Complete each problem below and print to pdf. Submit the pdf.

You will need to work with the three datasets attached to this assignment:

- poverty.csv
- poverty_2.csv
- real_estate.csv

Problem 1: Univariate Linear Regression

1) import the libraries you will need:

numpy pandas matplotlab.pyplot statsmodels.api

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn import datasets, linear_model, metrics
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
```

2) Import the date poverty.csv dataset

```
data = pd.read_csv('poverty.csv')
```

3) Print the dataset indexed upon the location column.

```
data = pd.read_csv('poverty.csv')
data.columns = [columns.replace(' ', '_').lower()
for columns in data.columns]
print(data.columns)
```

```
Index(['location', 'povpct', 'brth15to17', 'brth18to19', 'violcrime',
       'teenbrth'],
      dtype='object')
```

```
data[['location']][1:50]
```

	location
1	Alaska
2	Arizona
3	Arkansas
4	California
5	Colorado
6	Connecticut
7	Delaware
8	District_of_Columbia
9	Florida
10	Georgia
11	Hawaii
12	Idaho
13	Illinois
14	Indiana
15	Iowa
16	Kansas
17	Kentucky
18	Louisiana
19	Maine

4) Get useful descriptive statistial data on the dataset.

Hint: this is a single line, data._____

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   location    51 non-null     object 
 1   povpct      51 non-null     float64
 2   brth15to17  51 non-null     float64
 3   brth18to19  51 non-null     float64
 4   violcrime   51 non-null     float64
 5   teenbrth    51 non-null     float64
```

```
dtypes: float64(5), object(1)
memory usage: 2.5+ KB
```

32 New_York

5) Print the columns

```
print(data)
```

```
                location  povpct  brth15to17  brth18to19  violcrime  teenbrth
0                Alabama    20.1        31.5        88.7       11.2      54.5
1                 Alaska     7.1        18.9        73.7        9.1      39.5
2                Arizona    16.1        35.0       102.5       10.4      61.2
3               Arkansas    14.9        31.6       101.7       10.4      59.9
4             California    16.7        22.6        69.1       11.2      41.1
5               Colorado     8.8        26.2        79.1        5.8      47.0
6            Connecticut     9.7        14.1        45.1        4.6      25.8
7               Delaware    10.3        24.7        77.8        3.5      46.3
8   District_of_Columbia    22.0        44.8       101.5       65.0      69.1
9                Florida    16.2        23.2        78.4        7.3      44.5
10               Georgia    12.1        31.4        92.8        9.5      55.7
11                Hawaii    10.3        17.7        66.4        4.7      38.2
12                 Idaho    14.5        18.4        69.1        4.1      39.1
13              Illinois    12.4        23.4        70.5       10.3      42.2
14               Indiana     9.6        22.6        78.5        8.0      44.6
15                  Iowa    12.2        16.4        55.4        1.8      32.5
16                Kansas    10.8        21.4        74.2        6.2      43.0
17              Kentucky    14.7        26.5        84.8        7.2      51.0
18             Louisiana    19.7        31.7        96.1       17.0      58.1
19                 Maine    11.2        11.9        45.2        2.0      25.4
20              Maryland    10.1        20.0        59.6       11.8      35.4
21         Massachusetts    11.0        12.5        39.6        3.6      23.3
22              Michigan    12.2        18.0        60.8        8.5      34.8
23             Minnesota     9.2        14.2        47.3        3.9      27.5
24           Mississippi    23.5        37.6       103.3       12.9      64.7
25              Missouri     9.4        22.2        76.6        8.8      44.1
26               Montana    15.3        17.8        63.3        3.0      36.4
27              Nebraska     9.6        18.3        64.2        2.9      37.0
28                Nevada    11.1        28.0        96.7       10.7      53.9
29         New_Hampshire     5.3         8.1        39.0        1.8      20.0
30            New_Jersey     7.8        14.7        46.1        5.1      26.8
31            New_Mexico    25.3        37.8        99.5        8.8      62.4
32              New_York    16.5        15.7        50.1        8.5      29.5
33        North_Carolina    12.6        28.6        89.3        9.4      52.2
34          North_Dakota    12.0        11.7        48.7        0.9      27.2
35                  Ohio    11.5        20.1        69.4        5.4      39.5
36              Oklahoma    17.1        30.1        97.6       12.2      58.0
37                Oregon    11.2        18.2        64.8        4.1      36.8
38          Pennsylvania    12.2        17.2        53.7        6.3      31.6
39          Rhode_Island    10.6        19.6        59.0        3.3      35.6
40        South_Carolina    19.9        29.2        87.2        7.9      53.0
41          South_Dakota    14.5        17.3        67.8        1.8      38.0
42             Tennessee    15.5        28.2        94.2       10.6      54.3
43                 Texas    17.4        38.2       104.3        9.0      64.4
44                  Utah     8.4        17.8        62.4        3.9      36.8
```

```
45         Vermont   10.3   10.4   44.4   2.2   24.2
46        Virginia   10.2   19.0   66.0   7.6   37.6
47      Washington   12.5   16.8   57.6   5.1   33.0
48   West_Virginia   16.7   21.5   80.7   4.9   45.5
49       Wisconsin    8.5   15.9   57.1   4.3   32.3
50         Wyoming   12.2   17.7   72.1   2.1   39.9
```

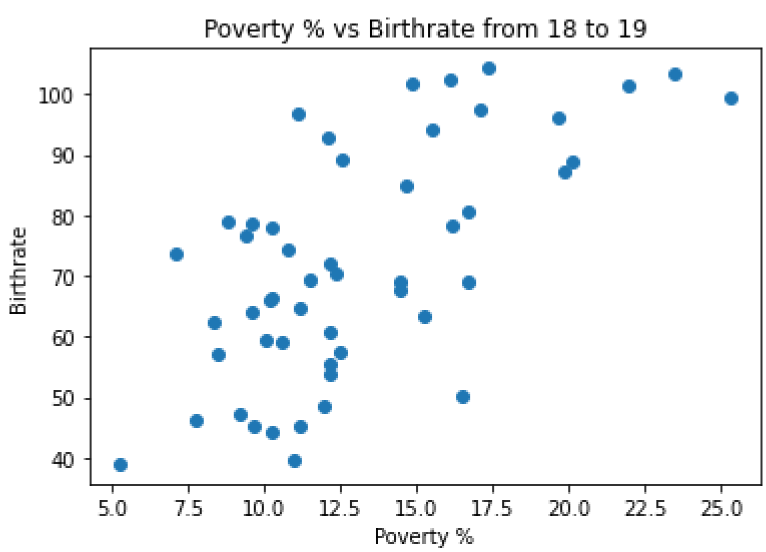
6) Create a regression line based upon the dependent and independent variables:

PovPct Brth18to19

In this step only create a scatterplot of the two variables, simply plotting the data.

Note: The variable PovPct is the percent of a state's population in 2000 living in households with incomes below the federally defined poverty level.

```
x = data['povpct']
y = data['brth18to19']

plt.scatter(x,y)
plt.title('Poverty % vs Birthrate from 18 to 19')
plt.xlabel("Poverty %")
plt.ylabel("Birthrate")
plt.show()
```



7) Lets create a new variable, x1, as well as the results variable:

Example would be

1. x1 = sm.add_constant(x)

2. results = sm.OLS(y, x1).fit()
3. results.summary()

This gives you the OLS Regression results, the coefficients table, and some additional tests. The data that you are interested in is the coefficient values. This is the value for the constant you created is b0, and birth19to19 is b1 in the regression equation.

```
x1 = sm.add_constant(x)
results = sm.OLS(y, x1).fit()
results.summary()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:117: FutureWarning:
  x = pd.concat(x[::order], 1)
```

OLS Regression Results

Dep. Variable:	brth18to19	**R-squared:**	0.422
Model:	OLS	**Adj. R-squared:**	0.410
Method:	Least Squares	**F-statistic:**	35.78
Date:	Fri, 18 Feb 2022	**Prob (F-statistic):**	2.50e-07
Time:	22:06:39	**Log-Likelihood:**	-207.98
No. Observations:	51	**AIC:**	420.0
Df Residuals:	49	**BIC:**	423.8
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	34.2124	6.641	5.151	0.000	20.866	47.559
povpct	2.8822	0.482	5.982	0.000	1.914	3.850

Omnibus:	1.175	**Durbin-Watson:**	2.161
Prob(Omnibus):	0.556	**Jarque-Bera (JB):**	0.988
Skew:	0.088	**Prob(JB):**	0.610
Kurtosis:	2.341	**Cond. No.**	45.1

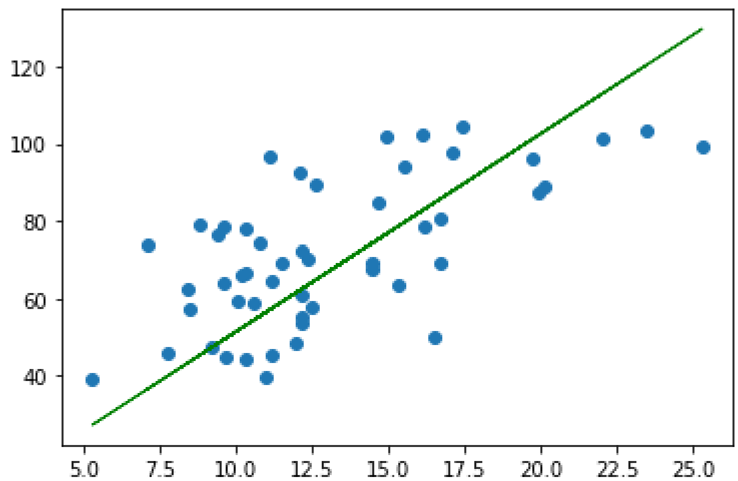
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

8) Taking the coeffient values for the new constant and the Y variable, create a scatterplot:

e.g. yhat = 0.1464*x + 0.25712 fig = plt.plot(x, yhat, lw=4, c='red', label = 'regression line')

```
x1 = data['povpct'].tolist()
y = data['brth18to19'].tolist()
plt.scatter(x1, y)
```

```
y = 5.123 * x + 0.1324

# plotting the regression line
plt.plot(x,y, lw = 1, c = 'green', label= 'regression line')
plt.show()
```

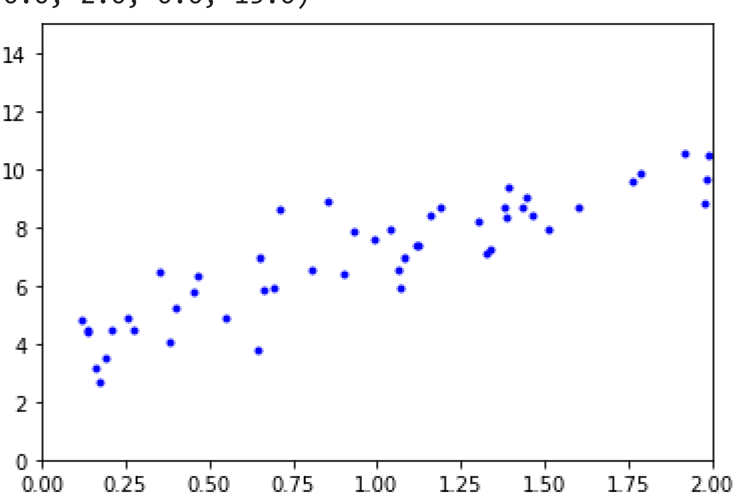


Problem 2: Implement code from lecture

1) Perform linear regression using the normal equation, as done in slides.

```
a = 2 * np.random.rand(50,1)
b = 4 + 3 * a + np.random.randn(50,1)


plt.plot(a, b, "b.")
plt.axis([0,2,0,15])
```

```
(0.0, 2.0, 0.0, 15.0)
```



```
a_b = np.c_[np.ones((50,1)), a]
theta_best = np.linalg.inv(a_b.T.dot(a_b)).dot(a_b.T).dot(b) # normal equation
```

```
theta_best
```

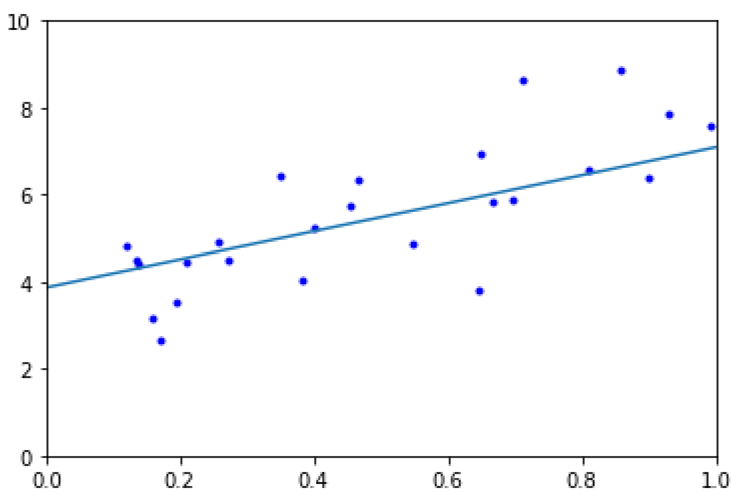
```
array([[3.86854428],
       [3.22018808]])
```

```
a_new = np.array([[0],[1]])
a_new_b = np.c_[np.ones((2, 1)), a_new]
b_predict = a_new_b.dot(theta_best)
b_predict
```

```
array([[3.86854428],
       [7.08873236]])
```

```
plt.axis([0,1,0,10])
plt.plot(a,b,"b.", b_predict)
```

```
[<matplotlib.lines.Line2D at 0x7fc721dd9a50>,
 <matplotlib.lines.Line2D at 0x7fc721de3b50>]
```



2) Perform linear regression using Scikit-Learn, as done in the slides.

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(a,b)
lin_reg.intercept_, lin_reg.coef_
```

```
(array([3.87012215]), array([[3.07711437]]))
```

Problem 3: Multivariate Linear Regression

In this problem we will continue using the poverty dataset. Do poverty and violent crimes affect teen pregnancy?

1) import the libraries you will need:

numpy pandas matplotlab.pyplot statsmodels.api

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import normalize
from mpl_toolkits.mplot3d import Axes3D
```

2) Import the dataset, poverty_2.csv, and print it.

```
data_2 = pd.read_csv('poverty_2.csv')
print(data_2)
```

```
    PovPct  ViolCrime  TeenBrth
0     20.1       11.2      54.5
1      7.1        9.1      39.5
2     16.1       10.4      61.2
3     14.9       10.4      59.9
4     16.7       11.2      41.1
5      8.8        5.8      47.0
6      9.7        4.6      25.8
7     10.3        3.5      46.3
8     22.0       65.0      69.1
9     16.2        7.3      44.5
10    12.1        9.5      55.7
11    10.3        4.7      38.2
12    14.5        4.1      39.1
13    12.4       10.3      42.2
14     9.6        8.0      44.6
15    12.2        1.8      32.5
16    10.8        6.2      43.0
17    14.7        7.2      51.0
18    19.7       17.0      58.1
19    11.2        2.0      25.4
20    10.1       11.8      35.4
21    11.0        3.6      23.3
22    12.2        8.5      34.8
23     9.2        3.9      27.5
24    23.5       12.9      64.7
25     9.4        8.8      44.1
```

```
26   15.3    3.0   36.4
27    9.6    2.9   37.0
28   11.1   10.7   53.9
29    5.3    1.8   20.0
30    7.8    5.1   26.8
31   25.3    8.8   62.4
32   16.5    8.5   29.5
33   12.6    9.4   52.2
34   12.0    0.9   27.2
35   11.5    5.4   39.5
36   17.1   12.2   58.0
37   11.2    4.1   36.8
38   12.2    6.3   31.6
39   10.6    3.3   35.6
40   19.9    7.9   53.0
41   14.5    1.8   38.0
42   15.5   10.6   54.3
43   17.4    9.0   64.4
44    8.4    3.9   36.8
45   10.3    2.2   24.2
46   10.2    7.6   37.6
47   12.5    5.1   33.0
48   16.7    4.9   45.5
49    8.5    4.3   32.3
50   12.2    2.1   39.9
```

3) We need to normalize the input variables.

```
data_2 = normalize(data_2, axis=0)
```

4) Split the data into input variables, X, and the output variable, Y.

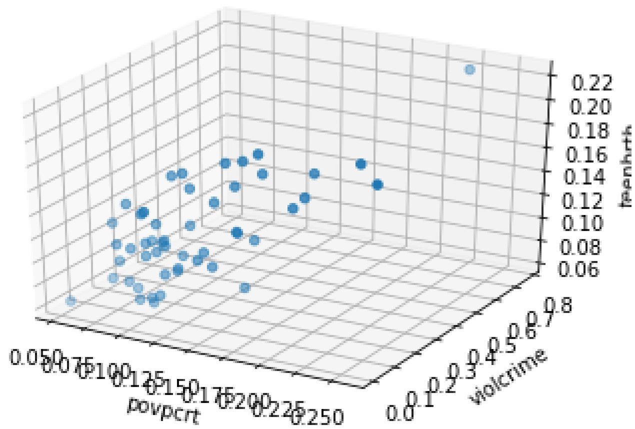
```
X = data_2[:, 0:2]
Y = data_2[:, 2:]
```

5) Graph the dataset with a seed of 42.

Replace the FILLINTHESEVALUES fields.

```
np.random.seed(42)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

xs = X[:, 0]
```

```
ys = X[:, 1]
zs = Y
ax.scatter(xs, ys, zs)

ax.set_xlabel('povpcrt')
ax.set_ylabel('violcrime')
ax.set_zlabel('teenbrth')

plt.show()
```



6) Implement Gradient Descent.

This section has be provided. Please run and understand the code.

```
# hyperparameters
learning_rate = 0.05
max_iteration = 500

#parameters
theta = np.zeros((data_2.shape[1], 1))


def hypothesis (theta, X) :
  tempX = np.ones((X.shape[0], X.shape[1] + 1))
  tempX[:,1:] = X
  return np.matmul(tempX, theta)


def loss (theta, X, Y) :
  return np.average(np.square(Y - hypothesis(theta, X))) / 2


def gradient (theta, X, Y) :
  tempX = np.ones((X.shape[0], X.shape[1] + 1))
  tempX[:,1:] = X
  d_theta = - np.average((Y - hypothesis(theta, X)) * tempX, axis= 0)
```

```
  d_theta = d_theta.reshape((d_theta.shape[0], 1))
```

```
def gradient_descent (theta, X, Y, learning_rate, max_iteration, gap) :
  cost = np.zeros(max_iteration)
  for i in range(max_iteration) :
    d_theta = gradient (theta, X, Y)
    theta = theta - learning_rate * d_theta
    cost[i] = loss(theta, X, Y)
    if i % gap == 0 :
      print ('iteration : ', i, ' loss : ', loss(theta, X, Y))
  return theta, cost
```

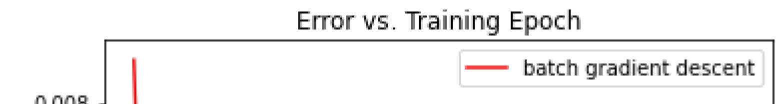
```
# Training model
theta, cost = gradient_descent (theta, X, Y, learning_rate, max_iteration, 100)
```

```
iteration :  0  loss :  0.008893757788504215
iteration :  100  loss :  0.0006811106575134702
iteration :  200  loss :  0.0006573219302696655
iteration :  300  loss :  0.0006360731168287809
iteration :  400  loss :  0.0006169026951758099
```

```
#optimal value is :
theta
```

```
array([[0.12381477],
       [0.04264512],
       [0.05698502]])
```

```
#plot cost
fig, ax = plt.subplots()
ax.plot(np.arange(max_iteration), cost, 'r')
ax.legend(loc='upper right', labels=['batch gradient descent'])
ax.set_xlabel('Iterations')
ax.set_ylabel('Cost')
ax.set_title('Error vs. Training Epoch')

plt.show()
```



7) Implement Stochastic Gradient Descent. Please run.

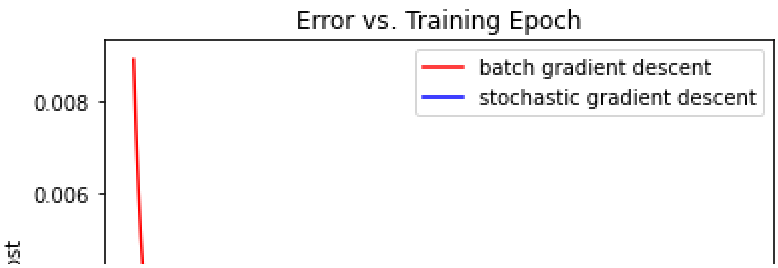
```
def stochastic_gradient_descent (theta, X, Y, learning_rate, max_iteration, gap) :
  cost = np.zeros(max_iteration)
  for i in range(max_iteration) :
    for j in range(X.shape[0]):
      d_theta = gradient (theta, X[j,:].reshape(1, X.shape[1]), Y[j,:].reshape(1, 1))
      theta = theta - learning_rate * d_theta

    cost[i] = loss(theta, X, Y)
    if i % gap == 0 :
      print ('iteration : ', i, ' loss : ', loss(theta, X, Y))
  return theta, cost
```

```
theta_stoc = np.zeros((data_2.shape[1], 1))
```

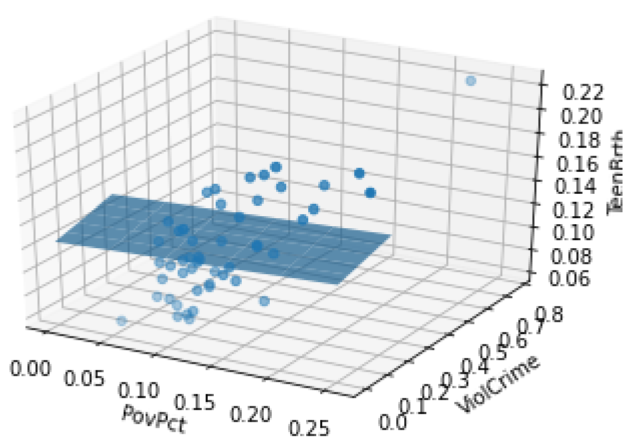
```
theta_stoc, cost_stoc = stochastic_gradient_descent (theta_stoc, X, Y, learning_rate, max_ite
```

```
iteration :  0  loss :  0.0007764556902156442
iteration :  100  loss :  0.0004037848207345314
iteration :  200  loss :  0.00036553095210465356
iteration :  300  loss :  0.000347847758744226
iteration :  400  loss :  0.00033956148785195
```

```
#plot the cost
fig, ax = plt.subplots()
ax.plot(np.arange(max_iteration), cost, 'r')
ax.plot(np.arange(max_iteration), cost_stoc, 'b')
#ax.plot(np.arange(max_iteration), mb_cost, 'g')
ax.legend(loc='upper right', labels=['batch gradient descent', 'stochastic gradient descent']
ax.set_xlabel('Iterations')
ax.set_ylabel('Cost')
ax.set_title('Error vs. Training Epoch')

plt.show()
```




```
np.random.seed(42)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

xs = X[:, 0]
ys = X[:, 1]
zs = Y
ax.scatter(xs, ys, zs)

ax.set_xlabel('PovPct')
ax.set_ylabel('ViolCrime')
ax.set_zlabel('TeenBrth')

# new
x = y = np.arange(0, 0.3, 0.05)
xp, yp = np.meshgrid(x, y)
z = np.array([hypothesis(theta, np.array([[x,y]]))[0, 0] for x,y in zip(np.ravel(xp), np.rave
zp = z.reshape(xp.shape)

ax.plot_surface(xp, yp, zp, alpha=0.7)

plt.show()
```



Problem 4, predict house price.

- import real_estate.csv
- Are there any null values in the dataset? Drop any missing data if exist.

- Create X as a 1-D array of the distance to the nearest MRT station, and y as the housing price
- What is the number of samples in the data set? To do this, you can look at the "shape" of X and y
- Split the data into train and test sets using sklearn's train_test_split, with test_size = 1/3
- Find the line of best fit using a Linear Regression and show the result of coefficients and intercept (you can use sklearn's linear regression)
- Using the predict method, make predictions for the test set and evaluate the performance (e.g., MSE or other metrics).

```
df = pd.read_csv("real_estate.csv")
print(df)
```

```
      No  X1 transaction date  ...  X6 longitude  Y house price of unit area
0      1             2012.917  ...     121.54024                        37.9
1      2             2012.917  ...     121.53951                        42.2
2      3             2013.583  ...     121.54391                        47.3
3      4             2013.500  ...     121.54391                        54.8
4      5             2012.833  ...     121.54245                        43.1
..   ...                  ...  ...           ...                         ...
409  410             2013.000  ...     121.50381                        15.4
410  411             2012.667  ...     121.54310                        50.0
411  412             2013.250  ...     121.53986                        40.6
412  413             2013.000  ...     121.54067                        52.5
413  414             2013.500  ...     121.54310                        63.9

[414 rows x 8 columns]
```

```
X = np.array (df[['X3 distance to the nearest MRT station']])
```

```
y1 = np.array(df[['Y house price of unit area']])
```

```
print (df['X1 transaction date'].isnull())
```

```
0      False
1      False
2      False
3      False
4      False
       ...
409    False
410    False
411    False
412    False
413    False
Name: X1 transaction date, Length: 414, dtype: bool
```

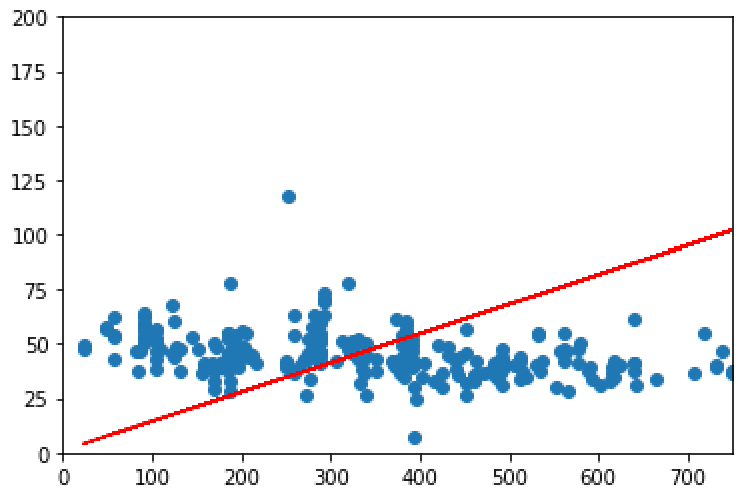
```
print (df.isnull().values.any())
```

```
False
```

```
print(df.shape)
```

```
(414, 8)
```

```
plt.axis([0,750,0,200])
plt.scatter(X, y1)

y1 = 0.1345 * X + 0.9818

# plotting the regression line
plt.plot(X,y1, lw = 1, c = 'red', label= 'regression line')
plt.show()
```




```
from sklearn.model_selection import train_test_split
X_train, X_test, y1_train, y1_test = train_test_split(X, y1, test_size = 0.33)
```

```
print(X_train.shape)
print(X_test.shape)
print(y1_train.shape)
print(y1_test.shape)
```

```
(277, 1)
(137, 1)
(277, 1)
(137, 1)
```

```
from sklearn.linear_model import LinearRegression
lin_reg2 = LinearRegression()
lin_reg2.fit(X_train,y1_train)
```

```
LinearRegression()
```

```
lin_reg2.intercept_,lin_reg2.coef_
```

```
(array([0.9818]), array([[0.1345]]))
```

```
X_pred = lin_reg2.predict(X_test)
print(X_pred)
```

```
[157.652511  ]
[ 18.60056025]
[ 35.707817  ]
[ 65.63564065]
[549.663521  ]
[134.9109013 ]
[ 39.8959856 ]
[  4.12679198]
[ 13.14814007]
[ 13.14814007]
[201.100046  ]
[ 66.72799585]
[ 52.6112928 ]
[ 80.05269565]
[ 67.18690985]
[155.1630505 ]
[ 50.91485775]
[ 28.44273225]
[ 23.86413705]
[415.227005  ]
[ 13.14814007]
[ 97.59230265]
[ 17.62522005]
[183.9204955 ]
[ 43.8239774 ]
[ 22.1796994 ]
[246.8907055 ]
[ 52.03228375]
[240.81951   ]
[547.9377515 ]
[ 39.8959856 ]
[ 26.19816935]
[ 39.8959856 ]
[ 15.07875845]
[ 26.06740845]
[ 30.14582505]
[ 53.8753507 ]
[191.2773765 ]
[ 39.8959856 ]
[260.7291415 ]
[289.803872  ]
[ 15.07875845]
```

```
 [ 67.18690985]
 [156.928363  ]
 [ 27.01891525]
 [200.7273465 ]
 [183.9204955 ]
 [565.5252405 ]
 [352.7618425 ]
 [ 53.5132498 ]
 [ 23.94513295]
 [ 34.6916695 ]
 [ 65.91276445]
 [261.8780405 ]
 [303.867461  ]
 [ 15.76946975]
 [ 17.3539739 ]
 [ 25.1183361 ]]
```

```
y1_pred = lin_reg2.predict(y1_test)
print(y1_pred)
```

```
 [  5.81117908]
 [ 36.61828328]
 [  9.28416698]
 [ 83.02084185]
 [ 14.81428043]
 [ 16.26847603]
 [  7.98014216]
 [  6.34781006]
 [ 12.26782086]
 [ 10.01843937]
 [ 22.03363428]
 [  4.6158441 ]
 [ 27.29561766]
 [  8.21329654]
 [  8.12874633]
 [ 32.92385624]
 [ 25.71910664]
 [ 10.76456993]
 [ 33.37202409]
 [ 74.67942758]
 [  2.75022484]
 [ 15.87425511]
 [  4.50545378]
 [  7.08317692]
 [ 27.13810585]
 [ 11.56932013]
 [ 69.34837562]
 [ 23.99313398]
 [  3.31590949]
 [ 18.56641355]
 [ 32.09579314]
 [ 41.8519735 ]
 [  2.75022484]
 [ 10.76456993]
 [ 39.96042078]
 [  5.78450139]
```

```
[  8.79396281]
[  2.01223291]
[ 27.44960187]
[ 11.2802922 ]
[ 39.96042078]
[ 22.08866482]
[  3.84429856]
[ 13.14501889]
[ 12.7049826 ]
[  4.46831992]
[ 34.89256388]
[ 42.50454309]
[  8.27170634]
[ 45.79034756]
[ 74.67942758]
[  6.34781006]
[ 39.14728214]
[  6.41425555]
[ 40.6433639 ]
[ 12.22378738]
[  4.82899247]
[ 10.01843937]
```

```
print(np.mean(X_train))
print(np.mean(X_test))
print(np.mean(y1_train))
print(np.mean(y1_test))
```

```
1043.8485331768954
1164.8367264233577
141.37942771229243
157.65233970394164
```

```
mean_squared_error(X_test,X_pred)
```

```
2142740.609563325
```

```
mean_squared_error(y1_test,y1_pred)
```

```
38762.71331215295
```

✓ 0s completed at 8:59 PM