Assignment2

February 18, 2022

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

1 Problem 1: Univariate Linear Regression

1.1 1) import the libraries you will need:

numpy pandas matplotlab.pyplot statsmodels.api

[1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

1.2 2) Import the date poverty.csv dataset

[2]:
```python
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving real_estate.csv to real_estate (4).csv
```

1.3 3) Print the dataset indexed upon the location column.

```
[ ]: poverty = pd.read_csv("poverty.csv",index_col = 0)
     poverty
```

```
[ ]:                       PovPct  Brth15to17  Brth18to19  ViolCrime  TeenBrth
     Location
     Alabama                 20.1        31.5        88.7       11.2      54.5
     Alaska                   7.1        18.9        73.7        9.1      39.5
     Arizona                 16.1        35.0       102.5       10.4      61.2
     Arkansas                14.9        31.6       101.7       10.4      59.9
     California              16.7        22.6        69.1       11.2      41.1
     Colorado                 8.8        26.2        79.1        5.8      47.0
     Connecticut              9.7        14.1        45.1        4.6      25.8
     Delaware                10.3        24.7        77.8        3.5      46.3
     District_of_Columbia    22.0        44.8       101.5       65.0      69.1
     Florida                 16.2        23.2        78.4        7.3      44.5
     Georgia                 12.1        31.4        92.8        9.5      55.7
     Hawaii                  10.3        17.7        66.4        4.7      38.2
     Idaho                   14.5        18.4        69.1        4.1      39.1
     Illinois                12.4        23.4        70.5       10.3      42.2
     Indiana                  9.6        22.6        78.5        8.0      44.6
     Iowa                    12.2        16.4        55.4        1.8      32.5
     Kansas                  10.8        21.4        74.2        6.2      43.0
     Kentucky                14.7        26.5        84.8        7.2      51.0
     Louisiana               19.7        31.7        96.1       17.0      58.1
     Maine                   11.2        11.9        45.2        2.0      25.4
     Maryland                10.1        20.0        59.6       11.8      35.4
     Massachusetts           11.0        12.5        39.6        3.6      23.3
     Michigan                12.2        18.0        60.8        8.5      34.8
     Minnesota                9.2        14.2        47.3        3.9      27.5
     Mississippi             23.5        37.6       103.3       12.9      64.7
     Missouri                 9.4        22.2        76.6        8.8      44.1
     Montana                 15.3        17.8        63.3        3.0      36.4
     Nebraska                 9.6        18.3        64.2        2.9      37.0
     Nevada                  11.1        28.0        96.7       10.7      53.9
     New_Hampshire            5.3         8.1        39.0        1.8      20.0
     New_Jersey               7.8        14.7        46.1        5.1      26.8
     New_Mexico              25.3        37.8        99.5        8.8      62.4
     New_York                16.5        15.7        50.1        8.5      29.5
     North_Carolina          12.6        28.6        89.3        9.4      52.2
     North_Dakota            12.0        11.7        48.7        0.9      27.2
     Ohio                    11.5        20.1        69.4        5.4      39.5
     Oklahoma                17.1        30.1        97.6       12.2      58.0
     Oregon                  11.2        18.2        64.8        4.1      36.8
     Pennsylvania            12.2        17.2        53.7        6.3      31.6
     Rhode_Island            10.6        19.6        59.0        3.3      35.6
     South_Carolina          19.9        29.2        87.2        7.9      53.0
```

```
South_Dakota         14.5        17.3        67.8        1.8       38.0
Tennessee            15.5        28.2        94.2       10.6       54.3
Texas                17.4        38.2       104.3        9.0       64.4
Utah                  8.4        17.8        62.4        3.9       36.8
Vermont              10.3        10.4        44.4        2.2       24.2
Virginia             10.2        19.0        66.0        7.6       37.6
Washington           12.5        16.8        57.6        5.1       33.0
West_Virginia        16.7        21.5        80.7        4.9       45.5
Wisconsin             8.5        15.9        57.1        4.3       32.3
Wyoming              12.2        17.7        72.1        2.1       39.9
```

1.4 4) Get useful descriptive statistial data on the dataset.

Hint: this is a single line, data._____

[ ]:
```
poverty.describe()
```

[ ]:
```
          PovPct  Brth15to17  Brth18to19  ViolCrime   TeenBrth
count  51.000000   51.000000   51.000000  51.000000  51.000000
mean   13.117647   22.282353   72.019608   7.854902  42.243137
std     4.277228    8.043499   18.975563   8.914131  12.318511
min     5.300000    8.100000   39.000000   0.900000  20.000000
25%    10.250000   17.250000   58.300000   3.900000  33.900000
50%    12.200000   20.000000   69.400000   6.300000  39.500000
75%    15.800000   28.100000   87.950000   9.450000  52.600000
max    25.300000   44.800000  104.300000  65.000000  69.100000
```

1.5 5) Print the columns

[ ]:
```
print(poverty.columns)
```

```
Index(['PovPct', 'Brth15to17', 'Brth18to19', 'ViolCrime', 'TeenBrth'],
dtype='object')
```

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

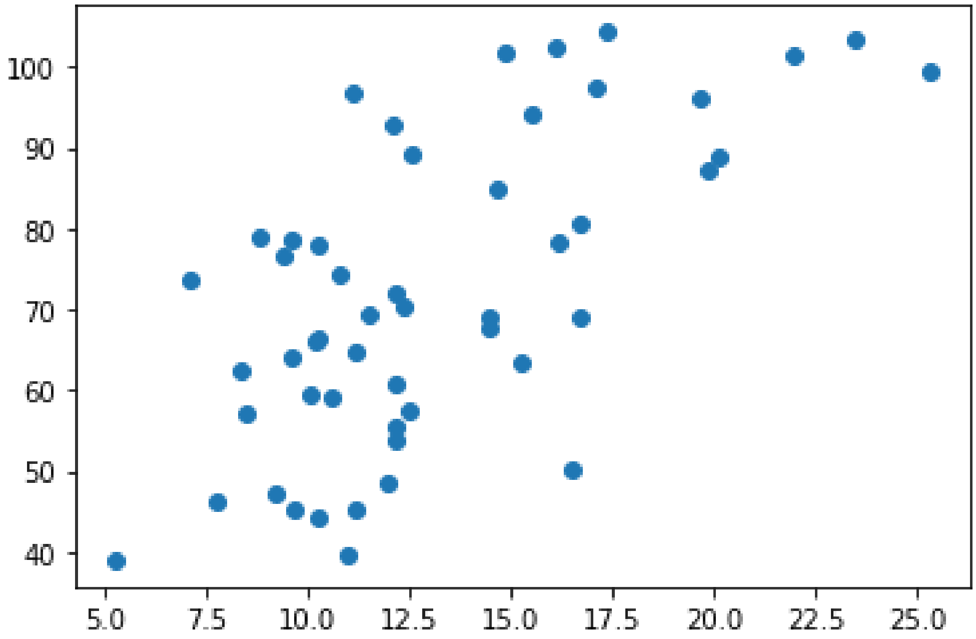
PovPct Brth18to19

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.

[ ]:
```
plt.scatter(poverty.PovPct, poverty.Brth18to19)
```

[ ]:
```
<matplotlib.collections.PathCollection at 0x7efc09f48d50>
```



1.7 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(poverty.PovPct)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:117:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

```
[ ]: results = sm.OLS(poverty.Brth18to19,x1).fit()
```

```
[ ]: results.summary()
```

```
[ ]: <class 'statsmodels.iolib.summary.Summary'>
     """
                                 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:                Thu, 17 Feb 2022   Prob (F-statistic):           2.50e-07
     Time:                        02:35:21   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.
"""
```

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

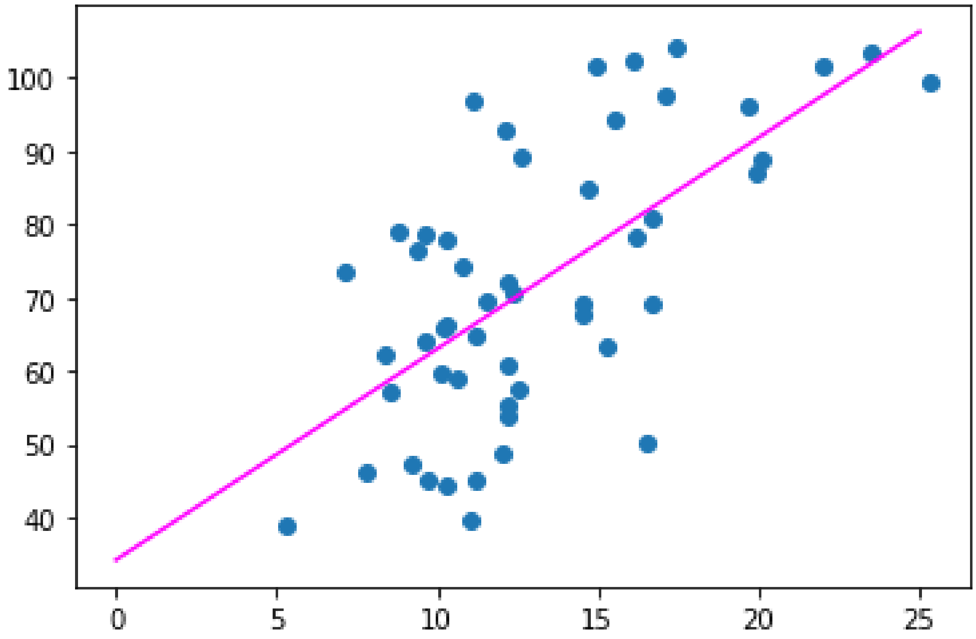
e.g. yhat = 0.1464*x + 0.25712 fig = plt.plot(x, yhat, lw=4, c='red', label = 'regression line')

[ ]:

```
plt.scatter(poverty.PovPct, poverty.Brth18to19)
plt.plot([0,25],[34.2124,2.8822*25+34.2124], color="magenta")
```

[ ]:

```
[<matplotlib.lines.Line2D at 0x7fb63c15ab10>]
```



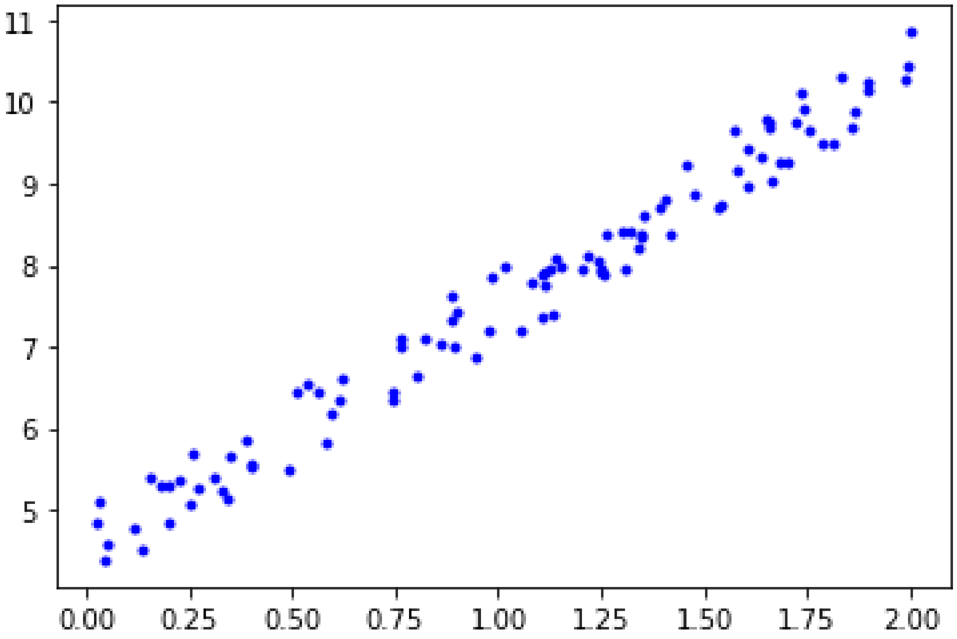
2 Problem 2: Implement code from lecture

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

[ ]:
```
X = 2* np.random.rand(100,1)
y=4+3*X + np.random.rand(100,1)
```

[ ]:
```
plt.plot(X,y,"b.")
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7fb641086710>]
```



[ ]:

```
X_b = np.c_[np.ones((100,1)),X]
```

[ ]:

```
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

[ ]:

```
theta_best
```

[ ]:

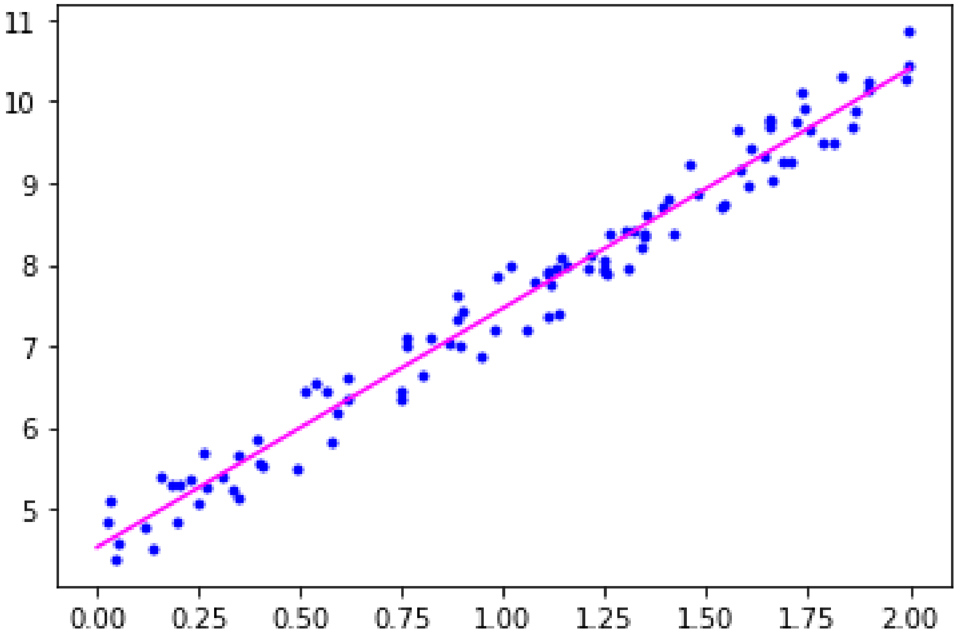
```
array([[4.52769162],
       [2.93813808]])
```

[ ]:

```
plt.plot(X,y,"b.")
plt.plot([0,2],[theta_best[0],theta_best[1]*2+theta_best[0]], color="magenta")
```

[ ]:

```
[<matplotlib.lines.Line2D at 0x7fb63be61190>]
```



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

[ ]:
```
from sklearn.linear_model import LinearRegression as lr
```

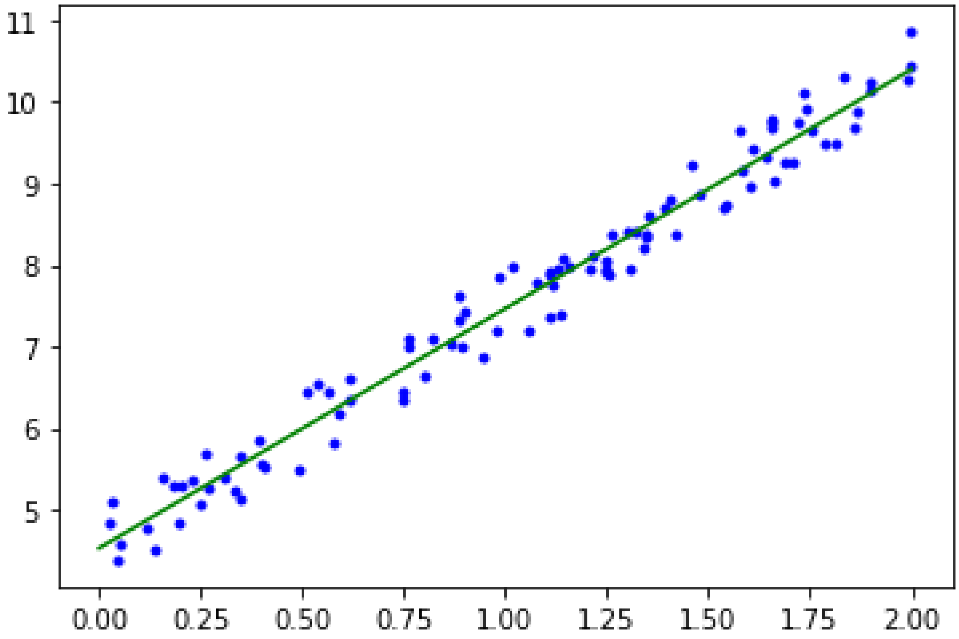
[ ]:
```
lin_reg = lr()
lin_reg.fit(X,y)
lin_reg.intercept_, lin_reg.coef_
```

[ ]:
```
(array([4.52769162]), array([[2.93813808]]))
```

[ ]:
```
plt.plot(X,y,"b.")
plt.plot([0,2],[lin_reg.intercept_,lin_reg.intercept_+2*lin_reg.coef_],␣
 ↪color="green")
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:65:
VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  ary = asanyarray(ary)
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7fb63bf92d50>]
```



3 Problem 3: Multivariate Linear Regression

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

3.1 1) import the libraries you will need:

numpy pandas matplotlab.pyplot statsmodels.api

[ ]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import normalize
```

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

[ ]:

```
poverty2 = pd.read_csv("poverty_2.csv")
poverty2
```

[ ]:

```
     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

3.3 3) We need to normalize the input variables.

[ ]:

```
poverty2 = normalize(poverty2)
```

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

[ ]:

```
X = poverty2[:,0:2]
Y = poverty2[:,2:]
x=X
y=Y
```

3.5 5) Graph the dataset with a seed of 42.

Replace the FILLINTHESEVALUES fields.

[ ]:

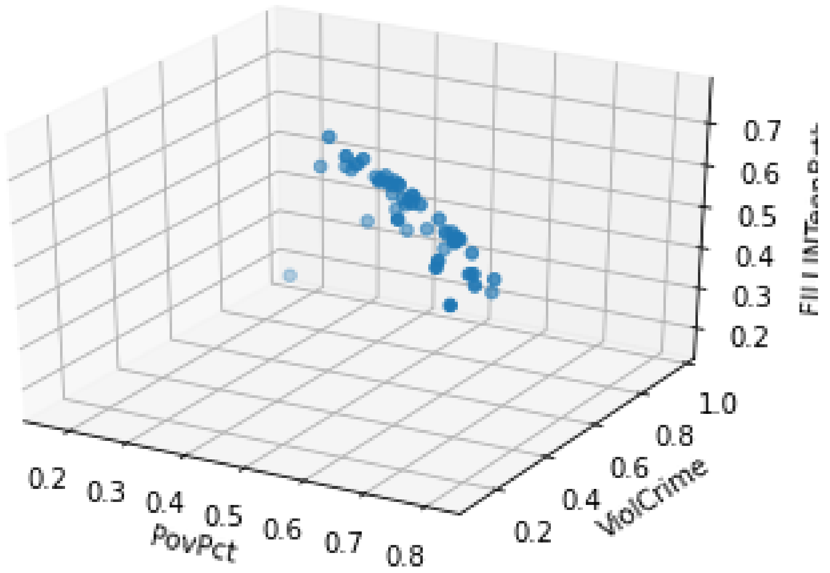
```
np.random.seed(42)

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('PovPct')
ax.set_ylabel('ViolCrime')
ax.set_zlabel('FILLINTeenBrth')

plt.show()
```



3.6 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((poverty2.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))
  return d_theta
```

[ ]:
```
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 :  0.14248615838353396
iteration :  100  loss :  0.005841062708110685
iteration :  200  loss :  0.005374637890296291
iteration :  300  loss :  0.005059239296919674
iteration :  400  loss :  0.0048406904218634165
```

[ ]:

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

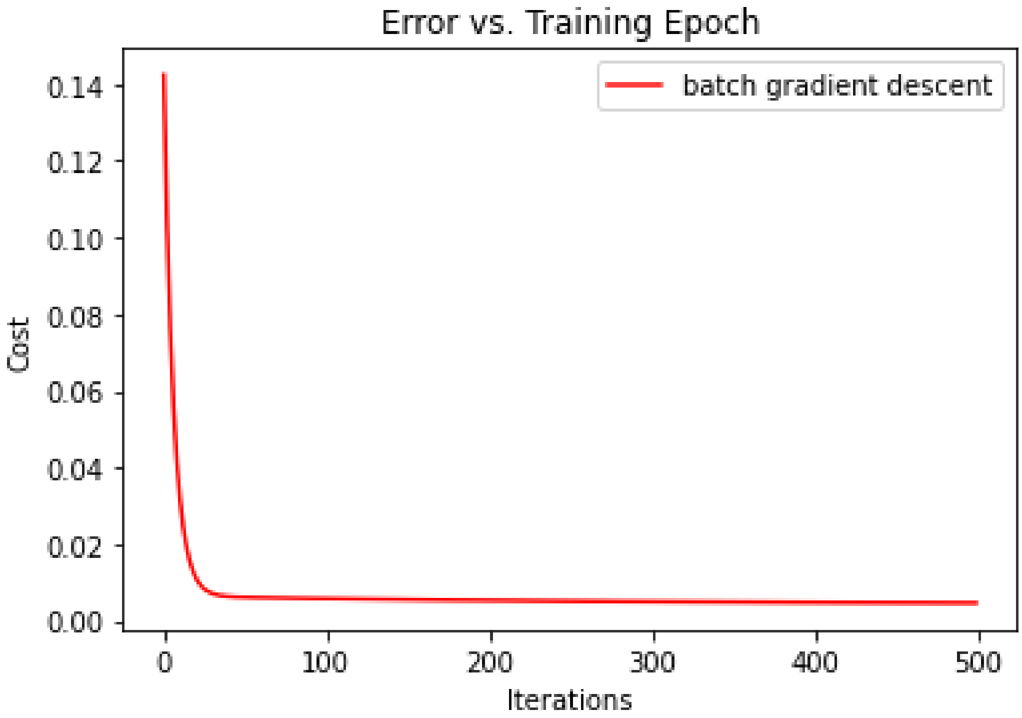
[ ]:

```
array([[0.41375839],
       [0.26569717],
       [0.00707897]])
```

[ ]:

```
#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()
```



3.7 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((poverty2.shape[1], 1))
```

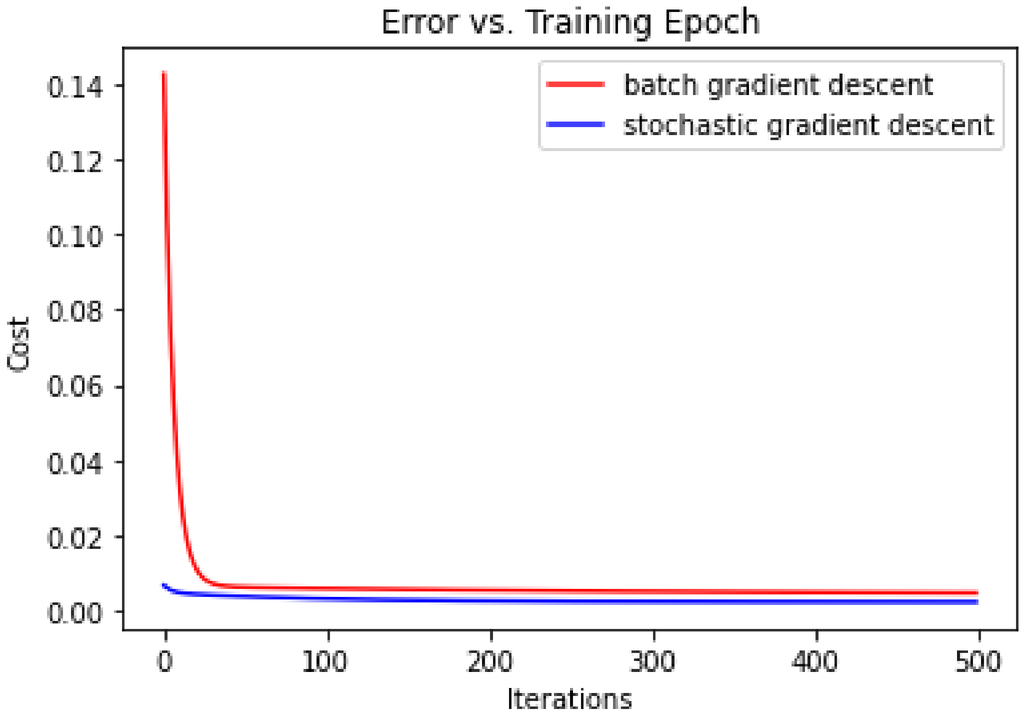
```
[ ]: theta_stoc, cost_stoc = stochastic_gradient_descent (theta_stoc, X, Y,␣
     ↪learning_rate, max_iteration, 100)
```

```
iteration :  0  loss :  0.0066577245043739405
iteration :  100  loss :  0.003102327706993443
iteration :  200  loss :  0.002532377208293092
iteration :  300  loss :  0.0023333911770596814
iteration :  400  loss :  0.0022626837845736957
```

[ ]:

```
#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'])#, 'mini-batch 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
```

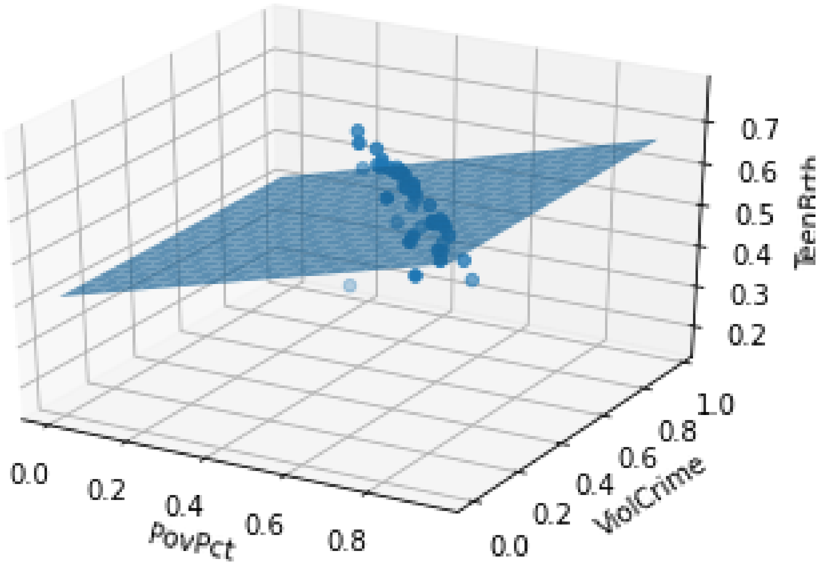
```
ax.scatter(xs, ys, zs)

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

# new
x = y = np.arange(0, 1, 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.ravel(yp))])
zp = z.reshape(xp.shape)

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

plt.show()
```



4 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).

[3]:
```
import sklearn as sk
import sklearn.model_selection as ms
import sklearn.metrics as me
```

[4]:
```
realEstate = pd.read_csv("real_estate.csv",index_col=0)
realEstate
```

[4]:
```
     X1 transaction date  ...  Y house price of unit area
No                        ...
1                2012.917  ...                        37.9
2                2012.917  ...                        42.2
3                2013.583  ...                        47.3
4                2013.500  ...                        54.8
5                2012.833  ...                        43.1
..                    ...  ...                         ...
410              2013.000  ...                        15.4
411              2012.667  ...                        50.0
412              2013.250  ...                        40.6
413              2013.000  ...                        52.5
414              2013.500  ...                        63.9

[414 rows x 7 columns]
```

[5]:
```
realEstate.dropna() ##No Missing data
```

[5]:
```
     X1 transaction date  ...  Y house price of unit area
No                        ...
1                2012.917  ...                        37.9
2                2012.917  ...                        42.2
3                2013.583  ...                        47.3
4                2013.500  ...                        54.8
5                2012.833  ...                        43.1
..                    ...  ...                         ...
410              2013.000  ...                        15.4
411              2012.667  ...                        50.0
412              2013.250  ...                        40.6
413              2013.000  ...                        52.5
414              2013.500  ...                        63.9

[414 rows x 7 columns]
```

[6]:
```
data=realEstate[["X3 distance to the nearest MRT station","Y house price of␣
 ↪unit area"]]
```

[7]:
```
data.shape
```

[7]: (414, 2)

[67]:
```
x = data[["X3 distance to the nearest MRT station"]]
y = data["Y house price of unit area"].to_numpy()
```

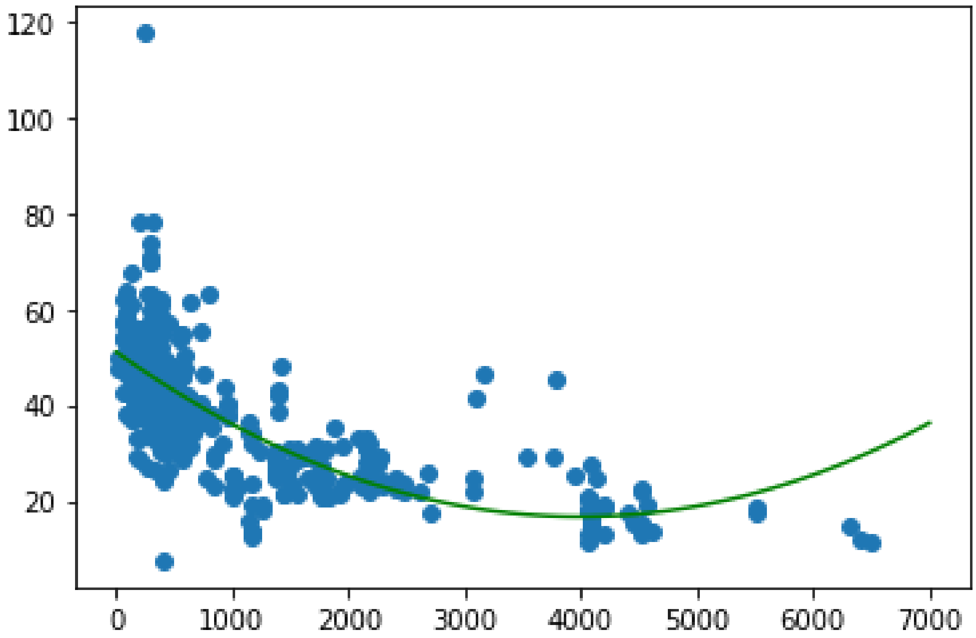
[11]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

[109]:
```
x_train,x_val,y_train,y_val = train_test_split(x,y,test_size=1/
 ↪3,random_state=42)
poly_features = PolynomialFeatures(degree=2,include_bias=False)
x_poly = poly_features.fit_transform(x_train)

lin_reg = LinearRegression()
lin_reg.fit(x_poly,y_train)
xp = np.linspace(0,7000,100).reshape(100,1)
xp_poly = poly_features.transform(xp)
yp = lin_reg.predict(xp_poly)
plt.plot(xp,yp,"green")
plt.scatter(x,y)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does
not have valid feature names, but PolynomialFeatures was fitted with feature
names
  "X does not have valid feature names, but"
```

[109]: <matplotlib.collections.PathCollection at 0x7f578280b050>



[110]:

```
x_val_poly = poly_features.transform(x_val)
y_val_predict = lin_reg.predict(x_val_poly)
me.mean_squared_error(y_val,y_val_predict)
```

[110]:

```
82.27443641911879
```

[112]:

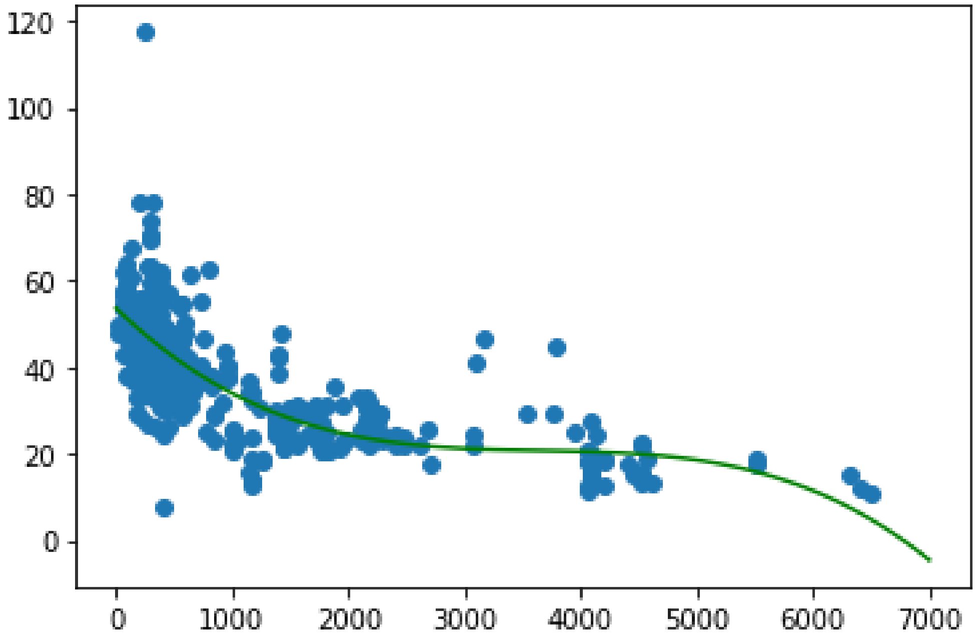
```
x_train,x_val,y_train,y_val = train_test_split(x,y,test_size=1/
 ↪3,random_state=42)
poly_features = PolynomialFeatures(degree=3,include_bias=False)
x_poly = poly_features.fit_transform(x_train)

lin_reg = LinearRegression()
lin_reg.fit(x_poly,y_train)
xp = np.linspace(0,7000,100).reshape(100,1)
xp_poly = poly_features.transform(xp)
yp = lin_reg.predict(xp_poly)
plt.plot(xp,yp,"green")
plt.scatter(x,y)
x_val_poly = poly_features.transform(x_val)
y_val_predict = lin_reg.predict(x_val_poly)
me.mean_squared_error(y_val,y_val_predict)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does
not have valid feature names, but PolynomialFeatures was fitted with feature
names
  "X does not have valid feature names, but"
```

[112]:

```
73.4172288261123
```



[113]:

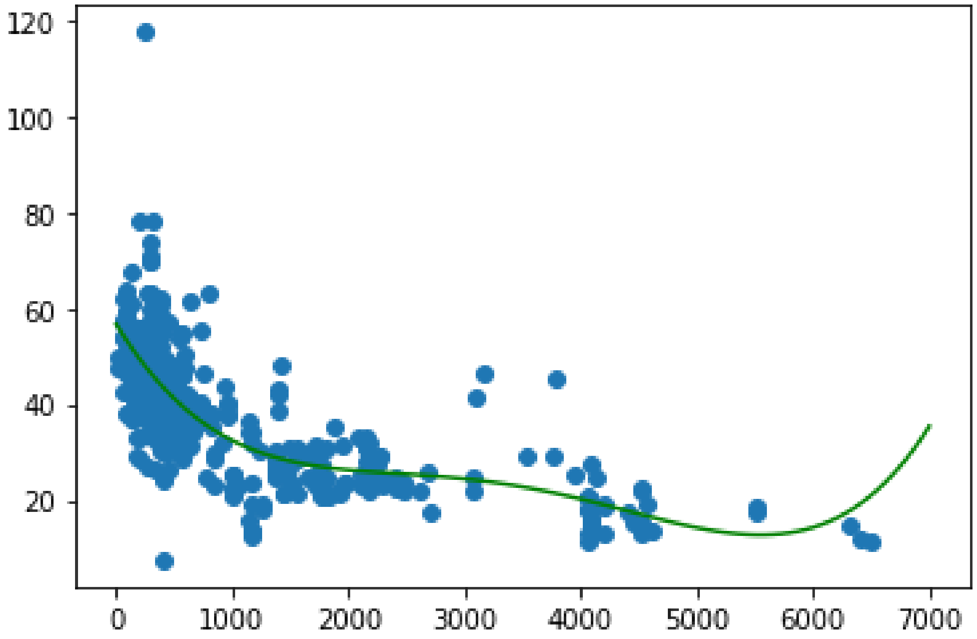
```
x_train,x_val,y_train,y_val = train_test_split(x,y,test_size=1/
 ↪3,random_state=42)
poly_features = PolynomialFeatures(degree=4,include_bias=False)
x_poly = poly_features.fit_transform(x_train)

lin_reg = LinearRegression()
lin_reg.fit(x_poly,y_train)
xp = np.linspace(0,7000,100).reshape(100,1)
xp_poly = poly_features.transform(xp)
yp = lin_reg.predict(xp_poly)
plt.plot(xp,yp,"green")
plt.scatter(x,y)
x_val_poly = poly_features.transform(x_val)
y_val_predict = lin_reg.predict(x_val_poly)
me.mean_squared_error(y_val,y_val_predict)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does
not have valid feature names, but PolynomialFeatures was fitted with feature
names
  "X does not have valid feature names, but"
```

[113]:

```
72.16393207172295
```



Quartic has the least squared error, got larger for 5 or 6 powered, MSE is 72.1639

[122]:
```
def colab_pdf(file_name, notebookpath="/content/drive/MyDrive/"):
    import os

    # Checking if file_name passed is a string.
    if not isinstance(file_name, str):
        raise TypeError(
            f"expected a string as file_name, but got {type(file_name)} instead.
 ↪"
        )

    # Using the defaults used by google.colab
    drive_mount_point = "/content/drive/"
    gdrive_home = os.path.join(drive_mount_point, "My Drive/")

    # If the drive is not already mounted, attempt to mount it.
    if not os.path.isdir(gdrive_home):
        from google.colab import drive

        drive.mount(drive_mount_point)

    # Check if the notebook exists in the Drive.
    if not os.path.isfile(os.path.join(notebookpath, file_name)):
        raise ValueError(f"file '{file_name}' not found in path␣
 ↪'{notebookpath}'.")
```

```
    # Installing all the recommended packages.
    get_ipython().system(
        "apt update >> /dev/null && apt install texlive-xetex␣
↪texlive-fonts-recommended texlive-generic-recommended >> /dev/null"
    )

    # If pdf with the same name exists, remove it.
    pdf_file = os.path.join(gdrive_home, file_name.split(".")[0] + ".pdf")

    if os.path.isfile(pdf_file):
        os.remove(pdf_file)

    # Attempt to convert to pdf and save it in Gdrive home dir using jupyter␣
↪nbconvert command.
    try:
        get_ipython().system(
            "jupyter nbconvert --output-dir='$gdrive_home'␣
↪'$notebookpath''$file_name' --to pdf"
        )
    except:
        return "nbconvert error"

    # Attempt to download the file to system.
    try:
        from google.colab import files

        file_name = file_name.split(".")[0] + ".pdf"
        files.download(gdrive_home + file_name)
    except:
        return "File Download Unsuccessful. Saved in Google Drive"

    return "File ready to be Downloaded and Saved to Drive"
```

[ ]:

```
colab_pdf("Assignment2.ipynb")
```

```
Mounted at /content/drive/

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```

[ ]:

```
```