1. Load Data & Perform General EDA

```
# I. import libraries
import pandas as pd
import numpy as np

%matplotlib inline
import matplotlib.pyplot as plt

import seaborn as sns

import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

from scipy import stats
import missingno as msno

# II. import the data to a dataframe and show number of rows & cols
df = pd.read_csv('ecommerce.csv')
print("Number of rows: " + str(df.shape[0]))
print("Number of columns:" + str(df.shape[1]))
```

```
Number of rows: 500
Number of columns:9
```

```
#III. Show the top 5 rows
df.head()
```

```
   Unnamed: 0         Email
Address  \
0           0  adkv@ota.com              89280 Mark Lane\nNew John,
MN 16131
1           1  gjun@syj.com  363 Amanda Cliff Apt. 638\nWest Angela,
KS 31437
2           2  qjyr@pkk.com          62008 Adam Lodge\nLake Pamela,
NY 30677
3           3  jkiu@xsb.com        950 Tami Island\nLake Aimeeview,
MT 93614
4           4  stvb@niy.com      08254 Kelly Squares\nNorth Lauren,
AR 78382

           Credit Card  Avg. Session Length  Time on App  Time on
Website  \
0     3544288738428794            35.497268    13.655651
40.577668
1     6546228325389133            32.926272    12.109461
38.268959
2  4406395951712628314            34.000915    12.330278
```

```
38.110597
3      30334036663133            35.305557    14.717514
37.721283
4    3582080469154498            34.330673    13.795189
38.536653

   Length of Membership  Yearly Amount Spent
0              4.582621           588.951054
1              3.164034           393.204933
2              4.604543           488.547505
3              3.620179           582.852344
4              4.946308           600.406092
```

```
#III. Show the last 5 rows
df.tail()
```

```
     Unnamed: 0         Email  \
495         495  xskz@gwj.com
496         496  awrc@iok.com
497         497  pndt@jyr.com
498         498  zvtz@onj.com
499         499  phqb@nlg.com

                                               Address          Credit
Card  \
495            7083 Wallace Rest\nNew Trevor, NM 70240
30206742023085
496  663 Christopher Garden\nLake Carrieberg, PA 70796
6011536844623717
497            1555 Chen Road\nBergerchester, NH 46418
4086276267550896697
498  5568 Robert Station Apt. 030\nTurnerstad, GA 9...
36218092488069
499       424 Mark Junctions\nDarrellchester, TX 09088
5427200269739116

     Avg. Session Length  Time on App  Time on Website  Length of
Membership  \
495            34.237660    14.566160        37.417985
4.246573
496            35.702529    12.695736        38.190268
4.076526
497            33.646777    12.499409        39.332576
5.458264
498            34.322501    13.391423        37.840086
2.836485
499            34.715981    13.418808        36.771016
3.235160

     Yearly Amount Spent
```

```
495          574.847438
496          530.049004
497          552.620145
498          457.469510
499          498.778642
```

```
# IV. call the describe method of dataframe to see summary statistics of the numerical columnsnew
new_df = df.select_dtypes(include = np.number)
new_df.describe()
```

```
       Unnamed: 0   Credit Card  Avg. Session Length  Time on App  \
count  500.000000  5.000000e+02           500.000000   500.000000   
mean   249.500000  3.706324e+17            34.053194    13.052488   
std    144.481833  1.235588e+18             0.992563     0.994216   
min      0.000000  5.018057e+11            30.532429     9.508152   
25%    124.750000  3.683275e+13            33.341822    12.388153   
50%    249.500000  3.513612e+15            34.082008    12.983231   
75%    374.250000  4.777131e+15            34.711985    13.753850   
max    499.000000  4.959148e+18            37.139662    16.126994   

       Time on Website  Length of Membership  Yearly Amount Spent  
count       500.000000            500.000000           500.000000  
mean         38.060445              4.033462           500.314038  
std           1.010489              0.999278            79.314782  
min          34.913847              0.769901           257.670582  
25%          37.349257              3.430450           446.038277  
50%          38.069367              4.033975           499.887875  
75%          38.716432              4.626502           550.313828  
max          41.005182              7.422689           766.518462  
```

Explain in words about the description of any two variables.

Credit card is not very reliable as numeric value, as its numbers are only used for identification, not quantitative purposes, same goes for Email and Address

Minimum length of membership can be less than a whole year (values have decimals, are not strictly ints)

```
# V. Missing value analysis

#Show a list with column wise count of missing values and display the list in count wise descending order
df.isnull().sum().sort_values(ascending = False)
```

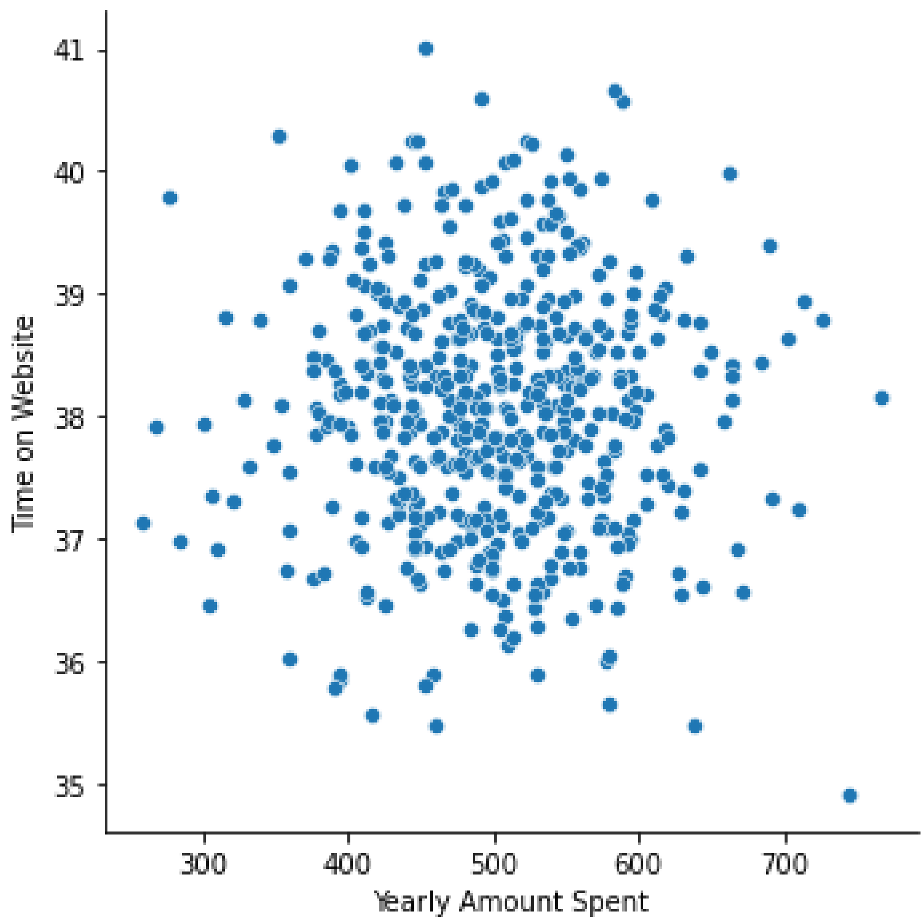
```
Unnamed: 0              0
Email                   0
Address                 0
Credit Card             0
Avg. Session Length     0
Time on App             0
```

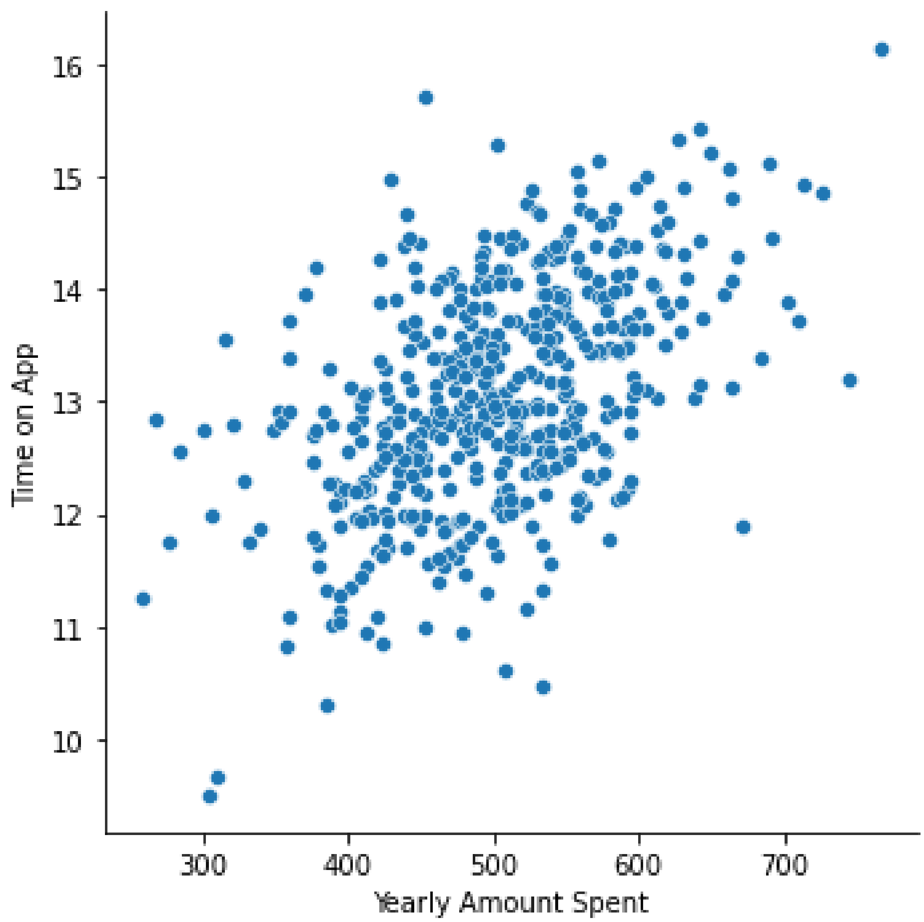
```
Time on Website         0
Length of Membership    0
Yearly Amount Spent     0
dtype: int64
```

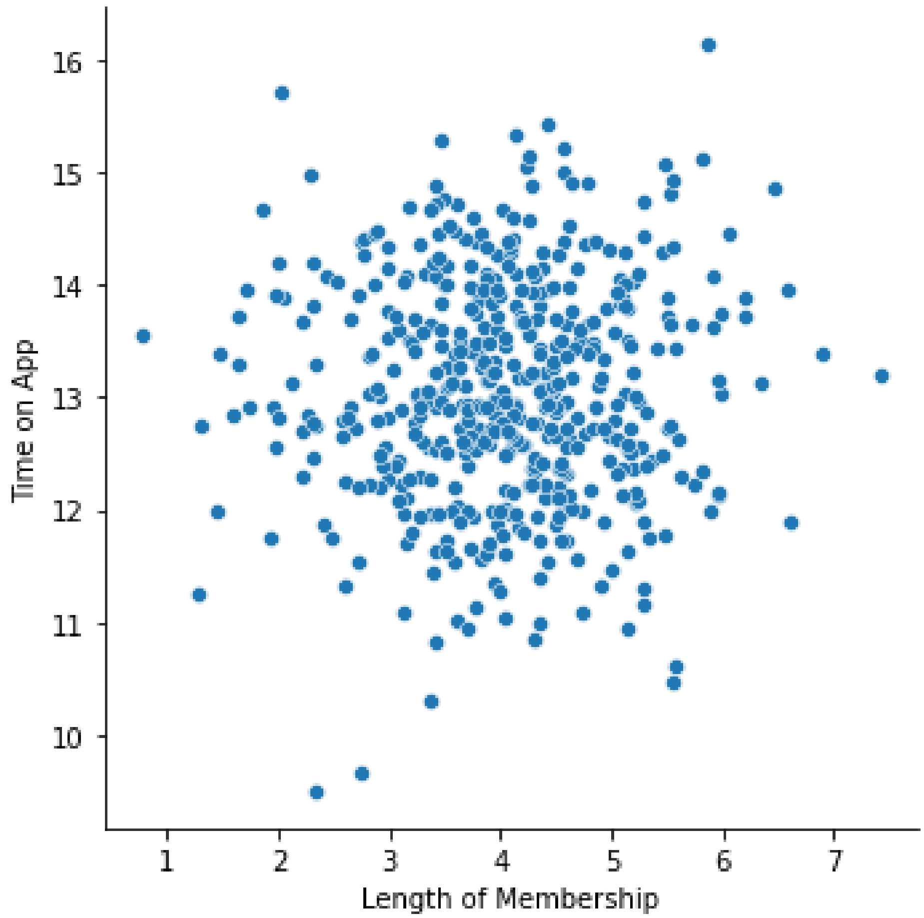
No missing values!

```
# VI. PLot various scatter plots to understand the data
    # Yearly amount spent vs time on website
sns.relplot(data = df, y = "Time on Website", x = "Yearly Amount Spent")
    # Yearly amount spent vs time on app
sns.relplot(data = df, y = "Time on App", x = "Yearly Amount Spent")
    # Length of membership vs time on app
sns.relplot(data = df, y = "Time on App", x = "Length of Membership")

    # IV. Seaborn pairplot
```

```
<seaborn.axisgrid.FacetGrid at 0x278922a3040>
```




16
15
14
13
12
11
10
Time on App
300
400
500
600
700
Yearly Amount Spent

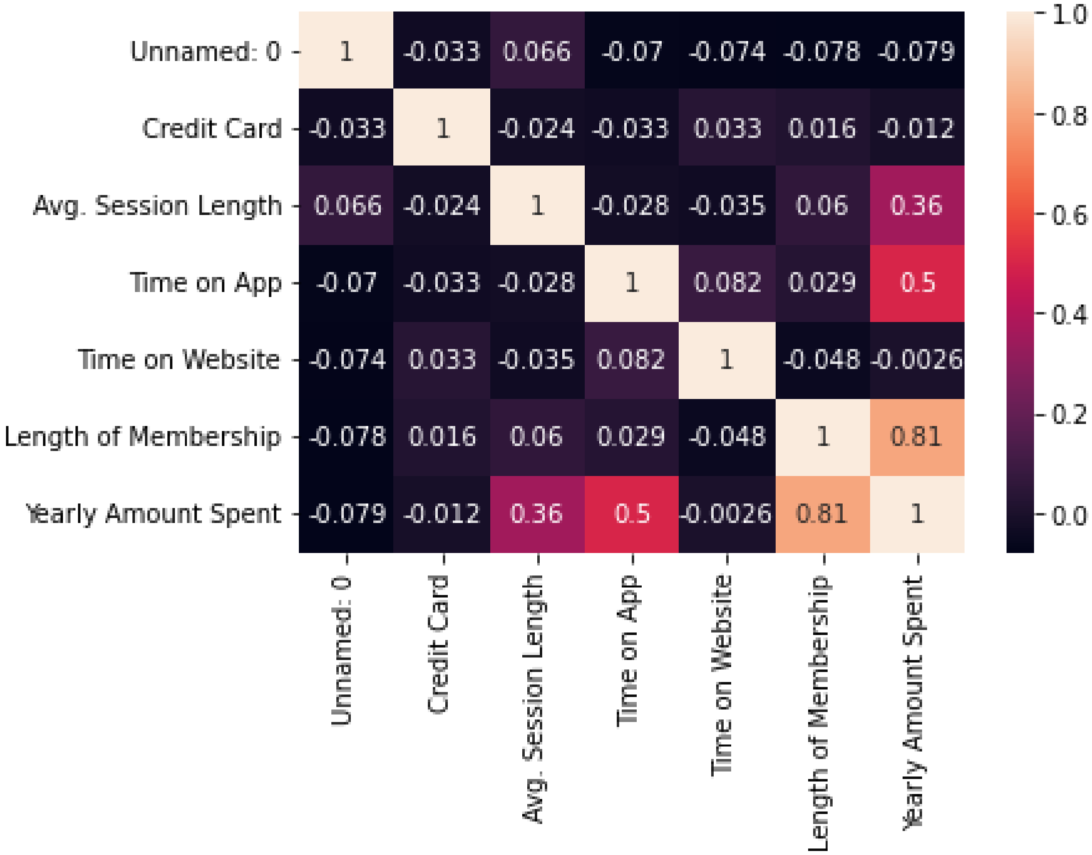


Based on the plots, what feature is mostly correlated with the yearly amount spent?

Time on app (middle plot shows positive correlation)

```
# V. Plot sns heatmap based on correlation (annot = True)
sns.heatmap(df.corr(), annot = True)
```

```
<AxesSubplot:>
```



Which columns must be removed based on the above plot?

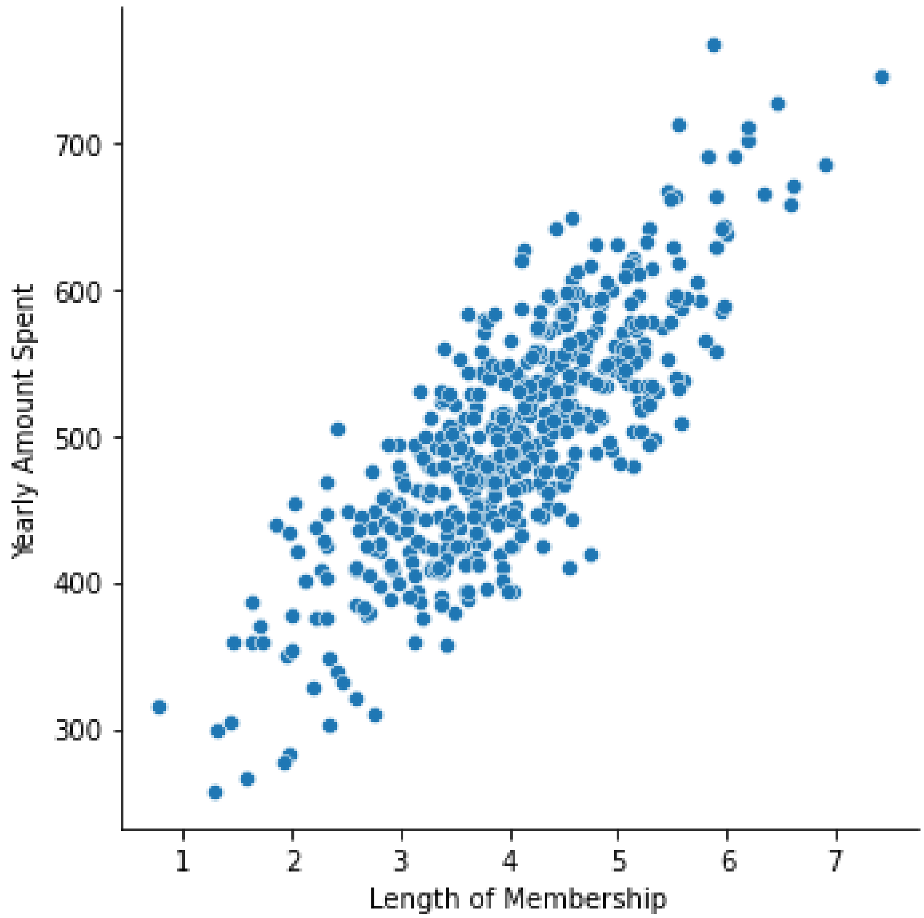
Unnamed (obviously), Credit Card, and Time on Website

Which column is the most interesting and related to yearly amount spent?

Length of Membership

```
#VI. Generate a scatter plot based on the column from the above
question against Yearly amount spent
sns.relplot(data = df, y = "Yearly Amount Spent", x = "Length of
Membership")
```

```
<seaborn.axisgrid.FacetGrid at 0x278923a8dc0>
```



2. Feature Selection & Pre-processing

```
# Drop unneccesary columns based on EDA & null analysis
df = df.drop(labels = ["Unnamed: 0", "Credit Card", "Time on Website",
"Email", "Address" ], axis = 1)
```

3. X/Y & Train/Test Split

```
# I. Use sklearn's StandardScaler
X = df[["Length of Membership"]]
y = df[["Yearly Amount Spent"]]

scaler = StandardScaler()
scaler.fit(X)

StandardScaler()

# II. Split data into train & test sets using sklearn's
train_test_split
# 30% of the data should be in the test set, with random_stat = 101
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =
.3, random_state = 101)

#Use sklearn's StandardScaler for scaling the X of training & test
```

```
sets. But not for y (target) train and test
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

4. Training Linear Model using SKLearn's LinearRegression

```
# I. Train a linear model using SKLearn LinearRegression
lr = LinearRegression()
# fit to training data
lr.fit(X_train_scaled,y_train)
```

```
LinearRegression()
```

```
# II. Show coefficient and intercept after training
print("Coefficient: ", lr.coef_)
print("Intercept: ", lr.intercept_)
```

```
Coefficient:  [[63.04345407]]
Intercept:  [499.20662197]
```

```
#III. Predict for the test data
y_pred = lr.predict(X_test_scaled)
print("Predicted value: ", y_pred , sep = "\n")
```

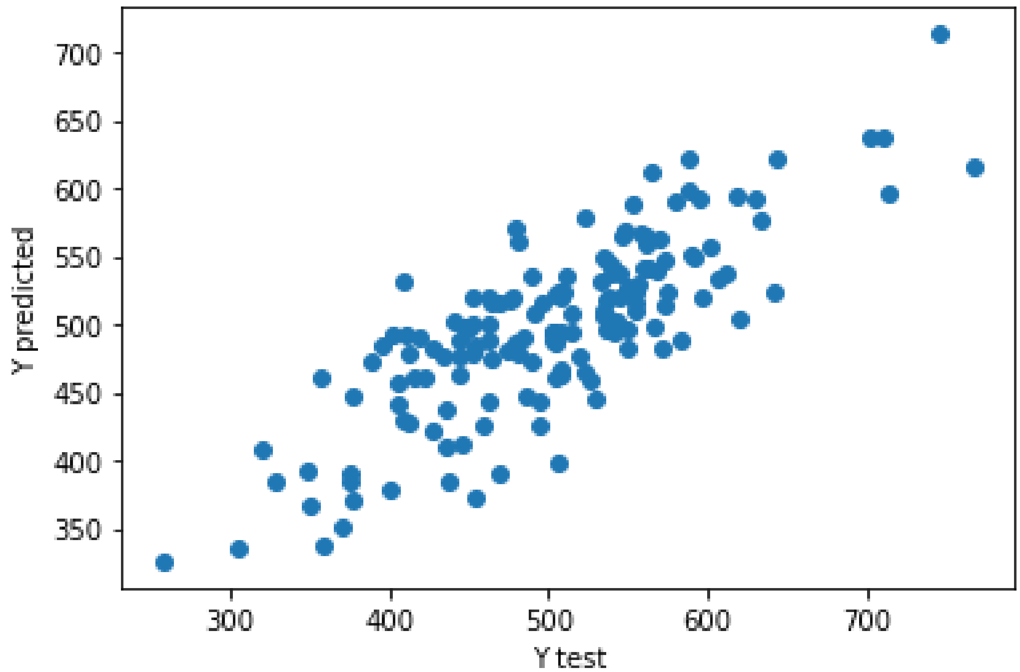
```
Predicted value: 
[[371.83579601]
 [492.96075167]
 [492.60079547]
 [556.85492114]
 [597.18095979]
 [517.76796611]
 [550.33474479]
 [596.0338348 ]
 [475.60618837]
 [549.76249278]
 [383.88613297]
 [426.30464324]
 [520.52313863]
 [442.00103108]
 [615.6676527 ]
 [542.71690157]
 [636.40350294]
 [495.01031549]
 [498.08419864]
 [513.64378058]
 [531.60128609]
 [527.06059832]
 [390.37124373]
 [507.00733385]
 [547.47234306]
 [479.13717947]
 [495.32653849]
```

```
[351.87390205]
[494.71253898]
[487.57296173]
[532.74594446]
[524.2877641 ]
[442.36699929]
[562.87881802]
[493.48109308]
[564.01579963]
[495.28814651]
[477.49162934]
[385.12216808]
[443.32544672]
[621.02972961]
[499.14698155]
[576.05646422]
[523.45641518]
[462.83013518]
[477.17641891]
[590.88744087]
[501.45389117]
[460.55042748]
[481.41539221]
[514.88365584]
[522.68590693]
[325.92061651]
[535.2716955 ]
[577.86347433]
[335.20258383]
[523.09376666]
[472.3953404 ]
[485.5027917 ]
[422.15386179]
[503.24826379]
[549.55491254]
[713.24387886]
[564.05981572]
[509.08710624]
[502.05935578]
[438.14968491]
[521.90541947]
[622.51228922]
[520.04457376]
[483.43131747]
[384.41581829]
[589.18608861]
[378.58323357]
[447.17330242]
[461.81729975]
[478.30763542]
```

```
[478.26156732]
[496.33751782]
[636.35381819]
[459.65017958]
[594.97481064]
[514.8912009 ]
[482.78527799]
[538.66116671]
[520.67054656]
[536.99018342]
[569.74952768]
[482.02057831]
[476.81858223]
[466.47569445]
[541.02304457]
[427.56789559]
[503.78341247]
[489.30599233]
[528.87954382]
[494.04640067]
[500.41840011]
[425.59555291]
[518.80827303]
[337.23131847]
[490.87436792]
[611.67394155]
[506.02931025]
[486.75791916]
[490.27619922]
[494.32951925]
[485.42448353]
[531.73307991]
[411.16873023]
[569.29276541]
[392.72622454]
[483.68634942]
[536.60049151]
[592.12620997]
[445.64237952]
[558.90132399]
[508.19869247]
[536.01671895]
[390.31754637]
[367.22186288]
[592.28102189]
[465.24964993]
[547.74130946]
[488.96311454]
[409.77808911]
[520.13521596]
```

```
 [397.35812755]
 [519.54227632]
 [461.19896249]
 [407.65817334]
 [462.52828029]
 [509.82634537]
 [499.43790585]
 [516.25162347]
 [370.4297317 ]
 [457.36987765]
 [517.68300759]
 [541.67284787]
 [460.86911302]
 [515.04592685]
 [536.95997195]
 [566.27814108]
 [488.11616301]
 [523.52529657]
 [561.13753352]
 [473.50669572]
 [519.81675571]
 [428.93996415]
 [446.54401758]]
```

```
#IV. Generate a scatter plot that shows the Y-test on x- axis and y-
pridected in y- axis
plt.scatter(y_test,y_pred, marker = 'o')
plt.plot()
plt.xlabel("Y test")
plt.ylabel("Y predicted")
plt.show()
```




```
#V. Use sklearn's metrics to print the value of MAE, MSE, RMSE, and
R^2
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import r2_score

# print MAE (mean absolute error)
print("MAE: ", mean_absolute_error(y_test, y_pred))

# print MSE (mean squared error)
print("MSE: ", mean_squared_error(y_test, y_pred))

# print RMSE (root mean squared error)
print("RMSE: ", sqrt(mean_squared_error(y_test, y_pred)))

# print r^2
print("R^2: ", r2_score(y_test,y_pred))
```

```
MAE:  40.90606096474333
MSE:  2551.5330908757346
RMSE:  50.51270227255452
R^2:  0.6484902730789126
```

```
# VI. Interpret the coefficient & which coefficient belongs to which
feature and explain any strategy that should help the business
```

5. Normal Equation

```
X_train_scaled.shape
```

```
(350, 1)
```

```
from sklearn.datasets import make_regression
#x, y = make_regression(n_samples = 100, n_features = 1, n_informative
= 1, noise = 10, random_state = 10)
x = X_train_scaled
# convert target variable array from 1d to 2d
y = y_train
#  adding x0= 1 to each new instance
x_new = np.array([np.ones(len(x)),x.flatten()]).T

theta_best_values =
np.linalg.inv(x_new.T.dot(x_new)).dot(x_new.T).dot(y)
```

The coefficient we got from linear regression is very close to the 2nd theta value, but the intercept does not line up

```
# prepare the test set before prediction
x_sample_new = np.array([np.ones(len(x)), x.flatten()]).T

#perform prediction for the test set
predict_value = x_sample_new.dot(theta_best_values)
predict_value
```

```
array([[516.74946326],
       [512.13789889],
       [570.52971241],
       [444.67342164],
       [531.91447696],
       [533.90205498],
       [438.47416049],
       [480.00767455],
       [485.65167522],
       [362.11657464],
       [478.49700147],
       [536.83228067],
       [505.57733377],
       [573.89317962],
       [478.32499421],
       [577.51799451],
       [498.29539089],
       [546.88814013],
       [541.19574367],
       [457.43054334],
       [473.64116992],
       [474.11442385],
       [477.03186791],
       [530.06540802],
       [569.47347086],
       [526.73822383],
       [485.10402344],
```

```
[534.13641664],
[459.77102297],
[516.19771925],
[529.89915615],
[537.32143882],
[573.95617557],
[544.89862536],
[572.48027948],
[448.29001218],
[507.39392424],
[659.81484678],
[440.40088902],
[513.25217029],
[391.36935146],
[499.84354331],
[495.89748211],
[432.43435895],
[531.27427157],
[612.03641777],
[556.58627228],
[485.04289092],
[518.1970066 ],
[532.19193601],
[503.79307294],
[480.10385152],
[470.06910897],
[369.67498365],
[569.0440582 ],
[490.61170194],
[421.78773911],
[403.5141523 ],
[387.69679972],
[550.82890638],
[444.60888948],
[466.99980402],
[439.43345216],
[491.69432975],
[606.137612  ],
[594.22835262],
[477.87154264],
[474.17355069],
[564.93710419],
[575.56497879],
[459.86716351],
[449.04727126],
[503.68645124],
[519.21438712],
[432.15722296],
[425.16856323],
[499.16873213],
```

```
[454.15608167],
[474.32877629],
[481.81774211],
[451.79801534],
[472.3042265 ],
[427.78097393],
[451.87413269],
[505.78959188],
[480.46754332],
[416.51080514],
[498.72318265],
[519.29643204],
[549.53552369],
[566.55693597],
[515.95861738],
[458.89110429],
[479.30128418],
[488.46757679],
[514.05373448],
[478.57448502],
[448.09416125],
[412.44901634],
[396.93970911],
[578.50939448],
[501.92623184],
[515.5788601 ],
[418.30764399],
[533.58123029],
[530.52963485],
[427.32086559],
[573.64869156],
[463.76621688],
[366.29408113],
[530.42606608],
[489.07972895],
[540.23548897],
[598.97815616],
[414.3723906 ],
[515.08754562],
[534.21667897],
[519.4792612 ],
[429.96004395],
[589.84781027],
[439.88952078],
[506.52120265],
[494.08668376],
[536.53739824],
[444.30035355],
[529.19702238],
[466.94936522],
```

```
[457.76381921],
[443.55036554],
[499.07657558],
[347.45281116],
[423.61487472],
[617.86309263],
[493.7522494 ],
[447.94435092],
[569.89876756],
[522.63722103],
[415.93006119],
[529.87270011],
[459.53293285],
[607.90721382],
[471.31641128],
[515.44246279],
[555.59539826],
[617.42409804],
[487.36310302],
[417.84674798],
[542.215401  ],
[571.92991979],
[533.61488434],
[564.74293053],
[653.50685497],
[512.99104615],
[590.46247838],
[508.51436125],
[551.99589309],
[487.3409148 ],
[457.5978572 ],
[547.46912497],
[528.78237193],
[422.60858084],
[569.60478579],
[680.31249104],
[504.69476661],
[594.79135829],
[529.8521872 ],
[533.83558197],
[568.61087276],
[644.90159143],
[553.71900818],
[459.8360354 ],
[400.91307017],
[543.71849344],
[450.91982782],
[482.93053615],
[497.39869578],
[471.17121882],
```

```
[596.95037739],
[371.03242655],
[451.5935847 ],
[475.93392006],
[460.71908077],
[369.62806047],
[493.27846019],
[448.79207948],
[562.26569522],
[455.89982765],
[457.49748294],
[620.52214449],
[544.58430514],
[459.09473326],
[581.43505009],
[498.10710782],
[392.14776782],
[600.5252132 ],
[504.69410986],
[415.55216832],
[493.74750646],
[574.39350008],
[470.96973103],
[500.7445138 ],
[408.35774294],
[466.34037198],
[391.04891389],
[523.26276193],
[579.59232841],
[432.63649484],
[558.36894924],
[536.91489142],
[353.61870735],
[582.52845119],
[627.59024287],
[490.39317602],
[465.28775242],
[388.79837888],
[488.35400325],
[578.8321427 ],
[540.87844006],
[487.63825513],
[505.56176695],
[523.5733219 ],
[374.52458759],
[528.26895491],
[468.62416107],
[497.53346818],
[586.78971029],
[520.89395436],
```

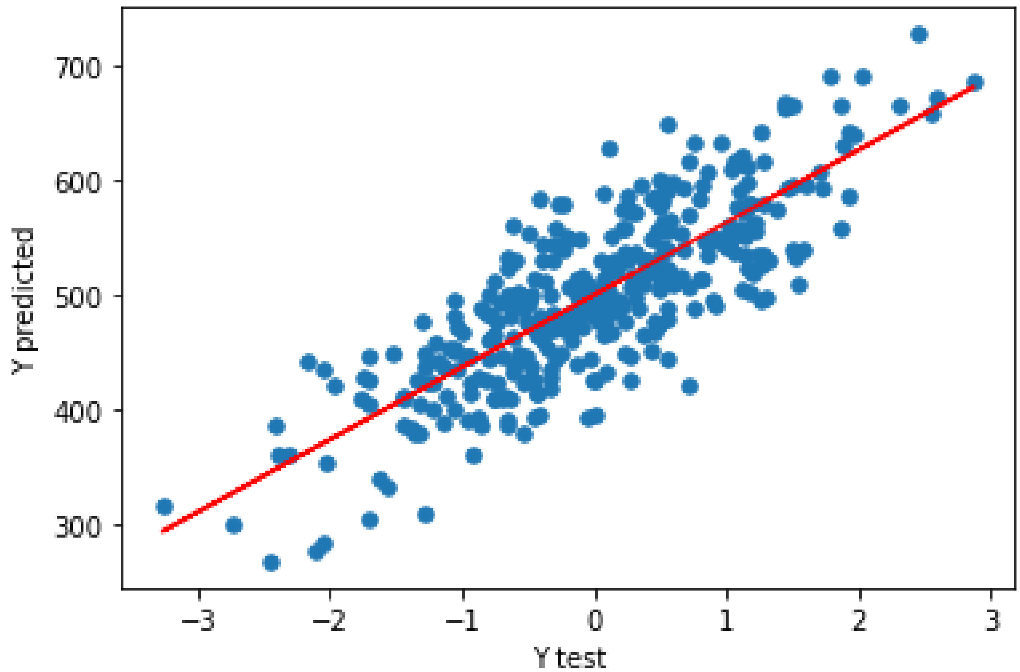
```
[441.40363249],
[554.14326035],
[575.84361797],
[502.47153357],
[591.83714856],
[450.47839273],
[572.51755296],
[504.18160201],
[550.75884372],
[559.61427826],
[413.85989672],
[484.00816293],
[491.80192455],
[594.34504338],
[527.56259737],
[433.59665004],
[344.55465113],
[463.27530274],
[408.34953081],
[473.10689753],
[523.83770468],
[452.22136811],
[483.16625527],
[492.05071603],
[528.49623123],
[438.6743931 ],
[495.32008158],
[524.74544822],
[468.45648415],
[326.67817867],
[433.41527021],
[347.99696718],
[468.71997747],
[535.82004119],
[507.7804234 ],
[407.81819819],
[466.16211079],
[499.35445575],
[520.61336052],
[490.3977391 ],
[567.85023199],
[516.46040152],
[499.30936252],
[432.84287297],
[469.45602746],
[509.80152214],
[473.7611343 ],
[484.99620076],
[479.9613702 ],
[505.26155317],
```

```
[616.53167241],
[562.98220809],
[471.12298159],
[482.88866472],
[620.49095974],
[526.07351548],
[457.20828829],
[449.94552962],
[533.88723658],
[514.23423421],
[454.76249083],
[549.62744696],
[568.20760453],
[461.4489025 ],
[293.10553483],
[391.22476534],
[419.24262214],
[574.73812817],
[463.60005358],
[500.1901863 ],
[528.73041285],
[496.5309749 ],
[556.09489154],
[533.08494988],
[503.04839529],
[466.8859643 ],
[561.53103098],
[579.35746604],
[566.18935891],
[593.33327769],
[460.90345436],
[500.9593307 ],
[567.53509747],
[465.81684232],
[553.2645222 ],
[455.28309177],
[514.33205022],
[535.36717845],
[570.6189587 ],
[505.61874787],
[465.70408321],
[418.70174059],
[527.56201674],
[526.81008279],
[437.9083782 ],
[578.62983337],
[463.40545999],
[522.17219842],
[511.1486825 ],
[453.14037034],
```

```
       [478.04669502],
       [475.81695809],
       [534.31100113],
       [662.24906849],
       [622.98785494],
       [488.36673726],
       [529.67915205],
       [508.37176738],
       [435.96488447],
       [428.43631257],
       [468.46150905],
       [512.66508098],
       [432.27376729],
       [571.06209553],
       [530.84112068],
       [479.542234  ],
       [473.13153523],
       [488.66033944],
       [491.70583496],
       [544.06688076],
       [463.92191445],
       [510.55808484],
       [579.7167859 ]])
```

```
# generate a scatter lpot that shows Y-test on x axis and y pred on y
axis
plt.scatter(x, y, s=30, marker = 'o')
plt.plot(x,predict_value, c = 'red')
plt.xlabel("Y test")
plt.ylabel("Y predicted")
```

```
Text(0, 0.5, 'Y predicted')
```




```
# use sklearn's metrics to print MAE, MSE, RMSE, and R^2

# print MAE (mean absolute error)
print("MAE: ", mean_absolute_error(x,predict_value))

# print MSE (mean squared error)
print("MSE: ", mean_squared_error(x,predict_value))

# print RMSE (root mean squared error)
print("RMSE: ", sqrt(mean_squared_error(x,predict_value)))

# print r^2
print("R^2: ", r2_score(x,predict_value))
```

```
MAE:  499.7149238163755
MSE:  253443.1390209243
RMSE:  503.4313647568299
R^2:  -261685.28878517618
```

What is the limitation of using the Normal Equation for regression?

Since the normal equation creates an inverse of a roughly n x n matrix, the algorithm for inverting the matrix runs O(n^2) so this method becomes computationally slow as the data set becomes larger.

6. Batch Gradient Descent

```
y_pred = np.empty(y_pred.shape)
# implement batch gradient descent
```

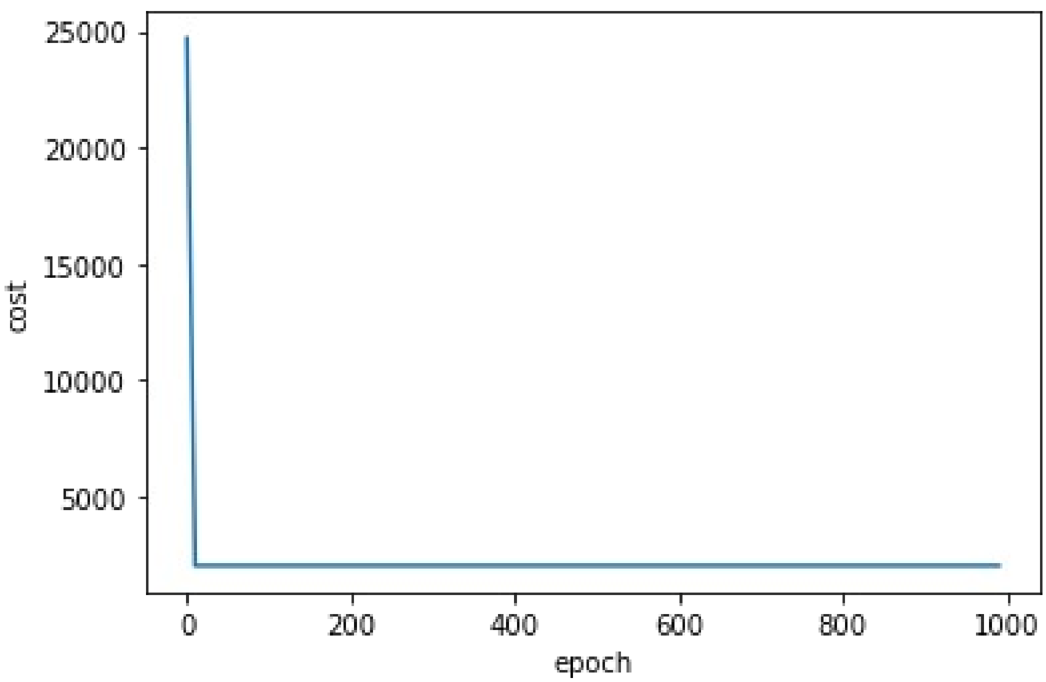
```python
cost_list = []
epoch_list = []
pred_list = []

eta = .1
n_iterations = 1000
m = 100
X_b = np.array([np.ones(len(x)),x.flatten()]).T
theta = np.random.randn(2,1)
for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta)- y)
    theta = theta - eta * gradients
    y_pred = np.dot(theta.T, x_new.T).reshape((350,1))
    # Calculate mean squared error (MSE)
    cost = np.mean(np.square(y_train-y_pred))

    if (iteration % 10 == 0):
        cost_list.append(cost)
        epoch_list.append(iteration)

#Display the theta values.
#Are they very close to the sklearn's linear regression?
theta
```

```
array([[499.20662197],
       [ 63.04345407]])
```

The theta values are very close to the ones obtained in linear regression!

```python
# plot step numbr (in x-axis) against the cost(y axis)

plt.xlabel("epoch")
plt.ylabel("cost")
plt.plot(epoch_list,cost_list)
```

```
[<matplotlib.lines.Line2D at 0x27891b5adc0>]
```




```
#Prepare the test set before prediction
X_b =
np.array([np.ones(len(X_test_scaled)),X_test_scaled.flatten()]).T

# scale y data
y_train_scaled = scaler.transform(y_train)
y_test_scaled = scaler.transform(y_test)

#Perform prediction for the test set
theta = np.random.randn(2,1)
for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta)- y_test_scaled)
    theta = theta - eta * gradients
    y_pred = np.dot(theta.T, X_b.T)
y_pred
```

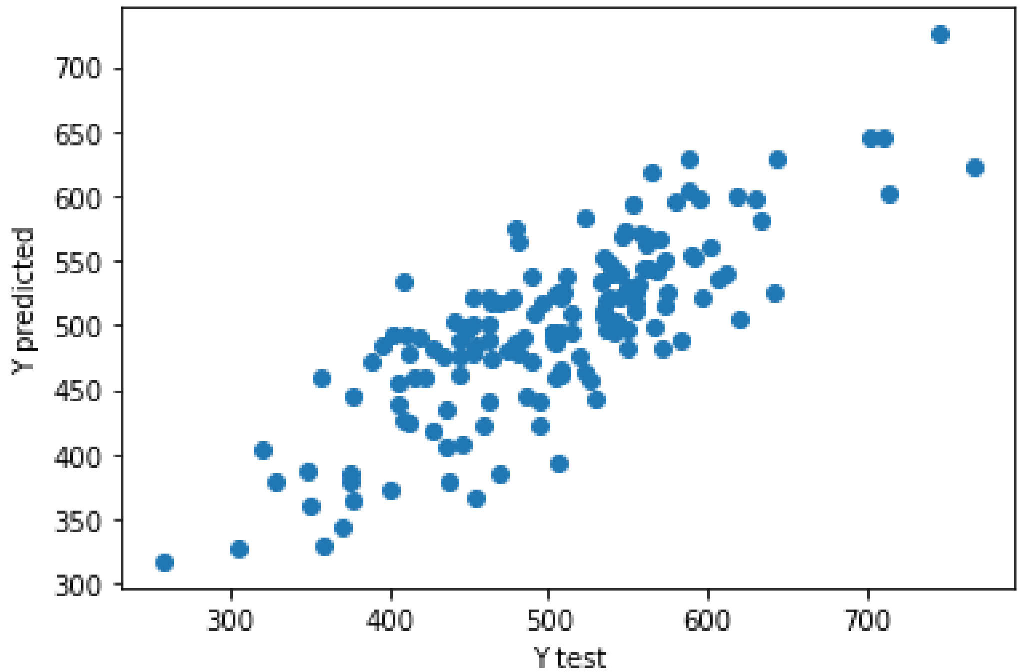
```
array([[365.37246317, 493.19839614, 492.81852615, 560.62737294,
        603.18436284, 519.3780163 , 553.74648187, 601.97377562,
        474.88372901, 553.14257125, 378.08945907, 422.85468207,
        522.28561281, 439.41943877, 622.69379221, 545.70719788,
        644.57680871, 495.36134767, 498.60528679, 515.02566909,
        533.97663492, 529.18474337, 384.93334462, 508.02207523,
        550.72572394, 478.61006458, 495.69506502, 344.30622041,
        495.04709731, 487.51253834, 535.18461904, 526.25850807,
        439.80565337, 566.98452894, 493.74752433, 568.18441161,
        495.6545491 , 476.87347793, 379.39387516, 440.81712485,
        628.35251448, 499.72686588, 580.89120004, 525.38116656,
        461.40086819, 476.54082917, 596.54266853, 502.1614003 ,
        458.99504056, 481.01431453, 516.33413777, 524.56803161,
```

```
       316.91712525, 537.85010182, 582.79817909, 326.71259741,
       524.9984553 , 471.49524779, 485.3278406 , 418.47426758,
       504.05504272, 552.92350708, 725.66821294, 568.23086279,
       510.21690655, 502.80036094, 435.35502516, 523.74436535,
       629.91709351, 521.78057237, 483.14176652, 378.64844809,
       594.74719262, 372.49318818, 444.87785487, 460.33199982,
       477.73462779, 477.68601108, 496.76197461, 644.52437525,
       458.04498839, 600.85616323, 516.34210025, 482.4599863 ,
       541.42708823, 522.44117576, 539.66366138, 574.23534582,
       481.65298126, 476.16319594, 465.24811017, 543.91963189,
       424.18782113, 504.61979735, 489.34144512, 531.10431812,
       494.34410634, 501.06862293, 422.10636285, 520.47587596,
       328.85356743, 490.9965877 , 618.47913759, 506.98994459,
       486.65240528, 490.36532662, 494.64288785, 485.2452002 ,
       534.11571995, 406.88140726, 573.75331415, 387.41860967,
       483.41090756, 539.25241061, 597.84996986, 443.26223696,
       562.78698862, 509.2793432 , 538.63634209, 384.87667657,
       360.50327413, 598.01334642, 463.95423738, 551.00957032,
       488.97959835, 405.41383193, 521.87622915, 392.30676273,
       521.2504864 , 459.67945438, 403.17663637, 461.08231383,
       510.9970425 , 500.03388494, 517.77778527, 363.88861137,
       455.63853366, 519.28835763, 544.60538416, 459.3313567 ,
       516.50538608, 539.63177853, 570.57191223, 488.08579105,
       525.45385867, 565.14691162, 472.66808645, 521.54015077,
       425.63579643, 444.2137562 ]])
```

```
y_pred.shape
```

```
(1, 150)
```

```
#IV. Generate a scatter plot that shows the Y-test on x- axis and y-
pridected in y- axis
plt.scatter(y_test,y_pred, marker = 'o')
plt.plot()
plt.xlabel("Y test")
plt.ylabel("Y predicted")
plt.show()
```




```
y_test.shape
```

```
(150, 1)
```

```
y_pred = y_pred.reshape(150,1)
```

```
# print MAE (mean absolute error)
print("MAE: ", mean_absolute_error(y_test, y_pred))

# print MSE (mean squared error)
print("MSE: ", mean_squared_error(y_test, y_pred))

# print RMSE (root mean squared error)
print("RMSE: ", sqrt(mean_squared_error(y_test, y_pred)))

# print r^2
print("R^2: ", r2_score(y_test,y_pred))
```

```
MAE:  40.7967716913424
MSE:  2535.792118506177
RMSE:  50.35664919855348
R^2:  0.6506588143840919
```

Short Question: How do derivatives help in the process of gradient descent?

Answer: Derivatives are used in order to define "steps" towards the local minimum of a given regression function. They help us determine our theta values at each point along the slope.

Advantages and Disadvantages of Batch Gradient Descent

Some advantages of BGD is that it is computationally efficient compared to running Linear Regressors and Normal Equation, and it generally runs faster than these as well. Some disadvantages are that it performs redundant computations on the data set and can also take a long time on very large data sets

7. Stochastic Gradient Descent

```
y_pred = np.empty(y_pred.shape)

# Implement Stochastic Gradient Descent and train our data set.
m = len(X_train_scaled)
n_epochs = 50
t0, t1 = 5, 50

cost_list = []
epoch_list = []
pred_list = []

def learning_schedule(t):
    return t0/ (t+t1)
theta = np.random.randn(2,1)
# perform prediction
for epoch in range(n_epochs):
    for i in range(m):
        if i == 101:
            i = 0
        rand_index = np.random.randint(m)
        xi = X_b[rand_index : rand_index +1]
        yi = y_train_scaled[rand_index : rand_index + 1]
        gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(epoch * m + i)
        theta = theta - eta * gradients
    y_pred = np.dot(theta.T, X_b.T)
    cost = np.mean(np.square(y_train_scaled-y_pred))
    if epoch % 10 == 0:
        cost_list.append(cost)
        epoch_list.append(epoch)


theta
```

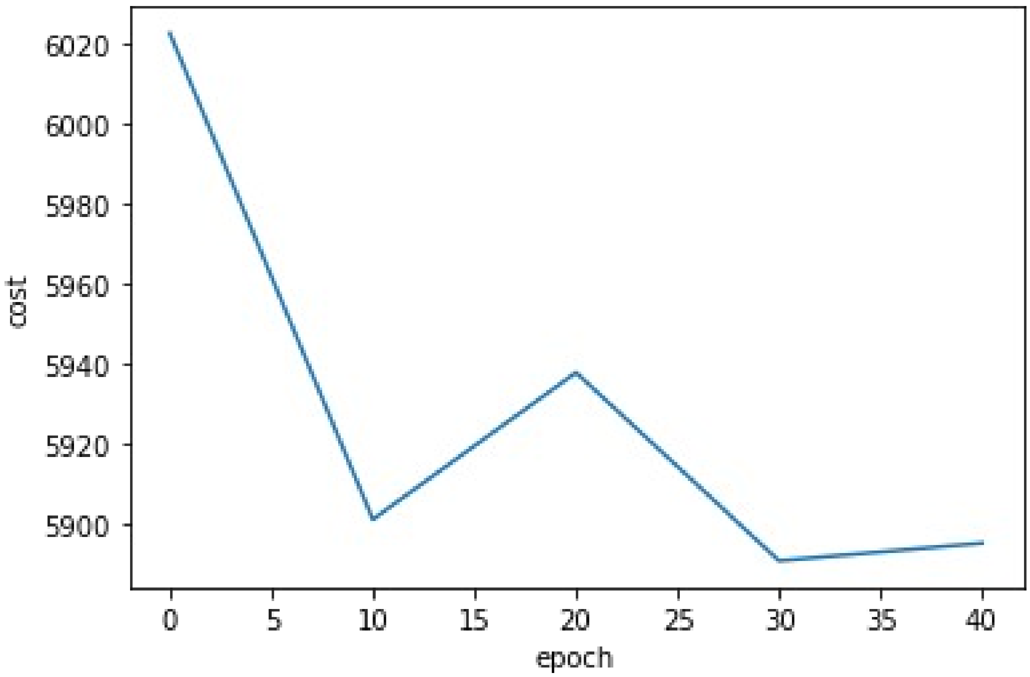
```
array([[497.02226112],
       [  4.82940055]])
```

Display the theta values. Are they very close to the sklearn's linear regression?

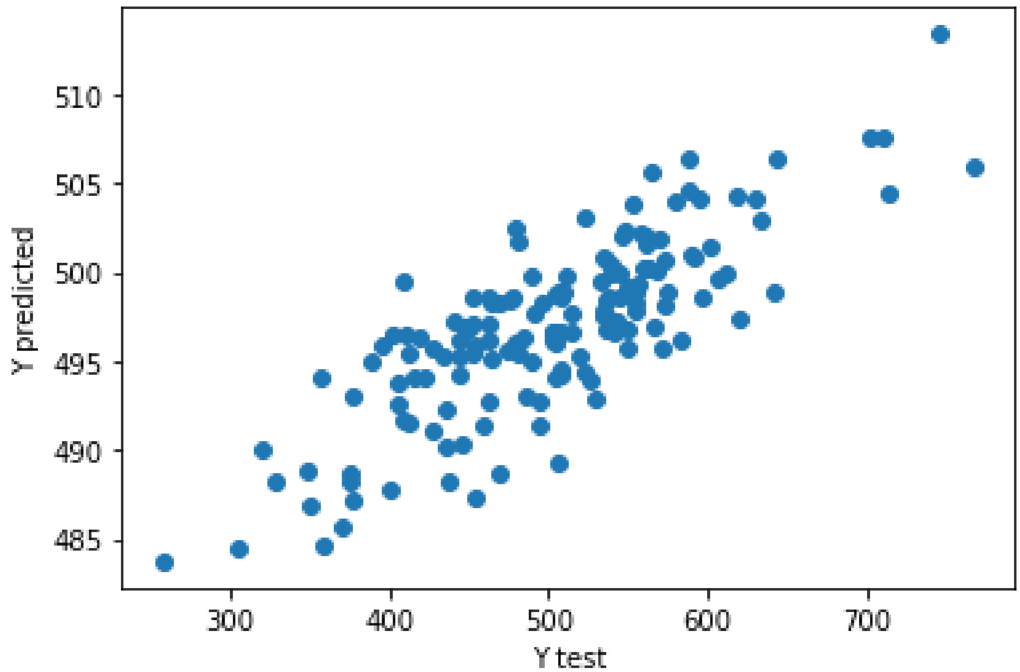
The first theta value is very close to the original coefficient we got from linear regression but the second (intercept) is not very close.

```
# plot step against cost
plt.xlabel("epoch")
plt.ylabel("cost")
plt.plot(epoch_list,cost_list)
```

```
[<matplotlib.lines.Line2D at 0x2789228f160>]
```




```
#IV. Generate a scatter plot that shows the Y-test on x- axis and y-pridected in y- axis
plt.scatter(y_test,y_pred, marker = 'o')
plt.plot()
plt.xlabel("Y test")
plt.ylabel("Y predicted")
plt.show()
```




```
y_pred = y_pred.reshape(150,1)

# print MAE (mean absolute error)
print("MAE: ", mean_absolute_error(y_test, y_pred))

# print MSE (mean squared error)
print("MSE: ", mean_squared_error(y_test, y_pred))

# print RMSE (root mean squared error)
print("RMSE: ", sqrt(mean_squared_error(y_test, y_pred)))

# print r^2
print("R^2: ", r2_score(y_test,y_pred))
```

```
MAE:  63.964944434041804
MSE:  6618.173544377079
RMSE:  81.3521575889483
R^2:  0.08825310413595144
```

What are the benefits and the limitations of using Stochastic gradient descent?

Some advantages are that is fast, and more effecient on larger data sets. Some disadvantages are that randomly picking instances can result in computational redundancy and missing several instances while repeating others. It generally converges more slowly

8. SGDRegressor

```
# Use sklearn's SGDRegressor to train a model for our data set. Put a
reasonable iteration and tolerance and learning steps so that we can
get coefficients close to normal equation
```

```
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor()
sgd_reg.fit(X_train_scaled,y_train_scaled.ravel())
```

```
SGDRegressor()
```

```
# Display the theta values. Are they very close to sklearn's linear
regression?

sgd_reg.intercept_
```

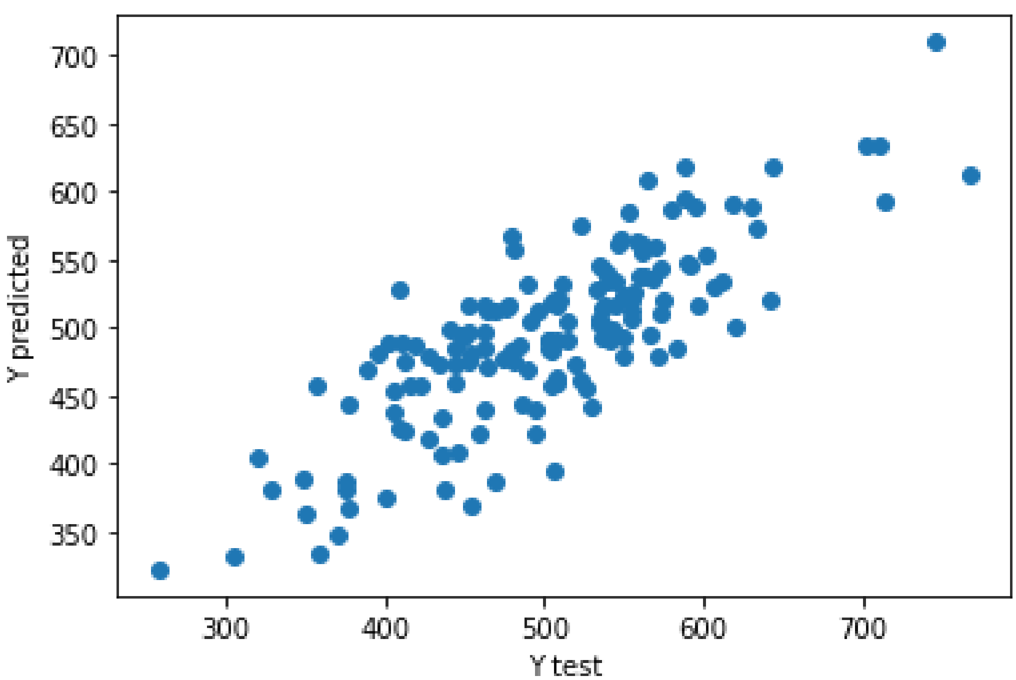
```
array([495.96644459])
```

```
sgd_reg.coef_
```

```
array([63.01223502])
```

Our theta values are very close to our original linear regression!

```
# Predict for the test data
y_pred = sgd_reg.predict(X_test_scaled)

# Generate a scatter plot that shows the Y test on the x-axis and y
predicted in the y-axis
plt.scatter(y_test,y_pred, marker = 'o')
plt.plot()
plt.xlabel("Y test")
plt.ylabel("Y predicted")
plt.show()
```



```python
# print MAE (mean absolute error)
print("MAE: ", mean_absolute_error(y_test, y_pred))

# print MSE (mean squared error)
print("MSE: ", mean_squared_error(y_test, y_pred))

# print RMSE (root mean squared error)
print("RMSE: ", sqrt(mean_squared_error(y_test, y_pred)))

# print r^2
print("R^2: ", r2_score(y_test,y_pred))
```

```
MAE:  41.13395355355476
MSE:  2586.172098302054
RMSE:  50.85442063677507
R^2:  0.6437182604858624
```

9. Mini-batch Gradient Descent

Briefly explain how mini-batch can overcome the limitations of Batch gradient descent and SGD.

By computing gradients on small mini-batches (random sets of instances), it gives us a performance boost over SGD and BGD, and we will generally be a bit closer to the minimum than SGD.

10. Polynomial of degree 2

```python
#Use sklearn's Polynomial features to degree = 2 on our training and test set
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2, include_bias = False)
X_train_poly = poly_features.fit_transform(X_train_scaled)

# Use linearRegression on the new polynomial features
lin_reg = LinearRegression()
lin_reg.fit(X_train_poly,y_train_scaled)
```

```
LinearRegression()
```

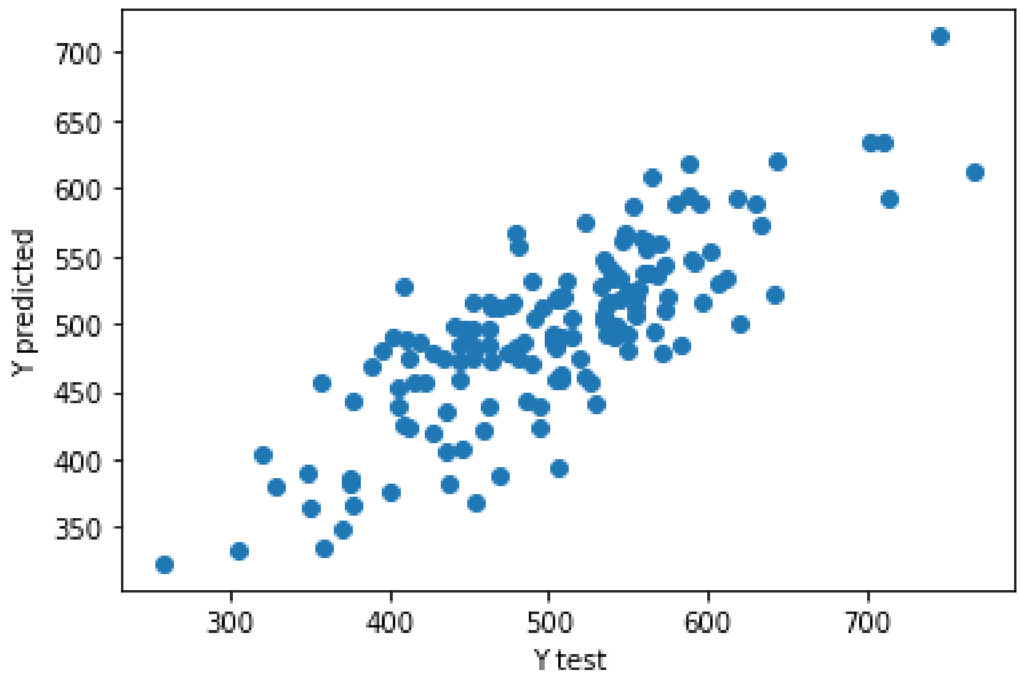
```python
lin_reg.intercept_, lin_reg.coef_
```

```
(array([495.86252367]), array([[63.16360087,  0.17018884]]))
```

Our theta values are very close to our original linear regression!

```python
#Predict for test set
X_test_poly = poly_features.transform(X_test_scaled)
y_new = lin_reg.predict(X_test_poly)
```

```
# Generate a scatter plot that shows the Y test on the x-axis and y
predicted in the y-axis
plt.scatter(y_test,y_new, marker = 'o')
plt.plot()
plt.xlabel("Y test")
plt.ylabel("Y predicted")
plt.show()
```




```
# print MAE (mean absolute error)
print("MAE: ", mean_absolute_error(y_test, y_new))

# print MSE (mean squared error)
print("MSE: ", mean_squared_error(y_test, y_new))

# print RMSE (root mean squared error)
print("RMSE: ", sqrt(mean_squared_error(y_test, y_new)))

# print r^2
print("R^2: ", r2_score(y_test,y_new))
```

```
MAE:  41.1012190142125
MSE:  2581.448959567414
RMSE:  50.80796157658182
R^2:  0.6443689395669062
```

11. Polynomial of degree 3

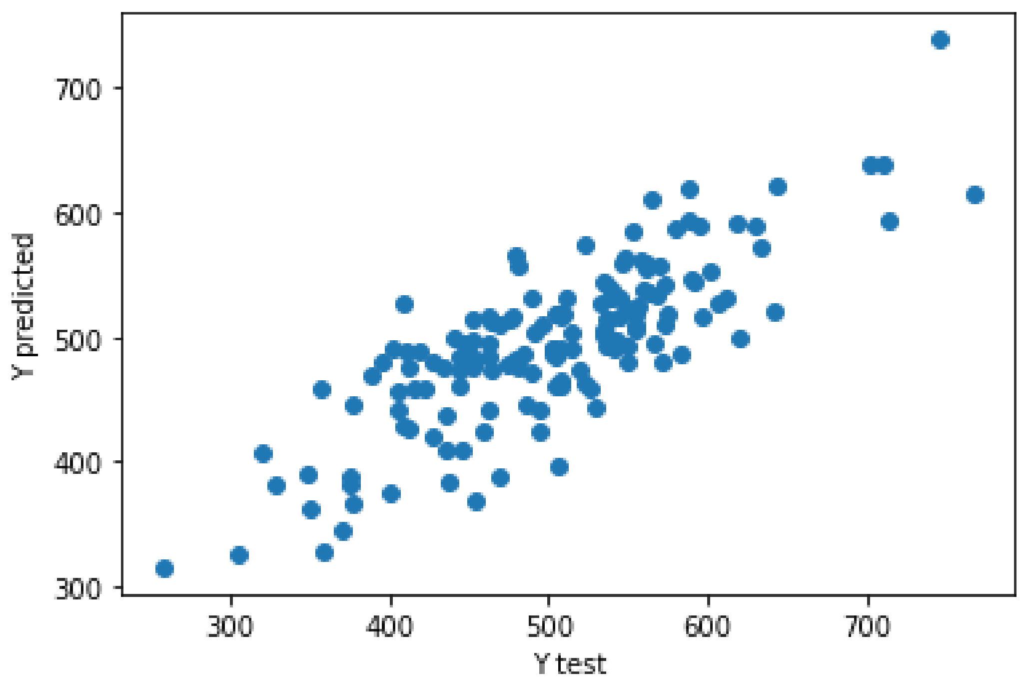
```
# Use sklearn's Polynomial features to degree = 3 on our training and
test set
```

```
poly_features = PolynomialFeatures(degree=3, include_bias = False)
X_train_poly = poly_features.fit_transform(X_train_scaled)

# Use linearRegression on the new polynomial features
lin_reg = LinearRegression()
lin_reg.fit(X_train_poly,y_train_scaled)
lin_reg.intercept_, lin_reg.coef_
```

```
(array([495.68330235]), array([[60.58201415,  0.42630355,
0.83917808]]))
```

Our theta values are very close to our original linear regression!¶

```
# Predict for test set
X_test_poly = poly_features.transform(X_test_scaled)
y_new = lin_reg.predict(X_test_poly)

# Generate a scatter plot that shows the Y test on the x-axis and y
predicted in the y-axis
plt.scatter(y_test,y_new, marker = 'o')
plt.plot()
plt.xlabel("Y test")
plt.ylabel("Y predicted")
plt.show()
```



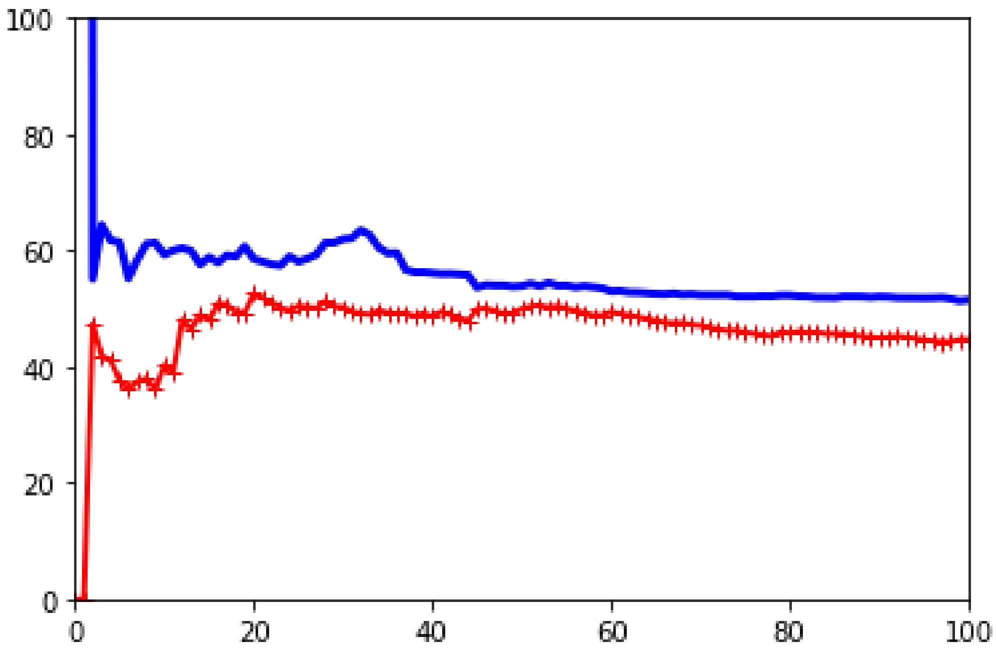

```
# print MAE (mean absolute error)
print("MAE: ", mean_absolute_error(y_test, y_new))

# print MSE (mean squared error)
print("MSE: ", mean_squared_error(y_test, y_new))
```

```
# print RMSE (root mean squared error)
print("RMSE: ", sqrt(mean_squared_error(y_test, y_new)))

# print r^2
print("R^2: ", r2_score(y_test,y_new))
```

```
MAE:  40.895180045340915
MSE:  2562.242363984242
RMSE:  50.61859701714619
R^2:  0.6470149194261003
```

12. Learning Curve

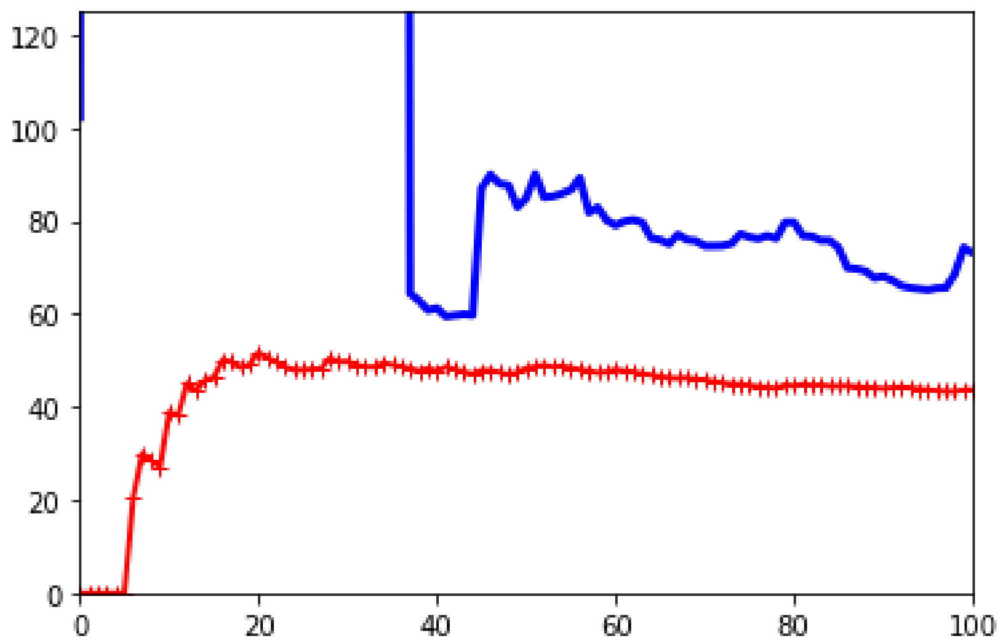
```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

def plot_learning_curves(model):
    train_errors, val_errors = [], []
    for m in range(1, len(X_train)):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_test)

train_errors.append(mean_squared_error(y_train[:m],y_train_predict))
        val_errors.append(mean_squared_error(y_test,y_val_predict))
    plt.plot(np.sqrt(train_errors), "r-+", linewidth = 2, label = 
"train")
    plt.plot(np.sqrt(val_errors), "b-", linewidth = 3, label = "val")


# Generate learning curve with linearRegression
lin_reg = LinearRegression()
plot_learning_curves(lin_reg)
plt.axis([0, 100, 0, 100])
plt.show()
```



Interpret the result:

Naturally the performance on training and testing data is poor on the first instance, but gradually the performance plateaus until the data becomes noisy and nonlinear. At this point, adding more instances would not improve the performance of this model, and shows potential underfitting issues.

```
# generate learning curve w/ polynomial regression
from sklearn.pipeline import Pipeline
polynomial_reg = Pipeline([
    ("poly_features", PolynomialFeatures(degree = 5, include_bias = 
False)),
    ("lin_reg", LinearRegression()),
])
plot_learning_curves(polynomial_reg)
plt.axis([0, 100, 0, 125])
plt.show()
```



Interpret the result:

The error on training data is much lower than that of our previous Linear Regression model, and the gap between the curves is (eventually) much larger than the previous. This gap shows possible overfitting, so we could overcome this by providing more training instances to the model.

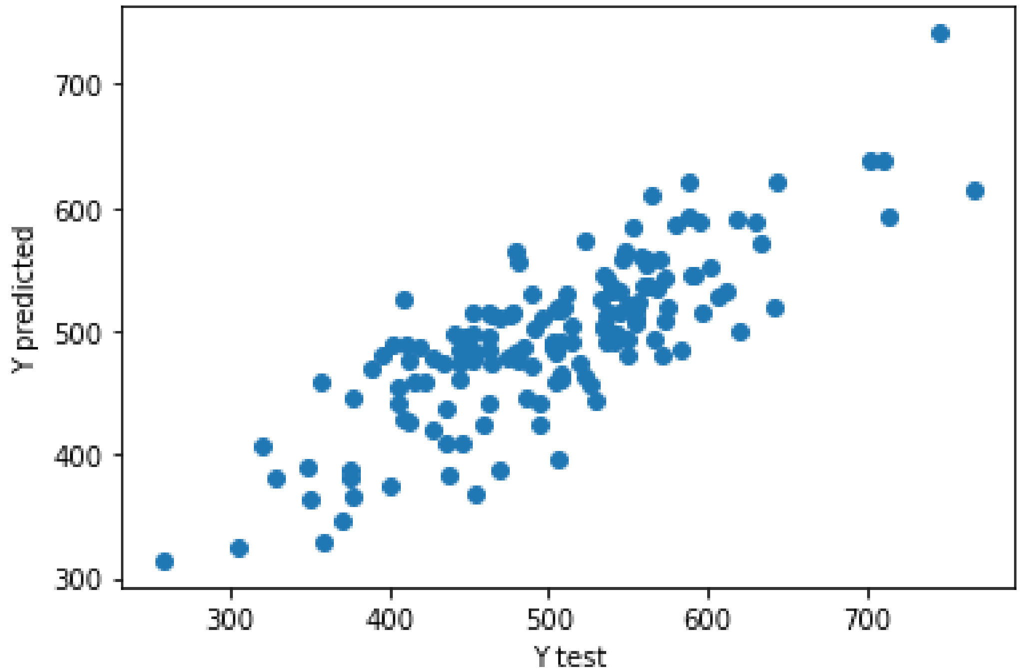
13. Regularization

The purpose of regularization: regularization helps us reduce error and prevent overfitting by properly fitting a function on a given training set and also reduces variance without adding a signficant bias.

14. Ridge Regression

```
# train ridge using polynomial degree 3 dataset
from sklearn.linear_model import Ridge
ridge_reg = Ridge(alpha = 1, solver = "cholesky")
ridge_reg.fit(X_train_poly,y_train_scaled)
# Predict for test set
y_pred = ridge_reg.predict(X_test_poly)

# Generate a scatter plot that shows the Y test on the x-axis and y
predicted in the y-axis
plt.scatter(y_test,y_pred, marker = 'o')
plt.plot()
plt.xlabel("Y test")
```

```python
plt.ylabel("Y predicted")
plt.show()
```



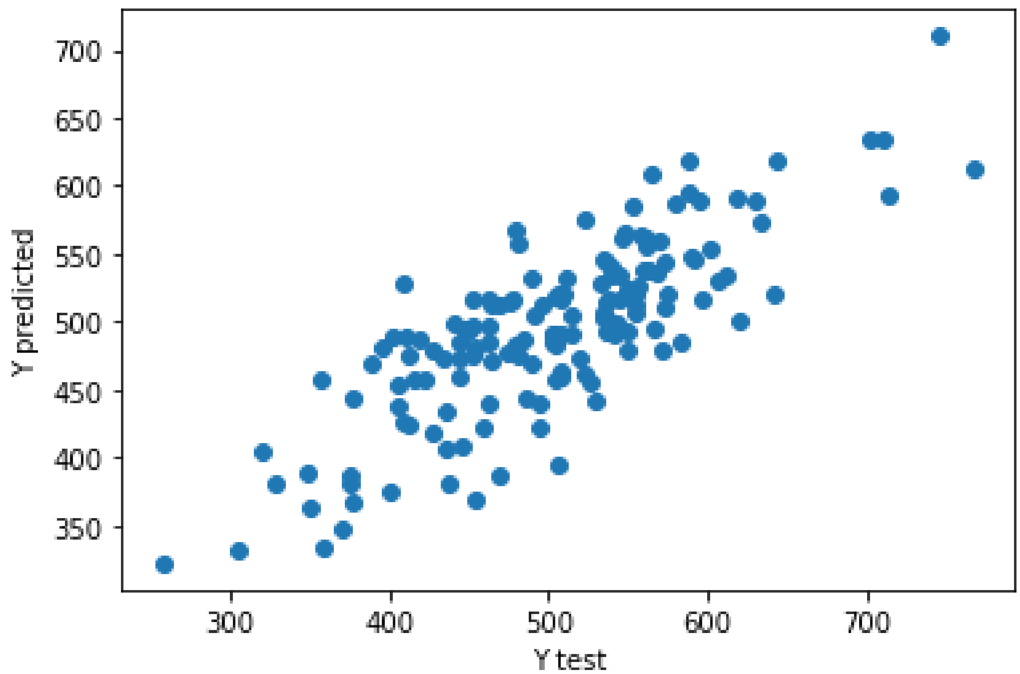

```python
# print MAE (mean absolute error)
print("MAE: ", mean_absolute_error(y_test, y_pred))

# print MSE (mean squared error)
print("MSE: ", mean_squared_error(y_test, y_pred))

# print RMSE (root mean squared error)
print("RMSE: ", sqrt(mean_squared_error(y_test, y_pred)))

# print r^2
print("R^2: ", r2_score(y_test,y_pred))
```

```
MAE:  40.887318202869054
MSE:  2562.111981772858
RMSE:  50.617309112326964
R^2:  0.6470328814175332
```

15. SGDRegressor for Ridge

```python
# Use sklearn's SGDRegressor for Ridge Regression
sgd_reg = SGDRegressor(penalty = "l2")
sgd_reg.fit(X_train_scaled,y_train_scaled.ravel())

# Predict for test set
y_pred = sgd_reg.predict(X_test_scaled)
```

```
# Generate a scatter plot that shows the Y test on the x-axis and y
predicted in the y-axis
plt.scatter(y_test,y_pred, marker = 'o')
plt.plot()
plt.xlabel("Y test")
plt.ylabel("Y predicted")
plt.show()
```



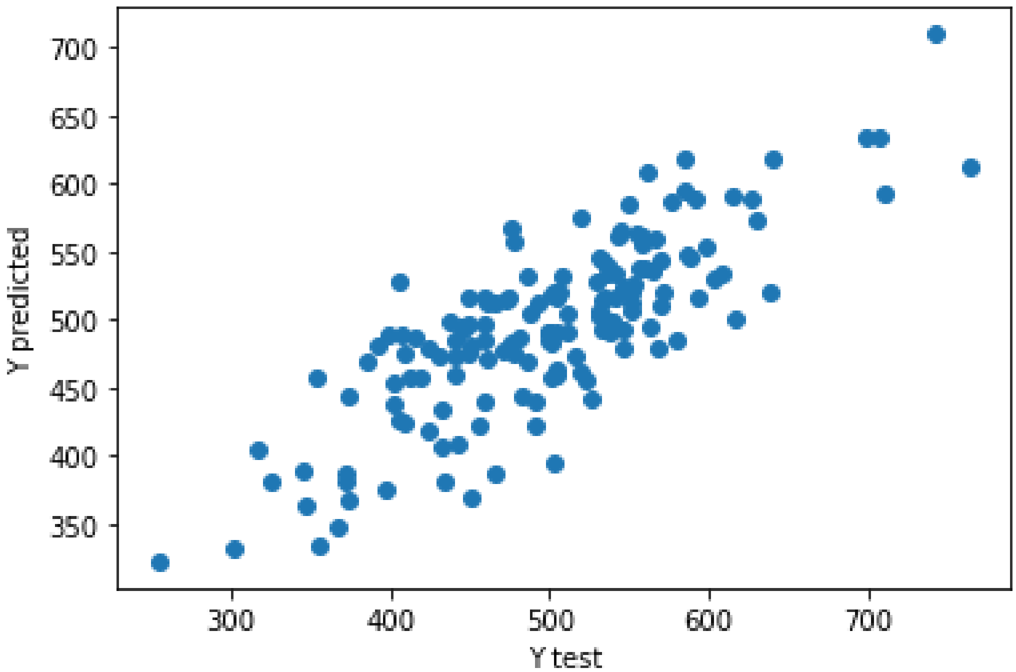

```
# print MAE (mean absolute error)
print("MAE: ", mean_absolute_error(y_test, y_pred))

# print MSE (mean squared error)
print("MSE: ", mean_squared_error(y_test, y_pred))

# print RMSE (root mean squared error)
print("RMSE: ", sqrt(mean_squared_error(y_test, y_pred)))

# print r^2
print("R^2: ", r2_score(y_test,y_pred))
```

```
MAE:  41.11879542768603
MSE:  2583.940906691265
RMSE:  50.83247885644831
R^2:  0.6440256386486705
```

16. Lasso Regression

```
from sklearn.linear_model import Lasso
lasso_reg =  Lasso(alpha = .1)
```

```
lasso_reg.fit(X_train_scaled, y_train_scaled)
y_pred = lasso_reg.predict(X_test_scaled)

plt.scatter(y_test_scaled,y_pred, marker = 'o')
plt.plot()
plt.xlabel("Y test")
plt.ylabel("Y predicted")
plt.show()
```

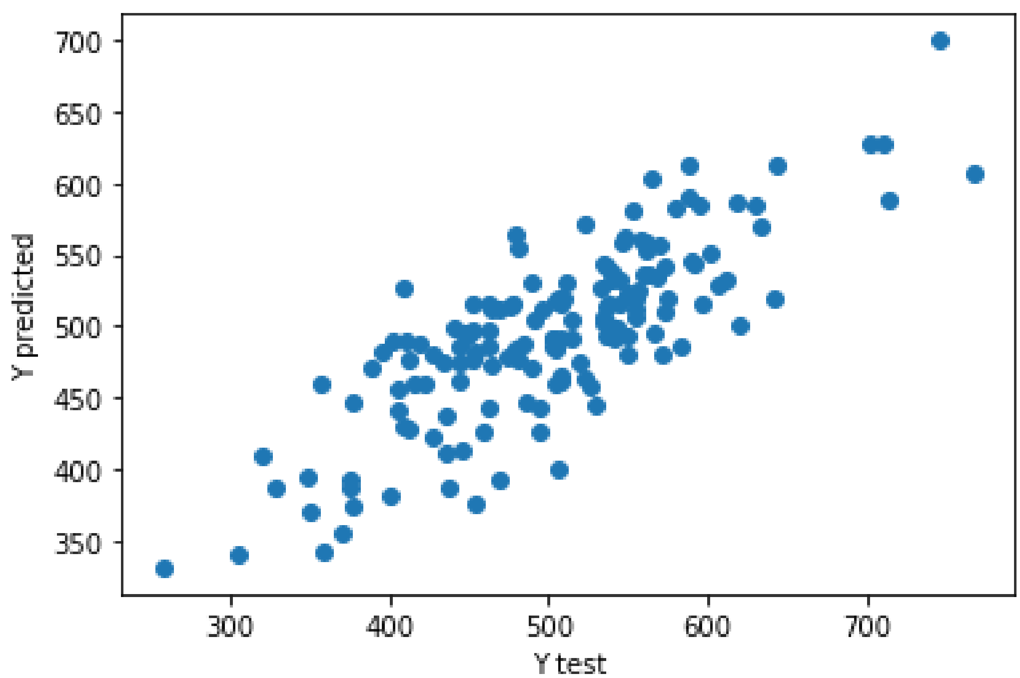



```
# print MAE (mean absolute error)
print("MAE: ", mean_absolute_error(y_test, y_pred))

# print MSE (mean squared error)
print("MSE: ", mean_squared_error(y_test, y_pred))

# print RMSE (root mean squared error)
print("RMSE: ", sqrt(mean_squared_error(y_test, y_pred)))

# print r^2
print("R^2: ", r2_score(y_test,y_pred))

MAE:  41.12644360978626
MSE:  2585.061202700001
RMSE:  50.843497152536635
R^2:  0.6438713020478598
```

How Lasso perform the regularization and how does that affect the thetas?

17. Elastic Net

```
# Use sklearn's ElasticNet

from sklearn.linear_model import ElasticNet
elastic_net = ElasticNet(alpha = .1, l1_ratio = .5)
elastic_net.fit(X_train_scaled, y_train_scaled)

# Predict for test set
y_pred = elastic_net.predict(X_test_scaled)

# Generate a scatter plot that shows the Y test in x axis and y
predicted in y axis
plt.scatter(y_test,y_pred, marker = 'o')
plt.plot()
plt.xlabel("Y test")
plt.ylabel("Y predicted")
plt.show()
```




```
# print MAE (mean absolute error)
print("MAE: ", mean_absolute_error(y_test, y_pred))

# print MSE (mean squared error)
print("MSE: ", mean_squared_error(y_test, y_pred))
```

```
# print RMSE (root mean squared error)
print("RMSE: ", sqrt(mean_squared_error(y_test, y_pred)))

# print r^2
print("R^2: ", r2_score(y_test,y_pred))
```

```
MAE:  41.32546460518151
MSE:  2615.8946151277673
RMSE:  51.14581718115146
R^2:  0.6396235639247342
```

How is ElasticNet different compared to Lasso and RIDGE perform the regularization and how does that affect the thetas?

Elastic Net's regularization is a combination of Ridge and Lasso Regression regularization methods. The difference is you can control r (the mix ratio). When r is 0, elastic net behaves the sames as ridge regression does, and when r is 1, it behaves just like lasso regression