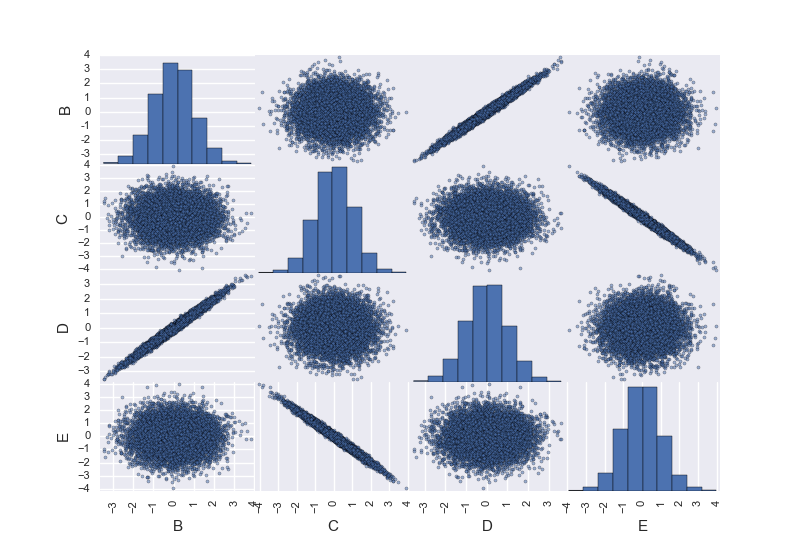
Lab 2

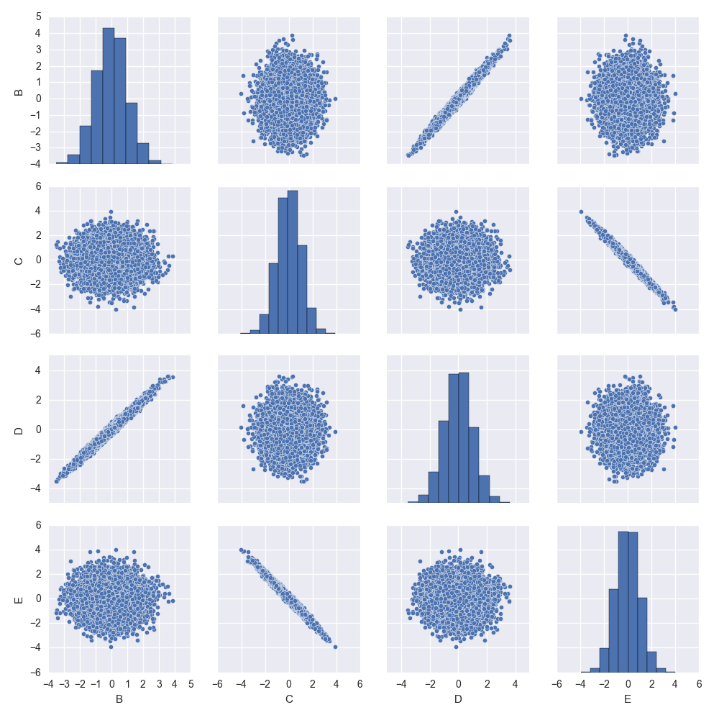
Aditya Kharosekar amk3587

Eyeunwana Nwoko ein79

# Problem 1

## Part a

Plot with Pandas

Plot with Seaborn

import pandas as pd

import numpy as np

from pandas.tools.plotting import scatter\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read\_csv("DF1.csv")

df.columns = ['A', 'B', 'C', 'D', 'E']

del df['A'] #deleting first column (Python adds this. Not needed)

df.drop(df.index[0]) #deleting first row (Python adds this. Not needed)

#with pandas

scatter\_matrix(df)

print(df.corr())

#with seaborn

sns.pairplot(df)

plt.show()

Columns 1 and 3 are pairwise correlated. Columns 2 and 4 are also pairwise correlated.

## Part b

The covariance matrix is a 4x4 matrix C. C[i][i] = variance of feature i, which is (1/n)∑(xi-xmean)2. C[i][j] are elements of this matrix which are not on the diagonal. C[i][j] = (1/n)∑(xi-xmean)(yi-ymean).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Features** | **A** | **B** | **C** | **D** |
| **A** | Variance(A) | Cov(A, B) | Cov(A, C) | Cov(A, D) |
| **B** | Cov(A, B) | Variance(B) | Cov(B, C) | Cov(B, D) |
| **C** | Cov(A, C) | Cov(B, C) | Variance(C) | Cov(C, D) |
| **D** | Cov(A, D) | Cov(B, D) | Cov(C, D) | Variance(D) |

print(np.cov(df, rowvar=False))

[[ 1.00155793 -0.00401176 0.99162409 0.00412485]

[-0.00401176 1.00537841 -0.00409877 -0.99545662]

[ 0.99162409 -0.00409877 1.00158867 0.00408108]

[ 0.00412485 -0.99545662 0.00408108 1.00516828]]

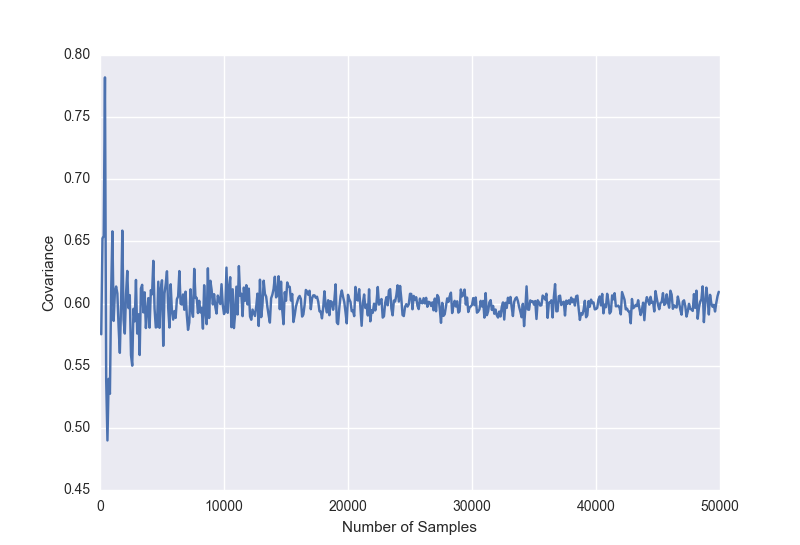
Let the above covariance matrix be called C. C[0][2] and C[2][0], which both represent the covariance between features A and C, have a value of 0.99, which represents a strong positive correlation. In the scatter matrix plot, these two scatter plots are tightly gathered around a line with positive slope, thereby showing a positive correlation.

Similarly, C[1][3] and C[3][1], which both represent the covariance between features B and D, have a value of -0.99, which represents a strong negative correlation. In the scatter matrix plot, these two scatter plots are tightly gathered around a line with negative slope, thereby showing a negative correlation.

In this way, the numbers agree with the plots.

## Part c

mean = [0,0,0]  
covMatrix = [[1, 0, 0], [0, 1, 0.6], [0, 0.6, 1]]  
  
cov = []  
samples = []  
**for** num **in** range(1, 500):  
 rv = np.random.multivariate\_normal(mean, covMatrix, size=(num\*100))  
 sum2 = 0.0  
 sum3 = 0.0  
 samples.append(num\*100)  
 **for** arr **in** rv:  
 sum2 += arr[1]  
 sum3 += arr[2]  
 mean2 = sum2 / (num\*100.0)  
 mean3 = sum3 / (num\*100.0)  
  
 diff2 = 0.0  
 diff3 = 0.0  
 tot = 0.0  
 **for** arr **in** rv:  
 diff2 = (arr[1] - mean2)  
 diff3 = (arr[2] - mean3)  
 tot += (diff2 \* diff3)  
 cov.append(tot / (num \* 100.0))  
plt.plot(samples, cov)  
plt.show()



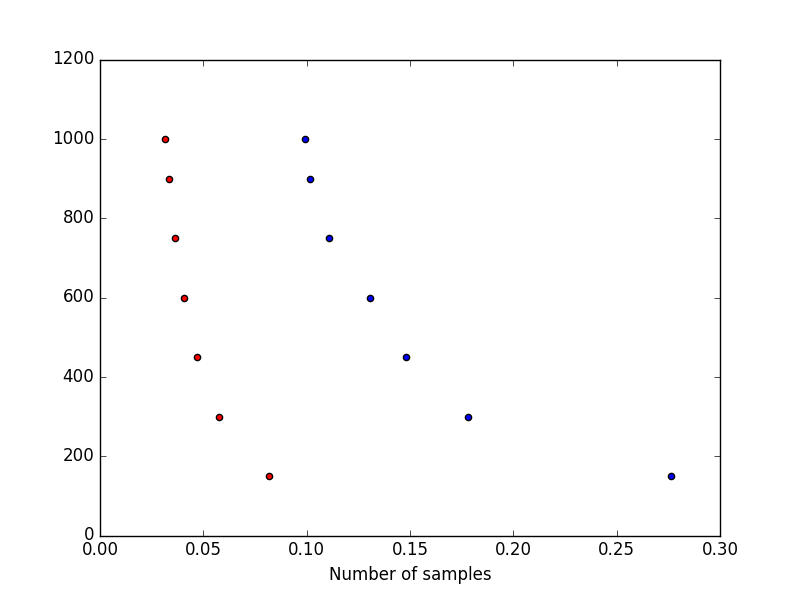
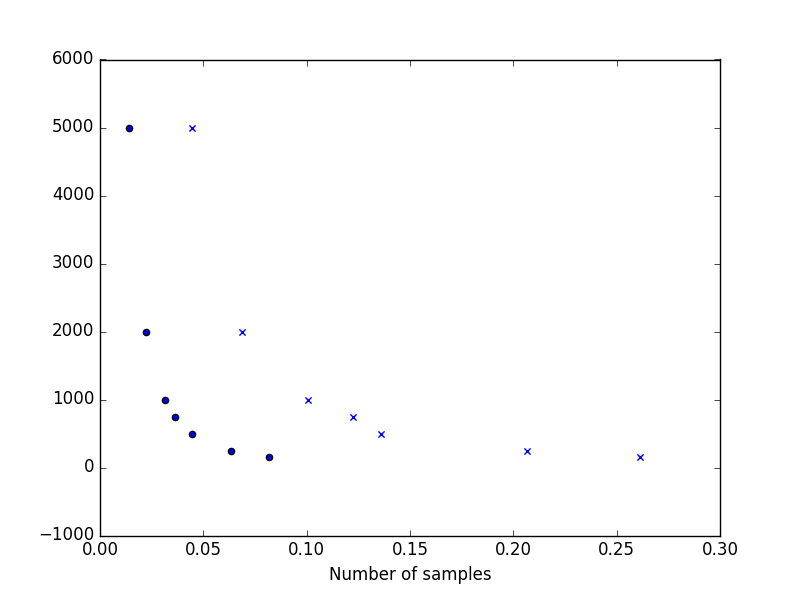
# Problem 3

## Part a

**def get\_emp\_stddev**(n):  
 errors = []  
 **for** num **in** range(n):  
 X = np.random.randn(n)  
 E = np.random.randn(n)  
 Y = -3 + E  
  
 #As this is one-dimensional, we can use the following formula  
 betahat = np.dot(X, Y) / np.dot(X, X)  
 errors.append(betahat - 0.0)  
  
 **return** np.std(errors)  
  
print(get\_emp\_stddev(150))

For n = 150, we get the empirical standard deviation to be 0.273479710848. This means that the Beta Hat which we find (-0.15) is not at all significant as this is less than the deviation.

## Part b

The red plot graphs the 1/sqrt(n) and the blue plot graphs the empirical standard deviation.

size = [150, 250, 500, 750, 1000, 2000, 5000]  
oneOver = []  
stddevs = []  
**for** i **in** size:  
 stddevs.append(get\_emp\_stddev(i))  
 oneOver.append(1/(i\*\*(1/2)))  
  
plt.scatter(stddevs, size, marker = 'x')  
plt.scatter(oneOver, size, marker = 'o')  
plt.xlabel("Number of samples")  
plt.show()  
  
print(stddevs)

# Problem 4

## Part a

**import** pandas **as** pd  
  
  
print("Enter k: ")  
k = int(input())  
print("Enter year: ")  
year = input()  
  
fileName = "Names/yob" + year + ".txt"  
df = pd.read\_csv(fileName, header=**None**)  
df.columns = ['A', 'B', 'C']  
male = df[df['B'] == 'M']  
female = df[df['B'] == 'F']  
print(male.head(k))  
print(female.head(k))

## Part b

My assumption is that the question is asking for the frequency of a name through the years.

print("Enter name: ")  
name = input()  
  
**for** num **in** range(1880, 2016):  
 fileName = "Names/yob" + str(num) + ".txt"  
 df = pd.read\_csv(fileName, header=**None**)  
 df.columns = ['A', 'B', 'C']  
 df\_partb = df[df['A'] == name]  
 print(df\_partb)  
 print()

## Part c

print("Enter name: ")  
name = input()  
**for** num **in** range(1880, 2016):  
 fileName = "Names/yob" + str(num) + ".txt"  
 df = pd.read\_csv(fileName, header=**None**)  
 df.columns = ['A', 'B', 'C']  
 df\_partB = df[df['A'] == name]  
 print("For " + str(num) + " : ")  
 print(df\_partB['C'].sum()/df['C'].sum())  
 print()

# Problem 5

To group tweets by state, we need to list of tags for each state. These tags are the common ways that people can talk about each state. For each state, the tags will include the state’s name, its abbreviation, and the names of the state’s biggest cities. For example –

NewYork = ['newyork', 'new york', 'ny', 'albany', 'nyc', 'buffalo']

California = ['ca', ' cali', 'california', 'los angeles', 'san francisco', 'san fran', 'san diego']

NorthCarolina = ['nc', 'northcarolina', 'north carolina', 'durham', 'charlotte', 'raleigh']

We will go through the user\_location column for each user, convert all the text to lower case, and search for the tags for any state. If any of these tags exist, we count that tweet for that particular state.