**RESEARCHING IN COMPUTING JOURNAL**

**NAME:-**

**ROLL NO:-**

**CLASS:- M.Sc. IT PART 1 (SEM 1)**

**COLLEGE NAME:-MULUND COLLEGE OF COMMERCE**

**SUBJECT:- RESEARCHING IN COMPUTING PRACTICALS**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Practical No** | | | **Name of the Practical** |
| **1** | **A** | | Write a program for obtaining descriptive  statistics of data. |
| **B** | | Import data from different data sources (from Excel, csv, mysql, sql server, oracle to  R/Python/Excel) |
| **2** | **A** | | Design a survey form for a given case study, collect the primary data and analyse it |
| **B** | | Perform suitable analysis of given secondary  data. |
| **3** | **A** | | Perform testing of hypothesis using one sample  t-test. |
| **B** | | Perform testing of hypothesis using two sample  t-test |
| **C** | | Perform testing of hypothesis using paired t-test. |
| **4** | **A** | | Perform testing of hypothesis using chi-squared  goodness-of-fit test. |
| **B** | | Perform testing of hypothesis using chi-squared  Test of Independence |
| **5** |  | | Perform testing of hypothesis using Z-test. |
| **6** | **A** | | Perform testing of hypothesis using one-way ANOVA |
| **B** | | Perform testing of hypothesis using two-way  ANOVA. |
| **C** | | Perform testing of hypothesis using multivariate  ANOVA (MANOVA). |
| **7** | **A** | | Perform the Random sampling for the given  data and analyse it. |
| **B** | | Perform the Stratified sampling for the given  data and analyse it. |
| **8** | | | Compute different types of correlation. |
| **9** | | **A** | Perform linear regression for prediction. |
| **B** | Perform polynomial regression for prediction. |
| **10** | | **A** | Perform multiple linear regression. |
| **B** | Perform Logistic. regression |

**Practical 1:**

1. **Write a program for obtaining descriptive statistics of data.**

**Code-**

import pandas as pd

#Create a Dictionary of series

d = {'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]), 'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#Create a DataFrame df = pd.DataFrame(d)

print(df) print('############ Sum ########## ')

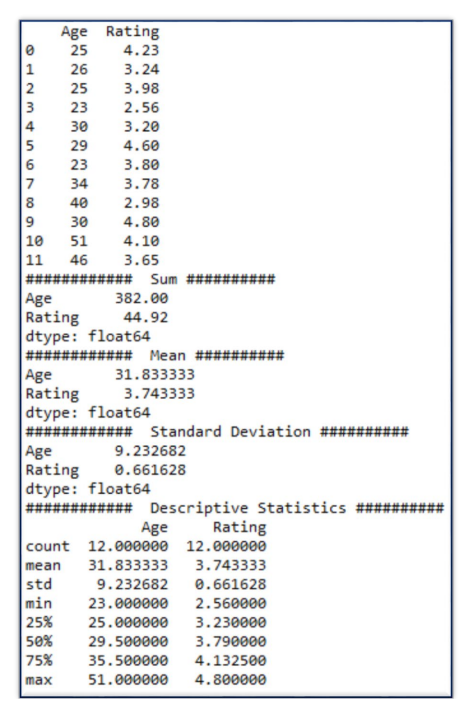
print (df.sum())

print('############ Mean ########## ')

print (df.mean())

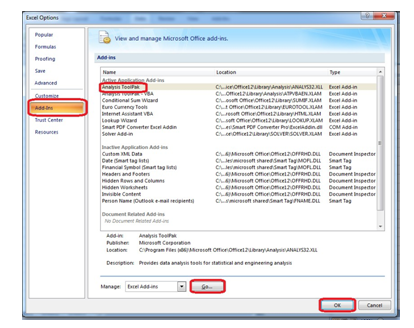
print('############ Standard Deviation ########## ') print (df.std()) print('############ Descriptive Statistics ########## ') print (df.describe()

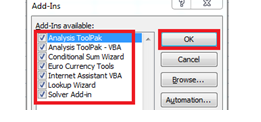
**Output-**

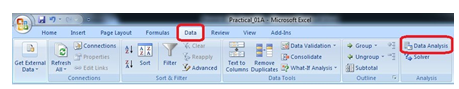
****

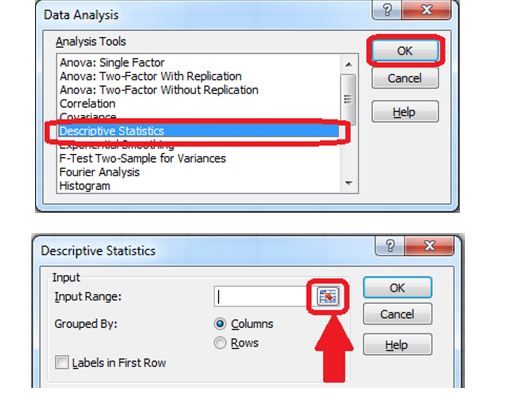
#### Using Excel

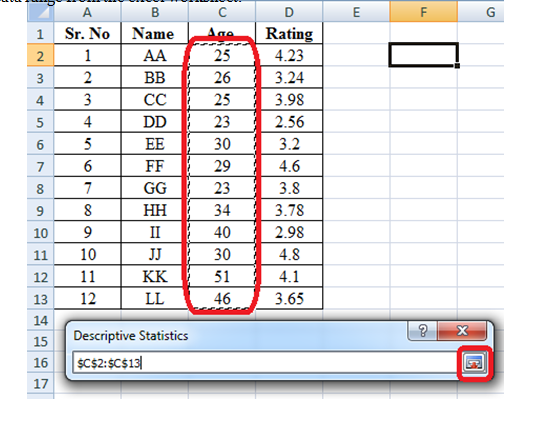
Go to File Menu  Options  Add-Ins Select Analysis ToolPak Press OK

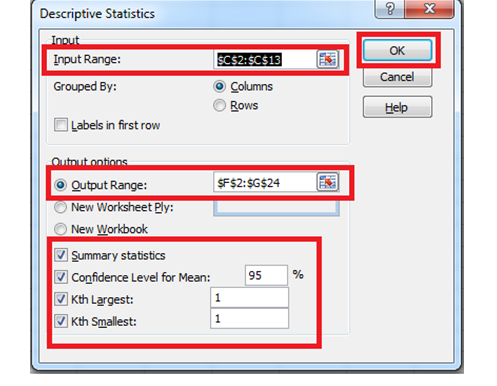




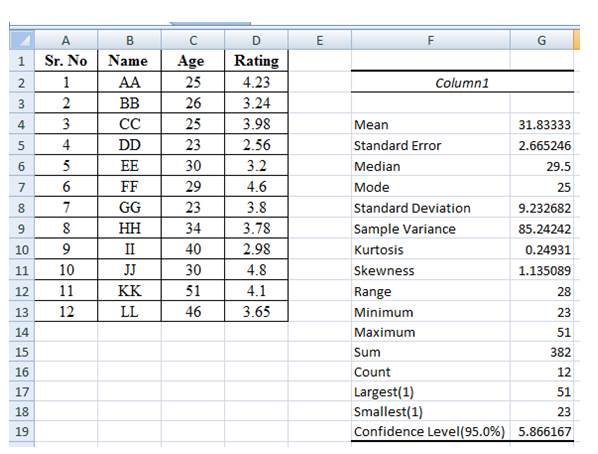


****

****

****

**Output-**

****

## Import data from different data sources (from Excel, csv, mysql, sql server, oracle to R/Python/Excel)

SQLite:

import sqlite3 as sq import pandas as pd

################################################################

Base='C:/VKHCG'

sDatabaseName=Base + '/01-Vermeulen/00-RawData/SQLite/vermeulen.db' conn = sq.connect(sDatabaseName) ################################################################

sFileName='C:/VKHCG/01-Vermeulen/01-Retrieve/01-EDS/02-Python/Retrieve\_IP\_DATA.csv' print('Loading :',sFileName) IP\_DATA\_ALL\_FIX=pd.read\_csv(sFileName,header=0,low\_memory=False) IP\_DATA\_ALL\_FIX.index.names = ['RowIDCSV']

sTable='IP\_DATA\_ALL'

print('Storing :',sDatabaseName,' Table:',sTable) IP\_DATA\_ALL\_FIX.to\_sql(sTable, conn, if\_exists="replace")

print('Loading :',sDatabaseName,' Table:',sTable) TestData=pd.read\_sql\_query("select \* from IP\_DATA\_ALL;", conn) print('################')

print('## Data Values')

print('################')

print(TestData)

print('################')

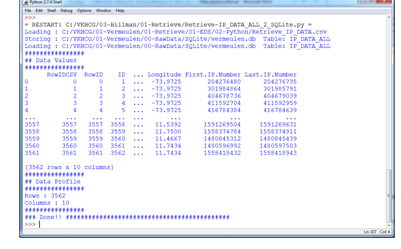
print('## Data Profile')

print('################')

print('Rows :',TestData.shape[0])

print('Columns :',TestData.shape[1]) print('################')

print('### Done!! ############################################')

****

**MySQL:**

Open MySql

Create a database “DataScience”

Create a python file and add the following code:

################ Connection With MySQL ######################

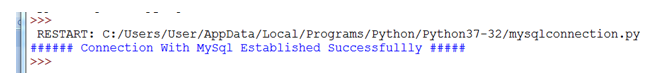
importmysql.connector

conn = mysql.connector.connect(host='localhost', database='DataScience',

user='root', password='root') conn.connect if(conn.is\_connected):

print('###### Connection With MySql Established Successfullly ##### ') else:

print('Not Connected -- Check Connection Properites')



Microsoft Excel

##################Retrieve-Country-Currency.py ################################################################

# -\*- coding: utf-8 -\*- ################################################################

importos

import pandas as pd

################################################################

Base='C:/VKHCG'

################################################################

sFileDir=Base + '/01-Vermeulen/01-Retrieve/01-EDS/02-Python'

#if not os.path.exists(sFileDir):

#os.makedirs(sFileDir)

################################################################

CurrencyRawData = pd.read\_excel('C:/VKHCG/01-Vermeulen/00-RawData/Country\_Currency.xlsx')

sColumns = ['Country or territory', 'Currency', 'ISO-4217']

CurrencyData = CurrencyRawData[sColumns]

CurrencyData.rename(columns={'Country or territory': 'Country', 'ISO-4217':

'CurrencyCode'}, inplace=True)

CurrencyData.dropna(subset=['Currency'],inplace=True)

CurrencyData['Country'] = CurrencyData['Country'].map(lambda x: x.strip())

CurrencyData['Currency'] = CurrencyData['Currency'].map(lambda x: x.strip())

CurrencyData['CurrencyCode'] = CurrencyData['CurrencyCode'].map(lambda x: x.strip())

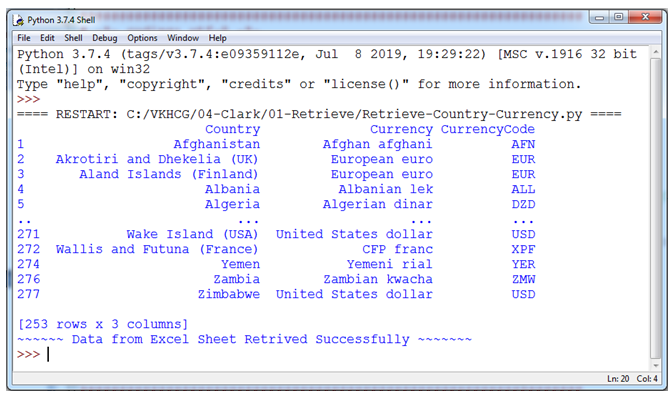
print(CurrencyData)

print('~~~~~~ Data from Excel Sheet Retrived Successfully ~~~~~~~ ')

sFileName=sFileDir + '/Retrieve-Country-Currency.csv'

CurrencyData.to\_csv(sFileName, index = False)

**OUTPUT:**

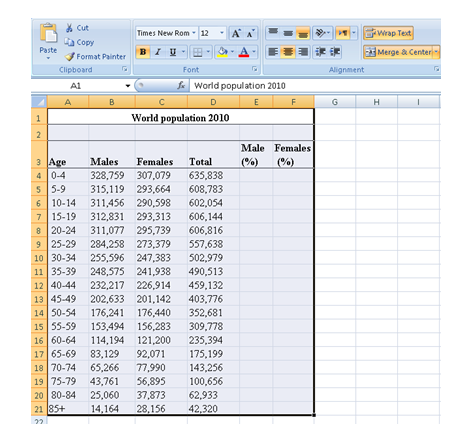


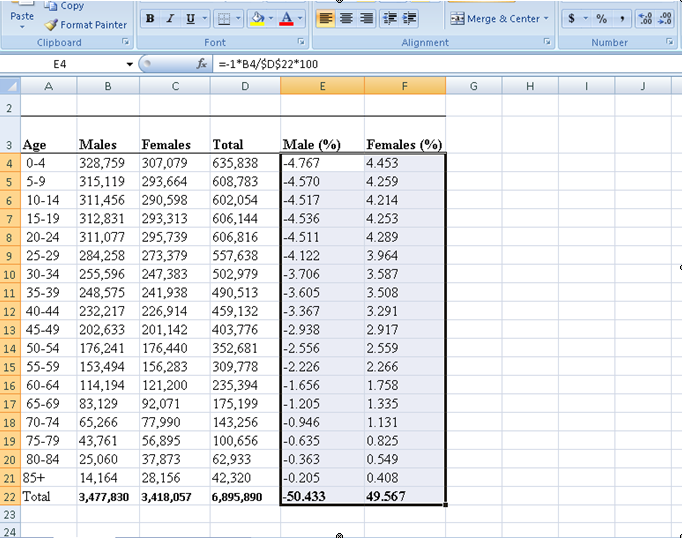
# Practical 2:

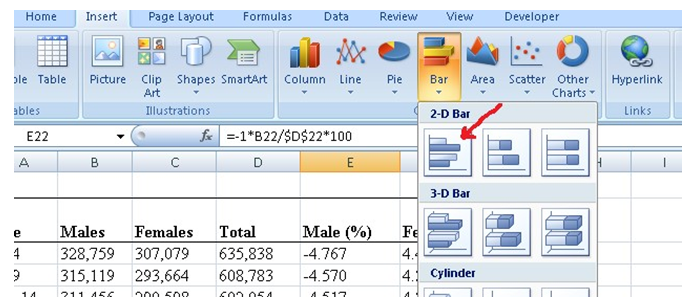
## Design a survey form for a given case study, collect the primary data and analyse it.

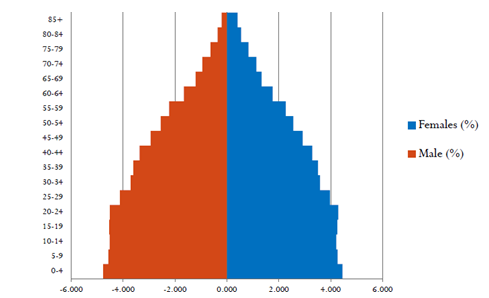
## Perform analysis of given secondary data.

### Steps in Secondary Data Analysis









# Practical 3:

## Perform testing of hypothesis using one sample t-test.

#### Program Code:

# -\*- coding: utf-8 -\*- """

Created on Mon Dec 16 18:01:46 2019 @author: Ahtesham Shaikh

"""

fromscipy.stats import ttest\_1samp importnumpy as np

ages = np.genfromtxt('ages.csv') print(ages)

ages\_mean = np.mean(ages) print(ages\_mean)

tset, pval = ttest\_1samp(ages, 30) print('p-values - ',pval)

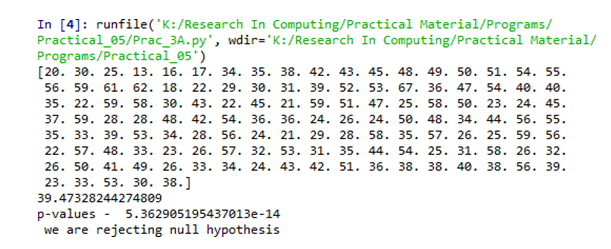
if pval< 0.05: # alpha value is 0.05

print(" we are rejecting null hypothesis")

else:

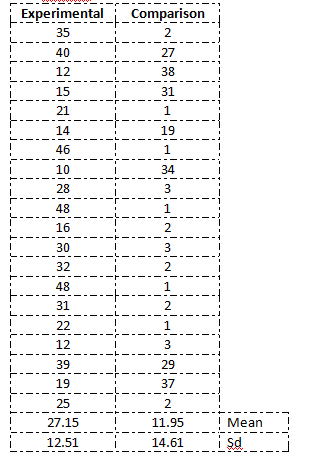
print("we are accepting null hypothesis")

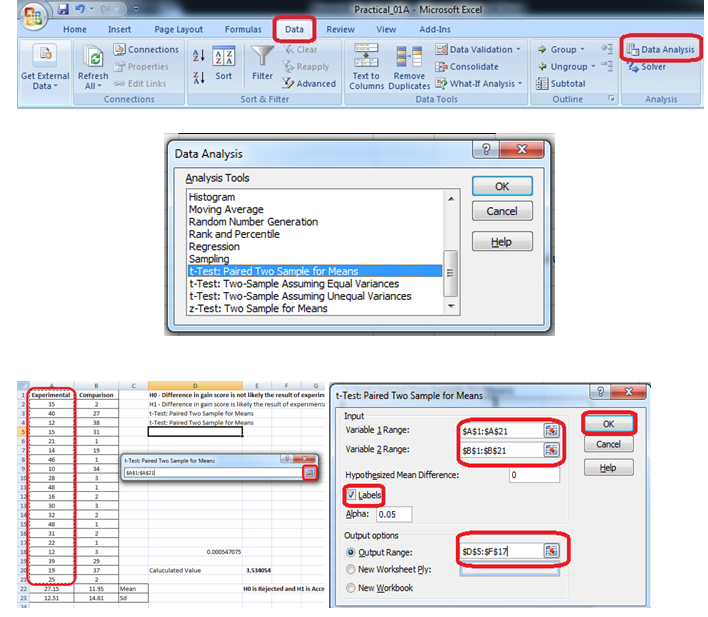
#### Output:



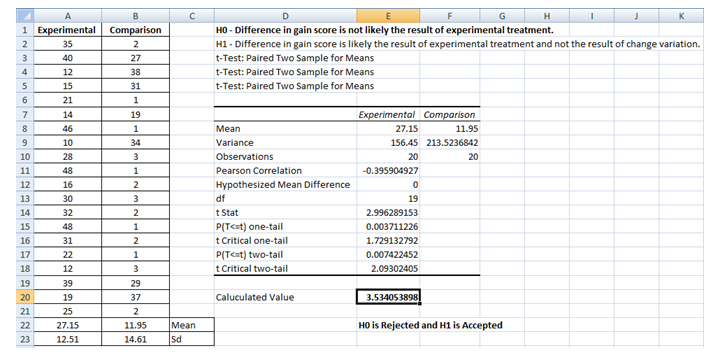
## Write a program for t-test comparing two means for independent samples.

**Two Sample t Test**

****

****

#### Output:



#### Using Python

importnumpy as np fromscipy import stats

fromnumpy.random import randn N = 20

#a = [35,40,12,15,21,14,46,10,28,48,16,30, 32,48,31,22,12,39,19,25]

#b = [2,27,31,38,1,19,1,34,3,1,2,1,3,1,2,1,3,29,37,2]

a = 5 \* randn(100) + 50 b = 5 \* randn(100) + 51 var\_a = a.var(ddof=1) var\_b = b.var(ddof=1)

s = np.sqrt((var\_a + var\_b)/2)

t = (a.mean() - b.mean())/(s\*np.sqrt(2/N))

df = 2\*N - 2

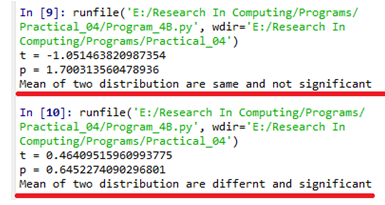
#p-value after comparison with the t p = 1 - stats.t.cdf(t,df=df)

print("t = " + str(t)) print("p = " + str(2\*p)) if t> p :

print('Mean of two distribution are differnt and significant') else:

print('Mean of two distribution are same and not significant')

#### Output:

****

## Perform testing of hypothesis using paired t-test.

#### Program Code:

from scipy import stats

import matplotlib.pyplot as plt

import pandas as pd

df = pd.read\_csv("blood\_pressure.csv")

print(df[['bp\_before','bp\_after']].describe())

#First let’s check for any significant outliers in

#each of the variables.

df[['bp\_before', 'bp\_after']].plot(kind='box') # This saves the plot as a png file

plt.savefig('boxplot\_outliers.png')

# make a histogram to differences between the two scores.

df['bp\_difference'] = df['bp\_before'] - df['bp\_after']

df['bp\_difference'].plot(kind='hist', title= 'Blood Pressure Difference Histogram') #Again, this saves the plot as a png file

plt.savefig('blood pressure difference histogram.png')

stats.probplot(df['bp\_difference'], plot= plt)

plt.title('Blood pressure Difference Q-Q Plot')

plt.savefig('blood pressure difference qq plot.png')

stats.shapiro(df['bp\_difference'])

stats.ttest\_rel(df['bp\_before'], df['bp\_after'])

#### Output:

#### 

# Practical 4:

## Perform testing of hypothesis using chi-squared goodness- of-fit test.

H0 : The population distribution of the variable is the same as the proposed distribution HA : The distributions are different

To calculate the Chi –Squred value for Windows go to cell D2 and type =((B2- C2)\*(B2-C2))/C2

To calculate the Chi –Squred value for Mac go to cell D3 and type =((B3-C3)\*(B3- C3))/C3

To calculate the Chi –Squred value for Mac go to cell D3 and type =((B4-C4)\*(B4- C4))/C4

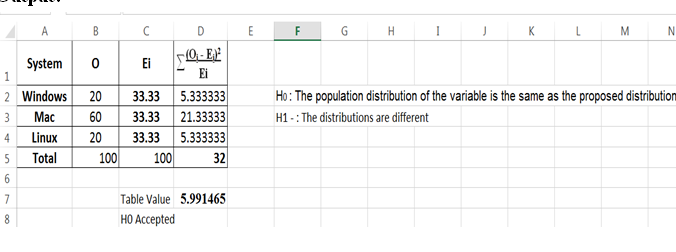
Go to Cell D5 for and type=SUM(D2:D4)

To get the table value for Chi-Square for α = 0.05 and dof = 2, go to cell D7 and type

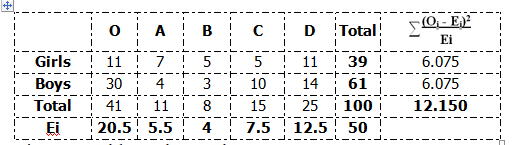
=CHIINV(0.05,2)

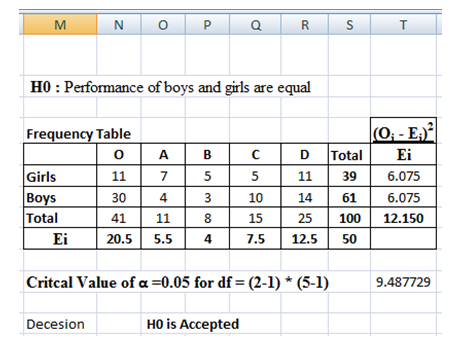
At cell D8 type =IF(D5>D7, "H0 Accepted","H0 Rejected")

#### Output:



## Perform testing of hypothesis using chi-squared test of independence.





#### Using Python

importnumpy as np

import pandas as pd

importscipy.stats as stats

np.random.seed(10)

stud\_grade = np.random.choice(a=["O","A","B","C","D"],

p=[0.20, 0.20 ,0.20, 0.20, 0.20], size=100)

stud\_gen = np.random.choice(a=["Male","Female"], p=[0.5, 0.5], size=100)

mscpart1 = pd.DataFrame({"Grades":stud\_grade, "Gender":stud\_gen})

print(mscpart1)

stud\_tab = pd.crosstab(mscpart1.Grades, mscpart1.Gender, margins=True)

stud\_tab.columns = ["Male", "Female", "row\_totals"]

stud\_tab.index = ["O", "A", "B", "C", "D", "col\_totals"]

observed = stud\_tab.iloc[0:5, 0:2 ]

print(observed)

expected = np.outer(stud\_tab["row\_totals"][0:5], stud\_tab.loc["col\_totals"][0:2]) / 100 print(expected)

chi\_squared\_stat = (((observed-expected)\*\*2)/expected).sum().sum()

print('Calculated : ',chi\_squared\_stat)

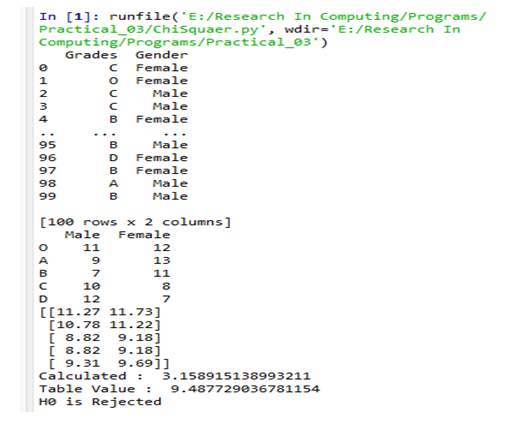
crit = stats.chi2.ppf(q=0.95, df=4)

print('Table Value : ',crit)

ifchi\_squared\_stat>= crit: print('H0 is Accepted ') else:

print('H0 is Rejected ')

**Output:**

****

# Practical 5:

## Perform testing of hypothesis using Z-test.

### Program Code for one-sample Z test.

from statsmodels.stats

import weightstats as stests

import pandas as pd

from scipy import stats

df = pd.read\_csv("blood\_pressure.csv")

df[['bp\_before','bp\_after']].describe() print(df)

ztest ,pval = stests.ztest(df['bp\_before'], x2=None, value=156)

print(float(pval))

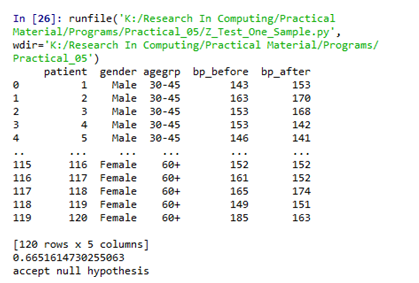
if pval<0.05:

print("reject null hypothesis")

else:

print("accept null hypothesis")

#### Output:



**Two-sample Z test**

import pandas as pd

from statsmodels.stats import weightstats as stests

df = pd.read\_csv("blood\_pressure.csv")

df[['bp\_before','bp\_after']].describe()

print(df)

ztest ,pval = stests.ztest(df['bp\_before'], x2=df['bp\_after'], value=0,alternative='two- sided')

print(float(pval))

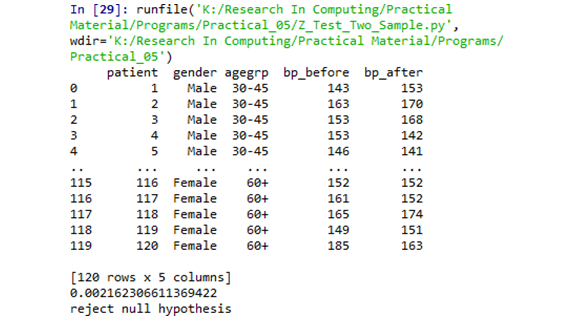
if pval<0.05:

print("reject null hypothesis")

else:

print("accept null hypothesis")

**Output:**

****

# Practical 6:

## Perform testing of hypothesis using One-way ANOVA.

**Code:**

import pandas as pd

importnumpy as np

importmatplotlib.pyplot as plt

importseaborn as sns fromscipy

import stats

data = pd.read\_csv("scores.csv")

data.head()

data['Borough'].value\_counts()

#### ############### There is no total score column, have to create it.

**########In addition, find the mean score of the each district across all schools.**

data['total\_score'] = data['Average Score (SAT Reading)'] + \ data['Average Score (SAT Math)'] + \

data['Average Score (SAT Writing)']

data = data[['Borough', 'total\_score']].dropna()

x = ['Brooklyn', 'Bronx', 'Manhattan', 'Queens', 'Staten Island'] district\_dict = {}

#Assigns each test score series to a dictionary key for district in x:

district\_dict[district] = data[data['Borough'] == district]['total\_score']

y = []

yerror = []

#Assigns the mean score and 95% confidence limit to each district for district in x:

y.append(district\_dict[district].mean()) yerror.append(1.96\*district\_dict[district].std()/np.sqrt(district\_dict[district].shape[0])) print(district + '\_std : {}'.format(district\_dict[district].std()))

sns.set(font\_scale=1.8)

fig = plt.figure(figsize=(10,5))

ax = sns.barplot(x, y, yerr=yerror) ax.set\_ylabel('Average Total SAT Score') plt.show()

#### ###################### Perform 1-way ANOVA

print(stats.f\_oneway(

district\_dict['Brooklyn'], district\_dict['Bronx'], \ district\_dict['Manhattan'], district\_dict['Queens'], \ district\_dict['Staten Island']

))

districts = ['Brooklyn', 'Bronx', 'Manhattan', 'Queens', 'Staten Island'] ss\_b = 0

for d in districts:

ss\_b += district\_dict[d].shape[0] \* \ np.sum((district\_dict[d].mean() - data['total\_score'].mean())\*\*2)

ss\_w = 0

for d in districts:

ss\_w += np.sum((district\_dict[d] - district\_dict[d].mean())\*\*2)

msb = ss\_b/4

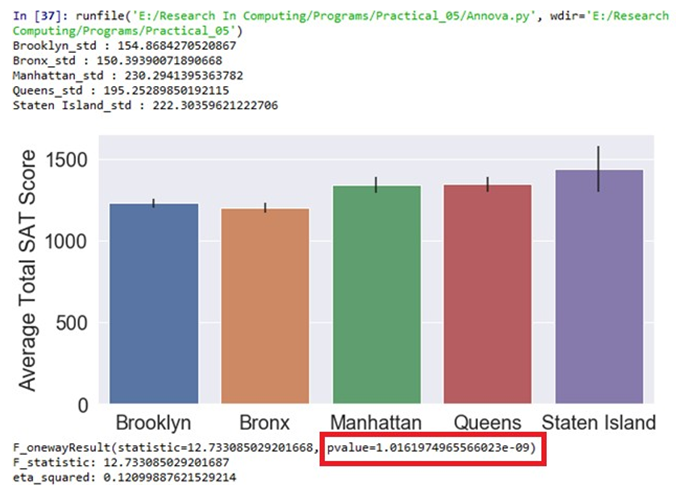
msw = ss\_w/(len(data)-5) f=msb/msw

print('F\_statistic: {}'.format(f))

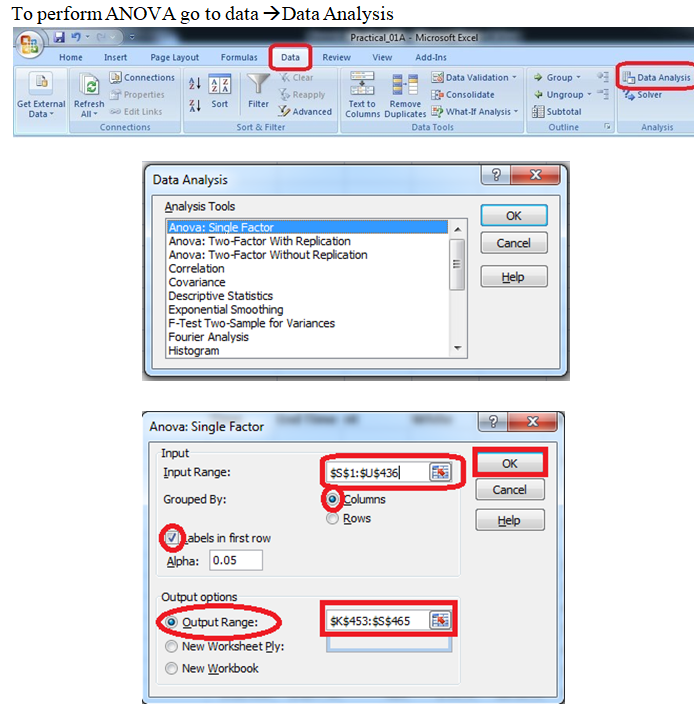
ss\_t = np.sum((data['total\_score']-data['total\_score'].mean())\*\*2) eta\_squared = ss\_b/ss\_t

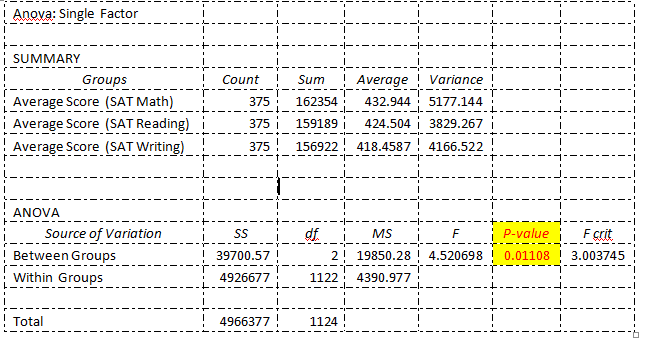
print('eta\_squared: {}'.format(eta\_squared))

#### Output:

****

# Using Excel

****

****

**B.Perform testing of hypothesis using Two-way ANOVA.**

#### Program Code:

import pandas as pd

import statsmodels.api as sm

from statsmodels.formula.api import ols

from statsmodels.stats.anova import anova\_lm

from statsmodels.graphics.factorplots import interaction\_plot import matplotlib.pyplot as plt

from scipy import stats

def eta\_squared(aov): aov['eta\_sq'] = 'NaN'

aov['eta\_sq'] = aov[:-1]['sum\_sq']/sum(aov['sum\_sq']) return aov

def omega\_squared(aov):

mse = aov['sum\_sq'][-1]/aov['df'][-1] aov['omega\_sq'] = 'NaN'

aov['omega\_sq'] = (aov[:-1]['sum\_sq']-(aov[:-

1]['df']\*mse))/(sum(aov['sum\_sq'])+mse) return aov

datafile = "ToothGrowth.csv" data = pd.read\_csv(datafile)

fig = interaction\_plot(data.dose, data.supp, data.len, colors=['red','blue'], markers=['D','^'], ms=10)

N = len(data.len)

df\_a = len(data.supp.unique()) - 1 df\_b = len(data.dose.unique()) - 1 df\_axb = df\_a\*df\_b

df\_w = N - (len(data.supp.unique())\*len(data.dose.unique())) grand\_mean = data['len'].mean()

#Sum of Squares A – supp

ssq\_a = sum([(data[data.supp ==l].len.mean()-grand\_mean)\*\*2 for l in data.supp]) #Sum of Squares B – supp

ssq\_b = sum([(data[data.dose ==l].len.mean()-grand\_mean)\*\*2 for l in data.dose]) #Sum of Squares Total

ssq\_t = sum((data.len - grand\_mean)\*\*2) vc = data[data.supp == 'VC']

oj = data[data.supp == 'OJ']

vc\_dose\_means = [vc[vc.dose == d].len.mean() for d in vc.dose]

oj\_dose\_means = [oj[oj.dose == d].len.mean() for d in oj.dose]

ssq\_w = sum((oj.len - oj\_dose\_means)\*\*2) +sum((vc.len - vc\_dose\_means)\*\*2) ssq\_axb = ssq\_t-ssq\_a-ssq\_b-ssq\_w

ms\_a = ssq\_a/df\_a #Mean Square A ms\_b = ssq\_b/df\_b #Mean Square B

ms\_axb = ssq\_axb/df\_axb #Mean Square AXB ms\_w = ssq\_w/df\_w

f\_a = ms\_a/ms\_w f\_b = ms\_b/ms\_w

f\_axb = ms\_axb/ms\_w

p\_a = stats.f.sf(f\_a, df\_a, df\_w) p\_b = stats.f.sf(f\_b, df\_b, df\_w)

p\_axb = stats.f.sf(f\_axb, df\_axb, df\_w)

results = {'sum\_sq':[ssq\_a, ssq\_b, ssq\_axb, ssq\_w], 'df':[df\_a, df\_b, df\_axb, df\_w],

'F':[f\_a, f\_b, f\_axb, 'NaN'],

'PR(>F)':[p\_a, p\_b, p\_axb, 'NaN']}

columns=['sum\_sq', 'df', 'F', 'PR(>F)']

aov\_table1 = pd.DataFrame(results, columns=columns, index=['supp', 'dose',

'supp:dose', 'Residual'])

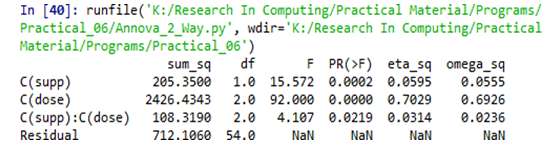
formula = 'len ~ C(supp) + C(dose) + C(supp):C(dose)' model = ols(formula, data).fit()

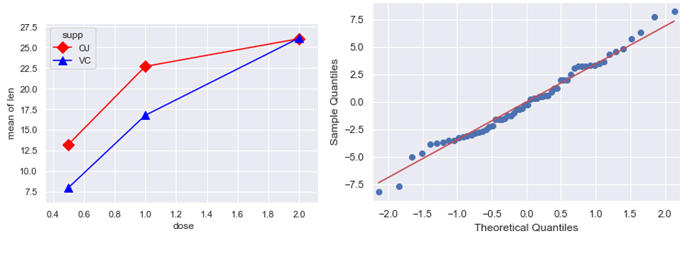
aov\_table = anova\_lm(model, typ=2) eta\_squared(aov\_table) omega\_squared(aov\_table) print(aov\_table.round(4))

res = model.resid

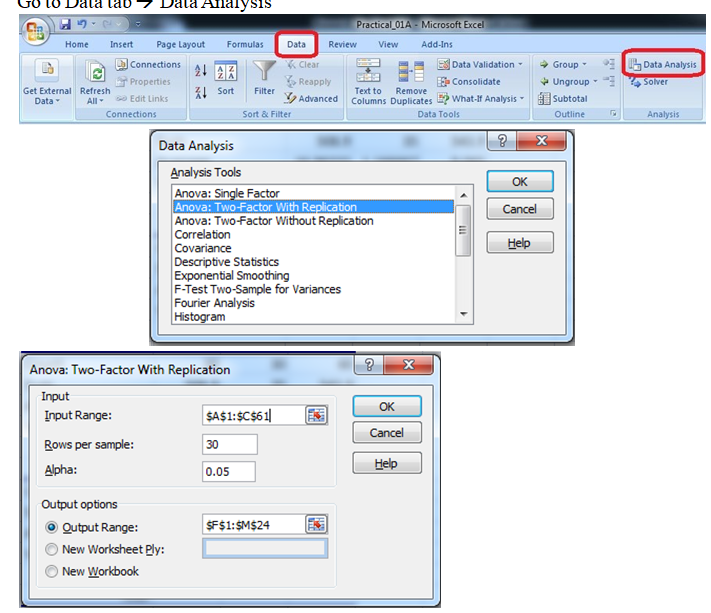
fig = sm.qqplot(res, line='s') plt.show()

## Output:

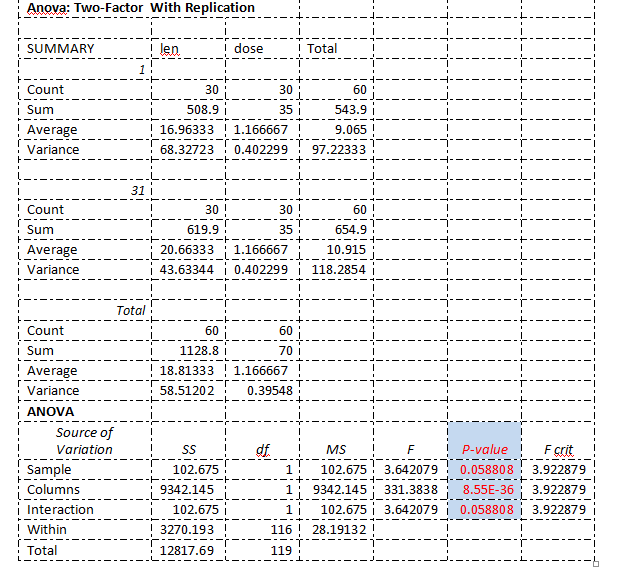




## Using Excel:



## Output:



## C.Perform testing of hypothesis using MANOVA.

#### Code:

import pandas as pd

fromstatsmodels.multivariate.manova import MANOVA

df = pd.read\_csv('iris.csv', index\_col=0)

df.columns = df.columns.str.replace(".", "\_")

df.head()

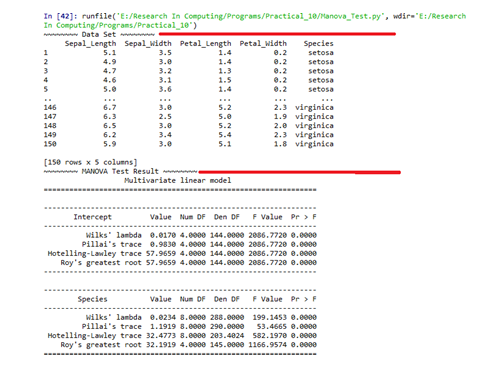
print('~~~~~~~~ Data Set ~~~~~~~~')

print(df)

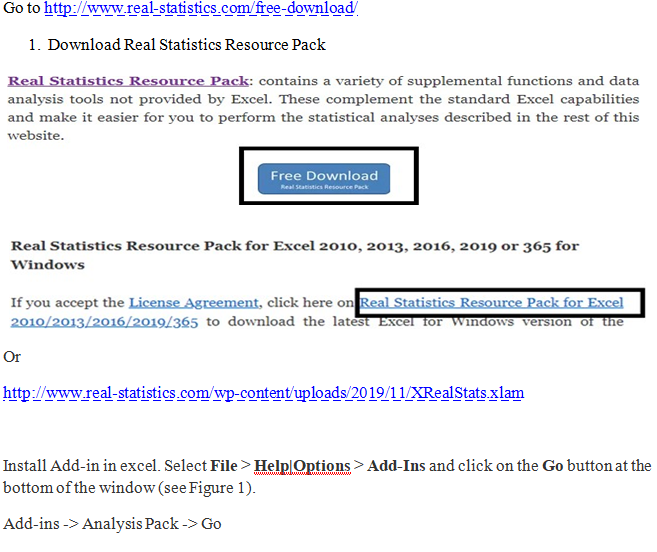
maov = MANOVA.from\_formula('Sepal\_Length + Sepal\_Width + \ Petal\_Length + Petal\_Width ~ Species', data=df)

print('~~~~~~~~ MANOVA Test Result ~~~~~~~~') print(maov.mv\_test())

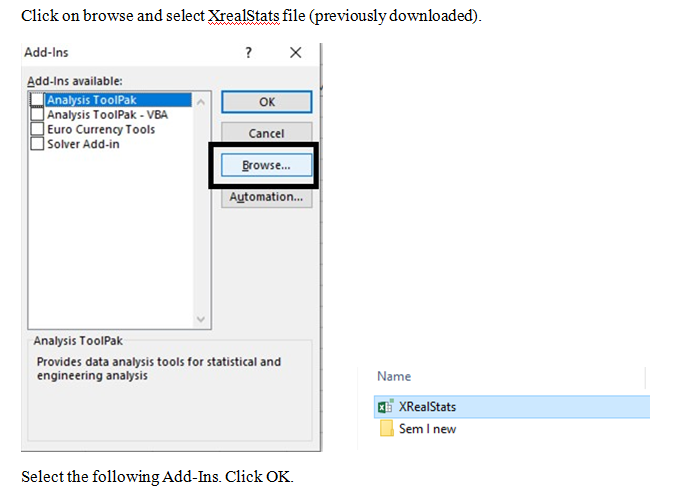
**Output:**

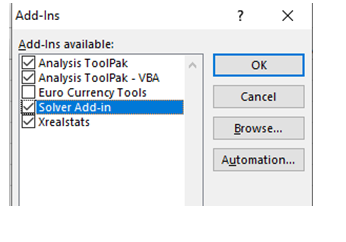


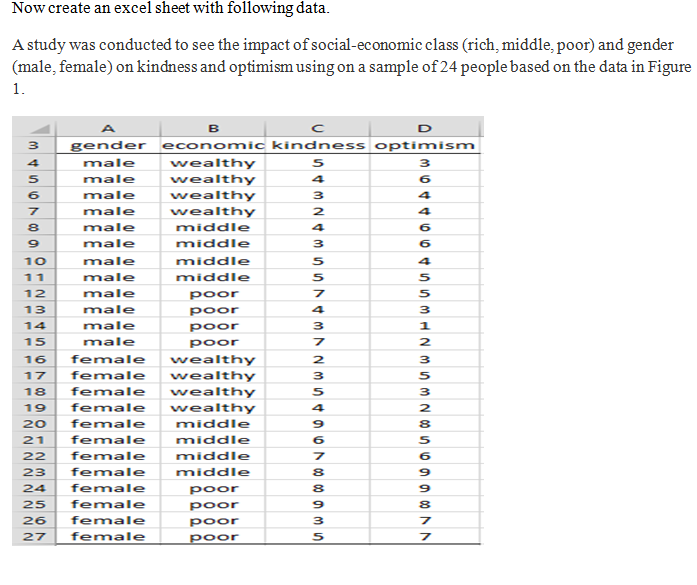
# Excel:

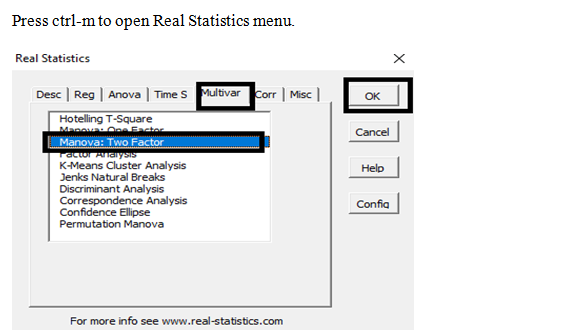


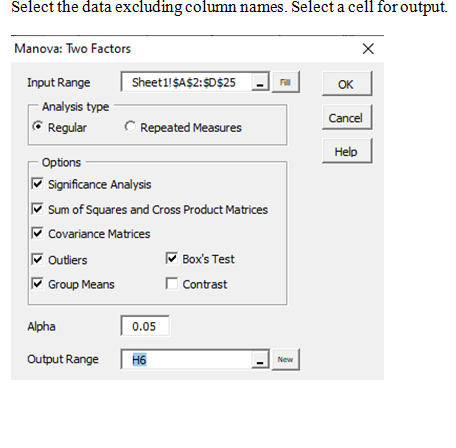




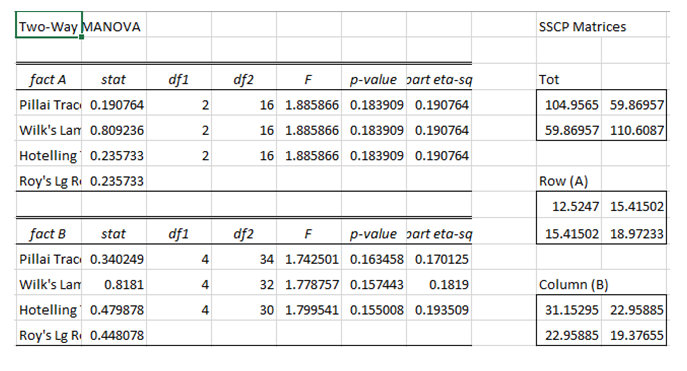






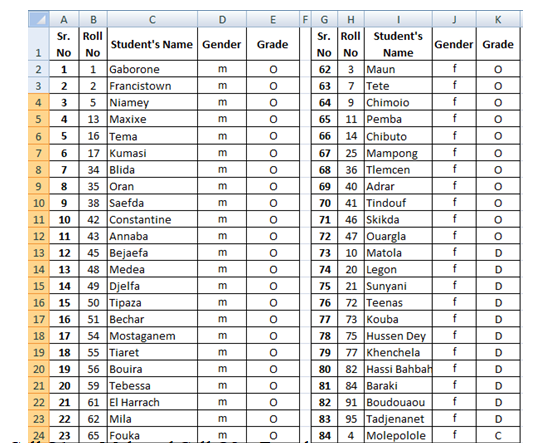


**Output:**

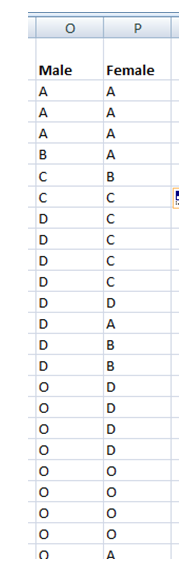
****

# Practical 7:

## Perform the Random sampling for the given data and analyse it.



#### Output:



## B.Perform the Stratified sampling for the given data and analyse it.

#### Program Code:

import pandas as pd importnumpy as np

importmatplotlib importmatplotlib.pyplot as plt

plt.rcParams['axes.labelsize'] = 14

plt.rcParams['xtick.labelsize'] = 12

plt.rcParams['ytick.labelsize'] = 12

importseaborn as sns color = sns.color\_palette() sns.set\_style('darkgrid')

importsklearn

fromsklearn.model\_selection import train\_test\_split

housing =pd.read\_csv('housing.csv')

print(housing.head())

print(housing.info())

#creating a heatmap of the attributes in the dataset

correlation\_matrix = housing.corr()

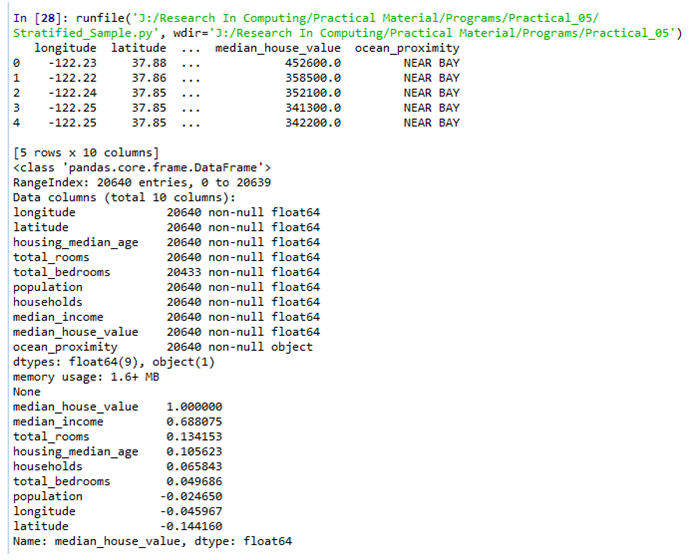
plt.subplots(figsize=(8,6))

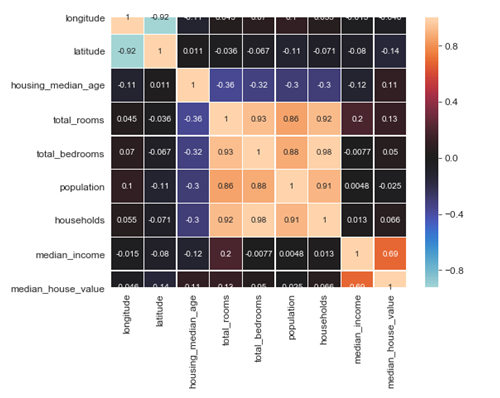
sns.heatmap(correlation\_matrix, center=0, annot=True, linewidths=.3)

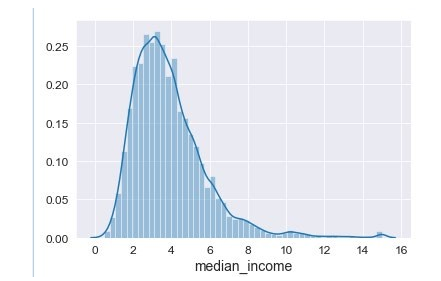
corr =housing.corr() print(corr['median\_house\_value'].sort\_values(ascending=False))

sns.distplot(housing.median\_income) plt.show()

**Output:**

****

****



# Practical 8:

## Write a program for computing different correlation.

#### Code:

importnumpy as np

importmatplotlib.pyplot as plt

np.random.seed(1)

# 1000 random integers between 0 and 50

x = np.random.randint(0, 50, 1000)

# Positive Correlation with some noise

y = x + np.random.normal(0, 10, 1000)

np.corrcoef(x, y)

matplotlib.style.use('ggplot')

plt.scatter(x, y)

plt.show()

#### Output:

#### 

**Negative Correlation:**

importnumpy as np

importmatplotlib.pyplot as plt

np.random.seed(1)

# 1000 random integers between 0 and 50

x = np.random.randint(0, 50, 1000)

# Negative Correlation with some noise

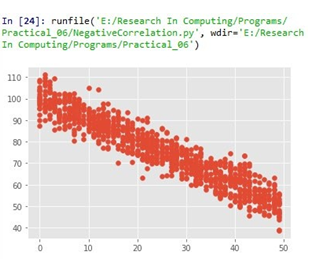
y = 100 - x + np.random.normal(0, 5, 1000)

np.corrcoef(x, y)

plt.scatter(x, y)

plt.show()

#### Output:

****

**No/Weak Correlation:**

importnumpy as np

importmatplotlib.pyplot as plt

np.random.seed(1)

x = np.random.randint(0, 50, 1000)

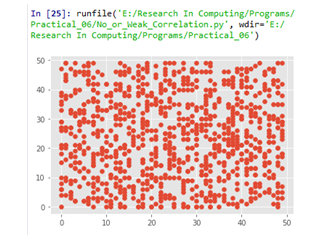
y = np.random.randint(0, 50, 1000)

np.corrcoef(x, y)

plt.scatter(x, y)

plt.show()

**Output:**

****

# Practical 9:

## Write a program to Perform linear regression for prediction.

import Quandl, math

import numpy as np

import pandas as pd

from sklearn import preprocessing, cross\_validation, svm

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

from matplotlib import style import datetime

style.use('ggplot')

df = Quandl.get("WIKI/GOOGL")

df = df[['Adj. Open', 'Adj. High', 'Adj. Low', 'Adj. Close', 'Adj. Volume']]

df['HL\_PCT'] = (df['Adj. High'] - df['Adj. Low']) / df['Adj. Close'] \* 100.0

df['PCT\_change'] = (df['Adj. Close'] - df['Adj. Open']) / df['Adj. Open'] \* 100.0

df = df[['Adj. Close', 'HL\_PCT', 'PCT\_change', 'Adj. Volume']]

forecast\_col = 'Adj. Close'

df.fillna(value=-99999, inplace=True)

forecast\_out = int(math.ceil(0.01 \* len(df)))

df['label'] = df[forecast\_col].shift(-forecast\_out)

X = np.array(df.drop(['label'], 1))

X = preprocessing.scale(X)

X\_lately = X[-forecast\_out:]

X = X[:-forecast\_out]

df.dropna(inplace=True)

y = np.array(df['label'])

X\_train, X\_test, y\_train, y\_test = cross\_validation.train\_test\_split(X, y, test\_size=0.2)

clf = LinearRegression(n\_jobs=-1)

clf.fit(X\_train, y\_train)

confidence = clf.score(X\_test, y\_test)

forecast\_set = clf.predict(X\_lately)

df['Forecast'] = np.nan

last\_date = df.iloc[-1].name

last\_unix = last\_date.timestamp()

one\_day = 86400

next\_unix = last\_unix + one\_day

for i in forecast\_set:

next\_date = datetime.datetime.fromtimestamp(next\_unix)

next\_unix += 86400

df.loc[next\_date] = [np.nan for \_ in range(len(df.columns)-1)]+[i]

df['Adj. Close'].plot()

df['Forecast'].plot()

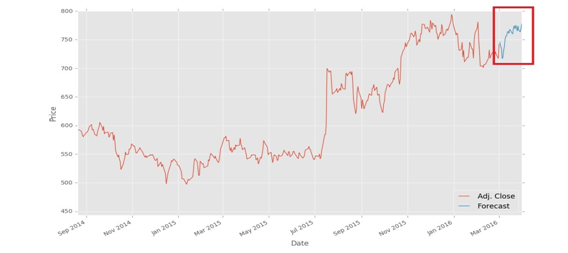
plt.legend(loc=4)

plt.xlabel('Date')

plt.ylabel('Price')

plt.show()

### Output:



## Perform polynomial regression for prediction.

importnumpy as np

importmatplotlib.pyplot as plt

defestimate\_coef(x, y):

# number of observations/points

n = np.size(x)

# mean of x and y vector

m\_x, m\_y = np.mean(x), np.mean(y)

# calculating cross-deviation and deviation about x

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

SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

# calculating regression coefficients

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x

return(b\_0, b\_1)

defplot\_regression\_line(x, y, b):

# plotting the actual points as scatter plot

plt.scatter(x, y, color = "m",

marker = "o", s = 30)

# predicted response vector

y\_pred = b[0] + b[1]\*x

# plotting the regression line

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

# putting labels

plt.xlabel('x')

plt.ylabel('y')

# function to show plot

plt.show()

def main():

# observations

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# estimating coefficients

b = estimate\_coef(x, y)

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

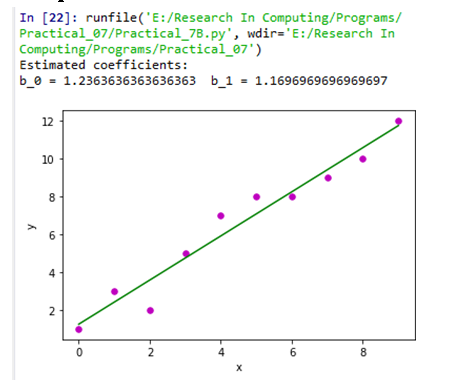
# plotting regression line

plot\_regression\_line(x, y, b)

if name == " main ":

main()

**Output:**

****

# Practical 10:

## Write a program for multiple linear regression analysis.

importnumpy as np

importmatplotlib as mpl

from mpl\_toolkits.mplot3d import Axes3D

importmatplotlib.pyplot as plt

defgenerate\_dataset(n):

x = []

y = []

random\_x1 = np.random.rand()

random\_x2 = np.random.rand()

for i in range(n):

x1 = i

x2 = i/2 + np.random.rand()\*n

x.append([1, x1, x2])

y.append(random\_x1 \* x1 + random\_x2 \* x2 + 1)

returnnp.array(x), np.array(y)

x, y = generate\_dataset(200)

mpl.rcParams['legend.fontsize'] = 12

fig = plt.figure()

ax = fig.gca(projection ='3d')

ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)

ax.legend()

ax.view\_init(45, 0)

plt.show()

defmse(coef, x, y):

returnnp.mean((np.dot(x, coef) - y)\*\*2)

def gradients(coef, x, y):

returnnp.mean(x.transpose()\*(np.dot(x, coef) - y), axis = 1)

defmultilinear\_regression(coef, x, y, lr, b1 = 0.9, b2 = 0.999, epsilon = 1e-8):

prev\_error = 0

m\_coef = np.zeros(coef.shape)

v\_coef = np.zeros(coef.shape)

moment\_m\_coef = np.zeros(coef.shape)

moment\_v\_coef = np.zeros(coef.shape)

t = 0

while True:

error = mse(coef, x, y)

if abs(error - prev\_error) <= epsilon:

break

prev\_error = error

grad = gradients(coef, x, y) t += 1

m\_coef = b1 \* m\_coef + (1-b1)\*grad

v\_coef = b2 \* v\_coef + (1-b2)\*grad\*\*2

moment\_m\_coef = m\_coef / (1-b1\*\*t)

moment\_v\_coef = v\_coef / (1-b2\*\*t)

delta = ((lr / moment\_v\_coef\*\*0.5 + 1e-8) \*

(b1 \* moment\_m\_coef + (1-b1)\*grad/(1-b1\*\*t)))

coef = np.subtract(coef, delta)

returncoef

coef = np.array([0, 0, 0])

c = multilinear\_regression(coef, x, y, 1e-1) fig = plt.figure()

ax = fig.gca(projection ='3d')

ax.scatter(x[:, 1], x[:, 2], y, label ='y',

s = 5, color ="dodgerblue")

ax.scatter(x[:, 1], x[:, 2], c[0] + c[1]\*x[:, 1] + c[2]\*x[:, 2],

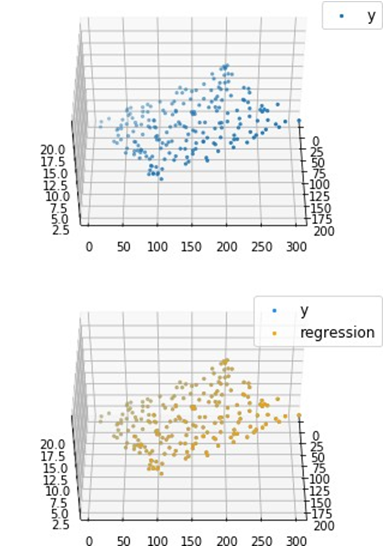
label ='regression', s = 5, color ="orange")

ax.view\_init(45, 0)

ax.legend()

plt.show()

# Output:



## Perform logistic regression analysis.

### Program Code:

import os

import numpy as np

import pandas as pd

import matplotlib

import matplotlib.pyplot as plt

import scipy.stats as stats

from sklearn import linear\_model

from sklearn import preprocessing

from sklearn import metrics

matplotlib.style.use('ggplot')

plt.figure(figsize=(9,9))

def sigmoid(t): # Define the sigmoid function

return (1/(1 + np.e\*\*(-t)))

plot\_range = np.arange(-6, 6, 0.1)

y\_values = sigmoid(plot\_range)

# Plot curve

plt.plot(plot\_range, # X-axis range

y\_values, # Predicted values

color="red")

titanic\_train = pd.read\_csv("titanic\_train.csv") # Read the data

char\_cabin = titanic\_train["Cabin"].astype(str) # Convert cabin to str

new\_Cabin = np.array([cabin[0] for cabin in char\_cabin]) # Take first letter

titanic\_train["Cabin"] = pd.Categorical(new\_Cabin) # Save the new cabin var

# Impute median Age for NA Age values

new\_age\_var = np.where(titanic\_train["Age"].isnull(), # Logical check 28, # Value if check is true

titanic\_train["Age"]) # Value if check is false titanic\_train["Age"] = new\_age\_var

label\_encoder = preprocessing.LabelEncoder()

# Convert Sex variable to numeric

encoded\_sex = label\_encoder.fit\_transform(titanic\_train["Sex"])

# Initialize logistic regression model

log\_model = linear\_model.LogisticRegression()

# Train the model

log\_model.fit(X = pd.DataFrame(encoded\_sex), y = titanic\_train["Survived"])

# Check trained model intercept print(log\_model.intercept\_)

# Check trained model coefficients print(log\_model.coef\_)

# Make predictions

preds = log\_model.predict\_proba(X= pd.DataFrame(encoded\_sex))

preds = pd.DataFrame(preds)

preds.columns = ["Death\_prob", "Survival\_prob"]

# Generate table of predictions vs Sex

pd.crosstab(titanic\_train["Sex"], preds.ix[:, "Survival\_prob"])

# Convert more variables to numeric

encoded\_class = label\_encoder.fit\_transform(titanic\_train["Pclass"])

encoded\_cabin = label\_encoder.fit\_transform(titanic\_train["Cabin"])

train\_features = pd.DataFrame([encoded\_class,

encoded\_cabin,

encoded\_sex, titanic\_train["Age"]]).T

# Initialize logistic regression model

log\_model = linear\_model.LogisticRegression()

# Train the model

log\_model.fit(X = train\_features ,

y = titanic\_train["Survived"])

# Check trained model intercept

print(log\_model.intercept\_)

# Check trained model coefficients

print(log\_model.coef\_)

# Make predictions

preds = log\_model.predict(X= train\_features)

# Generate table of predictions vs actual

pd.crosstab(preds,titanic\_train["Survived"])

log\_model.score(X = train\_features ,

y = titanic\_train["Survived"])

metrics.confusion\_matrix(y\_true=titanic\_train["Survived"],

# True labels y\_pred=preds) # Predicted labels

# View summary of common classification metrics

print(metrics.classification\_report(y\_true=titanic\_train["Survived"],

y\_pred=preds) )

# Read and prepare test data

titanic\_test = pd.read\_csv("titanic\_test.csv") # Read the data

char\_cabin = titanic\_test["Cabin"].astype(str) # Convert cabin to str

new\_Cabin = np.array([cabin[0] for cabin in char\_cabin]) # Take first letter

titanic\_test["Cabin"] = pd.Categorical(new\_Cabin) # Save the new cabin var

# Impute median Age for NA Age values

new\_age\_var = np.where(titanic\_test["Age"].isnull(),28,titanic\_test["Age"])

titanic\_test["Age"] = new\_age\_var

# Convert test variables to match model features

encoded\_sex = label\_encoder.fit\_transform(titanic\_test["Sex"])

encoded\_class = label\_encoder.fit\_transform(titanic\_test["Pclass"])

encoded\_cabin = label\_encoder.fit\_transform(titanic\_test["Cabin"])

test\_features = pd.DataFrame([encoded\_class,encoded\_cabin,encoded\_sex,titanic\_test["Age"]]).T

# Make test set predictions

test\_preds = log\_model.predict(X=test\_features)

# Create a submission for Kaggle

submission = pd.DataFrame({"PassengerId":titanic\_test["PassengerId"], "Survived":test\_preds})

# Save submission to CSV submission.to\_csv("tutorial\_logreg\_submission.csv",

index=False) # Do not save index values

print(pd)

### Output:

