Practical 1

In [62]:
```
x=5
y=10
z=x+y
z
```

Out[62]:
```
15
```

In [63]:
```
x=5
y=10
z=x-y
```

In [64]:
```
z
```

Out[64]:
```
-5
```

In [65]:
```
x=5
y=10
z=x*y
z
```

Out[65]:
```
50
```

In [66]:
```
x=5
y=10
z=x**y
z
```

Out[66]:
```
9765625
```

Practical 2

In [ ]:
```
x=10
```

In [ ]:
```
y=10
```

In [72]:
```
x = 5
y = 8

print("x == y:", x == y)
print("x != y:", x != y)
print("x < y:", x < y)
print("x > y:", x > y)
print("x <= y:", x <= y)
print("x >= y:", x >= y)
```

```
x == y: False
x != y: True
```

```
x < y: True
x > y: False
x <= y: True
x >= y: False
```

Practical 3

In [81]:
```
x='abc'
```

In [82]:
```
x
```

Out[82]:
```
'abc'
```

Practical 4

In [84]:
```
import numpy as np
array = np.arange(20)
array
```

Out[84]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [85]:
```
array = np.arange(27).reshape(3,3,3)
array
```

Out[85]:
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

       [[ 9, 10, 11],
        [12, 13, 14],
        [15, 16, 17]],

       [[18, 19, 20],
        [21, 22, 23],
        [24, 25, 26]]])
```

In [86]:
```
np.ones(5)
```

Out[86]:
```
array([1., 1., 1., 1., 1.])
```

In [87]:
```
 np.zeros(5)
```

Out[87]:
```
array([0., 0., 0., 0., 0.])
```

In [89]:
```
x = np.arange(9).reshape((3,3))
x
```

Out[89]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [91]:
```
np.diag(x)
```

Out[91]:
```
array([0, 4, 8])
```

practical 5

In [95]:
```
a=[[1,5,6,8],[1,2,5,9],[7,5,6,2]]
```

In [96]:
```
a
```

Out[96]:
```
[[1, 5, 6, 8], [1, 2, 5, 9], [7, 5, 6, 2]]
```

In [98]:
```
np.shape(a)
```

Out[98]:
```
(3, 4)
```

Practical 6

In [99]:
```
A = np.array([[2, 4], [5, -6]])
B = np.array([[9, -3], [3, 6]])
C = A + B      # element wise addition
print(C)
```

```
[[11  1]
 [ 8  0]]
```

In [101]:
```
A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
print(C)
```

```
[[ 36 -12]
 [ -1   2]]
```

In [102]:
```
A = np.array([[2, 4], [5, -6]])
B = np.array([[9, -3], [3, 6]])
C = A - B      # element wise addition
print(C)
```

```
[[ -7   7]
 [  2 -12]]
```

Practical 7

In [103]:
```
arr = [[14, 17, 12, 33, 44],
       [15, 6, 27, 8, 19],
       [23, 2, 54, 1, 4,]]

print("\nSum of arr : ", np.sum(arr))
```

```
Sum of arr :  279
```

In [104]:
```
np.min(arr), np.max(arr)
```

Out[104]:
```
(1, 54)
```

In [105]:
```
np.negative(arr)
```

Out[105]:
```
array([[-14, -17, -12, -33, -44],
       [-15,  -6, -27,  -8, -19],
       [-23,  -2, -54,  -1,  -4]])
```

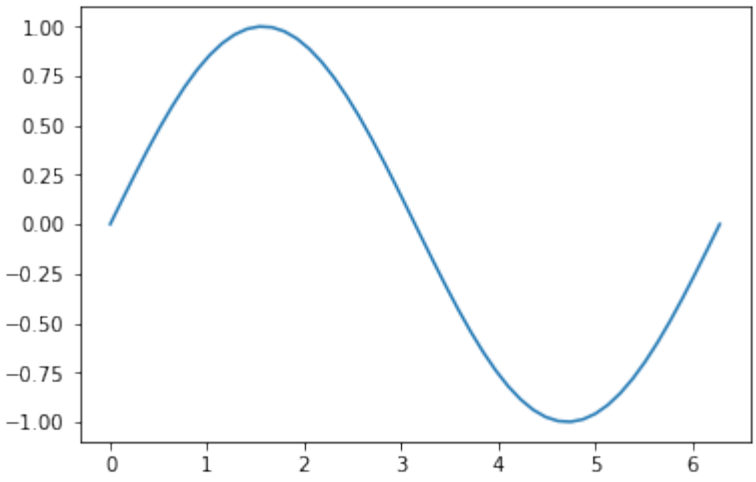
In [106]:
```
arr1 = [1, -3, 15, -466]
np.absolute(arr1)
```

Out[106]:
```
array([  1,   3,  15, 466])
```

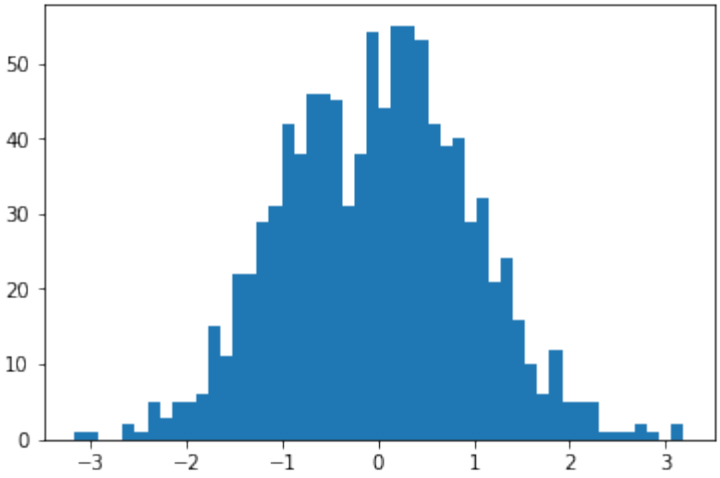
Practical 8

In [9]:
```
import matplotlib.pyplot as plt
import numpy as np
```

In [10]:
```
x = np.linspace(0, 2 * np.pi, 50)
plt.plot(x, np.sin(x))
plt.show()
```

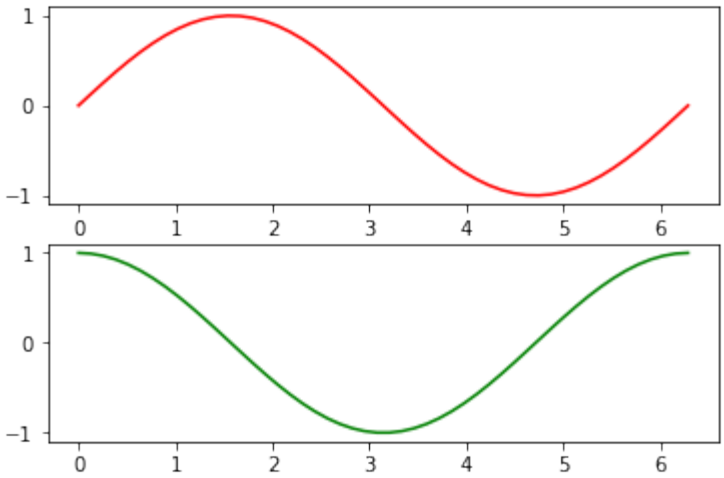


In [11]:
```
x = np.random.randn(1000)
plt.hist(x, 50)
plt.show()
```



Practical 9

In [12]:
```
x = np.linspace(0, 2 * np.pi, 50)
plt.subplot(2, 1, 1)
plt.plot(x, np.sin(x), 'r')
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x), 'g')
plt.show()
```



Practical 10

In [13]:
```
for i in [12, 16, 17, 24, 29, 30]:
    if i % 2 == 1:      # if the number is odd
        continue        # don't process it
    print(i)
print("done")
```

```
12
16
24
30
```

done

In [ ]:
```
num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

Practical 11

In [ ]:
```
# input two matrices
mat1 = ([1, 6, 5],[3 ,4, 8],[2, 12, 3])
mat2 = ([3, 4, 6],[5, 6, 7],[6,56, 7])


res = np.dot(mat1,mat2)


print(res)
```

In [ ]:
```
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

Y = [[5,8,1],
    [6,7,3],
    [4,5,9]]

result = [[0,0,0],
          [0,0,0],
          [0,0,0]]

for i in range(len(X)):

   for j in range(len(X[0])):
       result[i][j] = X[i][j] + Y[i][j]

for r in result:
   print(r)
```

In [ ]:
```
A = np.array([[15, 29, 24],
                [ 5, 23, 26],
                [30, 14, 44]])

A.T
```

Practical 12

and Practical 13

In [16]:
```
import pandas as pd
```

In [17]:
```
data_train=pd.read_csv("/home/desktop/Downloads/house.csv")
```

In [18]:
```
data_train.head()
```

Out[18]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterf
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	

5 rows × 21 columns

In [20]:
```
data_train.rename(columns ={'price': 'SalePrice'}, inplace =True)
```

In [21]:
```
data_train.head()
```

Out[21]:

	id	date	SalePrice	bedrooms	bathrooms	sqft_living	sqft_lot	floors	water
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	

5 rows × 21 columns

In [23]:
```
Y = data_train.SalePrice.values
```

In [24]:
```
feature_cols = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floor
s',
       'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'lon
g',
       'sqft_living15', 'sqft_lot15']
X=data_train[feature_cols]
```

In [25]:
```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test = train_test_split(X, Y, random_state=3)
```

In [26]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train, y_train)
```

Out[26]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [27]:
```
accuracy = regressor.score(x_test, y_test)
```

In [28]:
```
"Accuracy: {}%".format(int(round(accuracy * 100)))
```

Out[28]:
```
'Accuracy: 69%'
```

In [29]:
```
data_train.shape
```

Out[29]:
```
(21613, 21)
```

In [30]:
```
data_train.describe()
```

Out[30]:

	id	SalePrice	bedrooms	bathrooms	sqft_living	sqft_lot	f
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.00
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.49
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.53
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.00
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.00
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.50
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.00
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.50

Practical 14

In [32]:
```
df = pd.read_csv("/home/desktop/Downloads/diabetes.csv")
```

In [33]:
```
df.head()
```

Out[33]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21

4	0	137	40	35	168	43.1	2.288	33

In [34]:
```
X = ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BM
I','DiabetesPedigreeFunction','Age']
y = ['Output']
```

In [35]:
```
df2 = pd.DataFrame(data=df)
df2.head()
```

Out[35]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

In [36]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop('Outcome',axis
=1),df['Outcome'],
                                                    test_size=0.30, random
_state=101)
```

In [37]:
```
from sklearn.linear_model import LogisticRegression
```

In [39]:
```
logmodel = LogisticRegression()
```

In [40]:
```
logmodel.fit(X_train,y_train)
```

```
/home/desktop/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/
lo gistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs'
i0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[40]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [41]:
```
predictions = logmodel.predict(X_test)
```

In [43]:
```
from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.79      0.91      0.84       150
           1       0.76      0.56      0.64        81
```

```
   micro avg       0.78      0.78      0.78       231
   macro avg       0.78      0.73      0.74       231
weighted avg       0.78      0.78      0.77       231
```

Practical 15

In [46]:
```
from sklearn.linear_model import (LinearRegression, Ridge, Lasso, RandomizedLasso)
```

In [54]:
```
lasso = Lasso(alpha=0.2, normalize=True)
```

In [57]:
```
lasso.fit(X_train,y_train)
```

Out[57]:
```
Lasso(alpha=0.2, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=True, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
```

In [58]:
```
lasso_coef = lasso.coef_
print(lasso_coef)
```

```
[0. 0. 0. 0. 0. 0. 0. 0.]
```

In [59]:
```
ridge = Ridge(normalize=True)
```

In [60]:
```
ridge.fit(X_train,y_train)
```

Out[60]:
```
Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=True, random_state=None, solver='auto', tol=0.001)
```

In [ ]:
```

```