(A CHRISTIAN MINORITY INSTITUTION) JAISAKTHI EDUCATIONAL TRUST

BANGALORE TRUNK ROAD, VARADHARAJAPURAM, NASARATHPET, POONAMALLEE, CHENNAI 600 123

**DEPARTMENT OF**



# ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

**AD8412 – DATA ANALYTICS LABORATORY ACADEMIC YEAR: 2021 – 2022 (EVEN SEMESTER)**

Name of the Student :

Register Number :

Roll Number :

Year & Semester :



# REGISTER NUMBER:

Certified that this is a bonafide record of practical work done by

of II Year / III

Semester of B.Tech Artificial Intelligence and Data Science in AD8412 – DATA ANALYTICS LABORATORY during the academic year 2021 - 22.

# STAFF IN-CHARGE HEAD OF THE DEPARTMENT

Submitted for the Anna University practical examination held on at Panimalar Institute of Technology, Chennai – 600 123.

# INTERNAL EXAMINER EXTERNAL EXAMINER

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Expt. No.** | **Name of the Experiment** | **Page No** | **Date** | **Signature** |
| 1 | Demonstration the random sampling |  |  |  |
| 2 | Demonstrate the probability sampling from a known population |  |  |  |
| 3 | Implementation of Z-Test – One Sample Z-Test and Two Sample Z-Test |  |  |  |
| 4 | Implementation of Z-Test – using Titanic case study |  |  |  |
| 5 | Implementation of T-Test – one sample t-test |  |  |  |
| 6 | Implementation of T-Test – Two sample t-test and Paired T-Test |  |  |  |
| 7 | Implementation of VARIANCE ANALYSIS ( ANNOVA) |  |  |  |
| 8 | Demonstration of Linear Regression |  |  |  |
| 9 | Demonstration of Logistic Regression |  |  |  |
| 10 | Demonstration of Multiple-Linear Regression |  |  |  |
| 11 | Implementation of Time Series Analysis |  |  |  |

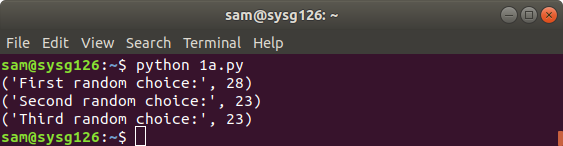
# PROGRAM

**1 A) Picking Random Items in a List using ‘random.choice()’**

bmi\_list = [29, 18, 20, 22, 19, 25, 30, 28,22, 21, 18, 19, 20, 20, 22, 23]

import random

print("First random choice:", random.choice(bmi\_list)) print("Second random choice:", random.choice(bmi\_list)) print("Third random choice:", random.choice(bmi\_list)) **OUTPUT:**

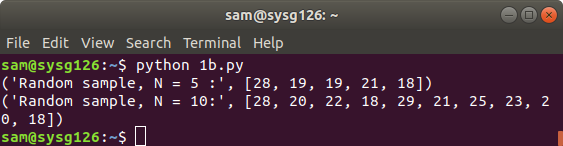


# 1 B) Picking Random Items in a List using ‘random.sample()’

print("Random sample, N = 5 :", random.sample(bmi\_list, 5))

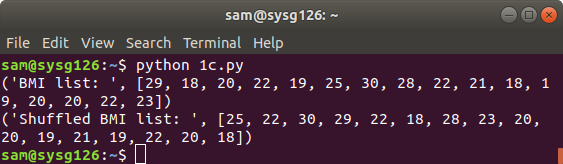
print("Random sample, N = 10:", random.sample(bmi\_list, 10))

# OUTPUT:



print("BMI list: ", bmi\_list) random.shuffle(bmi\_list) print("Shuffled BMI list: ", bmi\_list)

# OUTPUT:



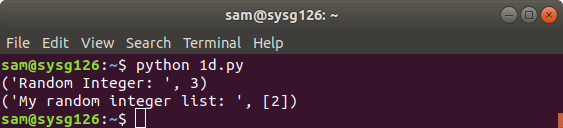
**1 D) Generating Random Integers using ‘random.randint()’** print("Random Integer: ", random.randint(1,5)) random\_ints\_list = []

fori in range(1,50):

n = random.randint(1,5) random\_ints\_list.append(n)

print("My random integer list: ", random\_ints\_list)

# OUTPUT:



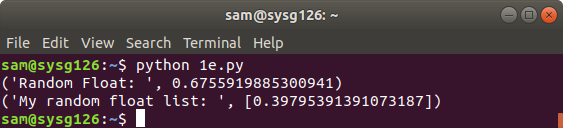
print("Random Float: ", random.random()) random\_float\_list = []

fori in range(1,5):

n = random.random() random\_float\_list.append(n)

print("My random float list: ", random\_float\_list)

# OUTPUT:



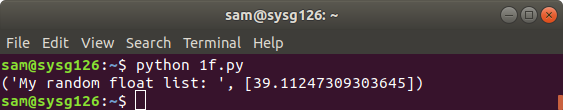
**Scale the random float numbers by multiplying our random number by 500**

random\_float\_list = [] fori in range(1,5):

n = random.random()\*500 random\_float\_list.append(n)

print("My random float list: ", random\_float\_list)

# OUTPUT:



**generate random numbers between 100 and 500**

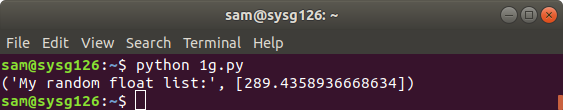
**r**andom\_float\_list = [] fori in range(1,10):

n = random.random()\*500 if n>=100.0:

random\_float\_list.append(n)

print("My random float list: ", random\_float\_list)

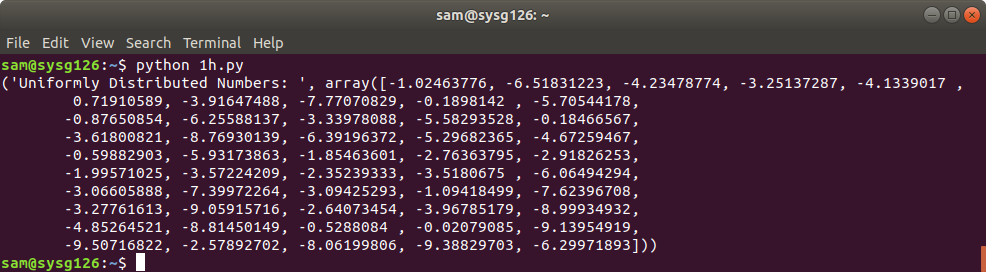
# OUTPUT:



**1F) Computing Uniformly Distributed Numbers with ‘random.uniform()’**

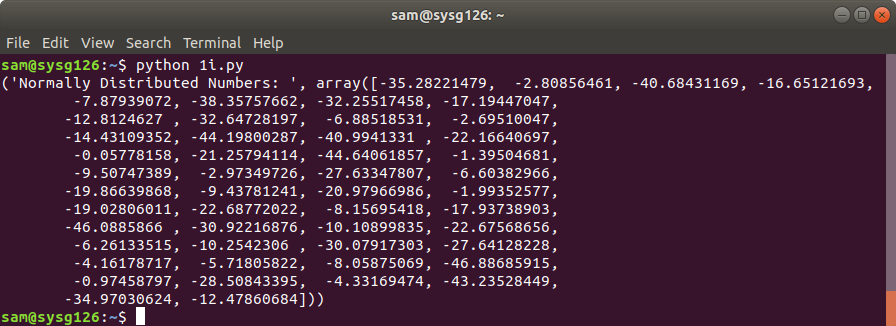
import numpy as np

uniform\_list = np.random.uniform(-10,1,50) print("Uniformly Distributed Numbers: ", uniform\_list) **OUTPUT:**



normal\_list = np.random.uniform(-50,0,50) print("Normally Distributed Numbers: ", normal\_list)

# OUTPUT:



**EX.NO 02**

**PROGRAM**

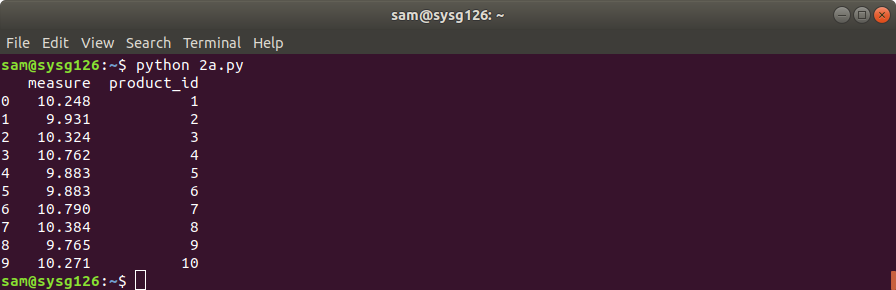
1. **A) Create Sample:**

import numpy as np import pandas as pd np.random.seed(42) number\_of\_products=10

data={'product\_id':np.arange(1,number\_of\_products+1).tolist(),'measure':np.round(np

.random.normal(loc=10,scale=0.5,size=number\_of\_products),3)} df=pd.DataFrame(data) real\_mean=round(df['measure'].mean(),3)

df print(data) **OUTPUT:**



# 2 B) Implement Simple Random Sampling:

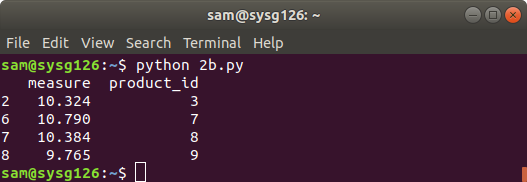
import numpy as np import pandas as pd np.random.seed(42) number\_of\_products=10

data={'product\_id':np.arange(1,number\_of\_products+1).tolist(),'measure':np.round(np

.random.normal(loc=10,scale=0.5,size=number\_of\_products),3)} df=pd.DataFrame(data) simple\_random\_sample=df.sample(n=4).sort\_values(by='product\_id') simple\_random\_mean=round(simple\_random\_sample['measure'].mean(),3) simple\_random\_sample

print(simple\_random\_sample)

# OUTPUT:



import numpy as np import pandas as pd number\_of\_students = 15

data = {'Id': np.arange(1, number\_of\_students+1).tolist(), 'height': [159, 171, 158, 162, 162, 177, 160, 175,

168, 171, 178, 178, 173, 177, 164]}

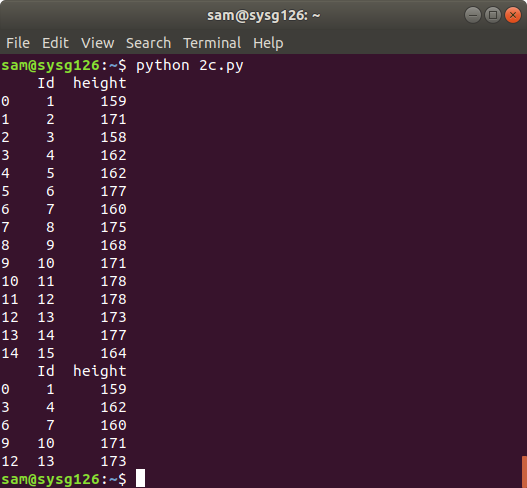
df = pd.DataFrame(data) print(df)

def systematic\_sampling(df, step):

indexes = np.arange(0, len(df), step=step) systematic\_sample = df.iloc[indexes] return systematic\_sample

systematic\_sample = systematic\_sampling(df, 3) print(systematic\_sample)

# OUTPUT:

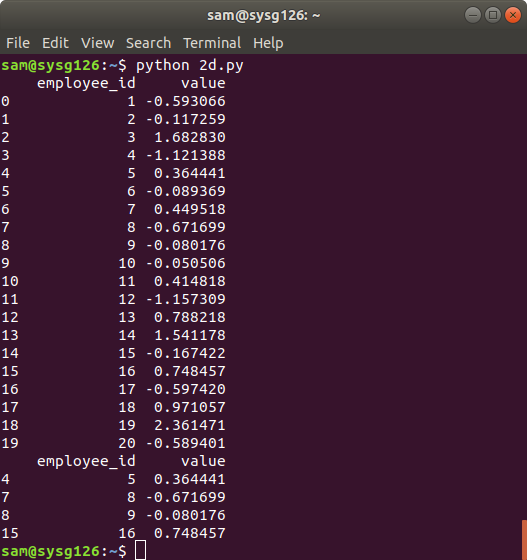


import numpy as np import pandas as pd

dic\_data={"employee\_id":np.arange(1,21),'value':np.random.randn(20)} df=pd.DataFrame(dic\_data)

print(df) samples=df.sample(4).sort\_values(by='employee\_id') print(samples)

# OUTPUT:



import pandas as pd import numpy as np number\_of\_products=6

data = {'product\_id':np.arange(1, number\_of\_products+1).tolist(), 'product\_strata':np.repeat([1,2], number\_of\_products/2).tolist(), 'measure':np.round(np.random.normal(loc=10, scale=0.5, size=number\_of\_products),3)}

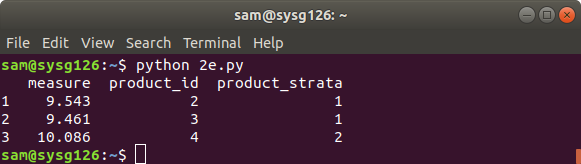
df = pd.DataFrame(data) df

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

split = StratifiedShuffleSplit(n\_splits=1, test\_size=0.4) for x, y in split.split(df, df['product\_strata']):

stratified\_random\_sample = df.iloc[y].sort\_values(by='product\_id') stratified\_random\_sample stratified\_random\_sample.groupby('product\_strata').mean().drop(['product\_id'],axis=1) print(stratified\_random\_sample)

# OUTPUT:



**PROGRAM:**

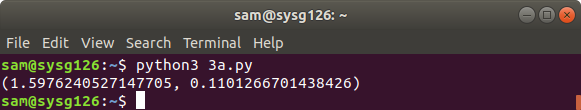
1. **A) One Sample Z-Test in Python:**

from statsmodels.stats.weightstats import ztest as ztest data = [88, 92, 94, 94, 96, 97, 97, 97, 99, 99,

105, 109, 109, 109, 110, 112, 112, 113, 114, 115]

ztest=ztest(data, value=100) print(ztest)

# OUTPUT:



1. **B) Two Sample Z-Test in Python:**

from statsmodels.stats.weightstats import ztest as ztest cityA = [82, 84, 85, 89, 91, 91, 92, 94, 99, 99,

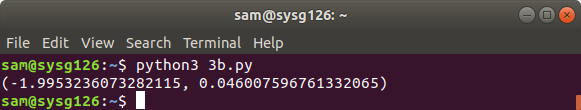
105, 109, 109, 109, 110, 112, 112, 113, 114, 114]

cityB = [90, 91, 91, 91, 95, 95, 99, 99, 108, 109,

109, 114, 115, 116, 117, 117, 128, 129, 130, 133]

ztest=ztest(cityA, cityB, value=0) print(ztest)

# OUTPUT:



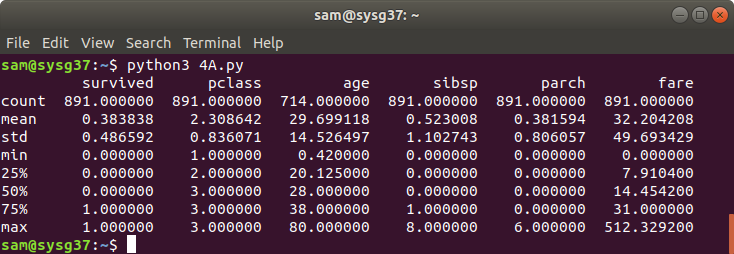
**PROGRAM:**

1. **A) verview Of Data Set:**

import pandas as pd import numpy as np import numpy.random import seaborn as sb

titanic\_trian\_data=sb.load\_dataset('titanic') titanic\_train\_data.describe()

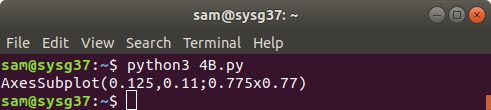
# OUTPUT:



import pandas as pd import numpy as np import seaborn as sb

titanic\_train\_data=sb.load\_dataset('titanic') temp=titanic\_train\_data[titanic\_train\_data['age'].notna()].age print(sb.histplot(temp))

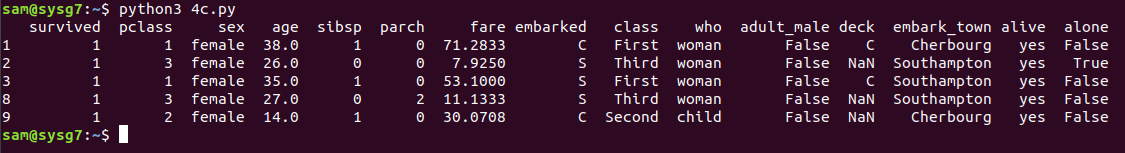
# OUTPUT:



**4 C) Survived Data:**

survived\_data = titanic\_train\_data(titanic\_train\_data['Survived']==1] survived\_data[survived\_data['Age'].notna ()].head()

# OUTPUT:



sample\_list = []

for i in range (60):

sample\_list\_temp = np. random. Choice (survived\_data[survived\_data['Age'] .notna()].Age,60) .mean()

sample\_list.append(sample\_list\_temp) sns.distplot (sample\_list)

# OUTPUT:



**4 E) Z test**

from statsmodels.stats.weightstats import ztest from statsmodels.stats.weightstats import zconfint

HØ

else:

ztest Score, p\_value= ztest (sample\_list, value = mean\_hØ, alternative='larger") print('p\_value’,p\_value)

print('ztest Score’,ztest Score)

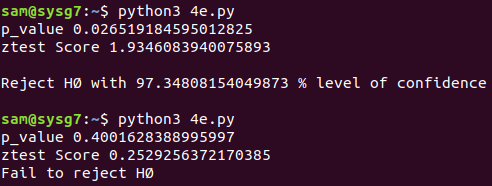
if (p\_value < 0.05): #The smaller the p-value, the stronger the evidence to reject the

print ()

print("Reject HØ with", (1-p\_value) \*100, '% level of confidence'); print ()

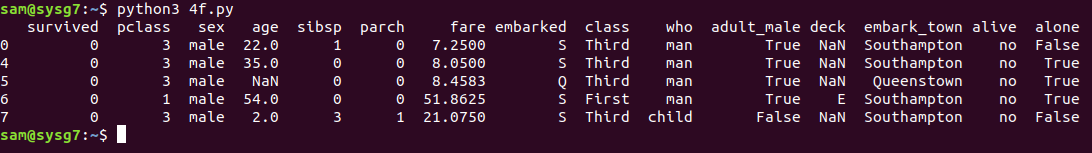
print("Fail to reject HØ"); print()

# OUTPUT:



sns.distplot (survived\_male\_data[survived\_male\_data.Age.notna ()1.Age)

# OUTPUT:



**4 G) Sample List male Using For Range (60)**

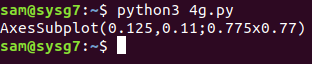
sample\_list\_male = []

for i in range (60):

sample\_list\_temp = np . random. Choice ( survived\_male\_data[survived\_male\_data.Age.notna()]. Age,60). Mean ()

sample\_list\_male.append(sample\_list\_temp) sns.distplot (sample\_list\_male)

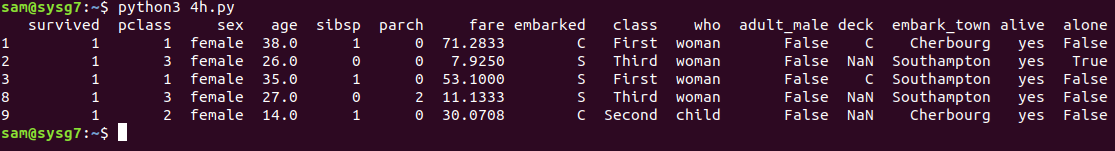
# OUTPUT:



**4 H) survived\_female\_data**

sns.distplot (survived\_female\_data[survived\_female\_data.Age.notna()]. Age)

# OUTPUT:



**4 I) Sample List Female Using For Range (60)**

sample\_list\_female = [ ]

for i in range(60):

sample\_list\_temp = np . random . choice ( survived\_female\_data[survived\_female\_date.Age . notna ()] . Age,60). Mean()

sample\_list\_female.append(sample\_list\_temp) sns.distplot (sample\_list\_female)

# OUTPUT:



**4 J) Z test**

from statsmodels.stats.weightstats import ztest from statsmodels.stats.weightstats import zconfint

Ztest\_Score, p\_value = ztest(x1=sample\_list\_male, x2 sample\_list\_female, value = 0, alternatives two-sided"]

print('p\_value,p\_value) print('ztest Score',ztest Score)

if (p\_value < 0.05): #The smaller the p-value, the stronger the evidence to reject the HØ print()

print("Reject HØ with", (1-p\_value)\*100,' %level of confidence’); print()

else:

print("Fail to reject HØ "); print()

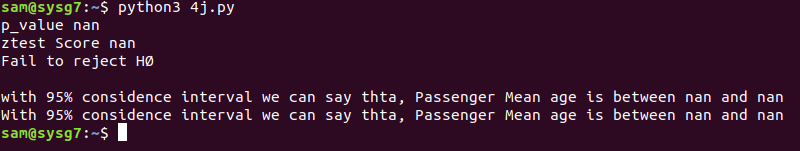
lower, upper = zconfint (x1=sample\_list\_male, value = 0,alpha=0.05, alternative = ‘two- sided")

print("with 95% considence interval we can say thta, Passenger Mean age is between, lower,"and", upper)

lower, upper = zconfint(x1=sample\_list\_female, value = 0,alpha 0.05, alternative="two- sided")

print("With 95% considence interval we can say thta, Passenger Mean age is between", lower, "and", upper)

# OUTPUT:



1. **K) z test**

from statsmodels.stats.weightstats import ztest from statsmodels.stats.weightstats import zconfint

ztest\_score,p\_value=ztest(x1=sample\_list\_male,x2=sample\_list\_female,value=0, alternative=’larger’)

print('p\_value,p\_value) print('ztest Score',ztest Score)

if (p\_value < 0.05): #The smaller the p-value, the stronger the evidence to reject

the HO

else:

print()

print("Reject HØ with", (1-p\_value)\*100,’%level of confidence'); print()

print("Fail to reject HØ’’); print()

lower, upper = zconfint(x1=sample\_list\_male, value = 0,alpha=0.05,

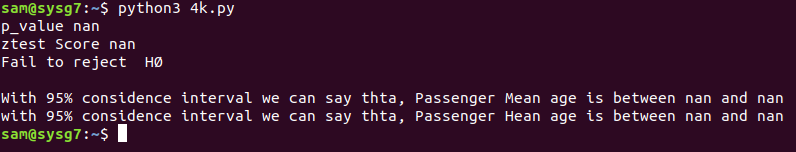
alternative='two-sided')

print("With 95% considence interval we can say thta, Passenger Mean age is between", lower,"and", upper)

lower, upper = zconfint (x1=sample\_list\_female, value = 0,alpha=0.05, alternatives=’ two-sided’)

print("with 95% considence interval we can say thta, Passenger Hean age is between", lower, "and", upper)

# OUTPUT:



p = Ø.5Ø

ztestScore,p\_value=proportions\_ztest(count=153,nobs=survived\_data.Age\_range.count(),val ue=p, alternative ‘large’

print("Below is the calculation using Library") print('p\_value,p\_value)

print('ztest Score’,ztest Score)

if (p\_value < 0.05): #The smaller the p-value, the stronger the evidence to reject the HØ print("Reject HØ with", (1-p\_value)\*100, '%level of confidence');

print() else:

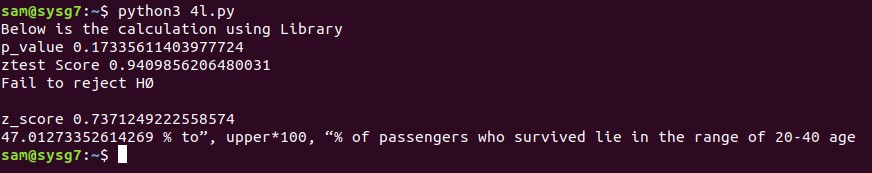
print("Fail to reject HØ "); print()

z\_score= ((153/290)-0.5)/(math.sqrt((0.5+0.5)/714)) print("z\_score,z\_score)

lower,upper=proportion\_confint(count=153,nobs=survived\_data.Age\_range.count(), method="normal)

print (lower \*100, "% to”, upper\*100, “% of passengers who survived lie in the range of 20- 40 age ")

# OUTPUT:



P = Ø.5 Ø

Ztest\_ Score,p\_value= proportions\_ztest (count=385, nobs=titanic\_train\_data.

Age\_range.count(),value=p,alternative=’large

print("Below is the calculation using Library") print('p\_value.p\_value)

print( ztest Scoreztest Score)

if (p\_value < 0.05): #The smaller the p-value, the stronger the evidence to reject the H Ø print("Reject H Ø with’’. (1-p\_value)\*100,’% level of confidence");

print() else:

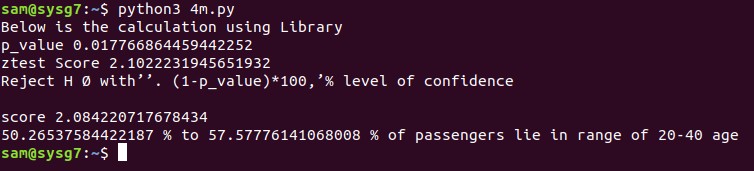
print("Fail to reject H Ø "); print()

z score = (0.539-0.5)/(math.sqrt((0.5\*0.5)/714)) print("score ,z\_score)

Lower, upper = proportion\_confint (count=385, nobs =titanic\_train\_data.Age\_range.count(), method="normal")

print (lower\*100, “% to, upper\*100,’’% of passengers lie in range of 20-40 age ")

# OUTPUT:



**PROGRAM:**

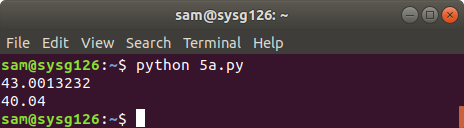
1. **A) ONE-SAMPLE T-TEST:**

import numpy as np from scipy import stats

population\_ages1=stats.poisson.rvs(loc=18,mu=35,size=1500000) population\_ages2=stats.poisson.rvs(loc=18,mu=10,size=1000000) population\_ages=np.concatenate((population\_ages1,population\_ages2)) minnesota\_ages1=stats.poisson.rvs(loc=18,mu=30,size=30) minnesota\_ages2=stats.poisson.rvs(loc=18,mu=10,size=20) minnesota\_ages=np.concatenate((minnesota\_ages1,minnesota\_ages2)) print(population\_ages.mean())

print(minnesota\_ages.mean())

# OUTPUT:

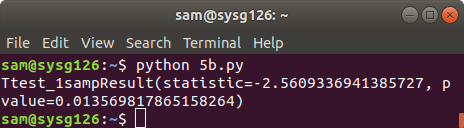


**5 B)**

import numpy as np from scipy import stats

population\_ages1=stats.poisson.rvs(loc=18,mu=35,size=1500000) population\_ages2=stats.poisson.rvs(loc=18,mu=10,size=1000000) population\_ages=np.concatenate((population\_ages1,population\_ages2)) minnesota\_ages1=stats.poisson.rvs(loc=18,mu=30,size=30) minnesota\_ages2=stats.poisson.rvs(loc=18,mu=10,size=20) minnesota\_ages=np.concatenate((minnesota\_ages1,minnesota\_ages2)) sa=stats.ttest\_1samp(a=minnesota\_ages,popmean=population\_ages.mean()) print(sa)

# OUTPUT:



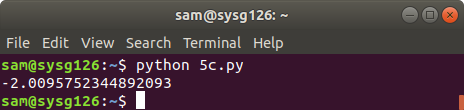
1. **C)**

import numpy as np from scipy import stats

population\_ages1=stats.poisson.rvs(loc=18,mu=35,size=1500000) population\_ages2=stats.poisson.rvs(loc=18,mu=10,size=1000000) population\_ages=np.concatenate((population\_ages1,population\_ages2)) minnesota\_ages1=stats.poisson.rvs(loc=18,mu=30,size=30) minnesota\_ages2=stats.poisson.rvs(loc=18,mu=10,size=20) minnesota\_ages=np.concatenate((minnesota\_ages1,minnesota\_ages2)) sa=stats.t.ppf(q=0.025, df=49)

print(sa)

# OUTPUT:

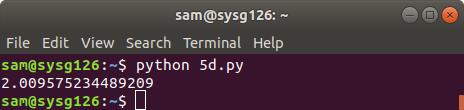


import numpy as np from scipy import stats

population\_ages1=stats.poisson.rvs(loc=18,mu=35,size=1500000) population\_ages2=stats.poisson.rvs(loc=18,mu=10,size=1000000) population\_ages=np.concatenate((population\_ages1,population\_ages2)) minnesota\_ages1=stats.poisson.rvs(loc=18,mu=30,size=30) minnesota\_ages2=stats.poisson.rvs(loc=18,mu=10,size=20) minnesota\_ages=np.concatenate((minnesota\_ages1,minnesota\_ages2)) sa=stats.t.ppf(q=0.975, df=49)

print(sa)

# OUTPUT:

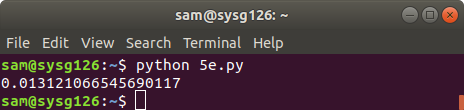


import numpy as np from scipy import stats

population\_ages1=stats.poisson.rvs(loc=18,mu=35,size=1500000) population\_ages2=stats.poisson.rvs(loc=18,mu=10,size=1000000) population\_ages=np.concatenate((population\_ages1,population\_ages2)) minnesota\_ages1=stats.poisson.rvs(loc=18,mu=30,size=30) minnesota\_ages2=stats.poisson.rvs(loc=18,mu=10,size=20) minnesota\_ages=np.concatenate((minnesota\_ages1,minnesota\_ages2)) sa=stats.t.cdf(x=-2.5742, df=49)\*2

print(sa)

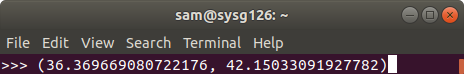
# OUTPUT:



import numpy as np from scipy import stats

population\_ages1=stats.poisson.rvs(loc=18,mu=35,size=1500000) population\_ages2=stats.poisson.rvs(loc=18,mu=10,size=1000000) population\_ages=np.concatenate((population\_ages1,population\_ages2)) minnesota\_ages1=stats.poisson.rvs(loc=18,mu=30,size=30) minnesota\_ages2=stats.poisson.rvs(loc=18,mu=10,size=20) minnesota\_ages=np.concatenate((minnesota\_ages1,minnesota\_ages2)) sa=stats.t.interval(0.95, df=49, loc=minnesota\_ages.mean(), scale=sigma) print(sa)

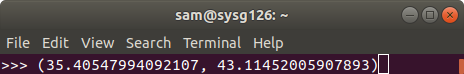
# OUTPUT:



import numpy as np from scipy import stats

population\_ages1=stats.poisson.rvs(loc=18,mu=35,size=1500000) population\_ages2=stats.poisson.rvs(loc=18,mu=10,size=1000000) population\_ages=np.concatenate((population\_ages1,population\_ages2)) minnesota\_ages1=stats.poisson.rvs(loc=18,mu=30,size=30) minnesota\_ages2=stats.poisson.rvs(loc=18,mu=10,size=20) minnesota\_ages=np.concatenate((minnesota\_ages1,minnesota\_ages2)) sa=stats.t.interval(alpha=0.99, df=49, loc=minnesota\_ages.mean(), scale=sigma) print(sa)

# OUTPUT:



**EX.NO 06 PROGRAM**

1. **A) Two-Sample T-Test:**

import pandas as pd import numpy as np from scipy import stats

from scipy.stats import norm import matplotlib.pyplot as plt np.random.seed(12)

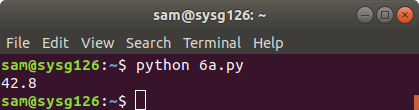
wisconsin\_ages1 = stats.poisson.rvs(loc=18, mu=33, size=30) wisconsin\_ages2 = stats.poisson.rvs(loc=18, mu=13, size=20) wisconsin\_ages = np.concatenate((wisconsin\_ages1, wisconsin\_ages2)) print(wisconsin\_ages.mean() )

stats.ttest\_ind(a= minnesota\_ages, b= wisconsin\_ages,

equal\_var=False) # Assume samples have equal variance?

Ttest\_indResult(statistic=-1.7083870793286842, pvalue=0.09073104343957748)

# OUTPUT:



**6 B) Paired T-Test:**

np.random.seed(11)

before= stats.norm.rvs(scale=30, loc=250, size=100)

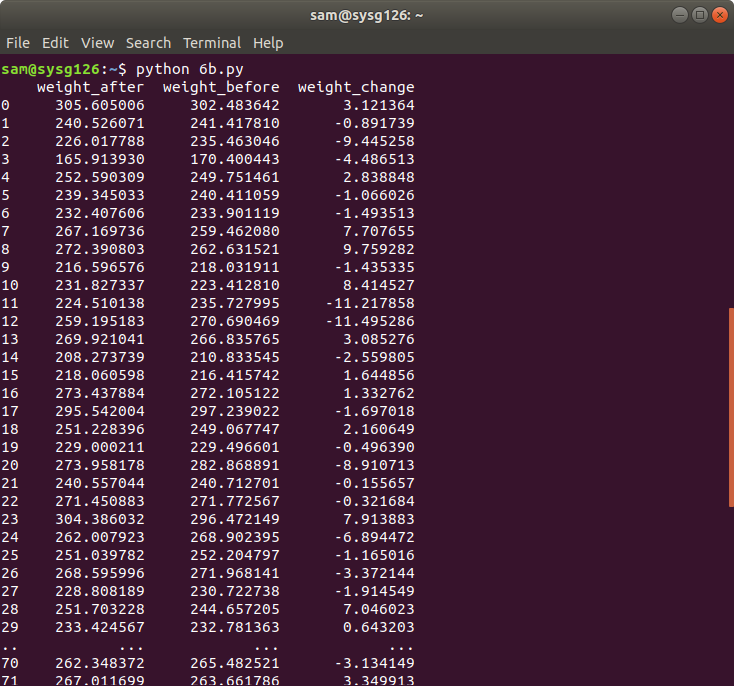
after = before + stats.norm.rvs(scale=5, loc=-1.25, size=100)

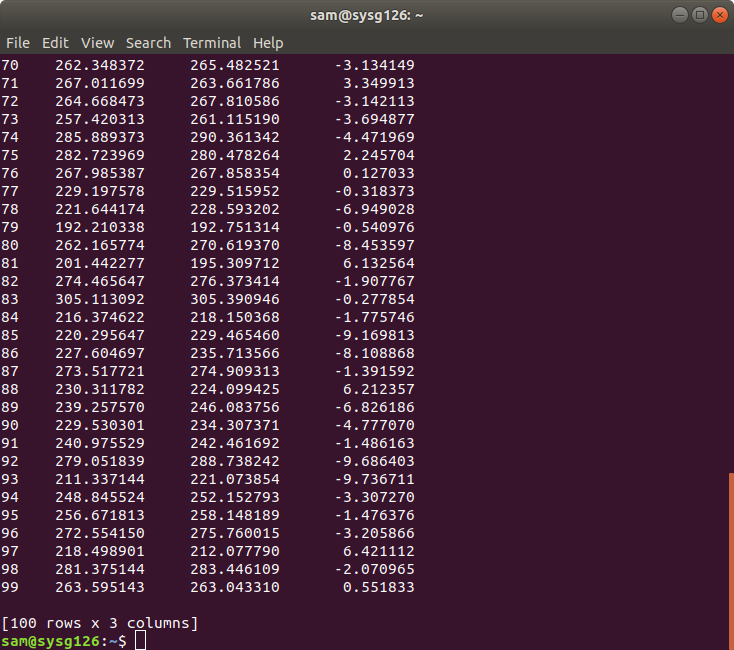
weight\_df = pd.DataFrame({"weight\_before":before, "weight\_after":after, "weight\_change":after-before})

weight\_df.describe()

stats.ttest\_rel(a = before, b = after) Ttest\_relResult(statistic=2.5720175998568284, pvalue=0.011596444318439857)

# OUTPUT:





**6 C) Type I and Type II Error:**

import pandas as pd import numpy as np from scipy import stats

from scipy.stats import norm import matplotlib.pyplot as plt

plt.figure(figsize=(12,10))

plt.fill\_between(x=np.arange(-4,-2,0.01),y1= stats.norm.pdf(np.arange(-4,-2,0.01))

,facecolor='red',alpha=0.35)

plt.fill\_between(x=np.arange(-2,2,0.01),y1= stats.norm.pdf(np.arange(-2,2,0.01))

,facecolor='grey',alpha=0.35)

plt.fill\_between(x=np.arange(2,4,0.01),y1= stats.norm.pdf(np.arange(2,4,0.01))

,facecolor='red',alpha=0.5)

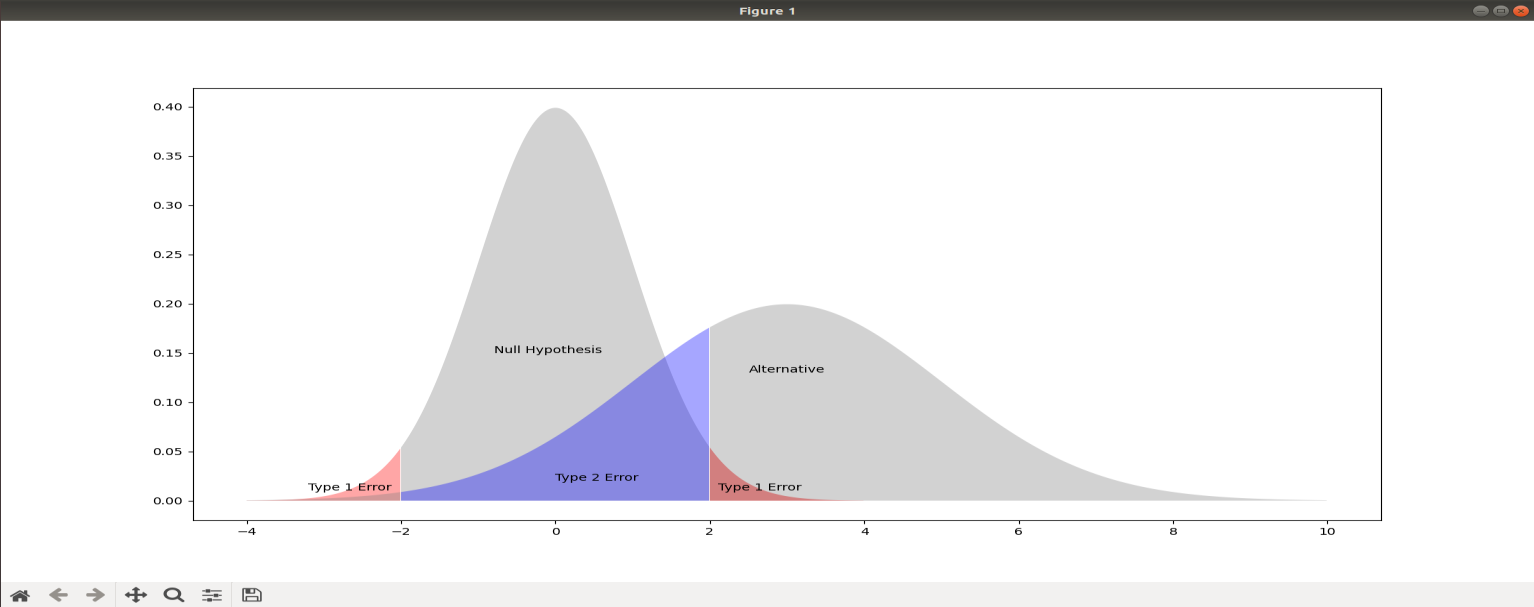
plt.fill\_between(x=np.arange(-4,-2,0.01),y1= stats.norm.pdf(np.arange(-4,-2,0.01),loc=3, scale=2) ,facecolor='grey',alpha=0.35)

plt.fill\_between(x=np.arange(-2,2,0.01),y1= stats.norm.pdf(np.arange(-2,2,0.01),loc=3, scale=2) ,facecolor='blue',alpha=0.35)

plt.fill\_between(x=np.arange(2,10,0.01),y1= stats.norm.pdf(np.arange(2,10,0.01),loc=3, scale=2),facecolor='grey',alpha=0.35)

plt.text(x=-0.8, y=0.15, s= "Null Hypothesis") plt.text(x=2.5, y=0.13, s= "Alternative") plt.text(x=2.1, y=0.01, s= "Type 1 Error") plt.text(x=-3.2, y=0.01, s= "Type 1 Error") plt.text(x=0, y=0.02, s= "Type 2 Error") plt.show()

# OUTPUT:



**PROGRAM:**

**i**mport pandas as pd import numpy as np import scipy.stats as stats a=[25,25,27,30,23,20]

b=[30,30,21,24,26,28]

c=[18,30,29,29,24,26]

list\_of\_tuples = list(zip(a, b,c))

df = pd.DataFrame(list\_of\_tuples, columns = ['A', 'B', 'C']) df

m1=np.mean(a) m2=np.mean(b) m3=np.mean(c)

print('Average mark for college A: {}'.format(m1)) print('Average mark for college B: {}'.format(m2)) print('Average mark for college C: {}'.format(m3)) m=(m1+m2+m3)/3

print('Overall mean: {}'.format(m)) SSb=6\*((m1-m)\*\*2+(m2-m)\*\*2+(m3-m)\*\*2)

print('Between-groups Sum of Squared Differences: {}'.format(SSb))

MSb=SSb/2

print('Between-groups Mean Square value: {}'.format(MSb)) err\_a=list(a-m1)

err\_b=list(b-m2) err\_c=list(c-m3) err=err\_a+err\_b+err\_c ssw=[]

for i in err: ssw.append(i\*\*2)

SSw=np.sum(ssw)

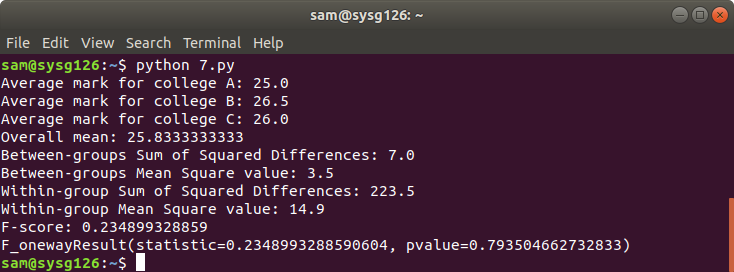
print('Within-group Sum of Squared Differences: {}'.format(SSw))

MSw=SSw/15

print('Within-group Mean Square value: {}'.format(MSw))

F=MSb/MSw

print('F-score: {}'.format(F)) print(stats.f\_oneway(a,b,c))



# PROGRAM:

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y): n = np.size(x)

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

plt.scatter(x, y, color = "m",marker = "o", s = 30) y\_pred = b[0] + b[1]\*x

plt.plot(x, y\_pred, color = "g") plt.xlabel('x')

plt.ylabel('y') plt.show()

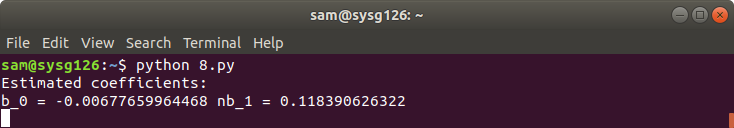
x = np.array([25, 23, 25, 31, 32, 25, 36, 27, 28, 29])

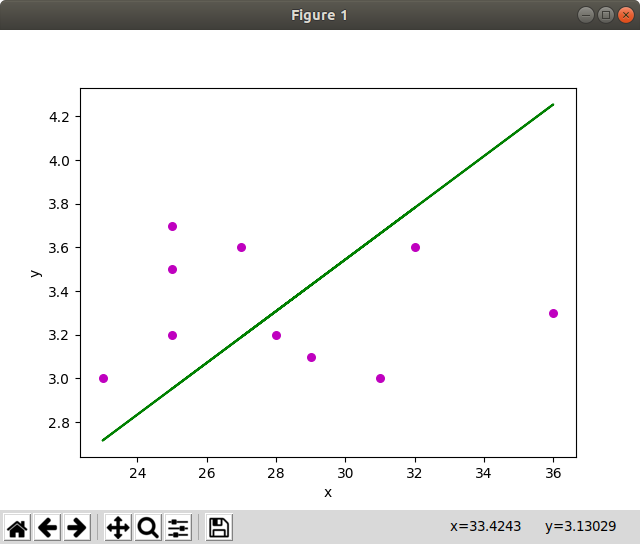
y = np.array([3.2, 3, 3.5, 3, 3.6, 3.7, 3.3, 3.6, 3.2, 3.1])

b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {} nb\_1 = {}".format(b[0], b[1])) plot\_regression\_line(x, y, b)

# OUTPUT:





**PROGRAM**

import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression from sklearn import metrics

import seaborn as sn

import matplotlib.pyplot as plt

candidates =

{'gmat':[780,750,690,710,680,730,690,720,740,690,610,690,710,680,770,610,580,650,540,5 90,620,600,550,550,570,670,660,580,650,660,640,620,660,660,680,650,670,580,590,690],

'gpa':[4,3.9,3.3,3.7,3.9,3.7,2.3,3.3,3.3,1.7,2.7,3.7,3.7,3.3,3.3,3,2.7,3.7,2.7,2.3,3.3,2,2.3,2.7,3,3

.3,3.7,2.3,3.7,3.3,3,2.7,4,3.3,3.3,2.3,2.7,3.3,1.7,3.7],

'work\_experience':[3,4,3,5,4,6,1,4,5,1,3,5,6,4,3,1,4,6,2,3,2,1,4,1,2,6,4,2,6,5,1,2,4,6,5,1,2,1,4,5

], 'admitted':[1,1,0,1,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,0,1,0,0,0,0,1,1,0,1,1,0,0,1,1,1,0,0,0,0,1]}

df = pd.DataFrame(candidates,columns= ['gmat', 'gpa','work\_experience','admitted']) print(df)

X = df[['gmat', 'gpa','work\_experience']] y = df['admitted']

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

print (y\_train)

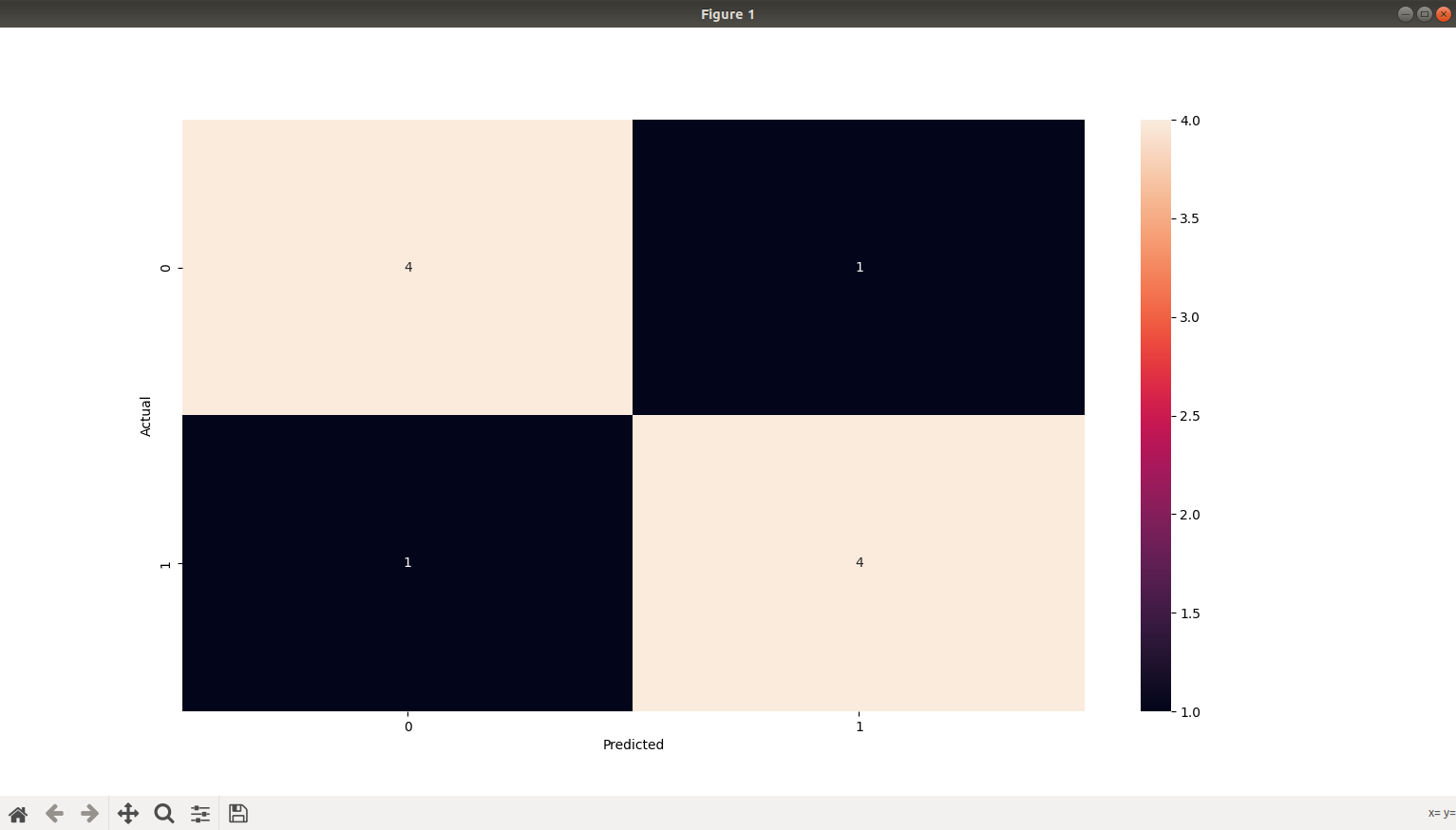
logistic\_regression= LogisticRegression() logistic\_regression.fit(X\_train,y\_train) y\_pred=logistic\_regression.predict(X\_test)

confusion\_matrix = pd.crosstab(y\_test, y\_pred, rownames=['Actual'], colnames=['Predicted']) sn.heatmap(confusion\_matrix, annot=True)

print('Accuracy: ',metrics.accuracy\_score(y\_test, y\_pred)) print (X\_test)

print (y\_pred)

print('confusion\_matrix:', confusion\_matrix, sep ='\n', end ='\n\n') plt.show()



# PROGRAM:

import matplotlib.pyplot as plt import numpy as np

from sklearn import datasets, linear\_model, metrics boston = datasets.load\_boston(return\_X\_y=False)

X = boston.data y = boston.target

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) reg = linear\_model.LinearRegression()

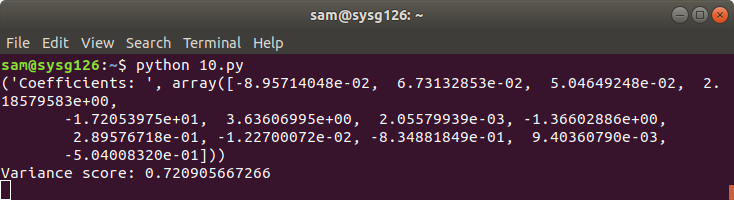
reg.fit(X\_train, y\_train) print('Coefficients: ', reg.coef\_)

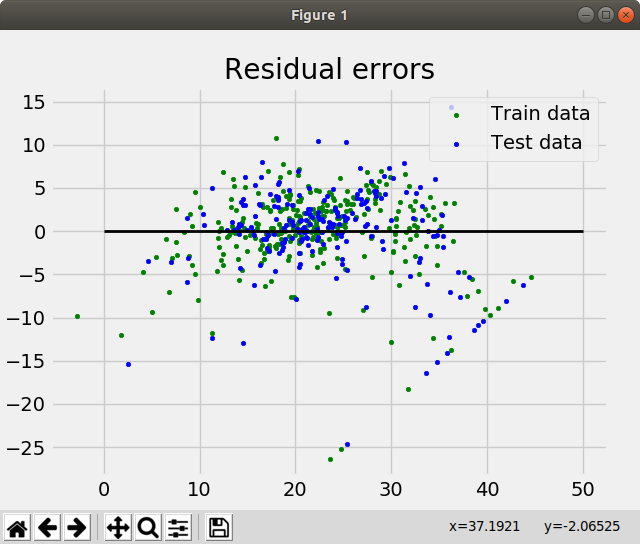
print('Variance score: {}'.format(reg.score(X\_test, y\_test))) plt.style.use('fivethirtyeight')

plt.scatter(reg.predict(X\_train), reg.predict(X\_train) - y\_train, color = "green", s = 10, label = 'Train data') plt.scatter(reg.predict(X\_test), reg.predict(X\_test) - y\_test, color = "blue", s = 10, label = 'Test data')

plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2) plt.legend(loc = 'upper right')

plt.title("Residual errors") plt.show()





# PROGRAM 11 A)

from dateutil.parser import parse import matplotlib as mpl

import matplotlib.pyplot as plt import seaborn as sns

import numpy as np import pandas as pd

plt.rcParams.update({'figure.figsize': (10, 7), 'figure.dpi': 120})

df = pd.read\_csv('https://raw.githubusercontent.com/selva86/datasets/master/a10.csv', parse\_dates=['date'])

df.head() df =

pd.read\_csv('https://raw.githubusercontent.com/selva86/datasets/master/MarketArrivals.csv') df = df.loc[df.market=='MUMBAI', :]

df.head().

import matplotlib.pyplot as plt

df = pd.read\_csv('https://raw.githubusercontent.com/selva86/datasets/master/a10.csv', parse\_dates=['date'], index\_col='date')

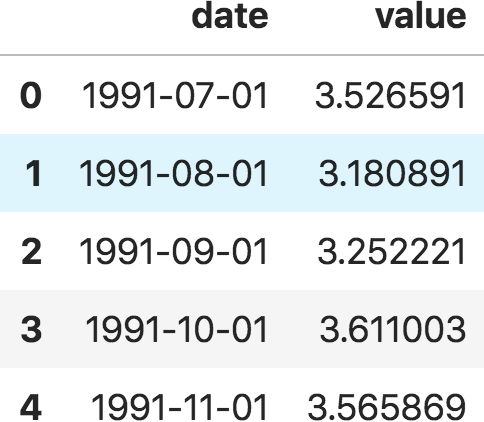
def plot\_df(df, x, y, title="", xlabel='Date', ylabel='Value', dpi=100): plt.figure(figsize=(16,5), dpi=dpi)

plt.plot(x, y, color='tab:red')

plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel) plt.show()

plot\_df(df, x=df.index, y=df.value, title='Monthly anti-diabetic drug sales in Australia from 1992 to 2008.')

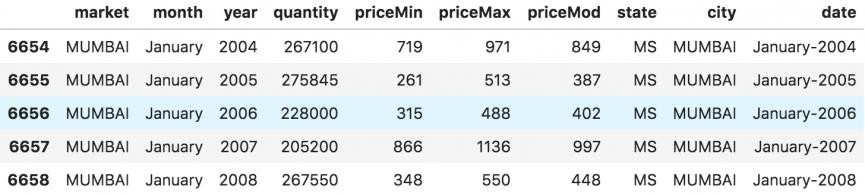
# OUTPUT



df = pd.read\_csv('<https://raw.githubusercontent.com/selva86/datasets/master/MarketArrivals.csv>') df = df.loc[df.market=='MUMBAI', :]

df.head()

# OUTPUT:



**11 C)**

import matplotlib.pyplot as plt

df = pd.read\_csv('<https://raw.githubusercontent.com/selva86/datasets/master/a10.csv>', parse\_dates=['date'], index\_col='date')

def plot\_df(df, x, y, title="", xlabel='Date', ylabel='Value', dpi=100): plt.figure(figsize=(16,5), dpi=dpi)

plt.plot(x, y, color='tab:red')

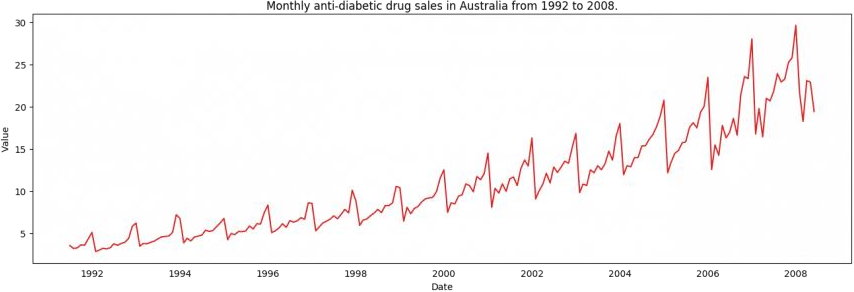
plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel) plt.show()

plot\_df(df, x=df.index, y=df.value, title='Monthly anti-diabetic drug sales in Australia from 1992 to 2008.')

df = pd.read\_csv('datasets/AirPassengers.csv', parse\_dates=['date']) x = df['date'].values

y1 = df['value'].values

# OUTPUT:



**11 D) PLOT**

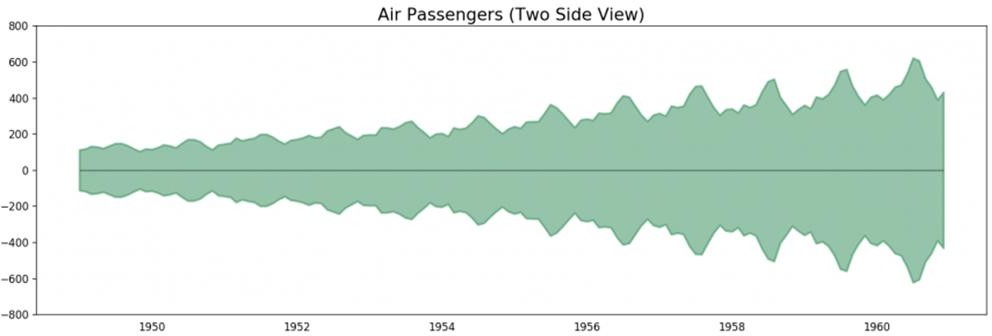
fig, ax = plt.subplots(1, 1, figsize=(16,5), dpi= 120)

plt.fill\_between(x, y1=y1, y2=-y1, alpha=0.5, linewidth=2, color='seagreen') plt.ylim(-800, 800)

plt.title('Air Passengers (Two Side View)', fontsize=16)

plt.hlines(y=0, xmin=np.min(df.date), xmax=np.max(df.date), linewidth=.5) plt.show()

# OUTPUT:



**11 E)**

df = pd.read\_csv('<https://raw.githubusercontent.com/selva86/datasets/master/a10.csv>', parse\_dates=['date'], index\_col='date')

df.reset\_index(inplace=True) df['year'] = [d.year for d in df.date]

df['month'] = [d.strftime('%b') for d in df.date] years = df['year'].unique() np.random.seed(100)

mycolors = np.random.choice(list(mpl.colors.XKCD\_COLORS.keys()), len(years), replace=False)

plt.figure(figsize=(16,12), dpi= 80) for i, y in enumerate(years):

if i > 0:

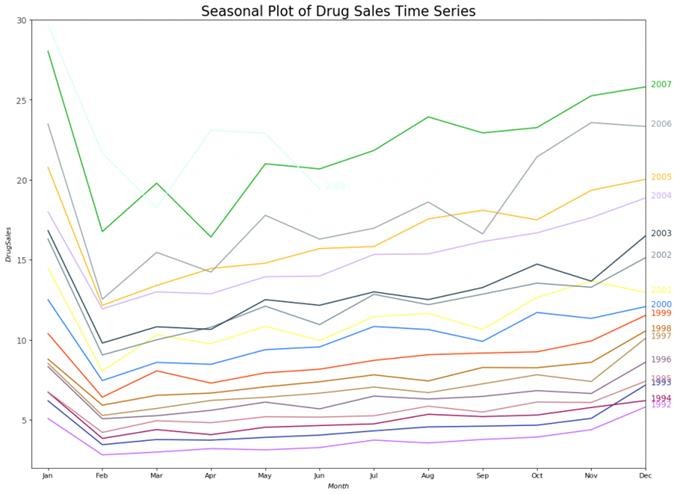
plt.plot('month', 'value', data=df.loc[df.year==y, :], color=mycolors[i], label=y) plt.text(df.loc[df.year==y, :].shape[0]-.9, df.loc[df.year==y, 'value'][-1:].values[0], y,

fontsize=12, color=mycolors[i])

plt.gca().set(xlim=(-0.3, 11), ylim=(2, 30), ylabel='$Drug Sales$', xlabel='$Month$') plt.yticks(fontsize=12, alpha=.7)

plt.title("Seasonal Plot of Drug Sales Time Series", fontsize=20) plt.show()

# OUTPUT:



**11 F)**

df = pd.read\_csv('<https://raw.githubusercontent.com/selva86/datasets/master/a10.csv>', parse\_dates=['date'], index\_col='date')

df.reset\_index(inplace=True) df['year'] = [d.year for d in df.date]

df['month'] = [d.strftime('%b') for d in df.date] years = df['year'].unique()

fig, axes = plt.subplots(1, 2, figsize=(20,7), dpi= 80) sns.boxplot(x='year', y='value', data=df, ax=axes[0])

sns.boxplot(x='month', y='value', data=df.loc[~df.year.isin([1991, 2008]), :]) axes[0].set\_title('Year-wise Box Plot\n(The Trend)', fontsize=18); axes[1].set\_title('Month-wise Box Plot\n(The Seasonality)', fontsize=18) plt.show()

# OUTPUT:

