In [1]:

```
#part (a)
import numpy as np
import scipy
import random
from scipy.stats import norm

random.seed(1)
x = scipy.stats.norm.rvs(0,1,100)
print(x)
type(x)
```

```
[ 0.97221479 -0.92740028 -0.63338234 -0.576785   -0.51394308 -0.81066485
  0.50877016 -1.24511252 -0.03315329 -0.31672968  0.34731047  0.37727811
 -0.749477   -0.35845986 -0.73594266  0.19585317 -1.31901384 -2.54999129
 -1.48009844  0.67231549 -0.80627534  0.99140209  0.18838675 -0.39416149
  0.16057233 -0.62531824 -0.75183729  0.62463442  0.55325876  1.03677831
  1.4242902  -2.05695552 -1.85543289  1.16253085 -0.28802843 -2.50679071
 -2.35706439  2.34382316  0.09240446 -1.11012116  0.61916749  0.70865674
 -0.72686415 -0.86596653  0.70313649 -0.59718063  0.08594065  0.65975115
 -0.99449372  0.71903647 -0.78020986  1.43170013 -0.37075467  0.34182477
 -0.9292206  -1.44132281  1.63085376  0.9782552   0.19592277 -0.62070193
 -0.96083161 -0.8377035  -0.49277161  0.92010711 -0.58262009 -0.00423353
 -0.12385248  0.85934768 -0.02088023 -0.18071929 -0.49545016  1.01441237
 -0.84777027 -0.45454538 -0.53220349 -0.11209985  2.25533352  0.04581569
  0.16757811  0.76471136 -0.41988988 -1.85102944  0.38149575 -0.20724744
  0.13250994 -0.42994976  0.18600503  0.11455694  0.19667599 -0.1624253
 -0.45903603  1.74160555 -0.50138967 -1.2435958  -0.48841424 -0.61793323
 -1.64076334 -0.26204921 -1.16822035  0.32033387]
```

```
numpy.ndarray
```

In [2]:

```
# part(b)
#i.e eps with 100 observations with mean=0 and standard deviation 0.25
eps = scipy.stats.norm.rvs(0,0.25,100)
print(eps)
```

```
[ 0.05134329  0.19175352  0.38992163  0.05443336  0.48360408  0.24973874
 -0.19169575  0.12362773 -0.04682272 -0.07134571  0.178674   -0.15468452
 -0.38596603 -0.26931792 -0.25112699 -0.03707416 -0.10588405 -0.24659402
 -0.22137272 -0.32473211 -0.26373246 -0.11456776 -0.06985854 -0.01103336
 -0.05339645  0.34171385  0.19228575 -0.29769024  0.3935461  -0.08723332
  0.3358643  -0.21260122  0.20357906  0.55893656  0.42641531 -0.21267006
 -0.30318128 -0.27665186  0.09000275  0.34220521 -0.01590712  0.1092139
 -0.00496411 -0.09527561  0.28386229  0.1691921   0.31493174  0.17205523
 -0.35325198  0.04741577 -0.06116971 -0.3442523  -0.17273676  0.12423984
  0.06550237 -0.19266313 -0.0174233   0.25442838 -0.04386765  0.03959369
  0.00716803  0.46013086 -0.60308423 -0.04665738  0.02335581  0.00479238
 -0.35824608 -0.24283514  0.43250509 -0.04210549 -0.32839784 -0.28566618
  0.28700475 -0.12793625 -0.50049858 -0.38392986 -0.07527084  0.08819376
  0.10829534  0.32689289 -0.23510999  0.16093872 -0.02285449 -0.14726861
  0.1258589   0.16623862 -0.05421325 -0.13792031 -0.09642576  0.00549327
  0.21160836 -0.29536019  0.12910054 -0.4938821   0.15604439 -0.20409732
  0.39097955  0.03636928  0.06479711  0.18805587]
```

In [3]:

```
# part (c)
# creating Y = -1 + 0.5 X + eps using X,eps as generted above
y = -1 + (0.5*x) + eps
print(y)
print(type(y))
print("Length of vectory y: ",np.size(y))
```

```
[-0.46254931 -1.27194662 -0.92676954 -1.23395914 -0.77336747 -1.15559369
 -0.93731068 -1.49892853 -1.06339936 -1.22971055 -0.64767076 -0.96604546
 -1.76070452 -1.44854785 -1.61909832 -0.93914758 -1.76539097 -2.52158966
 -1.96142194 -0.98857436 -1.66687013 -0.61886672 -0.97566516 -1.20811411
 -0.97311028 -0.97094527 -1.18363289 -0.98537303 -0.32982452 -0.56884417
  0.0480094  -2.24107898 -1.72413738  0.14020199 -0.7175989  -2.46606542
 -2.48171347 -0.10474028 -0.86379502 -1.21285537 -0.70632338 -0.53645773
 -1.36839619 -1.52825887 -0.36456947 -1.12939821 -0.64209794 -0.4980692
 -1.85049884 -0.593066   -1.45127464 -0.62840223 -1.3581141  -0.70484778
 -1.39910793 -1.91332454 -0.20199642 -0.25644402 -0.94590627 -1.27075728
 -1.47324777 -0.95872089 -1.84947004 -0.58660382 -1.26795424 -0.99732439
 -1.42017232 -0.8131613  -0.57793503 -1.13246514 -1.57612292 -0.77846
 -1.13688038 -1.35520893 -1.76660033 -1.43997979  0.05239593 -0.88889839
 -0.80791561 -0.29075143 -1.44505493 -1.764576   -0.83210662 -1.25089233
 -0.80788614 -1.04873626 -0.96121073 -1.08064184 -0.99808777 -1.07571938
 -1.01790966 -0.42455742 -1.12159429 -2.11568    -1.08816273 -1.51306394
 -1.42940212 -1.09465533 -1.51931306 -0.6517772 ]
<class 'numpy.ndarray'>
Length of vectory y:  100
```

In [4]:

```
# As observed from above length of vector Y is 100.
```

In [5]:
```
# part (d)
# Generating scatter plot of x,y values generated.
import matplotlib.pyplot as plt
plt.scatter(x,y)
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

In [6]:
```
# we observe that there is some kind of linear relationship between
# variables x and y by seeing the plot
```

In [7]:
```
# creating pandas dataframe from the above data.
import pandas as pd
# creating dictionary to pass to the dataFrame
XYdata = {'X':x, 'Y':y}
df = pd.DataFrame(XYdata)
print(df.head())
print(df.tail())
print(df.info())
print(df.describe())
```

```
          X         Y
0  0.972215 -0.462549
1 -0.927400 -1.271947
2 -0.633382 -0.926770
3 -0.576785 -1.233959
4 -0.513943 -0.773367
           X         Y
95 -0.617933 -1.513064
96 -1.640763 -1.429402
97 -0.262049 -1.094655
98 -1.168220 -1.519313
99  0.320334 -0.651777
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 2 columns):
X    100 non-null float64
Y    100 non-null float64
dtypes: float64(2)
memory usage: 1.7 KB
None
                X           Y
count  100.000000  100.000000
mean    -0.185999   -1.099306
std      0.950861    0.542683
min     -2.549991   -2.521590
25%     -0.750067   -1.441249
50%     -0.275039   -1.069559
75%      0.378333   -0.759425
max      2.343823    0.140202
```

In [8]:

```
# part (e)
import statsmodels.formula.api as sm
# fitting the data using ols model.
results = sm.ols('Y ~ X',df).fit()
type(results)


print(results.__dict__)
print("..................... Results .....................")
print(results.summary())
```

```
{'_results': <statsmodels.regression.linear_model.OLSResults object at 0x00000297EEA87C48>, '__doc__': "\n
for an OLS model.\n\n    Parameters\n    ----------\n    model : RegressionModel\n        The regression moc
rams : ndarray\n        The estimated parameters.\n    normalized_cov_params : ndarray\n        The normaliz
ters.\n    scale : float\n        The estimated scale of the residuals.\n    cov_type : str\n        The cov
ed in the results.\n    cov_kwds : dict\n        Additional keywords used in the covariance specification.\n
Flag indicating to use the Student's t in inference.\n    **kwargs\n        Additional keyword arguments use
results.\n\n    See Also\n    --------\n    RegressionResults\n        Results store for WLS and GLW models.
-----\n    Most of the methods and attributes are inherited from RegressionResults.\n    The special methods
able for OLS are:\n\n    - get_influence\n    - outlier_test\n    - el_test\n    - conf_int_el\n    "}
...................... Results .....................
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.798
Model:                            OLS   Adj. R-squared:                  0.796
Method:                 Least Squares   F-statistic:                     386.2
Date:                Mon, 27 Jan 2020   Prob (F-statistic):           9.02e-36
Time:                        01:05:29   Log-Likelihood:               -0.38756
No. Observations:                 100   AIC:                             4.775
Df Residuals:                      98   BIC:                             9.985
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -1.0045      0.025    -40.168      0.000      -1.054      -0.955
X              0.5097      0.026     19.653      0.000       0.458       0.561
==============================================================================
Omnibus:                        1.390   Durbin-Watson:                   1.843
Prob(Omnibus):                  0.499   Jarque-Bera (JB):                1.194
Skew:                           0.076   Prob(JB):                        0.550
Kurtosis:                       2.487   Cond. No.                         1.22
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [9]:

```
# ols (ordinary Least Square model) is used to estimate beta_1 and beta_0 such a way
# the square of the errors obtained from the model.
# estimated beta_0 is -1.0609 while the actual beta_0 is -1
# estimated beta_1 is 0.5051 while the actual beta_1 is 0.5
# value of r-squared is 0.825
# r-squared is represents how close the data are to the fitted regression line
# it is known as the coefficient of determination, Higher R-squared values
# represent smaller differences between the observed data and the fitted values.
```

In [10]:

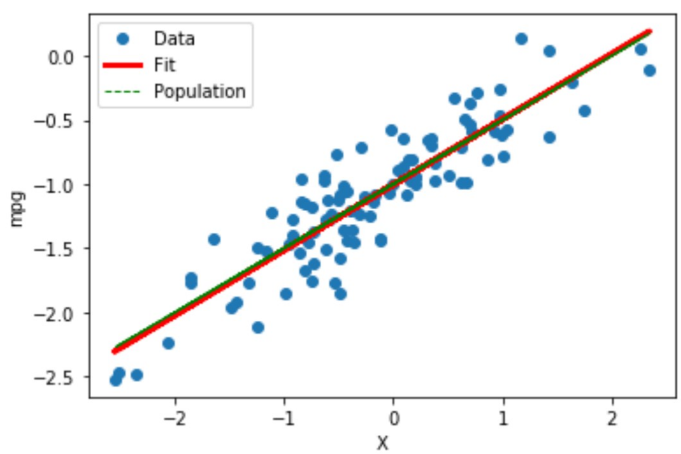
```
# part (f)
# scattered plot, least square line and population regression line
# on the same plot with legend. (done according to the requirement)
%matplotlib inline
y_true = df.Y.values.copy()
y_predicted = results.predict(df.X)
y_population = -1 + 0.5*df.X.values

plt.plot(df.X, y_true, 'o',label = 'Data')

plt.plot(df.X, y_predicted, 'r-',linewidth=3,label='Fit')

plt.plot(df.X, y_population, 'g--',linewidth=1,label='Population')
plt.xlabel('X')
plt.ylabel('mpg')
plt.legend(loc ='best')
print('Parameters: ',results.params)
print('Standard errors: ',results.bse)
print('R-square: ',results.rsquared)
```

```
Parameters:  Intercept   -1.004499
X            0.509716
dtype: float64
Standard errors:  Intercept    0.025007
X            0.025936
dtype: float64
R-square:  0.7976218044423379
```



In [11]:

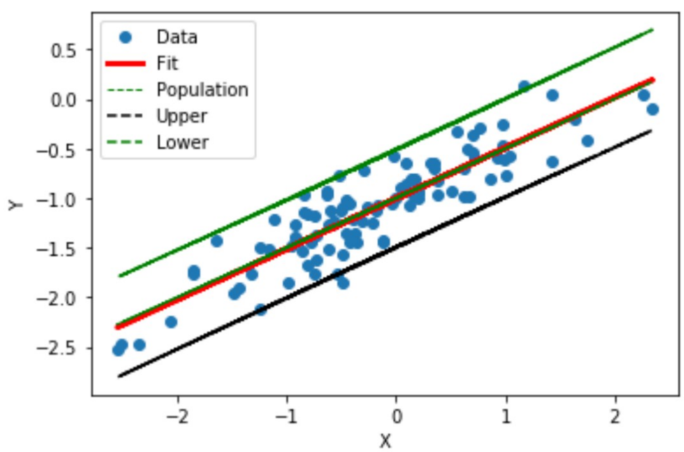
```
plt.plot(df.X, y_true, 'o',label = 'Data')


plt.plot(df.X, y_predicted, 'r-',linewidth=3,label='Fit')


plt.plot(df.X, y_population, 'g--',linewidth=1,label='Population')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend(loc ='best')
print('Parameters: ',results.params)
print('Standard errors: ',results.bse)
print('R-square: ',results.rsquared)
from statsmodels.sandbox.regression.predstd import wls_prediction_std
_, upper,lower = wls_prediction_std(results)
plt.plot(df.X, upper, '--k', label="Upper")
plt.plot(df.X, lower, '--g', label = "Lower")
plt.legend(loc='best')
```

```
Parameters:  Intercept   -1.004499
X            0.509716
dtype: float64
Standard errors:  Intercept    0.025007
X            0.025936
dtype: float64
R-square:  0.7976218044423379
```

```
<matplotlib.legend.Legend at 0x297eeb7f1c8>
```



In [12]:

```
# part (h) filling the blanks
import scipy
import random
from scipy.stats import norm
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.formula.api as sm
random.seed(1)
x_1 = scipy.stats.norm.rvs(0,1,100)
eps_1 = scipy.stats.norm.rvs(0,0.05,100)
y_1 = -1 + (0.5*x_1) + eps_1
XYdataLessNoise = {'X1':x_1,'Y1':y_1}
# creating a dataframe
dfLessNoise = pd.DataFrame(XYdataLessNoise)
#Perform linear regression on the Less noisy data
resultsLessNoise = sm.ols('Y1 ~ X1',dfLessNoise).fit()
print(resultsLessNoise.summary())
%matplotlib inline
#true, predicted and population data
y1_true = dfLessNoise.Y1.values.copy()
y1_predicted = resultsLessNoise.predict(dfLessNoise.X1)
y1_population = -1 + 0.5*dfLessNoise.X1.values

#plotting the data
plt.plot(dfLessNoise.X1, y1_true, 'o', label='Data')

#plot the prediction from the linear model
plt.plot(dfLessNoise.X1,y1_predicted,'r-',linewidth=3, label='Fit')
plt.plot(dfLessNoise.X1,y1_population, 'g--', linewidth=1, label='Population')
plt.xlabel('X1')
plt.ylabel('Y1')
plt.legend(loc='best')

print('Parameters: ',resultsLessNoise.params)

print('Standard errors: ',resultsLessNoise.bse)

print('R-square: ',resultsLessNoise.rsquared)

from statsmodels.sandbox.regression.predstd import wls_prediction_std
_, upper, lower = wls_prediction_std(resultsLessNoise)
plt.plot(dfLessNoise.X1, upper,'--k',label="Upper")
plt.plot(dfLessNoise.X1, lower,'--g',label="Lower")
plt.legend(loc="best")
```

In [13]:

```
# part (h) filling the blanks
import scipy
import random
from scipy.stats import norm
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.formula.api as sm
random.seed(1)
x_2 = scipy.stats.norm.rvs(0,1,100)
#observe standard deviation choosen for noise i.e 1.25
eps_2 = scipy.stats.norm.rvs(0,1.25,100)
y_2 = -1 + (0.5*x_2) + eps_2
XYdataMoreNoise = {'X2':x_2,'Y2':y_2}
# creating a dataframe
dfMoreNoise = pd.DataFrame(XYdataMoreNoise)
#Perform linear regression on the Less noisy data
resultsMoreNoise = sm.ols('Y2 ~ X2',dfMoreNoise).fit()
print(resultsMoreNoise.summary())
%matplotlib inline
#true, predicted and population data
y2_true = dfMoreNoise.Y2.values.copy()
y2_predicted = resultsMoreNoise.predict(dfMoreNoise.X2)
y2_population = -1 + 0.5*dfMoreNoise.X2.values

#plotting the data
plt.plot(dfMoreNoise.X2, y2_true, 'o', label='Data')

#plot the prediction from the linear model
plt.plot(dfMoreNoise.X2,y2_predicted,'r-',linewidth=3, label='Fit')
plt.plot(dfMoreNoise.X2,y2_population, 'g--', linewidth=1, label='Population')
plt.xlabel('X2')
plt.ylabel('Y2')
plt.legend(loc='best')

print('Parameters: ',resultsMoreNoise.params)

print('Standard errors: ',resultsMoreNoise.bse)

print('R-square: ',resultsMoreNoise.rsquared)

from statsmodels.sandbox.regression.predstd import wls_prediction_std
_, upper, lower = wls_prediction_std(resultsMoreNoise)
plt.plot(dfMoreNoise.X2, upper,'--k',label="Upper")
plt.plot(dfMoreNoise.X2, lower,'--k',label="Lower")
```

In [14]:

```
# for part (h) blanks were successfully filled and executed successfully.
```