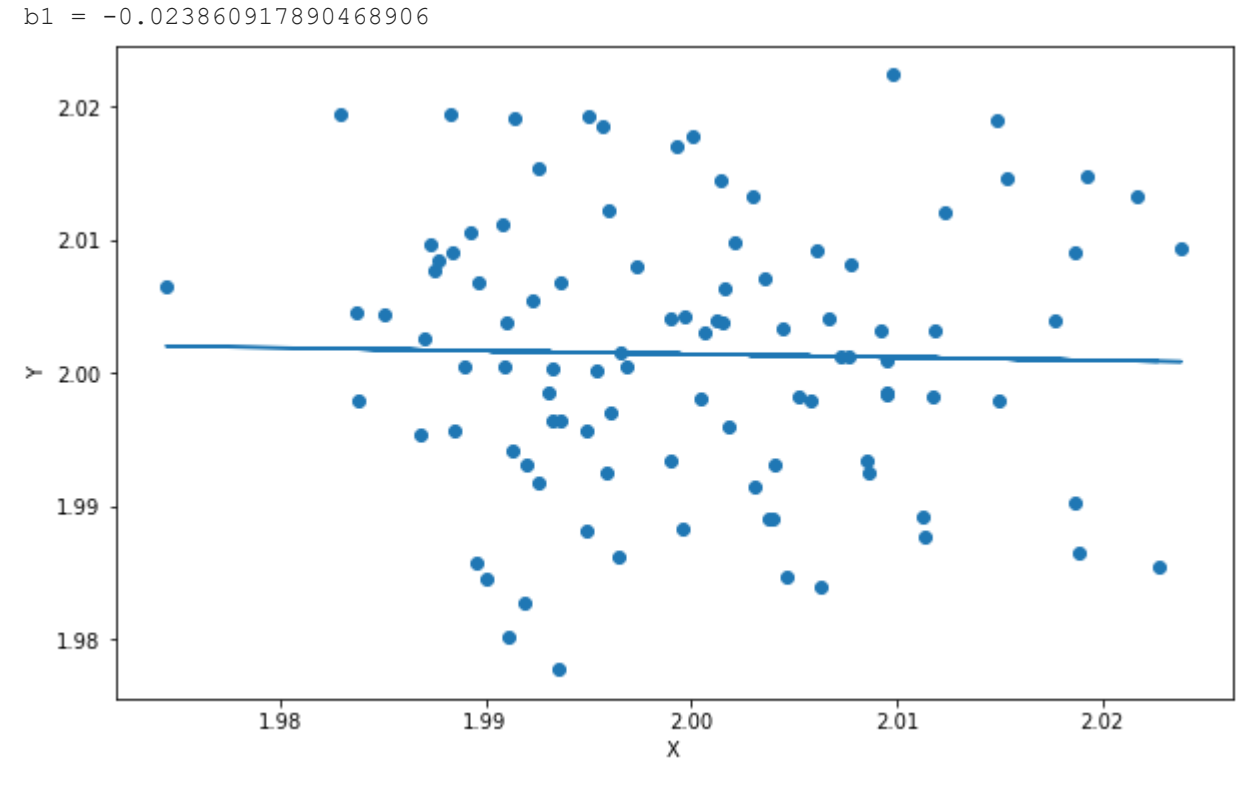


With setting B1 = 0 , the slope of regressor line becomes zero irrespective of the SD and B0 intercept is learnt with variations in SD

- Use numpy.linalg lstsq to replace step 2 for learning values of $\beta 0$ and $\beta 1$. Explain the difference between your values and the values from numpy.linalg lstsq.(1 point)

In [20]:
```python
# initializing the Data with normal distribution
np.random.seed(0)
rows = 100
col = 2
mean, sd = 2, 0.01
Data = np.random.normal(mean, sd,(rows,col))

x=Data[:,0].reshape((100,1)) # first column as input
y=Data[:,1] # second column as target#

bias_column = np.ones(shape=(100,1)) # bias column of ones
A = np.append(bias_column,x,axis=1)  # adding bias column with input for x0 feature

beta0,beta1 = np.linalg.lstsq(A, y,rcond=-1)[0] # linear regressor from numpy
print("b0 =", beta0)
print("b1 =", beta1)

Y_hat =(beta0 + beta1 * x) # predicting output

# plotting
plt.xlabel('X')
plt.ylabel('Y')
plt.plot(x,Y_hat)
plt.scatter(x,y)
plt.show()
```

```
b0 = 2.0491493939501377
b1 = -0.023860917890468906
```



My model implementation and one from numpy.linalg.lstsq returns same values of B0 and B1, hence same target predictions.

In [ ]: