Univariate Linear Regression ¶

Importing Numpy for mathematical operations, Pandas for dataframe, Matplotlib and Seaborn for data Visualisation

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv("boston.csv")
df
```

Out[1]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90
...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90

506 rows × 14 columns

Checking for Null Values

In [2]:

```
df.isnull().sum()
```

Out[2]:

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

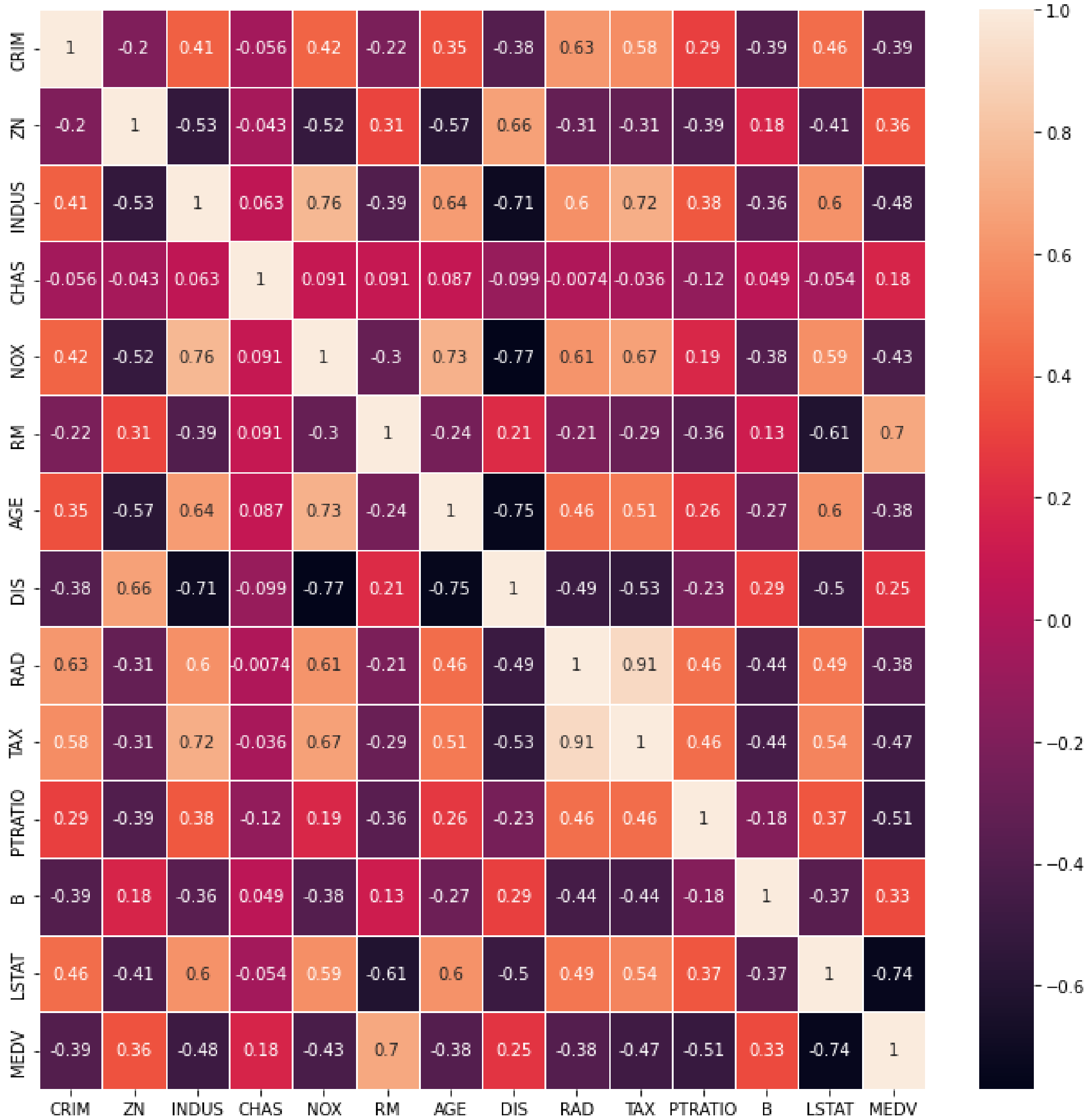
Visualising Data

In [3]:

```
import seaborn as sns
plt.figure(figsize=(12, 12))
sns.heatmap(df.corr(),annot=True,linewidths=1)
```

Out[3]:

<AxesSubplot:>



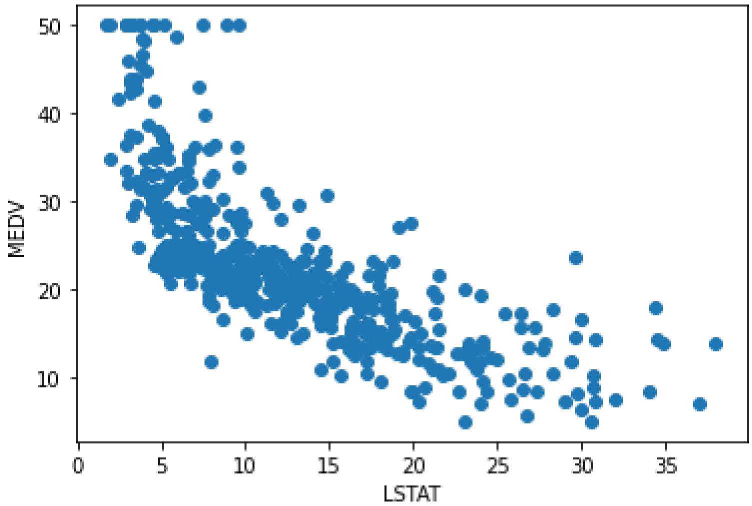
LSTAT vs MEDV Plot

In [4]:

```
plt.xlabel('LSTAT')
plt.ylabel('MEDV')
plt.scatter(df.LSTAT,df.MEDV)
```

Out[4]:

<matplotlib.collections.PathCollection at 0x1dc34a1cfd0>



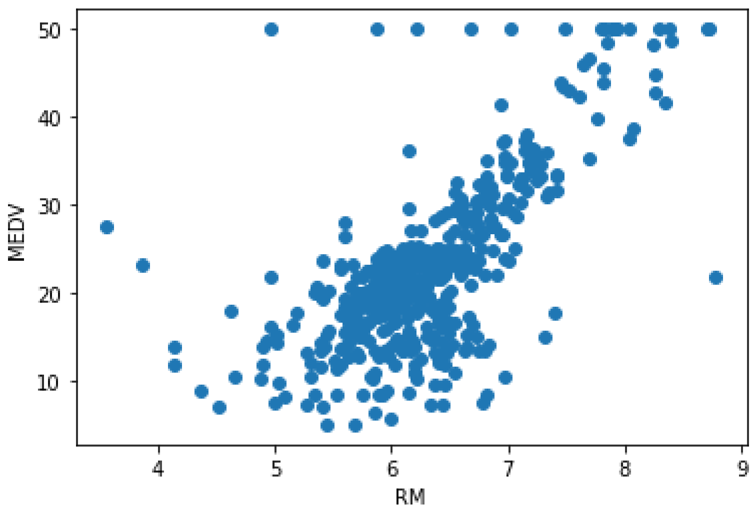
RM vs MEDV Plot

In [5]:

```
plt.xlabel('RM')
plt.ylabel('MEDV')
plt.scatter(df.RM,df.MEDV)
```

Out[5]:

```
<matplotlib.collections.PathCollection at 0x1dc34a92dc0>
```



Removing Outliers from Training Data

In [6]:

```
max_threshold=df['RM'].quantile(0.95)
min_threshold=df['RM'].quantile(0.05)
df=df[(df['RM']>min_threshold) & (df['RM']<max_threshold)] #x is training data with removed
df
```

Out[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90
...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90

454 rows × 14 columns

In [7]:

```
#Extracting target variable
target=df.MEDV
target
```

Out[7]:

```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: MEDV, Length: 454, dtype: float64
```

Splitting Train and Test Data

In [8]:

```
train=df.iloc[:350,:]
test=df.iloc[350:,:]
tgt_test=target.iloc[350:]
tgt_train=target.iloc[:350]
```

In [9]:

```
x=df
```

Building the Model

In [10]:

```
def model(X,Y,lr=0.0001,itr=1000):
    m=c=0
    n=len(X)
    cost_list= []
    for i in range (itr):
        h=m*X +c
        error=Y-h
        cost=(1/n)*sum(error**2)
        cost_list.append(cost)
        md=(-2/n)*np.sum((error)*X)
        cd=(-2/n)*np.sum(error)
        m=m-lr*md
        c=c-lr*cd
        print(cost)

    return m,c,cost_list
```

Calling the model and receiving the parameters

In [11]:

```
m,c,cost_list=model(x.RM,x.MEDV)
```

```
524.0601541850219
516.2954745582334
508.65605198057887
501.1398658452532
493.7449281412407
486.4692829274804
479.31100581553443
472.26820346058133
465.33901306063865
458.5216018638515
451.81416668373083
445.2149334222205
438.72215660044884
432.3341188970517
426.04913069394627
419.86552962943216
413.7816801584952
407.7959731202133
401.90682531213747
```

Printing Parameters

Slope

In [12]:

```
print("slope parameter")
m
```

```
slope parameter
```

Out[12]:

```
3.410725916661653
```

Intercept

In [13]:

```
print("intercept")
c
```

```
intercept
```

Out[13]:

```
0.49843630496613883
```

Prediction using test data

In [14]:

```
ypred=m*test.RM+c
```

Calculating MSE

In [15]:

```
mse=(np.square(np.subtract(ypred,tgt_test))).mean()
mse
```

Out[15]:

```
62.69464528254029
```

Visualising Result

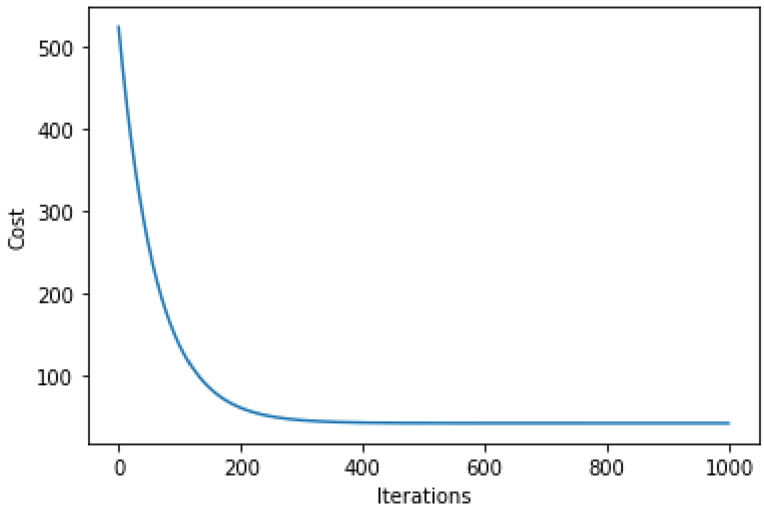
Ploting Cost curve

In [16]:

```
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.plot(cost_list)
```

Out[16]:

```
[<matplotlib.lines.Line2D at 0x1dc34b1baf0>]
```



Result :

MSE from Univariate Linear Regression : 62.69

Multivariate Linear Regression on Boston dataset

Importing required libraries and CSV file in Pandas Dataframe

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_csv("boston.csv")
df
```

Out[1]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90
...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90

506 rows × 14 columns

In [2]:

```
df.shape[1]
```

Out[2]:

14

Finding Correlation between features

In [3]:

```
target=df['MEDV'] #extracting target variable
target
df.corr()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996
MEDV	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929

Only Keeping the features with high correlation with target

In [4]:

```
X=df
X.drop(['RAD','B','DIS','CHAS','ZN','CRIM','TAX','MEDV','INDUS','NOX','PTRATIO','AGE'],axis
X
```

Out[4]:

	RM	LSTAT
0	6.575	4.98
1	6.421	9.14
2	7.185	4.03
3	6.998	2.94
4	7.147	5.33
...	...	...
501	6.593	9.67
502	6.120	9.08
503	6.976	5.64
504	6.794	6.48
505	6.030	7.88

506 rows × 2 columns

Only kept no of room(RM) and % lower status of the population(LSTAT)

Splitting Train and Test data

In [5]:

```
X.shape[1]
train=X.iloc[:300,:]
test=X.iloc[300:,:]
target_train=target.iloc[:300]
ytrue=target.iloc[300:]
```

Inserting 1s in train data

This is required since intercept(theta0) will also be calculated together with other weights.

In [6]:

```
ones = np.ones((train.shape[0],1))
train = np.hstack((ones,train))
ones = np.ones((test.shape[0],1))
test = np.hstack((ones,test))
print(train.shape)
```

```
(300, 3)
```

Building Model

In [7]:

```
def model(X,Y,lr=.0001,itr=3000):
    n=X.shape[1];
    theta=np.ones(n)
    err_list=[]

    for i in range (itr):
        h=np.dot(X,theta)
        e=np.sum(np.square(h-Y))
        err_list.append(e/X.shape[0])
        grad = (np.dot(X.T,(h-Y))/X.shape[0])
        theta = theta - lr*grad

    return theta,err_list
```

Calling Model

In [8]:

```
theta,err_list=model(train,target_train)
```

Printing the parameters

In [9]:

```
theta
```

Out[9]:

```
array([ 1.43462609,  4.76132763, -0.60958751])
```

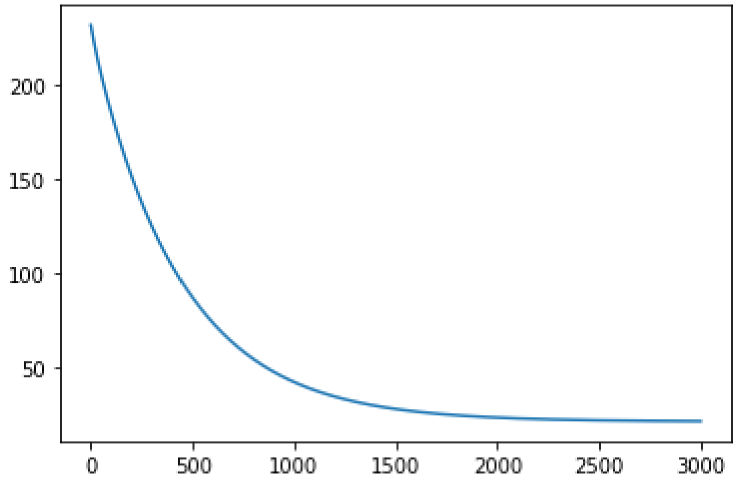
Error Plot

In [10]:

```
plt.plot(err_list)
```

Out[10]:

```
[<matplotlib.lines.Line2D at 0x22a153e1c70>]
```



Getting Prediction

In [11]:

```
ypred=np.dot(test,theta)
```

In [12]:

```
ypred
```

Out[12]:

```
array([30.44951206, 27.02069385, 27.07432536, 31.71562031, 31.66315139,
       27.49195325, 32.81964592, 29.45476509, 30.25850762, 23.79168725,
       17.40752232, 26.93814054, 22.96773683, 26.45336371, 27.04529257,
       21.58774389, 18.41937871, 19.2477976 , 25.50609245, 22.78057293,
       27.64188738, 27.60498488, 25.5039825 , 21.45572688, 28.24786729,
       28.95801953, 27.73916292, 22.60115785, 23.29650918, 27.11374166,
       25.14707261, 21.02558884, 25.37712284, 28.04471435, 27.36388776,
       25.29596506, 23.40490038, 23.0654084 , 25.09592051, 23.99378964,
       24.18716145, 32.56476403, 27.30077685, 28.93963759, 31.3537938 ,
       23.65029401, 21.79346274, 28.58246038, 29.37460574, 30.88300809,
       28.69030911, 29.41276515, 24.70159118, 30.72569459, 23.49084503,
       26.30246448, 20.28325322, 23.79409007, 23.6092159 , 22.81238686,
       27.14891358, 22.54763839, 20.75316815, 20.14024924, 40.01436475,
       14.04945074, 16.53087003, 11.70183327, 23.11116913, 30.98081723,
       33.03572171, 25.2216697 , 23.99428885,  3.59834188, -2.00903776,
       28.06135896, 18.92587986, 20.83544738, 17.37076835, 17.78755205,
       24.12055603, 19.7474108 , 13.40707071, 12.74568547,  3.56043988,
        7.77876095,  6.34518757,  5.74055994,  6.0043355 , 14.38828252,
       18.20604861, 18.80955797,  9.75846489, 21.67408563, 19.49780612,
       21.80903908, 20.12321957, 16.65499286,  8.75086386, 11.02857783,
       13.62203708, 19.24890912, 19.54544598, 14.85142259, 11.07902331,
       14.48493013,  6.90922744, 20.74185499, 12.08589324, 22.00160214,
       22.68265956, 20.19533724,  2.5185278 , 13.73875291,  0.4085197 ,
       14.36049102, 18.0046883 , 10.44929669, 17.22816041, 20.06390595,
       22.80349318, 20.46063599, 19.73142071, 16.2957156 , 17.47089275,
       14.63957453, 19.66206749, 22.11316946, 17.80320497, 17.13302923,
       20.90641028, 21.96599979, 24.69281841, 22.19102136, 21.74550555,
       18.81236569, 21.19450942, 14.60272414,  8.95493863, 14.27925453,
       15.65805049, 20.03654277, 20.93226589, 20.82111128, 14.80535127,
       17.57013286, 20.78164286, 21.1760665 , 19.831616  , 20.21693075,
       22.93762016, 22.31327498, 20.88912994, 26.4306264 , 22.06345614,
       21.45046739, 18.30006152, 19.37145455, 21.54214626, 21.42732306,
       23.33085568, 22.86239409, 22.98380351, 26.1724975 , 22.93896251,
       20.24164045, 19.31962241, 17.02047023, 18.59843214, 19.63857925,
       20.86755311, 23.24754469, 23.32961545, 27.56090262, 16.21643378,
       16.08286805, 20.91997983, 11.50388307, 19.89257475, 23.10124369,
       24.60786333, 28.85538028, 30.78115205, 22.51749407, 21.25648327,
       25.03869027, 21.41376238, 22.57048879, 16.39375663, 12.60064137,
        7.59151054, 18.90640307, 21.78365609, 21.28637692, 21.36595941,
       17.70261365, 14.21150214, 20.42657454, 22.21718651, 18.74568832,
       21.39575875, 26.93134796, 25.03889662, 31.21157409, 29.83295895,
       25.34188214])
```

Calculating MSE

In [13]:

```
MSE = np.square(np.subtract(ypred,ytrue)).mean()
```

In [14]:

```
MSE
```

Out[14]:

```
47.340435510898544
```

Closed Form

In [15]:

```
def normal(X, Y):
    theta = np.dot((np.linalg.inv(np.dot(X.T,X))), np.dot(X.T,Y))
    return theta
```

In [16]:

```
w=normal(train,target_train)
```

Printing parameters from closed form

In [17]:

```
w
```

Out[17]:

```
array([-36.37478967,   9.99214379,  -0.21425652])
```

Calculating MSE for Closed Form

In [18]:

```
MSE = np.square(np.subtract(np.dot(test,w),ytrue)).mean()
```

In [19]:

```
MSE
```

Out[19]:

```
77.77401808489346
```

Result

MSE from Multivariate : 47.34

MSE from Closed Form: 77.77

Breast Cancer with Single feature

Importing all required libraries and loading breast cancer dataset to pandas dataframe.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df=pd.read_csv("breast_cancer.csv")
```

In [2]:

```
df
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes:
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	
565	926682	M	20.13	28.25	131.20	1261.0	
566	926954	M	16.60	28.08	108.30	858.1	
567	927241	M	20.60	29.33	140.10	1265.0	
568	92751	B	7.76	24.54	47.92	181.0	

569 rows × 33 columns

Visualising Data and Pre-Processing

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

Droping id and Unnamed 32 column

id column is not required for prediction and unnamed 32 has all null values. So, dropping these two column.

In [4]:

```
X=df.drop(['id','Unnamed: 32'],axis=1)
X.replace({'M':1,'B':0},inplace=True)
```

In [5]:

```
X.info()
```

```
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

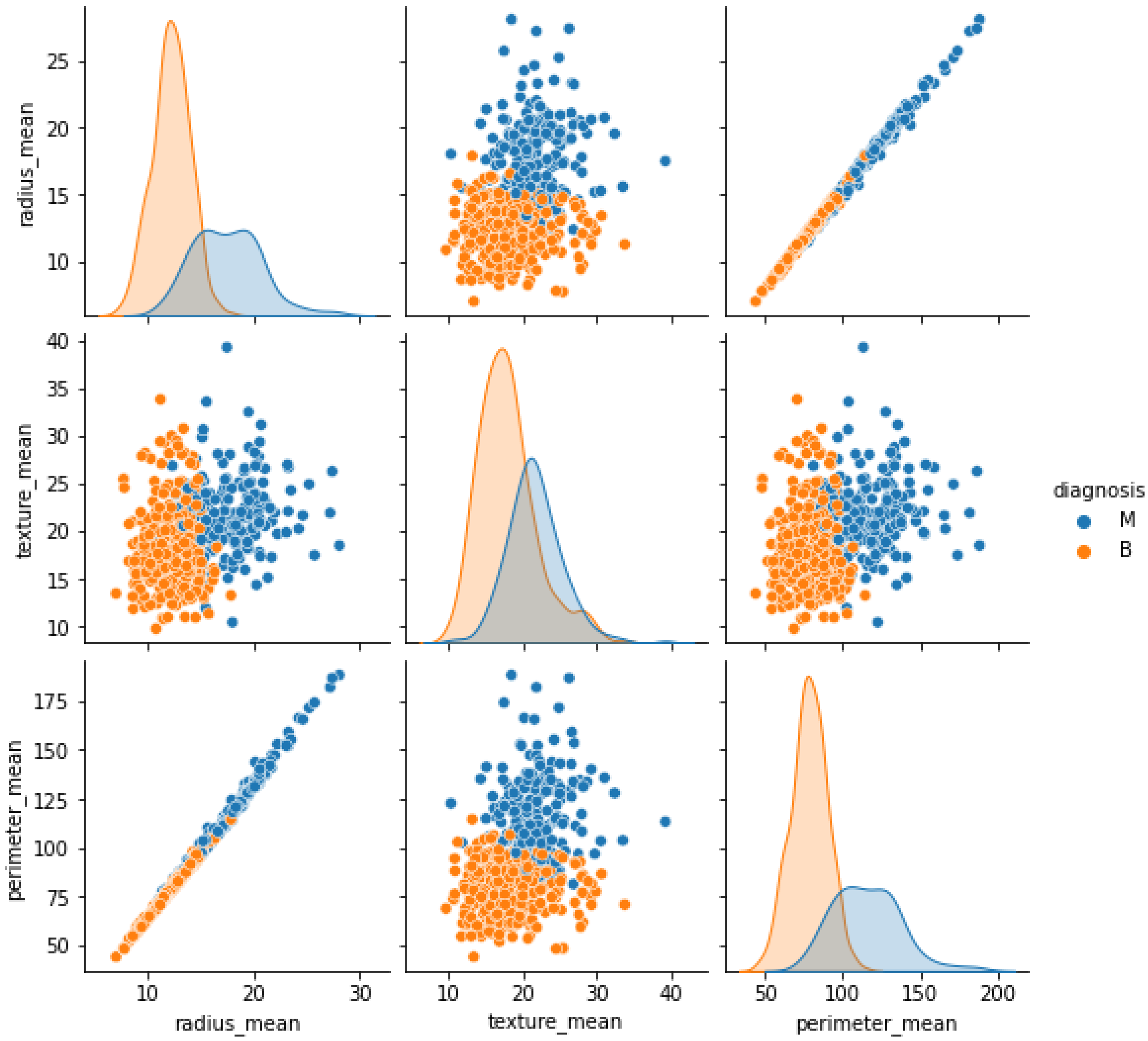
Trying to visualise first few columns of dataset

In [6]:

```
sns.pairplot(df.iloc[:,1:5],hue='diagnosis')
```

Out[6]:

```
<seaborn.axisgrid.PairGrid at 0x2abdaff9e50>
```

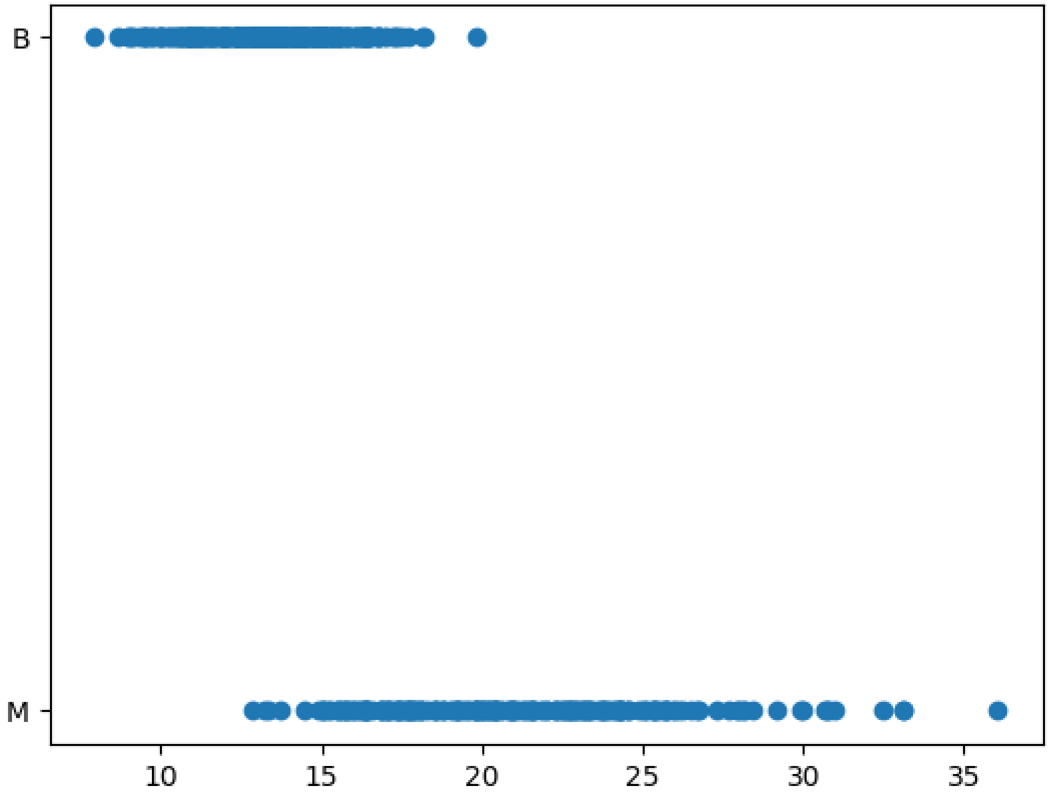


In [7]:

```
plt.scatter(df.radius_worst,df.diagnosis)
```

Out[7]:

```
<matplotlib.collections.PathCollection at 0x2abdf651ca0>
```



Feature Selection

Trying to select required features from analysing correlation matrix and heatmap

In [8]:

```
X.corr()
```

Out[8]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	sr
diagnosis	1.000000	0.730029	0.415185	0.742636	0.708984	
radius_mean	0.730029	1.000000	0.323782	0.997855	0.987357	
texture_mean	0.415185	0.323782	1.000000	0.329533	0.321086	
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.986507	
area_mean	0.708984	0.987357	0.321086	0.986507	1.000000	
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.177028	
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.685983	
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823269	
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.151293	
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	
radius_se	0.567134	0.679090	0.275869	0.691765	0.732562	
texture_se	-0.008303	-0.097317	0.386358	-0.086761	-0.066280	
perimeter_se	0.556141	0.674172	0.281673	0.693135	0.726628	
area_se	0.548236	0.735864	0.259845	0.744983	0.800086	
smoothness_se	-0.067016	-0.222600	0.006614	-0.202694	-0.166777	
compactness_se	0.292999	0.206000	0.191975	0.250744	0.212583	
concavity_se	0.253730	0.194204	0.143293	0.228082	0.207660	
concave points_se	0.408042	0.376169	0.163851	0.407217	0.372320	
symmetry_se	-0.006522	-0.104321	0.009127	-0.081629	-0.072497	
fractal_dimension_se	0.077972	-0.042641	0.054458	-0.005523	-0.019887	
radius_worst	0.776454	0.969539	0.352573	0.969476	0.962746	
texture_worst	0.456903	0.297008	0.912045	0.303038	0.287489	
perimeter_worst	0.782914	0.965137	0.358040	0.970387	0.959120	
area_worst	0.733825	0.941082	0.343546	0.941550	0.959213	
smoothness_worst	0.421465	0.119616	0.077503	0.150549	0.123523	
compactness_worst	0.590998	0.413463	0.277830	0.455774	0.390410	
concavity_worst	0.659610	0.526911	0.301025	0.563879	0.512606	
concave points_worst	0.793566	0.744214	0.295316	0.771241	0.722017	
symmetry_worst	0.416294	0.163953	0.105008	0.189115	0.143570	
fractal_dimension_worst	0.323872	0.007066	0.119205	0.051019	0.003738	

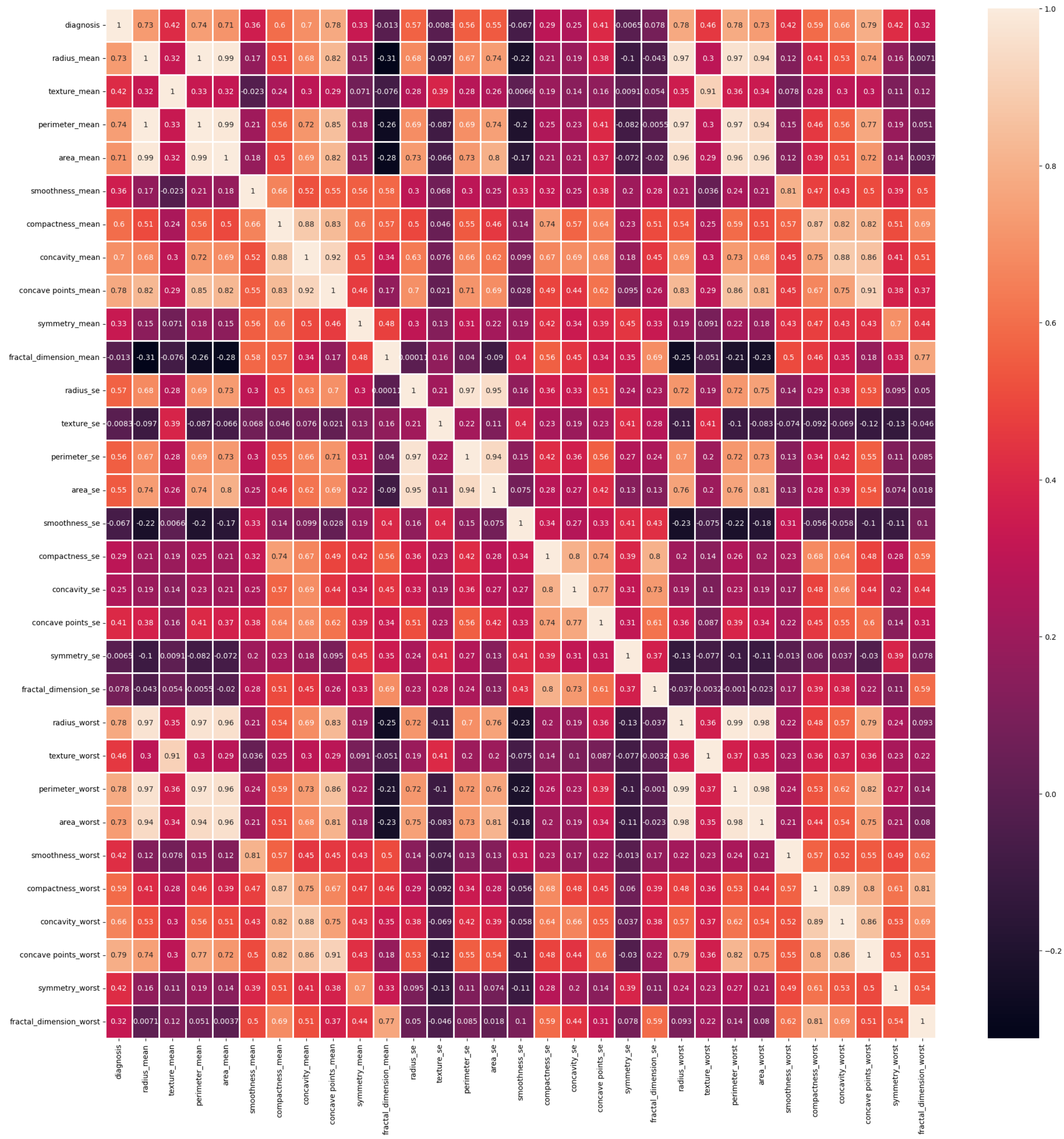
31 rows × 31 columns

In [9]:

```
plt.figure(figsize=(25, 25))
sns.heatmap(X.corr(),annot=True,linewidth=1)
```

Out[9]:

```
<AxesSubplot:>
```



Extract target

In [10]:

```
target=X.diagnosis
```

In [11]:

```
target
```

Out[11]:

```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64
```

Splitting into train and test

In [12]:

```
train=X.iloc[:400,:]
test=X.iloc[400:,:]
train_target=target.iloc[:400]
test_target=target.iloc[400:]
```

In [13]:

```
train
```

Out[13]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cc
0	1	17.99	10.38	122.80	1001.0	0.11840	
1	1	20.57	17.77	132.90	1326.0	0.08474	
2	1	19.69	21.25	130.00	1203.0	0.10960	
3	1	11.42	20.38	77.58	386.1	0.14250	
4	1	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	
395	0	14.06	17.18	89.75	609.1	0.08045	
396	0	13.51	18.89	88.10	558.1	0.10590	
397	0	12.80	17.46	83.05	508.3	0.08044	
398	0	11.06	14.83	70.31	378.2	0.07741	
399	0	11.80	17.26	75.26	431.9	0.09087	

400 rows × 31 columns

In [14]:

```
test
```

Out[14]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cc
400	1	17.91	21.02	124.40	994.0	0.12300	
401	0	11.93	10.91	76.14	442.7	0.08872	
402	0	12.96	18.29	84.18	525.2	0.07351	
403	0	12.94	16.17	83.18	507.6	0.09879	
404	0	12.34	14.95	78.29	469.1	0.08682	
...	...	...	...	...	...	...	
564	1	21.56	22.39	142.00	1479.0	0.11100	
565	1	20.13	28.25	131.20	1261.0	0.09780	
566	1	16.60	28.08	108.30	858.1	0.08455	
567	1	20.60	29.33	140.10	1265.0	0.11780	
568	0	7.76	24.54	47.92	181.0	0.05263	

169 rows × 31 columns

In [15]:

```
target
```

Out[15]:

```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64
```

Logistic Regression with single feture

In [16]:

```
train_uni=train.radius_worst
test_uni=test.radius_worst
```

In [17]:

```
def model_Uni(X,Y,lr=0.01,itr=100000):
    print("Starting model")
    m=X.shape[0]
    w=0
    cost_list=[]
    bias=0
    cost_prev = 1e10
    change = 10000
    while (change>1e-5):
        linear=w*X+bias                    #Calculating Linear estimation
        ypred=(1/(1+np.exp(-linear)))     #Putting Linear estimation on sigmoid function

        cost=np.sum(np.square(ypred-Y))
        cost_list.append(cost)
        dw=(1/m)*np.dot(X.T,(ypred-Y))
        db=(1/m)*np.sum(ypred-Y)
        #print(cost)
        w=w-lr*dw
        bias=bias-lr*db
        change = abs((cost - cost_prev))
        cost_prev = cost
        if change < 1e-6 :
            break
        itr=itr-1

    #plt.plot(ypred)
    return w,bias,cost_list
```

In [18]:

```
train_uni.shape
```

Out[18]:

```
(400,)
```

Calling Model and recieving parameters

In [19]:

```
m,c,cost_list=model_Uni(train_uni,train_target)
```

```
Starting model
```

In [20]:

```
m
```

Out[20]:

```
0.8658272452976603
```

In [21]:

```
c
```

Out[21]:

```
-14.144941014426436
```

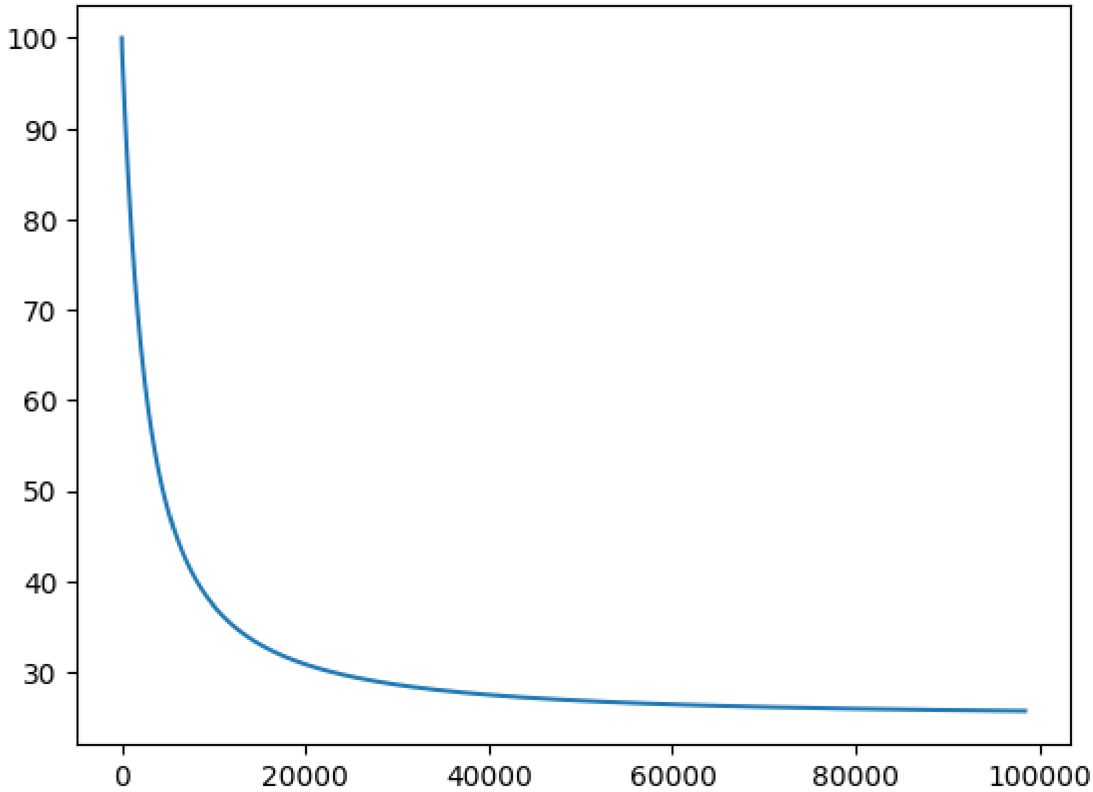
Error Plot

In [22]:

```
plt.plot(cost_list)
```

Out[22]:

```
[<matplotlib.lines.Line2D at 0x2abe29b1dc0>]
```



Predicting using Test Data

In [23]:

```
def predict(X,w=m,bias=c):
    n=X.shape[0]
    #ypred=[]
    for i in range (n):
        linear=w*X+bias
        ypred=1/(1+np.exp(-linear))
    ans=[]
    for i in ypred:
        if(i>0.5):
            ans.append(1)
        else:
            ans.append(0)
    print(ans)
    return ans
```

In [24]:

```
test_uni.shape[0]
```

Out[24]:

```
169
```

Testing Output

In [25]:

```
ypred=predict(test_uni)
```

```
[1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1,
1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0]
```

In [26]:

```
unique_elements, counts_elements = np.unique(ypred, return_counts=True)
print("Frequency of unique values of the said array:")
print(np.asarray((unique_elements, counts_elements)))
```

```
Frequency of unique values of the said array:
[[  0   1]
 [118  51]]
```

In [27]:

```
unique_elements, counts_elements = np.unique(test_target, return_counts=True)
print("Frequency of unique values of the said array:")
print(np.asarray((unique_elements, counts_elements)))
```

```
Frequency of unique values of the said array:
[[  0   1]
 [130  39]]
```

Calculating F1 Score

In [28]:

```
from sklearn.metrics import f1_score
f1_score(test_target, ypred, average='micro') #Using Sklearn for metric calculation
```

Out[28]:

```
0.9053254437869822
```

Result

F1 Score: 0.90

Logistic Regression with multiple features ¶

Importing Pandas, Numpy, Matplotlib, Seaborn libraries for creating Dataframe, doing mathematical operations efficiently, ploting graphs for better visualisation of data and results respectively.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df=pd.read_csv("breast_cancer.csv")
```

In [2]:

```
df
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	
565	926682	M	20.13	28.25	131.20	1261.0	
566	926954	M	16.60	28.08	108.30	858.1	
567	927241	M	20.60	29.33	140.10	1265.0	
568	92751	B	7.76	24.54	47.92	181.0	

569 rows × 33 columns

Pre-processing

Deleting unnecessary colomns

In [3]:

```
X=df.drop(['id','Unnamed: 32'],axis=1)
X
```

Out[3]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cc
0	M	17.99	10.38	122.80	1001.0	0.11840	
1	M	20.57	17.77	132.90	1326.0	0.08474	
2	M	19.69	21.25	130.00	1203.0	0.10960	
3	M	11.42	20.38	77.58	386.1	0.14250	
4	M	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	
564	M	21.56	22.39	142.00	1479.0	0.11100	
565	M	20.13	28.25	131.20	1261.0	0.09780	
566	M	16.60	28.08	108.30	858.1	0.08455	
567	M	20.60	29.33	140.10	1265.0	0.11780	
568	B	7.76	24.54	47.92	181.0	0.05263	

569 rows × 31 columns

Replacing characters of target feature to 1s and 0s

There are two categories in target column M for Malignant and B for Benign. Replacing M with 1 and B with 0.

In [4]:

```
X=X.replace({'M':1,'B':0})
X
```

Out[4]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cc
0	1	17.99	10.38	122.80	1001.0	0.11840	
1	1	20.57	17.77	132.90	1326.0	0.08474	
2	1	19.69	21.25	130.00	1203.0	0.10960	
3	1	11.42	20.38	77.58	386.1	0.14250	
4	1	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	
564	1	21.56	22.39	142.00	1479.0	0.11100	
565	1	20.13	28.25	131.20	1261.0	0.09780	
566	1	16.60	28.08	108.30	858.1	0.08455	
567	1	20.60	29.33	140.10	1265.0	0.11780	
568	0	7.76	24.54	47.92	181.0	0.05263	

569 rows × 31 columns

Checking for Correlation

In [5]:

```
X.corr()
```

Out[5]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	sı
diagnosis	1.000000	0.730029	0.415185	0.742636	0.708984	
radius_mean	0.730029	1.000000	0.323782	0.997855	0.987357	
texture_mean	0.415185	0.323782	1.000000	0.329533	0.321086	
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.986507	
area_mean	0.708984	0.987357	0.321086	0.986507	1.000000	
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.177028	
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.685983	
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823269	
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.151293	
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	
radius_se	0.567134	0.679090	0.275869	0.691765	0.732562	
texture_se	-0.008303	-0.097317	0.386358	-0.086761	-0.066280	
perimeter_se	0.556141	0.674172	0.281673	0.693135	0.726628	
area_se	0.548236	0.735864	0.259845	0.744983	0.800086	
smoothness_se	-0.067016	-0.222600	0.006614	-0.202694	-0.166777	
compactness_se	0.292999	0.206000	0.191975	0.250744	0.212583	
concavity_se	0.253730	0.194204	0.143293	0.228082	0.207660	
concave points_se	0.408042	0.376169	0.163851	0.407217	0.372320	
symmetry_se	-0.006522	-0.104321	0.009127	-0.081629	-0.072497	
fractal_dimension_se	0.077972	-0.042641	0.054458	-0.005523	-0.019887	
radius_worst	0.776454	0.969539	0.352573	0.969476	0.962746	
texture_worst	0.456903	0.297008	0.912045	0.303038	0.287489	
perimeter_worst	0.782914	0.965137	0.358040	0.970387	0.959120	
area_worst	0.733825	0.941082	0.343546	0.941550	0.959213	
smoothness_worst	0.421465	0.119616	0.077503	0.150549	0.123523	
compactness_worst	0.590998	0.413463	0.277830	0.455774	0.390410	
concavity_worst	0.659610	0.526911	0.301025	0.563879	0.512606	
concave points_worst	0.793566	0.744214	0.295316	0.771241	0.722017	
symmetry_worst	0.416294	0.163953	0.105008	0.189115	0.143570	
fractal_dimension_worst	0.323872	0.007066	0.119205	0.051019	0.003738	

31 rows × 31 columns

Extracting target Variable

In [6]:

```
target=X[['diagnosis']]
```

Only kept radius mean and concavity worst features for trainig and predicting

In [7]:

```
X=X[['radius_mean','concavity_worst']]
#X=X.values
X
```

Out[7]:

	radius_mean	concavity_worst
0	17.99	0.7119
1	20.57	0.2416
2	19.69	0.4504
3	11.42	0.6869
4	20.29	0.4000
...	...	...
564	21.56	0.4107
565	20.13	0.3215
566	16.60	0.3403
567	20.60	0.9387
568	7.76	0.0000

569 rows × 2 columns

Splitting Data for Train and Test & inserting a colomn of 1 in feature matix

In [10]:

```
xtrain=X.iloc[:400,:]
xtest=X.iloc[400:,:]
ytrain=target.iloc[:400]
ytest=target.iloc[400:]

ones=np.ones((xtrain.shape[0],1))
xtrain=np.hstack((ones,xtrain))
print(xtrain.shape)

ones=np.ones((xtest.shape[0],1))
xtest=np.hstack((ones,xtest))
xtest.shape
```

```
(400, 3)
```

Out[10]:

```
(169, 3)
```

In [11]:

```
print(xtrain.shape,xtest.shape,ytrain.shape,ytest.shape)
```

```
(400, 3) (169, 3) (400, 1) (169, 1)
```

In [12]:

```
theta=np.zeros((xtrain.shape[1],1))
theta.shape
```

Out[12]:

```
(3, 1)
```

Building Model

In [13]:

```
def model(x,y,theta,lr=0.007,itr=100000):
    cost_list=[]
    n=x.shape[0]

    for i in range (itr):
        l=np.dot(x,theta)
        h=1/(1+np.exp(-l))
        err=h-y
        cost=-(1/n)*np.sum((y*np.log(h))+((1-y)*np.log(1-h)))
#         print(cost)
        cost_list.append(cost)

        grad=(1/n)*(np.dot(x.T,err))
        theta=theta-lr*grad

    return theta,cost_list
```

Calling Model

In [14]:

```
theta,cost=model(xtrain,ytrain,theta)
```

In [15]:

```
cost
```

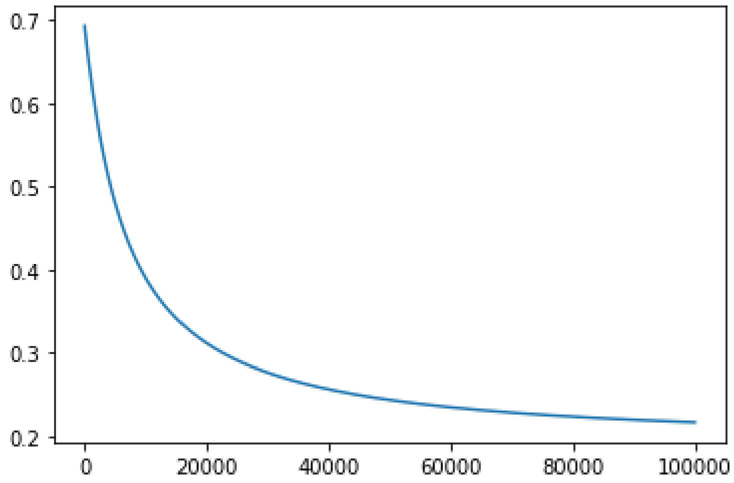
```
diagnosis    0.686821
dtype: float64,
diagnosis    0.686755
dtype: float64,
diagnosis    0.686689
dtype: float64,

diagnosis    0.686623
dtype: float64,
diagnosis    0.686556
dtype: float64,
diagnosis    0.68649
dtype: float64,
diagnosis    0.686424
dtype: float64,
diagnosis    0.686358
dtype: float64,
diagnosis    0.686292
dtype: float64,
diagnosis    0.686226
```

Plotting the change of cost with increasing in iterations

In [16]:

```
plt.plot(cost)
plt.show()
```



Prediction

In [17]:

```
def predict(x,theta):
    l=np.dot(x,theta)
    ypred=1/(1+np.exp(-l))
    ans=[]

    for i in ypred:
        if(i>=0.5):
            ans.append(1)
        else:
            ans.append(0)

    #print(ypred)
    return ans
```

xtest contains test data and theta are model parameters

In [18]:

```
ypred=predict(xtest,theta)
```

Result

Printing prediced value of Test Data

In [19]:

```
ypred
```

Out[19]:

```
[1,
 0,
 0,
 0,
 0,
 0,
 1,
 0,
 1,
 0,
 0,
 0,
 0,
 1,
 0,
 0,
 0,
 1,
```

F1 Score

In [21]:

```
from sklearn.metrics import f1_score
f1=f1_score(ytest,ypred,average='micro')
f1
```

Out[21]:

```
0.9171597633136095
```

Accuracy

In [22]:

```

# ypred=np.array(ypred, dtype='int64')
# y_= ypred > 0.5
# y_=np.array(y_, dtype='int64')
# acc=(1-np.sum(np.absolute(y_-ytest))/ytest.shape[0])
# acc

def accuracy_1(X, Y, theta):

    lines_1 = np.dot(X,theta)
    y_pred_1 = 1/(1+np.exp(-lines_1))

    y_pred_1 = y_pred_1 > 0.5

    y_pred_1 = np.array(y_pred_1, dtype='int64')
    acc = (1-np.sum(np.absolute(y_pred_1-Y))/Y.shape[0])*100

    print("Accuracy of the model is : ", round(acc,2), "%")
    return y_pred_1
```

In [23]:

```
A = accuracy_1(xtest,ytest,theta)
```

```
Accuracy of the model is :  diagnosis    91.72
dtype: float64 %
```

Result Summary

F1 Score : 0.9171

Accuracy : 91.72

Naive Baye's Classifier

Importing Pandas, Numpy, Matplotlib, Seaborn libraries for creating Dataframe, doing mathematical operations efficiently, ploting graphs for better visualisation of data and results respectively.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df=pd.read_csv("breast_cancer.csv")
df
```

Out[1]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	
565	926682	M	20.13	28.25	131.20	1261.0	
566	926954	M	16.60	28.08	108.30	858.1	
567	927241	M	20.60	29.33	140.10	1265.0	
568	92751	B	7.76	24.54	47.92	181.0	

569 rows × 33 columns

In [ ]:

```

```

Pre-Processing

Data contains columns like id and Unnamed 32. Id is not required for any type of prediction or training and Unnamed 32 is column of all NULL valus. So, dropping those two columns

In [2]:

```
X=df.drop(['id','Unnamed: 32'],axis=1)
X
```

Out[2]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cc
0	M	17.99	10.38	122.80	1001.0	0.11840	
1	M	20.57	17.77	132.90	1326.0	0.08474	
2	M	19.69	21.25	130.00	1203.0	0.10960	
3	M	11.42	20.38	77.58	386.1	0.14250	
4	M	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	
564	M	21.56	22.39	142.00	1479.0	0.11100	
565	M	20.13	28.25	131.20	1261.0	0.09780	
566	M	16.60	28.08	108.30	858.1	0.08455	
567	M	20.60	29.33	140.10	1265.0	0.11780	
568	B	7.76	24.54	47.92	181.0	0.05263	

569 rows × 31 columns

Our target variable 'diagnosis' is a character feature where M denotes Malignant or Cancerous and B denotes Benign or Not cancerous. Changin M to 1 and B to 0 for training purposes.

In [3]:

```
X=X.replace({'M':1,'B':0})
X
```

Out[3]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cc
0	1	17.99	10.38	122.80	1001.0	0.11840	
1	1	20.57	17.77	132.90	1326.0	0.08474	
2	1	19.69	21.25	130.00	1203.0	0.10960	
3	1	11.42	20.38	77.58	386.1	0.14250	
4	1	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	
564	1	21.56	22.39	142.00	1479.0	0.11100	
565	1	20.13	28.25	131.20	1261.0	0.09780	
566	1	16.60	28.08	108.30	858.1	0.08455	
567	1	20.60	29.33	140.10	1265.0	0.11780	
568	0	7.76	24.54	47.92	181.0	0.05263	

569 rows × 31 columns

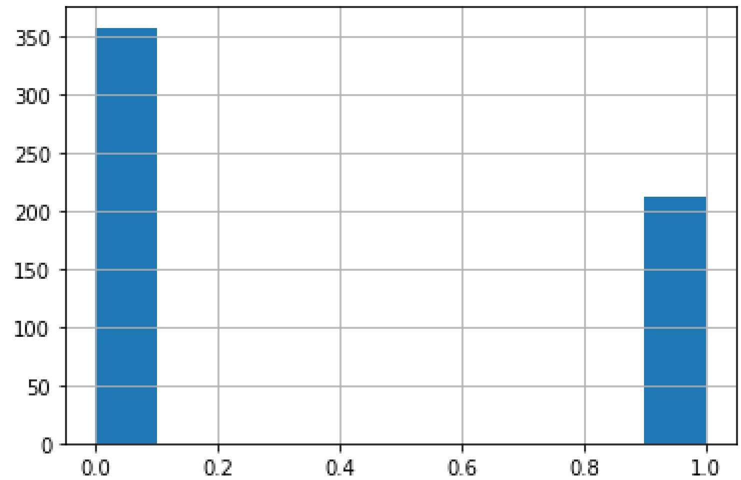
Looking at target distribution(Diagnosis)

In [4]:

```
X["diagnosis"].hist()
```

Out[4]:

<AxesSubplot:>



No. of Benign is more that no. of malignant. Data is not properly distributed.

Extracting Target variable

In [5]:

```
target=X[['diagnosis']]
target
```

Out[5]:

	diagnosis
0	1
1	1
2	1
3	1
4	1
...	...
564	1
565	1
566	1
567	1
568	0

569 rows × 1 columns

Feature Selection

Now we need to select the input features for training the model. For this Pearson Correlation technique is used. First we analyse the using correlation matrix and heatmap. Only selecting the features which has highest correlation with output feature and also exluding the features which are highly correlated among themselves only.

In [6]:

```
X.corr()
```

Out[6]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	sr
diagnosis	1.000000	0.730029	0.415185	0.742636	0.708984	
radius_mean	0.730029	1.000000	0.323782	0.997855	0.987357	
texture_mean	0.415185	0.323782	1.000000	0.329533	0.321086	
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.986507	
area_mean	0.708984	0.987357	0.321086	0.986507	1.000000	
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.177028	
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.685983	
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823269	
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.151293	
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	
radius_se	0.567134	0.679090	0.275869	0.691765	0.732562	
texture_se	-0.008303	-0.097317	0.386358	-0.086761	-0.066280	
perimeter_se	0.556141	0.674172	0.281673	0.693135	0.726628	
area_se	0.548236	0.735864	0.259845	0.744983	0.800086	
smoothness_se	-0.067016	-0.222600	0.006614	-0.202694	-0.166777	
compactness_se	0.292999	0.206000	0.191975	0.250744	0.212583	
concavity_se	0.253730	0.194204	0.143293	0.228082	0.207660	
concave points_se	0.408042	0.376169	0.163851	0.407217	0.372320	
symmetry_se	-0.006522	-0.104321	0.009127	-0.081629	-0.072497	
fractal_dimension_se	0.077972	-0.042641	0.054458	-0.005523	-0.019887	
radius_worst	0.776454	0.969539	0.352573	0.969476	0.962746	
texture_worst	0.456903	0.297008	0.912045	0.303038	0.287489	
perimeter_worst	0.782914	0.965137	0.358040	0.970387	0.959120	
area_worst	0.733825	0.941082	0.343546	0.941550	0.959213	
smoothness_worst	0.421465	0.119616	0.077503	0.150549	0.123523	
compactness_worst	0.590998	0.413463	0.277830	0.455774	0.390410	
concavity_worst	0.659610	0.526911	0.301025	0.563879	0.512606	
concave points_worst	0.793566	0.744214	0.295316	0.771241	0.722017	
symmetry_worst	0.416294	0.163953	0.105008	0.189115	0.143570	
fractal_dimension_worst	0.323872	0.007066	0.119205	0.051019	0.003738	

31 rows × 31 columns

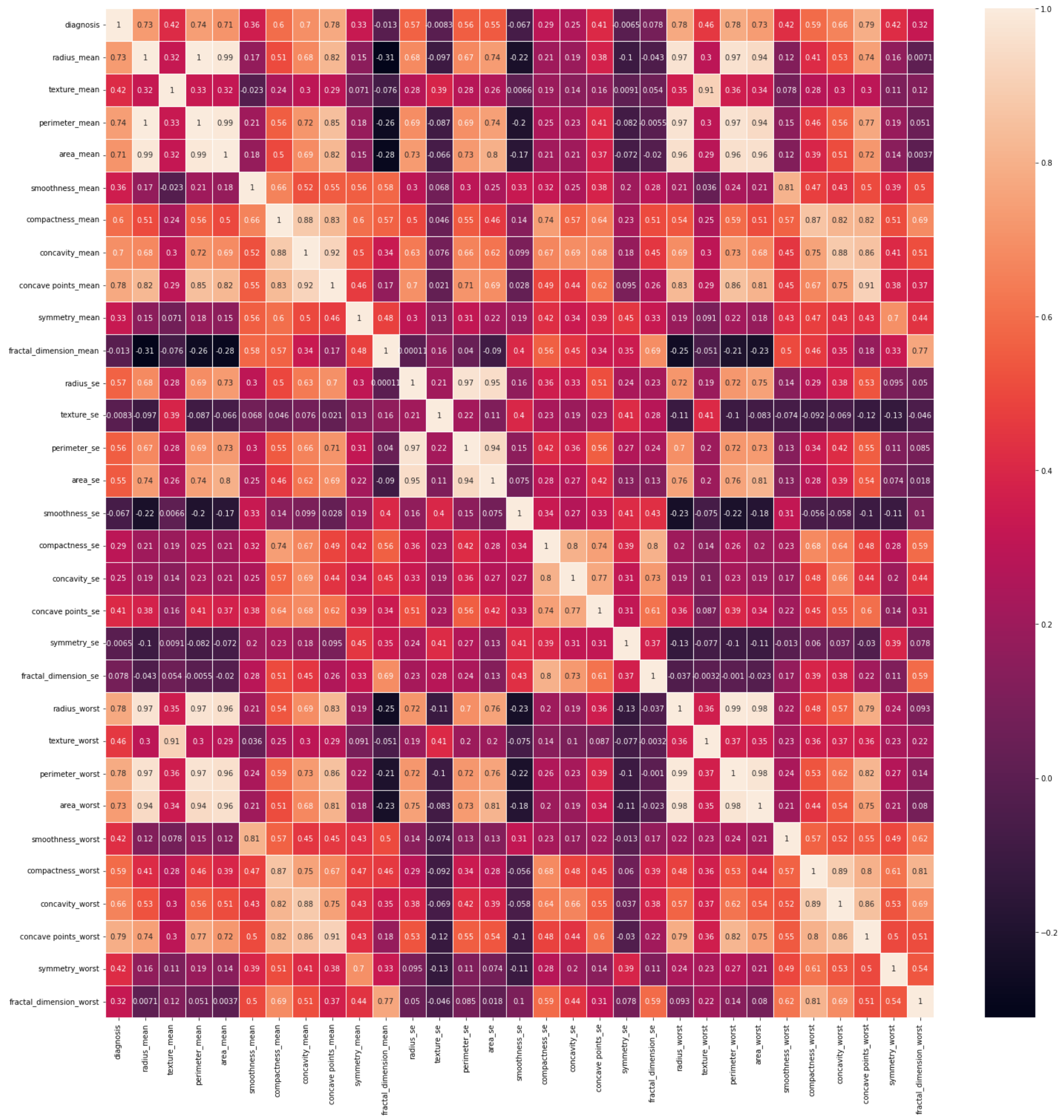
Looking for correlation from heatmap

In [7]:

```
plt.figure(figsize=(25, 25))
sns.heatmap(X.corr(),annot=True,linewidth=1)
```

Out[7]:

```
<AxesSubplot:>
```



The features related to area, perimeter, radius are highly correlated to diagnosis, but also they are correlated to themselves because all are related to radius only. So, taking only one among them.

Choosing radius_mean, texture_mean, smoothness_mean ans features

In [8]:

```
features=X[['radius_mean','texture_mean','smoothness_mean']]
features
```

Out[8]:

	radius_mean	texture_mean	smoothness_mean
0	17.99	10.38	0.11840
1	20.57	17.77	0.08474
2	19.69	21.25	0.10960
3	11.42	20.38	0.14250
4	20.29	14.34	0.10030
...	...	...	...
564	21.56	22.39	0.11100
565	20.13	28.25	0.09780
566	16.60	28.08	0.08455
567	20.60	29.33	0.11780
568	7.76	24.54	0.05263

569 rows × 3 columns

Splitting train and test data

In [9]:

```
xtrain=features.iloc[:400,:]
xtest=features.iloc[400:,:]
ytrain=target.iloc[:400]
ytest=target.iloc[400:]

#print(xtrain,xtest,ytrain,ytest)
```

In [10]:

```
result = pd.concat([features, target], axis=1)
result
train=result.iloc[:400,:]
test=result.iloc[400:,:]
```

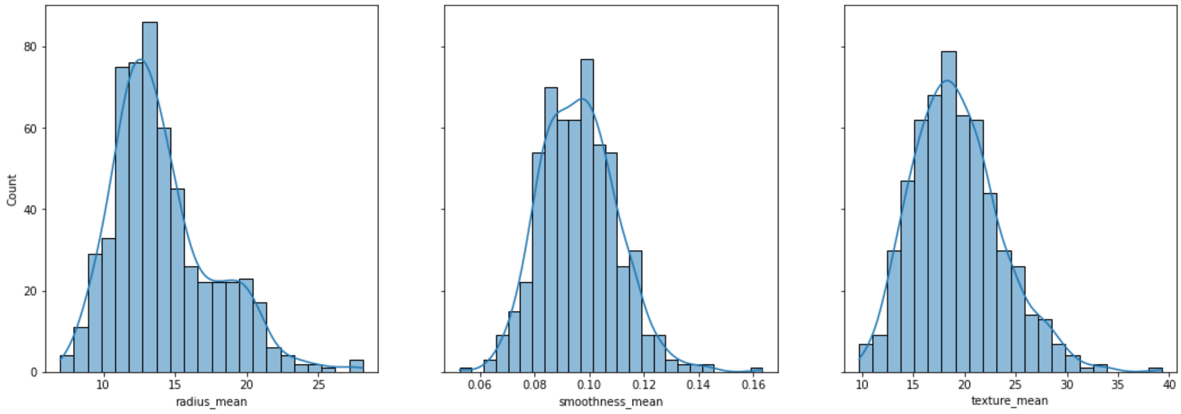
Checking the Distribution of the Data

In [19]:

```
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)
sns.histplot(features, ax=axes[0], x="radius_mean", kde=True)
sns.histplot(features, ax=axes[1], x="smoothness_mean", kde=True)
sns.histplot(features, ax=axes[2], x="texture_mean", kde=True)
```

Out[19]:

```
<AxesSubplot:xlabel='texture_mean', ylabel='Count'>
```



All the features are following almost Normal/Gaussian Distribution

calculating Prior Probabilities

P(Y=y) for all y

In [12]:

```
def cal_prior(train):
    #Extracting all classes
    classes=sorted(list(train['diagnosis'].unique()))

    #Lengh=Total length of the training data
    length=len(train)

    #Calculation prior of all the classes
    prior=[]
    for c in classes:
        #n=No of c in train data
        n=len(train[train['diagnosis']==c])
        prior.append(n/length)

    return prior
```

Calculate P(X=x|Y=y) using Gaussian distribution

In [13]:

```
def gaussian_likelihood(train,feature,feature_val,label):

    train = train[train['diagnosis']==label]

    mean, std = train[feature].mean(), train[feature].std()

    x = (1 / (np.sqrt(2 * np.pi) * std)) *  np.exp(-((feature_val-mean)**2 / (2 * std**2 ))

    return x
```

Naive_Baye's Model

In [14]:

```
def naive_bayes(train, X):
    features = list(train.columns)[:-1]

    prior = cal_prior(train)

    ypred = []
    for x in X:
        labels = sorted(list(train['diagnosis'].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):

                likelihood[j] *= gaussian_likelihood(train, features[i], x[i], labels[j])

        post_prob = [1]*len(labels)

        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        ypred.append(np.argmax(post_prob))


    return np.array(ypred)
```

Calling Model on Test data

In [15]:

```
Y_pred = naive_bayes(train, xtest.values)
```

Printing predicted value

In [16]:

```
Y_pred
```

Out[16]:

```
array([1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0], dtype=int64)
```

Result

Calculating F1 Score

In [17]:

```
from sklearn.metrics import confusion_matrix, f1_score
print(f1_score(ytest, Y_pred,average='micro'))
```

```
0.8402366863905325
```

F1 Score: 0.84

Result Analysis

Metrics	Linear Regression (Univariate)	Linear Regression (Multi-variate)	Linear Regression (Multi-variate) Closed Form	Logistic Regression (Uni-variate)	Logistic Regression (Multi-variate)	Naïve Bayes Classifier
MSE	62.69	47.34	77.77	NA	NA	NA
F1 Score	NA	NA	NA	0.9	**0.91**	**0.8**