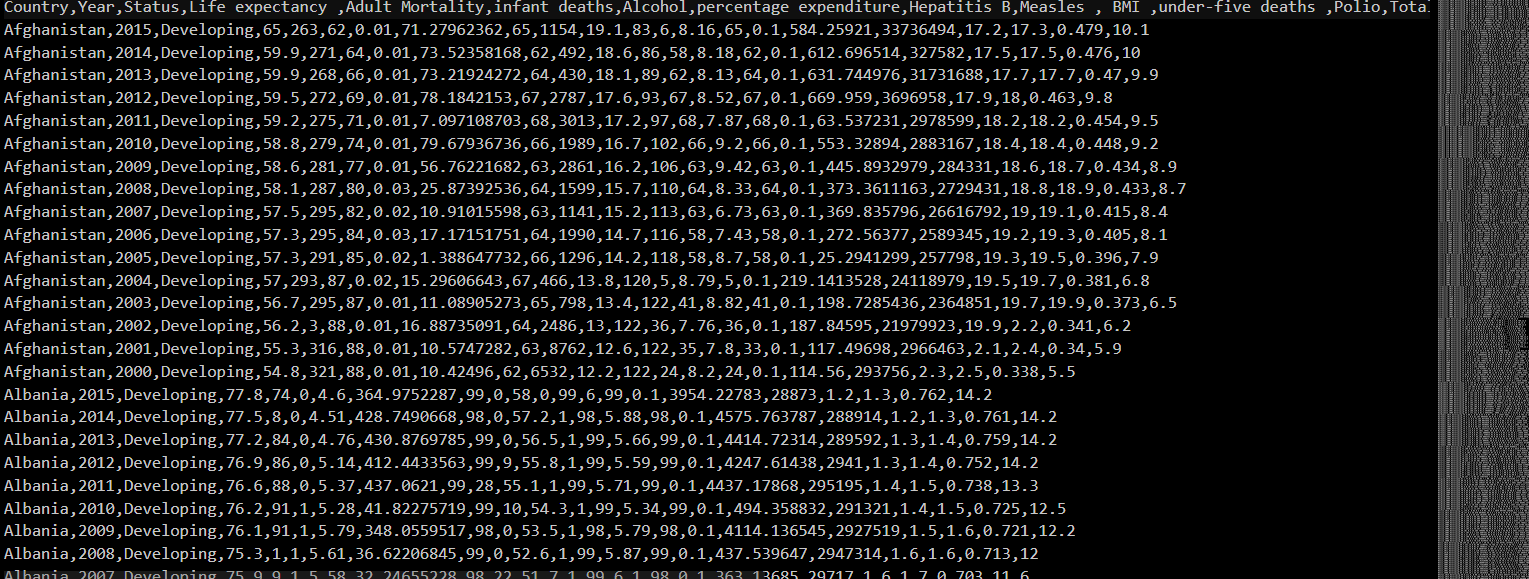
Original CSV



Code we use to Obtain Pakistan’s data from Original csv and create a new CSV on which later we will work on

header=[]

pakistan\_data = []

with *open*('life.csv',newline='') as csvfile:

file = csv.*reader*(*csvfile*)

for row in file:

if row[0]=='Country':

newRow=['','','','','','','','','']

for index in *range*(0,9):

newRow[index]=row[index]

header.*append*(*newRow*)

if row[0]=='Pakistan':

newRow=['','','','','','','','','']

for index in *range*(0,9):

newRow[index]=row[index]

pakistan\_data.*append*(*newRow*)

data = [\*header,\*pakistan\_data]

with *open*('pakistanLife.csv','w',newline='') as file:

writer = csv.*writer*(file)

for row in data:

writer.*writerow*(*row*)

Displaying Pakistan’s Comma Separated Values

We Use Code

ple=pandas.*read\_csv*('pakistanLife.csv')

*print*(*ple*.*head*(16))

Output:



As we can see there are a missing values such as 0.00000000 and NaN(Not a Number)

Now we need to Mark down these Values We use NaN instead of all Zeros

We can import NaN from Numpy

Side note:

NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices.

We use this code to mark all zeros to NaN

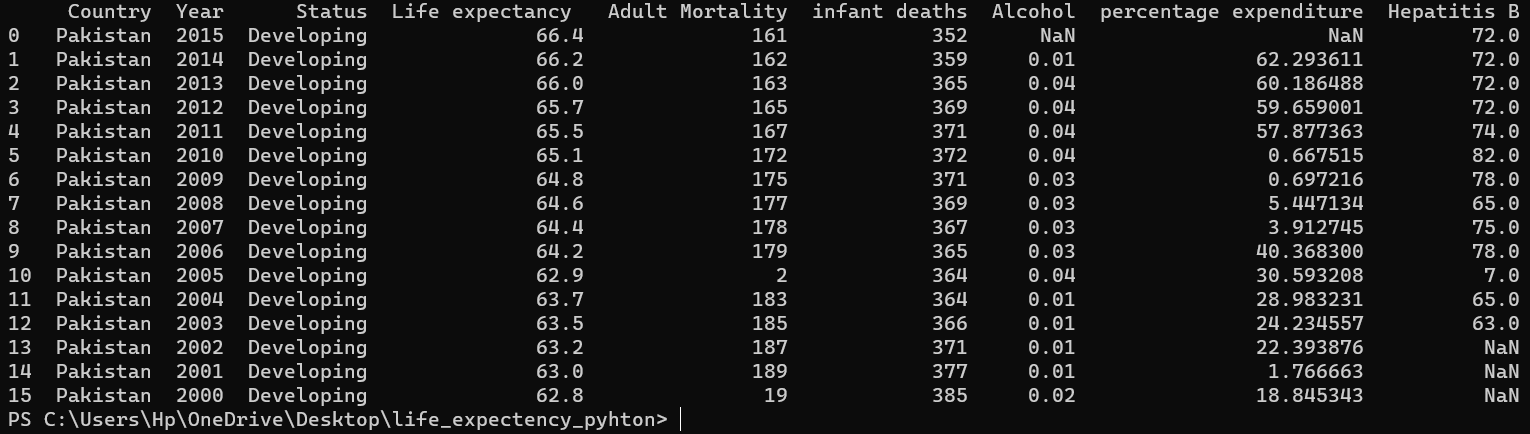
ple=pandas.*read\_csv*('pakistanLife.csv')

num\_missing = ple[['percentage expenditure']].*replace*(0,*NaN*)

*print*(*ple*.*describe*())

*print*(*ple*.*head*(16))

Output:



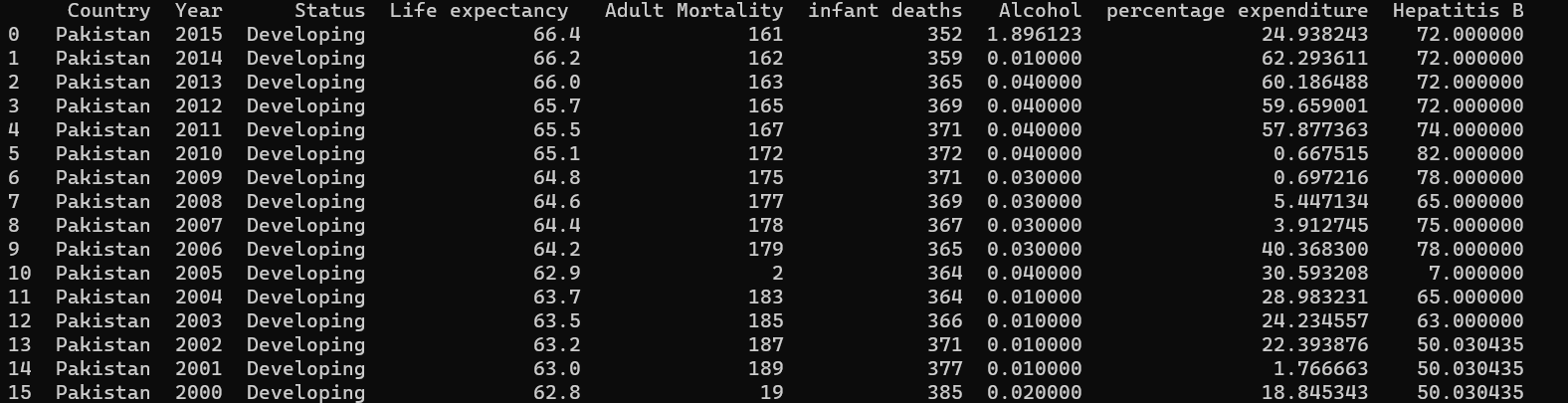
We successfully marked the missing values now to perform imputation

Mean Inputation

df[['Hepatitis B']] = df[['Hepatitis B']].*replace*(*nan*,*df*.*describe*().*mean*()[6])

df[['Alcohol']] = df[['Alcohol']].*replace*(*nan*,*df*.*describe*().*mean*()[4])

df[['percentage expenditure']] = df[['percentage expenditure']].*replace*(0,*df*.*describe*().*mean*()[5])



After it we have our data ready for Simple Linear Regression.

from numpy import nan

import pandas as pd

import csv

header=[]

pakistan\_data = []

with *open*('life.csv',newline='') as csvfile:

file = csv.*reader*(*csvfile*)

for row in file:

if row[0]=='Country':

newRow=['','','','','','','','','']

for index in *range*(0,9):

newRow[index]=row[index]

header.*append*(*newRow*)

if row[0]=='Pakistan':

newRow=['','','','','','','','','']

for index in *range*(0,9):

newRow[index]=row[index]

pakistan\_data.*append*(*newRow*)

data = [\*header,\*pakistan\_data]

with *open*('pakistanLife.csv','w',newline='') as file:

writer = csv.*writer*(file)

for row in data:

writer.*writerow*(*row*)

df = pd.*read\_csv*("pakistanLife.csv")

df[['Hepatitis B']] = df[['Hepatitis B']].*replace*(*nan*,*df*.*describe*().*mean*()[6])

df[['Alcohol']] = df[['Alcohol']].*replace*(*nan*,*df*.*describe*().*mean*()[4])

df[['percentage expenditure']] = df[['percentage expenditure']].*replace*(0,*df*.*describe*().*mean*()[5])

*print*(*df*.*head*(16))

X=df['Hepatitis B'].values.*tolist*()

Y=df['Life expectancy '].values.*tolist*()

*print*(*X*)

*print*(*Y*)

X\_concat=[]

for i in *range*(*len*(*X*)):

X\_concat\_row=[]

X\_concat\_row.*append*(1)

X\_concat\_row.*append*(*X*[*i*])

X\_concat.*append*(*X\_concat\_row*)

*print*(*X\_concat*[:10])

*def* multiply(A,B): #*Can be scalar-scalar,scalar-vector,scalar-matrix,vector-matrix,matrix-matrix multiply*

#*Position also important in case vector-matrix, matrix-matrix*

#*This function assume that vector is column vector*

if *type*(*A*)==int or *type*(*A*)==float or *type*(*B*)==int or *type*(*B*)==float: #*Check if at least one argument is scalar*

if (*type*(*A*)==int or *type*(*A*)==float) and (*type*(*B*)==int or *type*(*B*)==float): #*Check if it scalar-scalar multiply*

result=A\*B

#*If not scalar-scalar multiply, check another argument. It must be vector or matrix*

elif *type*(*A*)==list: #*Check whether if A is matrix or vector*

if *type*(*A*[0])==list: #*Check whether if A is matrix*

result=[]

for row in *range*(*len*(*A*)):

row\_result=[]

for column in *range*(*len*(*A*)):

row\_result.*append*(*B*\**A*[*row*][*column*])

result.*append*(*row\_result*)

else: #*So A must be vector*

result=[]

for element in A:

result.*append*(*B*\**element*)

else: #*So B must be vector/matrix*

if *type*(*B*[0])==list: #*Check whether if B is matrix*

result=[]

for row in *range*(*len*(*B*)):

row\_result=[]

for column in *range*(*len*(*B*)):

row\_result.*append*(*A*\**B*[*row*][*column*])

result.*append*(*row\_result*)

else: #*So B must be vector*

result=[]

for element in B:

result.*append*(*A*\**element*)

else: #*No scalar argument*

if *type*(*A*[0])==list and *type*(*B*[0])==list: #*Check whether if A is matrix and B is matrix*

if *len*(*A*[0])==*len*(*B*):

result=[]

for row\_A in *range*(*len*(*A*)):

row\_result=[]

for column\_B in *range*(*len*(*B*[0])):

for\_sum\_list=[]

for column\_A in *range*(*len*(*A*[0])):

for\_sum\_list.*append*(*A*[*row\_A*][*column\_A*]\**B*[*column\_A*][*column\_B*])

row\_result.*append*(*sum*(*for\_sum\_list*))

result.*append*(*row\_result*)

else:

*print*('Cannot multiply')

elif *type*(*A*[0])==list and *type*(*B*)==list: #*Check whether if A is matrix and B is vector*

if *len*(*A*[0])==*len*(*B*):

result=[]

for row in *range*(*len*(*A*)):

for\_sum\_list=[]

for column in *range*(*len*(*A*[0])):

for\_sum\_list.*append*(*A*[*row*][*column*]\**B*[*column*])

result.*append*(*sum*(*for\_sum\_list*))

else:

*print*('Cannot multiply')

# *Case A vector and B matrix is possible if A is row vector but I assume that all vector is column vector, so I skip this case*

# *Case A vector and B vector also same reason*

else:

*print*('Maybe something went wrong')

return result

*def* transpose(A):

result=[]

for column in *range*(*len*(*A*[0])):

row\_transpose=[]

for row in *range*(*len*(*A*)):

row\_transpose.*append*(*A*[*row*][*column*])

result.*append*(*row\_transpose*)

return result

*def* det(A):

if *len*(*A*[0])==*len*(*A*): #*Check whether if column and row are equal*

if *len*(*A*)==2:

result=(A[0][0]\*A[1][1])-(A[0][1]\*A[1][0])

else:

for\_sum\_list=[]

for row1 in *range*(*len*(*A*)):

minor=[]

for row2 in *range*(*len*(*A*)):

if row1==row2:

pass

else:

minor.*append*(*A*[*row2*][1:])

for\_sum\_list.*append*(*A*[*row1*][0]\*(-1)\*\*(*row1*+0)\**det*(*minor*))

result=*sum*(*for\_sum\_list*)

else:

*print*('Cannot find determinant')

return result

*def* cofactor(A): #*create cofactor matrix*

if *len*(*A*[0])==*len*(*A*):

result=[]

for row1 in *range*(*len*(*A*)):

cofactor\_row=[]

for column1 in *range*(*len*(*A*[0])): #*row1 and column1 indicate cofactor position*

minor=[]

for row2 in *range*(*len*(*A*)): #*row2 and column2 use to find minor*

if row1==row2:

pass

else:

if column1==0:

minor.*append*(*A*[*row2*][1:])

elif column1==*len*(*A*[0])-1:

minor.*append*(*A*[*row2*][:*len*(*A*[0])-1])

else:

storage=[]

for column2 in *range*(*len*(*A*)):

if column1==column2:

pass

else:

storage.*append*(*A*[*row2*][*column2*])

minor.*append*(*storage*)

cofactor\_row.*append*((-1)\*\*(*row1*+*column1*)\**det*(*minor*))

result.*append*(*cofactor\_row*)

else:

*print*('Cannot find cofactor matrix')

return result

*def* inverse(A):

if *det*(*A*)!=0:

if *len*(*A*)==2:

adjugate=[]

adjugate\_row=[]

adjugate\_row.*append*(*A*[1][1])

adjugate\_row.*append*(-*A*[0][1])

adjugate.*append*(*adjugate\_row*)

adjugate\_row=[]

adjugate\_row.*append*(-*A*[1][0])

adjugate\_row.*append*(*A*[0][0])

adjugate.*append*(*adjugate\_row*)

result=*multiply*(1/*det*(*A*),*adjugate*)

else:

result=*multiply*((1/*det*(*A*)),*transpose*(*cofactor*(*A*)))

else:

*print*('cannot find inverse')

return result

B=*multiply*(*inverse*(*multiply*(*transpose*(*X\_concat*),*X\_concat*)),*multiply*(*transpose*(*X\_concat*),*Y*)) #*B=(XT \* X)^−1 \* XT \* Y where XT is XTranspose and x = [X1,X2] and X1 is always 1*

*print*(*B*)

*print*('Hence we use best formula according to the model to find value of Y which corresponds to given X')

*print*(*f*'Formula y = *{B*[0]*}* + *{B*[1]*}*x')

Output

