MIS 382N: Business Data Science | Fall 2019
Homework One

Rutwick Bhawsar | Shivang Arya

Libraries Imported

```
import numpy as np
import matplotlib.pyplot as plt
import math
import pandas as pd
from sklearn.decomposition import PCA
```

1. Create 1000 samples from a Gaussian distribution with mean -10 and standard deviation 5. Create another 1000 samples from another independent Gaussian with mean 10 and standard deviation 5.
   a. Take the sum of 2 these Gaussians by adding the two sets of 1000 points, point by point, and plot the histogram of the resulting 1000 points. What do you observe?
   b. Estimate the mean and the variance of the sum.

<u>Answer</u>

Creating a NumPy array of 1000 samples with mean -10 and std. deviation 5

```
mu = -10
sigma = 5
arr1 = np.random.normal(mu,sigma, 1000)
arr2 = np.random.normal(-mu,sigma, 1000)
```

Adding the two NumPy arrays and getting the mean and standard deviation for the purpose of plotting

```
arr3 = np.add(arr1,arr2)
mu = arr3.mean()
sigma = arr3.std()
```

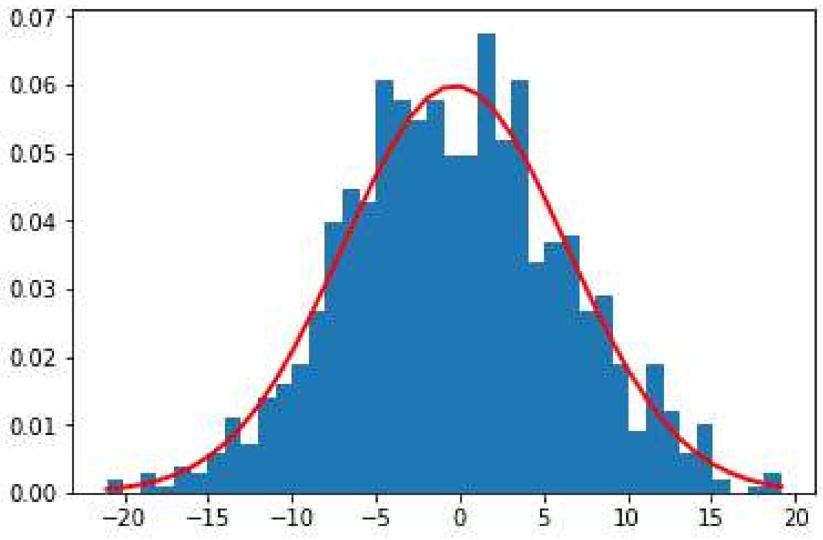
Plotting Histogram

```
count, bins, ignored = plt.hist(arr3, 40, normed=True)
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *  np.exp( - (bins - mu)**2 / (2 * sigma**2) ), linewidth=2, color='r')
plt.show()
```

Calculating and showing mean and variance

```
print("Mean : ",arr3.mean())
print("Variance : ",arr3.var())
```

Output



```
Mean :  -0.2953283731254982
Variance :  44.44237249218313
```

2. Central Limit Theorem. Let Xi be an iid Bernoulli random variable with value {-1,1}. Look at the random variable. $Z_n = \frac{1}{\sqrt{n}} \sum X_i$
   By taking 1000 draws from Zn, plot its histogram. Check that for small n (say, 5-10) Zn does not look that much like a Gaussian, but when n is bigger (already by the time n = 30 or 50) it looks much more like a Gaussian. Check also for much bigger n: n = 250, to see that at this point, one can really see the bell curve.

Answer

Defining a funtion which returns the value of Z_n.

```
def iid(n):
   temparr = [-1,1]
   arr1=np.random.choice(temparr,size = n)
   summ = np.sum(arr1)
   rootn = math.sqrt(n)
   z = summ/rootn
   return z
arr = np.array([])
```

A for loop to take 1000 draws and 1000 values of Z_n. Then appending the value to an array.

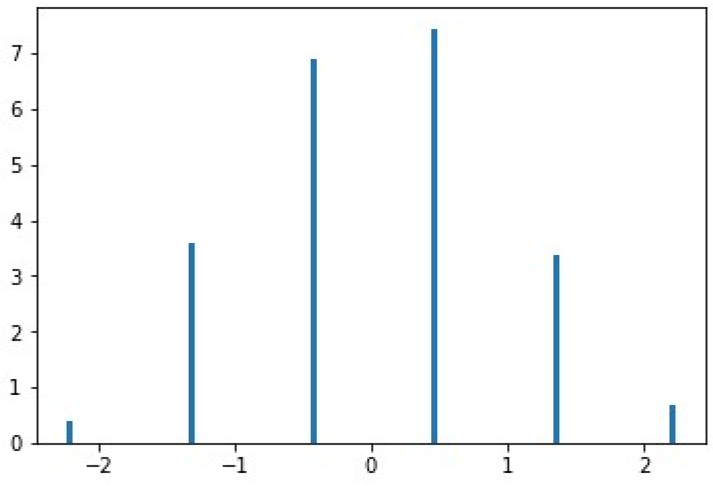
```
for i in range(1000):
    zt=iid(5)
    arr = np.append(arr,zt)
```

Plotting the array.

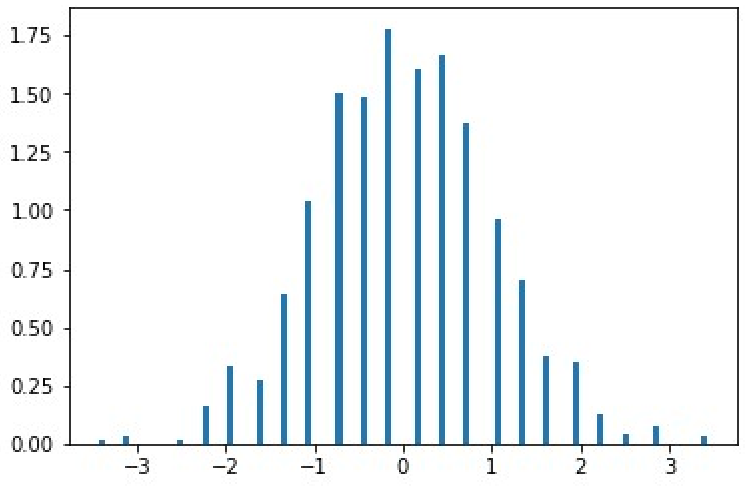
```
plt.hist(arr,density="true",bins=100)
plt.show()
```

Output

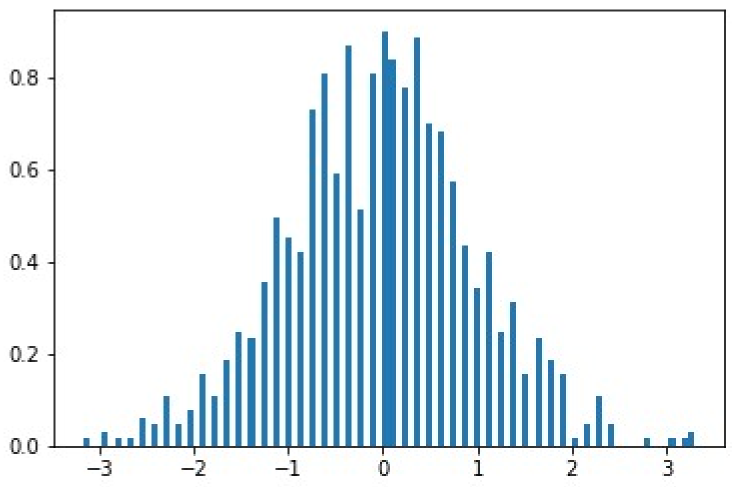
1. Output when n = 5



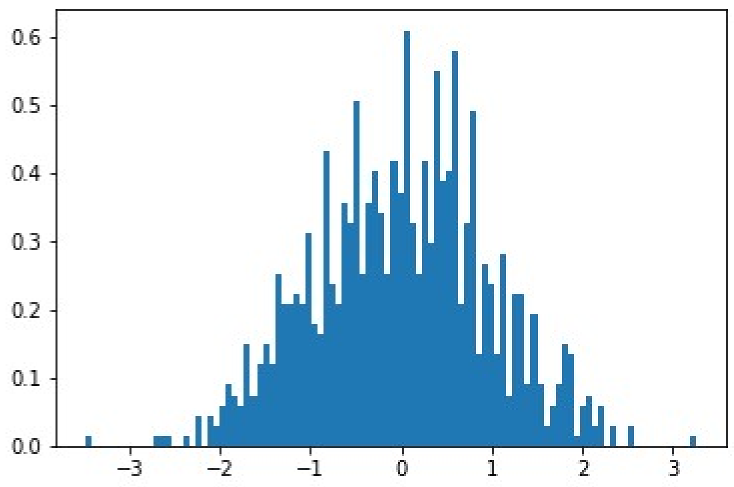
2. Output when n = 45



3. Output when n =250



4. Output when n = 5000



3. Estimate the mean and standard deviation from 1 dimensional data: generate 25,000 samples from a Gaussian distribution with mean 0 and standard deviation 5. Then estimate the mean and standard deviation of this Gaussian using elementary numpy commands, i.e., addition, multiplication, division (do not use a command that takes data and returns the mean or standard deviation).

<u>Answer</u>

Creating a NumPy array of 25000 samples with mean 0 and std. deviation 5.

```
mu = 0
sigma = 5
n = 25000
array = np.random.normal(mu,sigma,n)
```

Calculating the mean by adding all the elements in the array and dividing by total number of elements.

```
mean = np.sum(array)/n
print("Mean : ",mean)
```

Printing the mean using NumPy functions for verification

```
print("Mean with function for verification : ", np.mean(array))
```

Calculating the std. deviation by using the formula

$$s = \sqrt{\frac{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}{n-1}}$$

```
stddev = np.sqrt(np.sum(np.subtract(array,mean)**2)/n)
print("\n\nStd. Deviation : ",stddev)
```

Printing the std. deviation using NumPy functions for verification

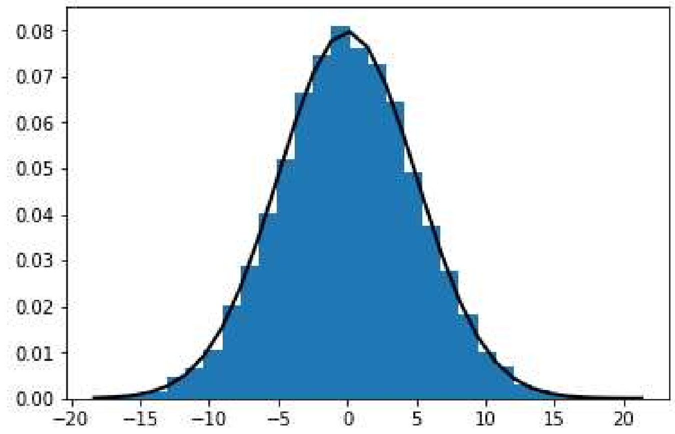
```
print("Std. Deviation with function for verification : ",np.std(array))
```

Plotting the histogram

```
count, bins, ignored = plt.hist(array, 30, normed=True)
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) * np.exp( - (bins - mu)**2 / (2 * sigma**2) ), linewidth=2,
color='black')
plt.show()
```

Output

```
Mean :  0.080847936745O3259
Mean with function for verification :  0.08084793674503259


Std. Deviation :  4.9977962497816675
Std. Deviation with function for verification :  4.9977962497816675
```



4. Estimate the mean and covariance matrix for multi-dimensional data: generate 10,000 samples of 2 dimensional data from the Gaussian distribution

$$\begin{pmatrix} X_i \\ Y_i \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} -5 \\ 5 \end{pmatrix}, \begin{pmatrix} 20 & .8 \\ .8 & 30 \end{pmatrix}\right)$$

   Then, estimate the mean and covariance matrix for this multi-dimensional data using elementary numpy commands, i.e., addition, multiplication, division (do not use a command that takes data and returns the mean or standard deviation). (Recall: this notation is a compact way of specifying the distribution of two Gaussian random variables that are correlated. Specially, the notation indicates that X has mean -5 and 1 variance 20, Y has mean 5 and variance 30, and the covariance of X and Y is 0:8, i.e., E[(X + 5)(Y - 5)] = 0.8.)

Answer

Creating two NumPy array of 10000 samples. One with mean -5 and std. deviation = square root of $\sqrt{20}$. Another with mean 5 and std. deviation is $\sqrt{30}$ ($Std.Dev = \sqrt{Variance}$)

```
mu1 = -5
sigma1 = math.sqrt(20)
mu2 = 5
sigma2 = math.sqrt(30)
n = 10000
ar1 = np.random.normal(mu1,sigma1,n)
ar2 = np.random.normal(mu2,sigma2,n)
```

Taking the mean of both the arrays. Used for covariance matrix.

```
mean1 = np.sum(ar1)/n
mean2 = np.sum(ar2)/n
```

Creating a 2D array with both the previous arrays.

```
arr = np.array([ar1,ar2])
```

Taking the mean and verifying it

```
mean = (np.sum(arr)/(n*2))
print("Mean : ",mean)
print("Mean with function for verification : ",np.mean(arr))
```

Calculating Cov_{xy} , Cov_{xx} , Cov_{yy}

*covariancexy = (np.sum(np.subtract(ar1,mean1)*np.subtract(ar2,mean2))/n)*
*covariancexx = np.sum(np.subtract(ar1,mean1)**2)/n*
*covarianceyy = np.sum(np.subtract(ar2,mean2)**2)/n*

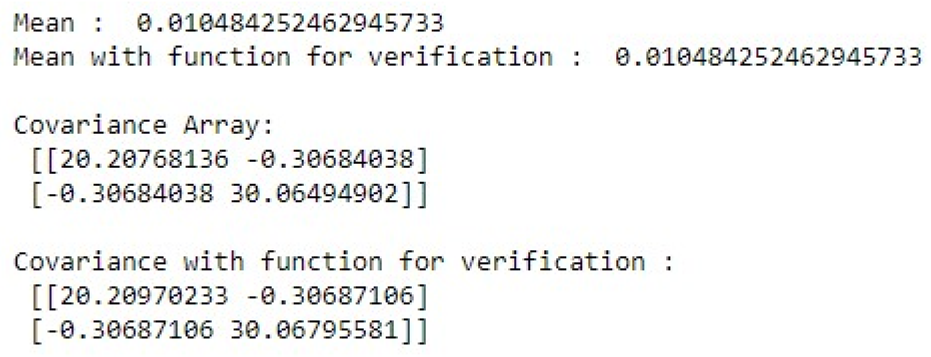
Putting the values in the Covariance Matrix

covarr = np.array([[covariancexx,covariancexy], [covariancexy,covarianceyy]])

Printing and Verifying the matrix

print("\nCovariance Array: \n", covarr)
print("\nCovariance with function for verification : \n",np.cov(arr))

Output



```
Mean :  0.010484252462945733
Mean with function for verification :  0.010484252462945733

Covariance Array:
 [[20.20768136 -0.30684038]
 [-0.30684038 30.06494902]]

Covariance with function for verification :
 [[20.20970233 -0.30687106]
 [-0.30687106 30.06795581]]
```

5. Download from Canvas/Files the dataset PatientData.csv.

   Each row is a patient and the last column is the condition that the patient has. Do data exploration using Pandas and other visualization tools to understand what you can about the dataset. For example:

   (a) How many patients and how many features are there?
   (b) What is the meaning of the first 4 features? See if you can understand what they mean.
   (c) Are there missing values? Replace them with the average of the corresponding feature column
   (d) How could you test which features strongly influence the patient condition and which do not? List what you think are the three most important features.

Answer

Read the .csv file in Pandas dataframe df and store a backup of in in df2.

df = pd.read_csv('PatientData.csv',header=None)
df2 = df

a. Print the shape of the dataframe to identify the number of rows and column

print(df.shape)

Output

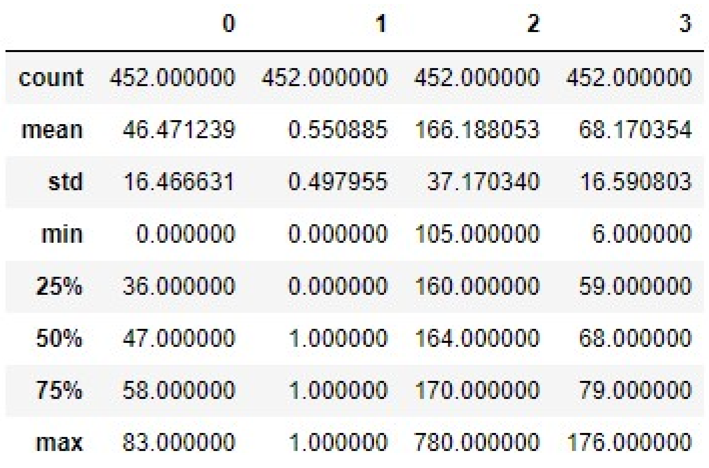
(452 , 280)

The total Rows = 452
The total Columns = 280

b. We followed the following steps to determine the meaning of the first 4 columns.

i. First we extracted the first 4 column from the entire dataframe. Then described it.

df = df[[0,1,2,3]].describe()



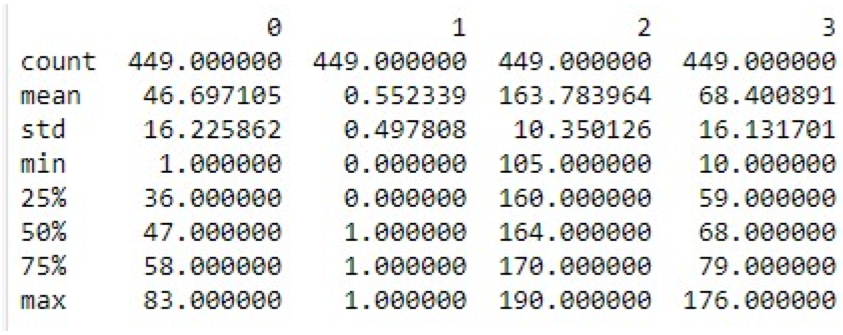
	0	1	2	3
count	452.000000	452.000000	452.000000	452.000000
mean	46.471239	0.550885	166.188053	68.170354
std	16.466631	0.497955	37.170340	16.590803
min	0.000000	0.000000	105.000000	6.000000
25%	36.000000	0.000000	160.000000	59.000000
50%	47.000000	1.000000	164.000000	68.000000
75%	58.000000	1.000000	170.000000	79.000000
max	83.000000	1.000000	780.000000	176.000000

ii. We sorted the data to find the outliers.

df = df.sort_values(2,ascending = False)

iii. We found that the top 4 rows in the 3rd column were noise and so we eliminated then. Then we described the table again.

df = df.iloc[3:]
df.descibe()



```
                0           1           2           3
count  449.000000  449.000000  449.000000  449.000000
mean    46.697105    0.552339  163.783964   68.400891
std     16.225862    0.497808   10.350126   16.131701
min      1.000000    0.000000  105.000000   10.000000
25%     36.000000    0.000000  160.000000   59.000000
50%     47.000000    1.000000  164.000000   68.000000
75%     58.000000    1.000000  170.000000   79.000000
max     83.000000    1.000000  190.000000  176.000000
```

iv. Based on the raw data, mean, std. deviation and minimum and maximum values we deduced the following.

<u>First Column: Age</u>. Because the max is 83 and min is 1 and the average is 46.6.
<u>Second Column: Gender.</u> Because the columns had only 0s and 1s so we realized that it was a categorical variable. Since the mean was close to 0.5 we assumed that there were approximately equal male and females.
<u>Third Column: Height in cms</u>. Because avg. was 163, min was 105 and max was 190 which is what we figured to be the average heights.
<u>Fourth Column: Weight in kgs,</u> Because firstly the units are in S.I. Secondly looking at min is 10kgs, maximum is 176kgs which is slightly high but it's believable.

v. To confirm our deduction found correlation amongst these column since in reality age, height and weight are related.

Finding the correlation between what we think is age and height in cms

print("\n",df[0].corr(df[2]))

Finding the correlation between what we think is age and weight in kgs

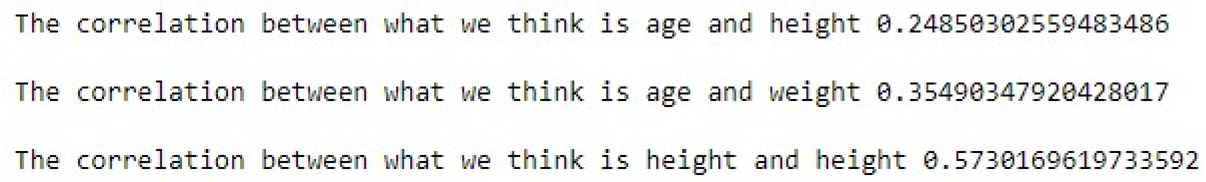
print(df[0].corr(df[3]))

Finding the correlation between what we think is height and weight

print(df[2].corr(df[3]))

Our assumptions were confirmed since we found a positive correlation amongst all of them.

Output



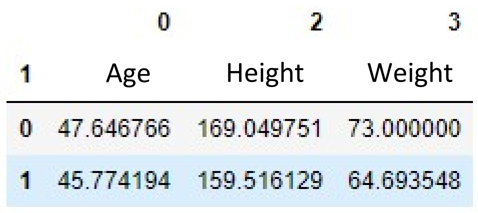
```
The correlation between what we think is age and height 0.24850302559483486

The correlation between what we think is age and weight 0.35490347920428017

The correlation between what we think is height and height 0.5730169619733592
```

vi. We first grouped the data on the basis on gender and got the means.

We then inferred that 0 is Male and 1 is Female. Since Avg. Height and Avg. Weight of 0s was higher than those of 1.

df_gender=df[[0,1,2,3]].groupby(1).mean()
df_gender.head()

Output



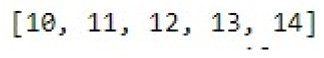
```
          0           2          3
1       Age      Height     Weight
0  47.646766  169.049751  73.000000
1  45.774194  159.516129  64.693548
```

c. We observed the dataset and observed that the missing values were denoted by '?' instead of NaN. First we replaced the '?' with NaN

df = df[:].replace('?', np.nan)

Then we figured out which columns had missing values

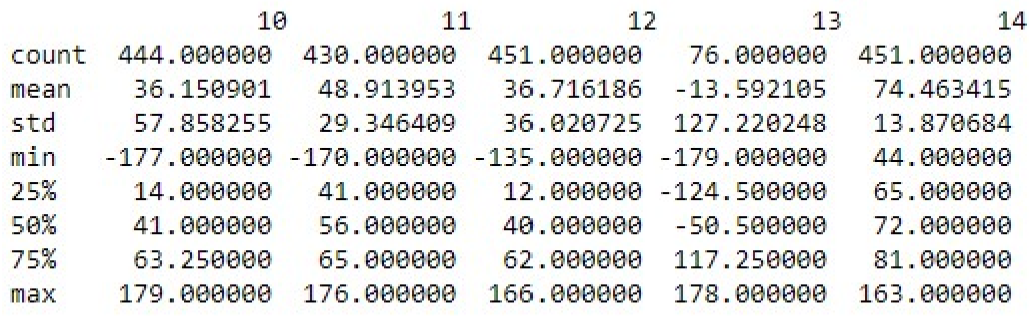
print(df.columns[df.isna().any()].tolist())



```
[10, 11, 12, 13, 14]
```

After describing these columns we realized that the mean wasn't getting calculated. This was because the values in the columns were stored as strings. So we explicitly converted them to float and then described them.

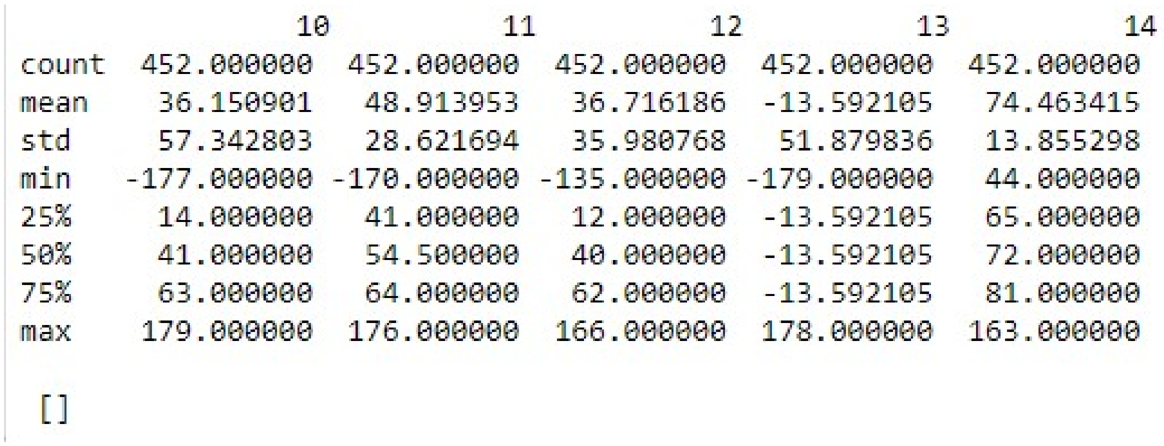
df.iloc[:,10:15]=df.iloc[:,10:15].astype(float)
print(df.iloc[:,10:15].describe())



```
               10          11          12          13          14
count  444.000000  430.000000  451.000000   76.000000  451.000000
mean    36.150901   48.913953   36.716186  -13.592105   74.463415
std     57.858255   29.346409   36.020725  127.220248   13.870684
min   -177.000000 -170.000000 -135.000000 -179.000000   44.000000
25%     14.000000   41.000000   12.000000 -124.500000   65.000000
50%     41.000000   56.000000   40.000000  -50.500000   72.000000
75%     63.250000   65.000000   62.000000  117.250000   81.000000
max    179.000000  176.000000  166.000000  178.000000  163.000000
```

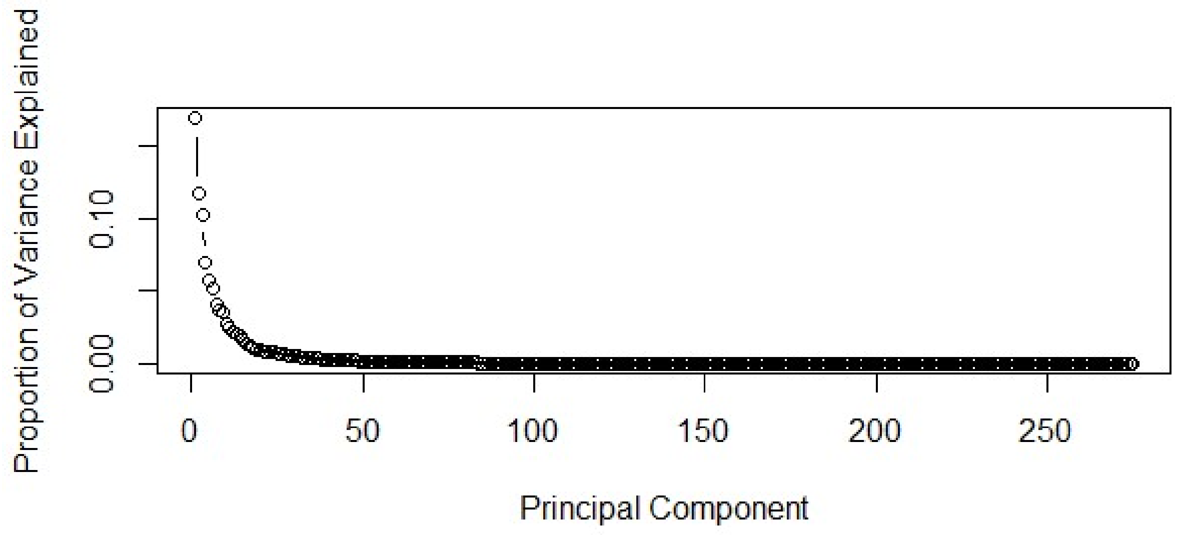
We replaced NaN with the mean of the corresponding column. To confirm the replacement, we described it again, saw that the mean was unchanged and checked if there are any columns with missing values and the answer was no.

```
df = df.fillna(df.mean(axis = 0, skipna = True))
print("\n",df.iloc[:,10:15].describe())
print("\n",df.columns[df.isna().any()].tolist())
```



```
               10          11          12          13          14
count  452.000000  452.000000  452.000000  452.000000  452.000000
mean    36.150901   48.913953   36.716186  -13.592105   74.463415
std     57.342803   28.621694   35.980768   51.879836   13.855298
min   -177.000000 -170.000000 -135.000000 -179.000000   44.000000
25%     14.000000   41.000000   12.000000  -13.592105   65.000000
50%     41.000000   54.500000   40.000000  -13.592105   72.000000
75%     63.000000   64.000000   62.000000  -13.592105   81.000000
max    179.000000  176.000000  166.000000  178.000000  163.000000

 []
```

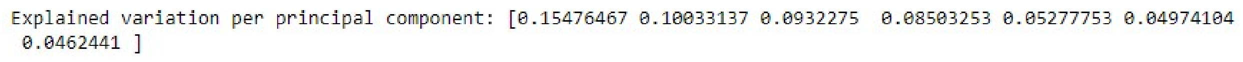
d. For finding the most important features in our dataset we used Principal Component Analysis. We extracted 7 components based on the Proportion of Variance explained graph.



pca = PCA(n_components = 7)
pca
pca.fit(df)

Proportion of Variance explained by each principal component.

print('Explained variation per principal component: {}'.format(pca.explained_variance_ratio_))



```
Explained variation per principal component: [0.15476467 0.10033137 0.0932275  0.08503253 0.05277753 0.04974104
 0.0462441 ]
```

We found out the features which have the maximum influence on our 7 principal components.

Features	PC1
9	0.444096
12	0.304351
258	0.230049
13	0.219639
248	0.174147
40	0.172991
187	0.170932
268	0.168229
257	0.167631
64	0.165991

Features	PC2
248	0.412921
247	0.359577
238	0.280944
258	0.275484
9	0.233061
237	0.231513
257	0.227793
12	0.169756
87	0.168455
99	0.160757

....

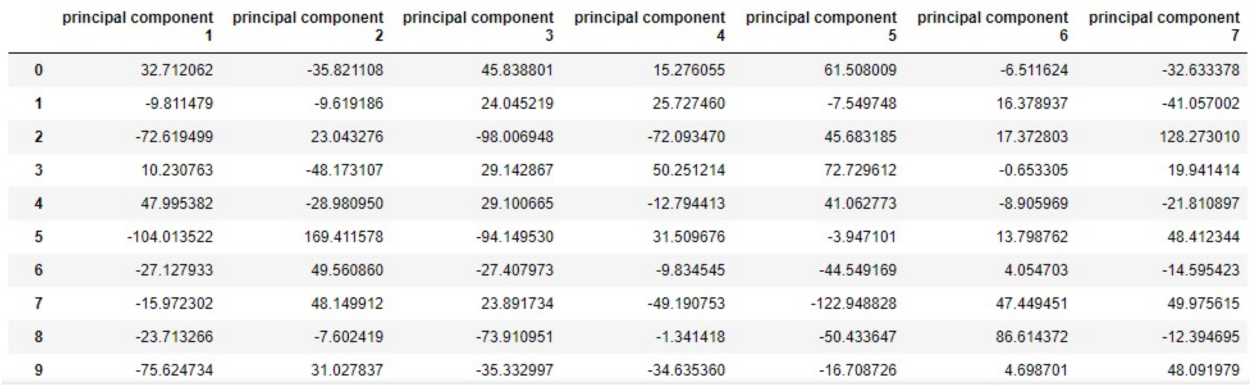
Features	PC7
7	0.351471
56	0.271770
5	0.245042
4	0.227163
6	0.218120
77	0.201978
29	0.192762
92	0.190674
17	0.187015
149	0.177752

The last column in the dataset is the dependent variable. This is stored in another dataset.

dfultimate=df.iloc[:,-1]

We fit all the rows in our dataset to the 7 principle components.

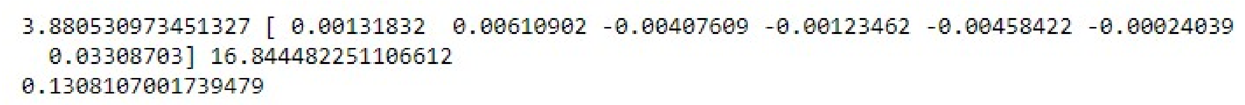
print(fitted)
principal_new = pd.DataFrame(data = fitted, columns = ['principal component 1', 'principal component 2', 'principal component 3', 'principal component 4', 'principal component 5', 'principal component 6', 'principal component 7'])
principal_new



	principal component 1	principal component 2	principal component 3	principal component 4	principal component 5	principal component 6	principal component 7
0	32.712062	-35.821108	45.838801	15.276055	61.508009	-6.511624	-32.633378
1	-9.811479	-9.619186	24.045219	25.727460	-7.549748	16.378937	-41.057002
2	-72.619499	23.043276	-98.006948	-72.093470	45.683185	17.372803	128.273010
3	10.230763	-48.173107	29.142867	50.251214	72.729612	-0.653305	19.941414
4	47.995382	-28.980950	29.100665	-12.794413	41.062773	-8.905969	-21.810897
5	-104.013522	169.411578	-94.149530	31.509676	-3.947101	13.798762	48.412344
6	-27.127933	49.560860	-27.407973	-9.834545	-44.549169	4.054703	-14.595423
7	-15.972302	48.149912	23.891734	-49.190753	-122.948828	47.449451	49.975615
8	-23.713266	-7.602419	-73.910951	-1.341418	-50.433647	86.614372	-12.394695
9	-75.624734	31.027837	-35.332997	-34.635360	-16.708726	4.698701	48.091979

We are finding the regression equation for all 7 of our principle components with respect to the dependent variable i.e. Last Column.

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(principal_new, dfultimate)
predicted = regr.predict(principal_new)
mse = np.mean((predicted-dfultimate)**2)
print (regr.intercept_, regr.coef_,mse)
print(regr.score(principal_new, dfultimate))
```



```
3.880530973451327 [ 0.00131832  0.00610902 -0.00407609 -0.00123462 -0.00458422 -0.00024039
  0.03308703] 16.844482251106612
0.1308107001739479
```

What we understand here is that the 7th Principal Component has the most effect on the independent variable by a large extent (coefficient of regression is more than 10 times higher than the other principal components).

Hence, we choose the three features that affect Principal Component 7 the most:
1. Feature 7 in Python (8th column in CSV)
2. Feature 56 in Python (57th column in CSV)
3. Feature 5 in Python (6th column in CSV)

Part II – Written Question on next page.

P.T.O

1.

	X=0	X=1	Total
Y=0	1/4	1/4	1/2
Y=1	1/6	1/3	1/2
Total	5/12	7/12	

(a) → $P(X=1) = ?$

From above table, it is clear that

$$P(X=1) = \frac{7}{12}$$

(b) → $P(X=1 \mid Y=1) = ?$

From above table,

$$P(X=1 \mid Y=1) = \frac{1}{3} \times \frac{1}{\left(\frac{1}{3}+\frac{1}{6}\right)} = \frac{2}{3}$$

(c) → Variance of random variable X?

According to X's probability distribution table, in 12 chances there is a high probability of X being 1 7 times and 0 for the remaining 5.

Therefore,

mean $(x) = 0\left(\frac{5}{12}\right) + 1\left(\frac{7}{12}\right)$

$\Rightarrow \mu_x = \frac{7}{12}$

x	$(x-\mu)^2$
1	25/144
1	25/144
1	25/144
1	25/144
1	25/144
1	25/144
1	25/144
0	49/144
0	49/144
0	49/144
0	49/144
0	49/144

$$\Sigma(x-\mu)^2 = \frac{(25\times7)}{144} + \frac{(49\times5)}{144}$$

$$= \frac{175}{144} + \frac{245}{144} = \frac{420}{144}$$

$$\text{Variance } (\sigma_x^2) = \frac{\Sigma(x-\mu)^2}{n}$$

$$= \frac{420}{144} \times \frac{1}{12}$$

$$= \frac{35}{144} \text{ or } 0.2430$$

(d) → variance of X given that $Y=1$?
$(\sigma^2_{x|y=1})$

Conditional distribution of x given that $Y=1$

	$X=0$	$X=1$	Total
$Y=1$	$\frac{1/6}{1/6+1/3} = \frac{1}{3}$	$\frac{1/3}{1/6+1/3} = \frac{2}{3}$	1

mean of X given $Y=1$

$$(\mu_{x|y=1}) = \frac{1}{3}(0) + \frac{2}{3}(1)$$

$$= \frac{2}{3}$$

X	$(X-\mu)^2$
0	4/9
1	1/9
1	1/9

$$\Sigma(x-\mu)^2 = \frac{6}{9} = \frac{2}{3}$$

$$\sigma^2_{x|y=1} = \frac{\Sigma(x-\mu)^2}{n}$$

$$\Rightarrow \sigma^2_{x|y=1} = \frac{2}{3} \times \frac{1}{3} = \frac{2}{9}$$

(e) → $E[x^3 + x^2 + 3y^7 \mid y=1] = ?$

Given that $y=1$, we can use the same conditional distribution as the previous part of the question.

x	x^2	x^3	$3y^7$	$x^3+x^2+3y^7$
0	0	0	3	3
1	1	1	3	5
1	1	1	3	5

$$E[x^3 + x^2 + 3y^7 \mid y=1] = \frac{\Sigma(x^3 + x^2 + 3y^7)}{n}$$

$$= \frac{13}{3}$$

2. $v_1 = [1, 1, 1]$

$v_2 = [1, 0, 0]$

$v_1 \times v_2$ will give a vector perpendicular to both of them.

Therefore we can project points on the subspace covered by v_1 & v_2 by subtracting the perpendicular vector from the points.

$$v_1 \times v_2 = \begin{vmatrix} i & j & k \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{vmatrix}$$

$$= i(0) - j(-1) + k(-1)$$

$$\bar{v}_3 = 0\hat{i} + 1\hat{j} - 1\hat{k}$$

Projection of $P_1\,[3,3,3]$ on R^3

$$= [3,3,3] - \frac{P_1 \cdot [0,1,-1]}{(\sqrt{2})^2}\,[0,1,-1]$$

$$= [3,3,3] - [0,0,0]$$

$$= [3,3,3]$$

Projection of $\underset{[1,2,3]}{P2}$ on R^3

$$= [1, 2, 3] - \frac{[1, 2, 3] \cdot [0, +1, -1]}{(\sqrt{2})^2} [0\ 1\ -1]$$

$$= [1\ 2\ 3] - \frac{0 + 2 - 3}{2} [0\ 1\ -1]$$

$$= [1\ 2\ 3] - [0\ \tfrac{-1}{2}\ \tfrac{1}{2}]$$

$$= [1\ 2.5\ 2.5]$$

Projection of $\underset{[0,0,1]}{P3}$ on R^3

$$= [0, 0, 1] - \frac{[0\ 0\ 1] \cdot [0\ 1\ -1]}{(\sqrt{2})^2} [0\ 1\ -1]$$

$$= [0\ 0\ 1] - \frac{0 + 0 - 1}{2} [0\ 1\ -1]$$

$$= [0\ 0\ 1] - [0\ \tfrac{-1}{2}\ \tfrac{1}{2}]$$

$$= [0\ \tfrac{1}{2}\ \tfrac{1}{2}]$$

3 → Given $P(X) = \frac{2}{3}$, $n = 100$

$X \rightarrow$ event of getting a Heads
$n \rightarrow$ number of coin tosses
$x \rightarrow$ outcome we want

To calculate: $P(X \leq x)$
where $x = 50$

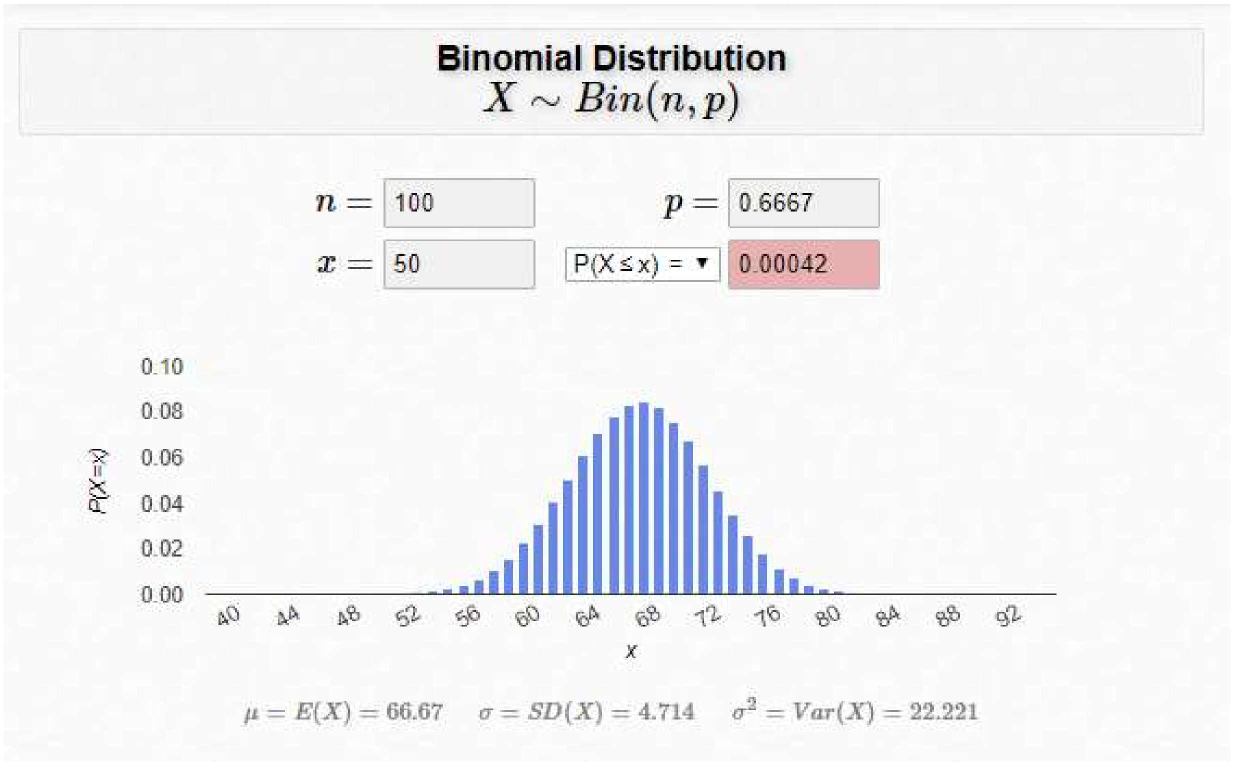
Using the Binomial Distribution,

$$P(X \leq 50) = {}^{100}C_0 \left(\frac{2}{3}\right)^0 \left(\frac{1}{3}\right)^{100} + {}^{100}C_1 \left(\frac{2}{3}\right) \left(\frac{1}{3}\right)^{99}$$

$$\ldots\ldots {}^{100}C_{50} \left(\frac{2}{3}\right)^{50} \left(\frac{1}{3}\right)^{50}$$

$= 0.00004182$

≈ 0.00042



Binomial Distribution

$X \sim Bin(n, p)$

$n =$ 100 $p =$ 0.6667

$x =$ 50 P(X ≤ x) = 0.00042

$\mu = E(X) = 66.67 \quad \sigma = SD(X) = 4.714 \quad \sigma^2 = Var(X) = 22.221$

Confirmed Question 3 using Binomial Distribution Calculator online.