IBM Advance Data Science Capstone Project

Import Required Libraries

In [19]:

```
# import data wrangling and visual libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv), data manipulation as in SQL
import matplotlib.pyplot as plt # this is used for the plot the graph
import seaborn as sns # used for plot interactive graph. I like it most for plot
%matplotlib inline

# to check the dircteory
import os
print(os.listdir("../Data"))
```

Loading Data

In [67]:

```
# Read Field Names File and Show Top 2 Records
with open('../Data/field_names.txt', 'r') as the_file:
    col_names = [line.strip() for line in the_file.readlines()]
print(col_names, '\n')

# Read Data File and Show Top 2 Records
data = pd.read_csv("../Data/breast-cancer.csv",header=None)
print(data.head(2))
```

```
['ID', 'diagnosis', 'radius_mean', 'radius_sd_error', 'radius_worst', 'texture_mean', 'te
xture_sd_error', 'texture_worst', 'perimeter_mean', 'perimeter_sd_error', 'perimeter_wors
t', 'area_mean', 'area_sd_error', 'area_worst', 'smoothness_mean', 'smoothness_sd_error',
'smoothness_worst', 'compactness_mean', 'compactness_sd_error', 'compactness_worst', 'con
cavity_mean', 'concavity_sd_error', 'concavity_worst', 'concave_points_mean', 'concave_po
ints_sd_error', 'concave_points_worst', 'symmetry_mean', 'symmetry_sd_error', 'symmetry_w
orst', 'fractal_dimension_mean', 'fractal_dimension_sd_error', 'fractal_dimension_worst']


        0  1      2      3      4       5        6        7       8        9   \
0  842302  M  17.99  10.38  122.8  1001.0  0.11840  0.27760  0.3001  0.14710
1  842517  M  20.57  17.77  132.9  1326.0  0.08474  0.07864  0.0869  0.07017

    ...       22     23     24      25      26      27      28      29   \
0   ...     25.38  17.33  184.6  2019.0  0.1622  0.6656  0.7119  0.2654
1   ...     24.99  23.41  158.8  1956.0  0.1238  0.1866  0.2416  0.1860

       30       31
0  0.4601  0.11890
1  0.2750  0.08902

[2 rows x 32 columns]
```

In [68]:

```
# set Column Names and display top 3 records
```

```python
data.columns = col_names
print(data.head(3))
# to see last 3 records
print(data.tail(3))
```

```
         ID diagnosis  radius_mean  radius_sd_error  radius_worst  \
0    842302         M        17.99            10.38         122.8
1    842517         M        20.57            17.77         132.9
2  84300903         M        19.69            21.25         130.0

   texture_mean  texture_sd_error  texture_worst  perimeter_mean  \
0        1001.0           0.11840        0.27760          0.3001
1        1326.0           0.08474        0.07864          0.0869
2        1203.0           0.10960        0.15990          0.1974

   perimeter_sd_error           ...              concavity_worst  \
0             0.14710           ...                        25.38
1             0.07017           ...                        24.99
2             0.12790           ...                        23.57

   concave_points_mean  concave_points_sd_error  concave_points_worst  \
0                17.33                    184.6                2019.0
1                23.41                    158.8                1956.0
2                25.53                    152.5                1709.0

   symmetry_mean  symmetry_sd_error  symmetry_worst  fractal_dimension_mean\
0         0.1622             0.6656          0.7119  0.2654
1         0.1238             0.1866          0.2416                  0.1860
2         0.1444             0.4245          0.4504                  0.2430

   fractal_dimension_sd_error  fractal_dimension_worst
0                      0.4601                  0.11890
1                      0.2750                  0.08902
2                      0.3613                  0.08758

[3 rows x 32 columns]
         ID diagnosis  radius_mean  radius_sd_error  radius_worst  \
566  926954         M        16.60            28.08        108.30
567  927241         M        20.60            29.33        140.10
568   92751         B         7.76            24.54         47.92

     texture_mean  texture_sd_error  texture_worst  perimeter_mean  \
566         858.1           0.08455        0.10230         0.09251
567        1265.0           0.11780        0.27700         0.35140
568         181.0           0.05263        0.04362         0.00000

     perimeter_sd_error           ...              concavity_worst  \
566             0.05302           ...                       18.980
567             0.15200           ...                       25.740
568             0.00000           ...                        9.456

     concave_points_mean  concave_points_sd_error  concave_points_worst  \
566                34.12                   126.70                1124.0
567                39.42                   184.60                1821.0
568                30.37                    59.16                 268.6

     symmetry_mean  symmetry_sd_error  symmetry_worst  fractal_dimension_mean  \
566        0.11390            0.30940          0.3403                  0.1418
567        0.16500            0.86810          0.9387                  0.2650
568        0.08996            0.06444          0.0000                  0.0000

     fractal_dimension_sd_error  fractal_dimension_worst
566                      0.2218                  0.07820
567                      0.4087                  0.12400
568                      0.2871                  0.07039

[3 rows x 32 columns]
```

Data Wrangling

We have successfully loaded data. Now lets look at the type of data we have.

In [52]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
ID                               569 non-null int64
diagnosis                        569 non-null object
radius_mean                      569 non-null float64
radius_sd_error                  569 non-null float64
radius_worst                     569 non-null float64
texture_mean                     569 non-null float64
texture_sd_error                 569 non-null float64
texture_worst                    569 non-null float64
perimeter_mean                   569 non-null float64
perimeter_sd_error               569 non-null float64
perimeter_worst                  569 non-null float64
area_mean                        569 non-null float64
area_sd_error                    569 non-null float64
area_worst                       569 non-null float64
smoothness_mean                  569 non-null float64
smoothness_sd_error              569 non-null float64
smoothness_worst                 569 non-null float64
compactness_mean                 569 non-null float64
compactness_sd_error    569    non-null    float64
compactness_worst                569 non-null float64
concavity_mean                   569 non-null float64
concavity_sd_error               569 non-null float64
concavity_worst                  569 non-null float64
concave_points_mean              569 non-null float64
concave_points_sd_error   569   non-null   float64
concave_points_worst    569    non-null    float64
symmetry_mean                    569 non-null float64
symmetry_sd_error                569 non-null float64
symmetry_worst                   569 non-null float64
fractal_dimension_mean    569   non-null   float64
fractal_dimension_sd_error  569  non-null  float64
fractal_dimension_worst   569   non-null   float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.3+ KB
```

So we have 569 records against 32 columns and all of them have 569 non-null records and the data type is float64.

Lets keep the Diagnosis data and drop ID and diagnosis columns as they are not needed

In [57]:

```
# y includes our labels and x includes our features
y = data.diagnosis                          # M or B
list = ['ID','diagnosis']
x = data.drop(list,axis = 1 )
x.head()
```

Remaining features are representing the 3 computations (Mean, Standard Deviation Error and Worst) against single feautre. Lets group them into 3 categories.

In [84]:

```
#list of column names that match with Mean
mean_cols = [col for col in x.columns if '_mean' in col]
print(mean_cols,'\n')

#list of column names that match with SD
sd_cols = [col for col in x.columns if '_sd' in col]
print(mean_cols, '\n')
```

```
#list of column names that match with Mean
worst_cols = [col for col in x.columns if '_worst' in col]
print(mean_cols)
```

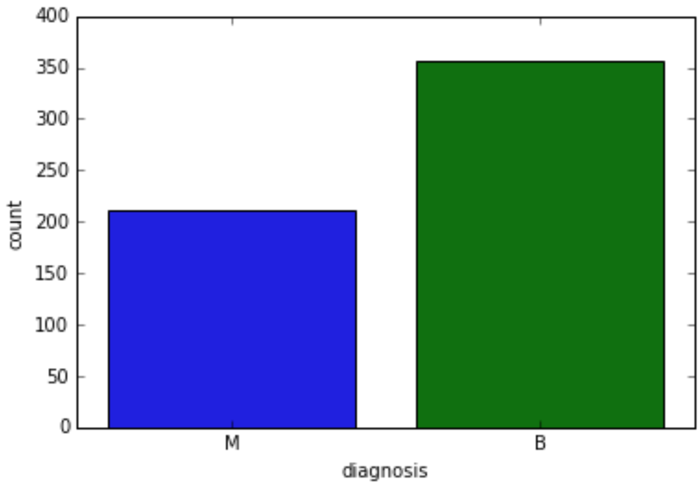
```
['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compac
tness_mean', 'concavity_mean', 'concave_points_mean', 'symmetry_mean', 'fractal_dimension
_mean']

['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compac
tness_mean', 'concavity_mean', 'concave_points_mean', 'symmetry_mean', 'fractal_dimension
_mean']

['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compac
tness_mean', 'concavity_mean', 'concave_points_mean', 'symmetry_mean', 'fractal_dimension
_mean']
```

Before moving forward to the data analysis of features. Lets have a quick look of the labels we are going to predict

In [71]:

```
ax = sns.countplot(y,label="Count")       # M = 212, B = 357
B, M = y.value_counts()
print('Number of Benign: ',B)
print('Number of Malignant : ',M)
```

```
Number of Benign:  357
Number of Malignant :  212
```



So far, we are unfimilar with the data and its features, and what they are representing. In real world, we come acroos to many different problems where we don't know the meanining of features but to imagine in our minds. What we must know is the distribution of data like **variance, standart deviation, number of sample (count) or max min values**. These type of information helps to understand the data, how normally distributed it is, or it has skewed distribution.

In [85]:

```
x.describe()
```

Out[85]:

	radius_mean	radius_sd_error	radius_worst	texture_mean	texture_sd_error	texture_worst	perimeter_mean	perimeter_
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	56
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	

	radius_mean	radius_sd_error	radius_worst	texture_mean	texture_sd_error	texture_worst	perimeter_mean	perimeter_
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	

8 rows × 30 columns

The summary statistics helps us to understand is we need standirdization or normalization before visualization, feature selection or classificaiton.

Before moving to Exploratory Analysis, lets develop a function to generate bootstrap samples.

In [161]:

```
def bootstrap_resample(X, n=None):
    """ Bootstrap resample an array_like
    Parameters
    ----------
    X : array_like
      data to resample
    n : int, optional
      length of resampled array, equal to len(X) if n==None
    Results
    -------
    returns X_resamples
    """
    if isinstance(X, pd.Series):
        X = X.copy()
        X.index = range(len(X.index))
    if n == None:
        n = len(X)

    resample_i = np.floor(np.random.rand(n)*len(X)).astype(int)
    X_resample = np.array(X[resample_i])
    return X_resample
```

In [181]:

```
# Create new df variable for resampled data
df_resampled = pd.DataFrame(index=df.index, columns=df.columns, dtype=df.dtypes)
for col in x.columns:
    df_resampled[col] = bootstrap_resample(x[col])

# original data
x.ix[:50,:50]
```

Out[181]:

	radius_mean	radius_sd_error	radius_worst	texture_mean	texture_sd_error	texture_worst	perimeter_mean	perimeter_sd_
0	17.990	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14
1	20.570	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07
2	19.690	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12
3	11.420	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10
4	20.290	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10
5	12.450	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08
6	18.250	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07
7	13.710	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05
8	13.000	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.09
9	12.460	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.08
10	16.020	23.24	102.70	797.8	0.08206	0.06669	0.03299	0.03
11	15.780	17.89	103.60	781.0	0.09710	0.12920	0.09954	0.06
12	19.170	24.80	132.40	1123.0	0.09740	0.24580	0.20650	0.11

	radius_mean	radius_sd_error	radius_worst	texture_mean	texture_sd_error	texture_worst	perimeter_mean	perimeter_sd
13	15.850	23.95	103.70	782.7	0.08401	0.10020	0.09938	0.0
14	13.730	22.61	93.60	578.3	0.11310	0.22930	0.21280	0.08
15	14.540	27.54	96.73	658.8	0.11390	0.15950	0.16390	0.07
16	14.680	20.13	94.74	684.5	0.09867	0.07200	0.07395	0.05
17	16.130	20.68	108.10	798.8	0.11700	0.20220	0.17220	0.10
18	19.810	22.15	130.00	1260.0	0.09831	0.10270	0.14790	0.09
19	13.540	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.04
20	13.080	15.71	85.63	520.0	0.10750	0.12700	0.04568	0.03
21	9.504	12.44	60.34	273.9	0.10240	0.06492	0.02956	0.02
22	15.340	14.26	102.50	704.4	0.10730	0.21350	0.20770	0.09
23	21.160	23.04	137.20	1404.0	0.09428	0.10220	0.10970	0.08
24	16.650	21.38	110.00	904.6	0.11210	0.14570	0.15250	0.09
25	17.140	16.40	116.00	912.7	0.11860	0.22760	0.22290	0.14
26	14.580	21.53	97.41	644.8	0.10540	0.18680	0.14250	0.08
27	18.610	20.25	122.10	1094.0	0.09440	0.10660	0.14900	0.07
28	15.300	25.27	102.40	732.4	0.10820	0.16970	0.16830	0.08
29	17.570	15.05	115.00	955.1	0.09847	0.11570	0.09875	0.07
30	18.630	25.11	124.80	1088.0	0.10640	0.18870	0.23190	0.12
31	11.840	18.70	77.93	440.6	0.11090	0.15160	0.12180	0.05
32	17.020	23.98	112.80	899.3	0.11970	0.14960	0.24170	0.12
33	19.270	26.47	127.90	1162.0	0.09401	0.17190	0.16570	0.07
34	16.130	17.88	107.00	807.2	0.10400	0.15590	0.13540	0.07
35	16.740	21.59	110.10	869.5	0.09610	0.13360	0.13480	0.06
36	14.250	21.72	93.63	633.0	0.09823	0.10980	0.13190	0.05
37	13.030	18.42	82.61	523.8	0.08983	0.03766	0.02562	0.02
38	14.990	25.20	95.54	698.8	0.09387	0.05131	0.02398	0.02
39	13.480	20.82	88.40	559.2	0.10160	0.12550	0.10630	0.05
40	13.440	21.58	86.18	563.0	0.08162	0.06031	0.03110	0.02
41	10.950	21.35	71.90	371.1	0.12270	0.12180	0.10440	0.05
42	19.070	24.81	128.30	1104.0	0.09081	0.21900	0.21070	0.09
43	13.280	20.28	87.32	545.2	0.10410	0.14360	0.09847	0.06
44	13.170	21.81	85.42	531.5	0.09714	0.10470	0.08259	0.05
45	18.650	17.60	123.70	1076.0	0.10990	0.16860	0.19740	0.10
46	8.196	16.84	51.71	201.9	0.08600	0.05943	0.01588	0.00
47	13.170	18.66	85.98	534.6	0.11580	0.12310	0.12260	0.07
48	12.050	14.63	78.04	449.3	0.10310	0.09092	0.06592	0.02
49	13.490	22.30	86.91	561.0	0.08752	0.07698	0.04751	0.03
50	11.760	21.60	74.72	427.9	0.08637	0.04966	0.01657	0.01

51 rows × 30 columns

In [182]:

```
#sample data
df_resampled.ix[:50,:50]
```

Out[182]:

	concavity_worst	concave_points_worst	radius_worst	radius_mean	radius_sd_error	texture_mean	texture_sd_error	textur

	concavity_worst	concave_points_worst	radius_worst	radius_mean	radius_sd_error	texture_mean	texture_sd_error	textur
0	17.560	1233.0	73.21	16.810	23.04	651.9	0.10990	
1	24.560	1479.0	143.70	12.320	10.89	420.3	0.11220	
2	12.360	331.6	97.03	11.930	15.79	408.2	0.07966	
3	16.510	471.4	130.70	9.567	17.68	378.2	0.09714	
4	13.030	567.7	87.32	13.660	18.59	324.9	0.10490	
5	17.380	973.1	125.50	16.030	16.83	1878.0	0.12370	
6	12.780	474.2	78.04	19.730	27.08	992.1	0.11860	
7	14.400	939.7	76.83	9.755	16.85	1214.0	0.08974	
8	14.990	614.9	103.70	19.160	39.28	221.2	0.08223	
9	25.700	2232.0	66.20	8.888	14.78	1311.0	0.09989	
10	11.060	1025.0	82.50	9.777	18.84	633.0	0.09905	
11	13.780	806.9	84.06	11.200	24.04	556.7	0.10640	
12	21.200	367.0	78.99	11.900	20.28	1482.0	0.09029	
13	13.330	268.6	98.92	28.110	20.66	685.9	0.07838	
14	12.840	527.2	70.15	19.690	12.83	321.4	0.06995	
15	12.410	2384.0	91.43	9.567	16.68	800.0	0.08675	
16	11.600	762.6	73.16	9.504	18.14	748.9	0.07557	
17	14.190	808.2	82.69	11.500	27.08	552.4	0.09427	
18	11.250	302.0	66.52	16.130	16.70	250.5	0.07941	
19	11.920	869.3	121.30	10.510	19.24	349.6	0.11410	
20	16.890	2403.0	69.28	19.190	22.44	678.1	0.12480	
21	15.770	1866.0	129.70	14.060	21.35	507.6	0.10280	
22	11.150	826.0	88.73	15.100	14.96	538.4	0.09198	
23	11.980	436.6	87.38	12.540	12.22	448.6	0.08915	
24	22.930	2384.0	70.67	16.170	14.23	666.0	0.11840	
25	19.590	1872.0	100.30	13.430	15.70	515.9	0.10630	
26	17.790	1349.0	72.17	17.290	16.02	420.3	0.07376	
27	19.770	745.5	95.77	12.050	23.77	427.3	0.09950	
28	25.580	674.7	81.35	12.560	15.62	541.6	0.11700	
29	12.680	395.4	61.06	13.340	24.98	537.3	0.07937	
30	12.470	328.1	92.41	15.280	18.90	471.3	0.08054	
31	11.930	1646.0	85.63	12.700	18.00	651.0	0.10020	
32	14.200	543.4	92.41	7.729	14.96	920.6	0.09579	
33	15.400	1025.0	130.70	21.560	20.86	514.3	0.09168	
34	15.790	1600.0	98.17	10.750	13.98	458.4	0.09037	
35	15.750	760.2	94.25	13.340	16.94	537.9	0.09831	
36	13.740	300.2	93.86	21.090	16.85	477.4	0.10890	
37	11.020	591.2	97.65	13.710	19.34	857.6	0.08206	
38	12.760	1261.0	124.40	19.810	25.13	747.2	0.08801	
39	9.965	310.1	82.61	14.920	11.89	493.8	0.10070	
40	17.870	470.9	107.00	23.090	24.99	311.7	0.09879	
41	13.600	374.4	96.12	12.060	19.59	1052.0	0.10420	
42	36.040	856.9	71.24	13.170	26.29	1206.0	0.10120	
43	20.270	455.7	61.93	20.340	20.18	476.3	0.08817	
44	24.330	1298.0	85.48	12.200	21.01	481.9	0.16340	

45	concavity_worst	concave_points_worst	radius_worst	radius_mean	radius_sd_error	texture_mean	texture_sd_error	textur
45	16.790	[illegible]	78.41	10.250	16.49	455.3	0.12780	
46	10.510	719.8	75.17	11.300	16.67	571.8	0.10380	
47	17.010	351.9	89.46	19.190	19.12	320.8	0.10420	
48	25.300	811.3	76.31	9.683	12.96	201.9	0.12570	
49	18.810	626.9	60.11	11.140	18.77	1265.0	0.12370	
50	20.580	605.5	105.10	17.080	21.68	386.3	0.09699	

51 rows × 30 columns

In [192]:

```
print("\n radius_mean of Original & Sampled dataset")
x.radius_mean.mean(), df_resampled.radius_mean.mean()
```

```
 radius_mean of Original & Sampled dataset
```

Out[192]:

```
(14.127291739894563, 14.144697715289995)
```

Visualization

For data visualization, we are going to use seaborn plots. Violin and Swarm plots usually helps us to understand data easily.

First, we need to perform normalization/standirdization of data. Because differences between values of features are very high to observe inplot

In [95]:

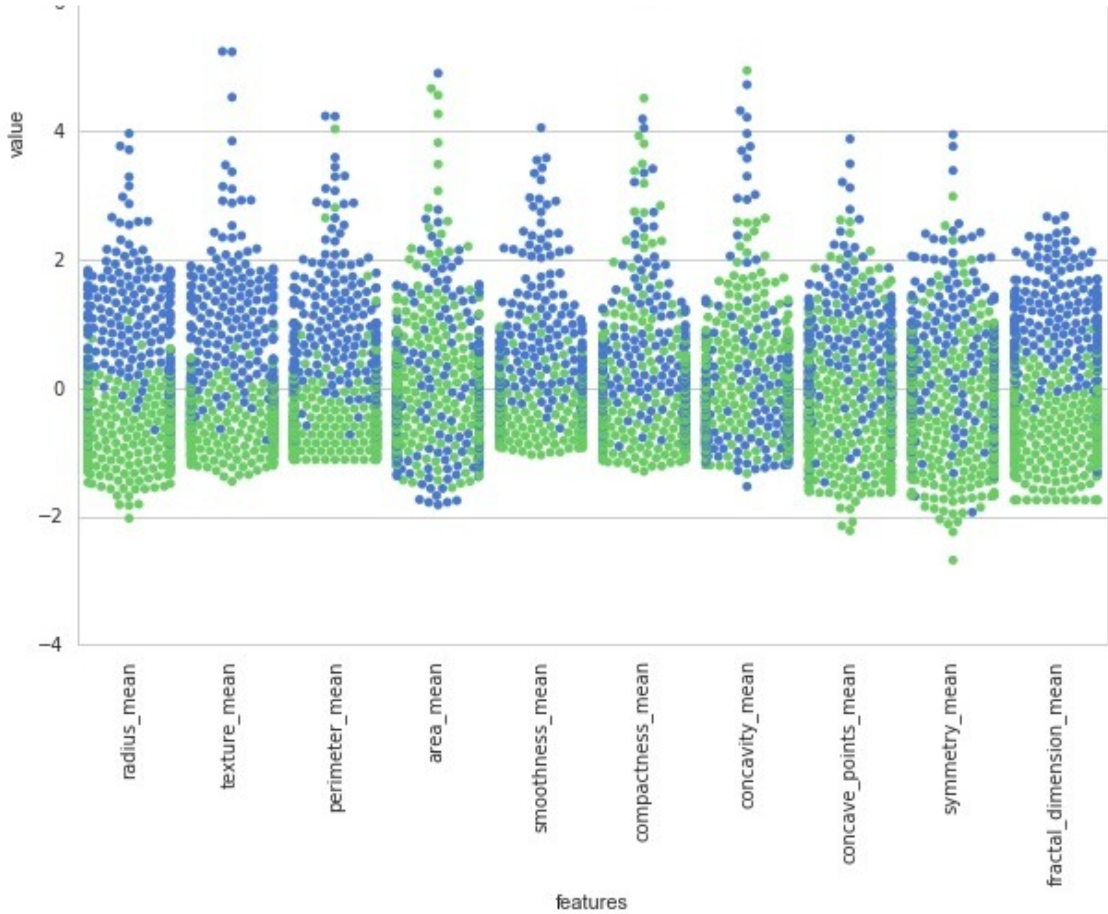
```
data = x
data_nor= (data - data.mean()) / (data.std())    # standardization
```

In [118]:

```
sns.set(style="whitegrid", palette="muted")
data = pd.concat([y,data_nor[mean_cols]],axis=1) # concat data to form new
data = pd.melt(data,id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(10,10))
import time
tic = time.time()
sns.swarmplot(x="features", y="value", hue="diagnosis", data=data)

plt.xticks(rotation=90)
```

Out[118]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), <a list of 10 Text xticklabel objects>)
```

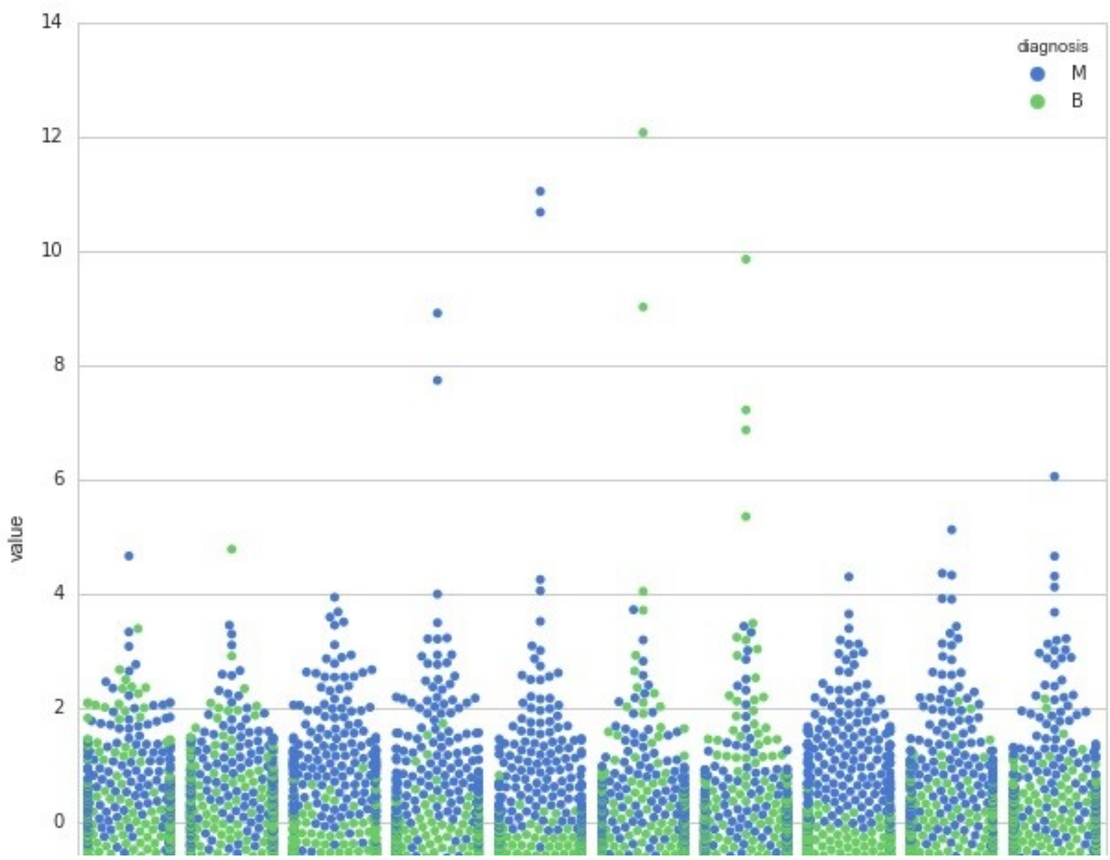


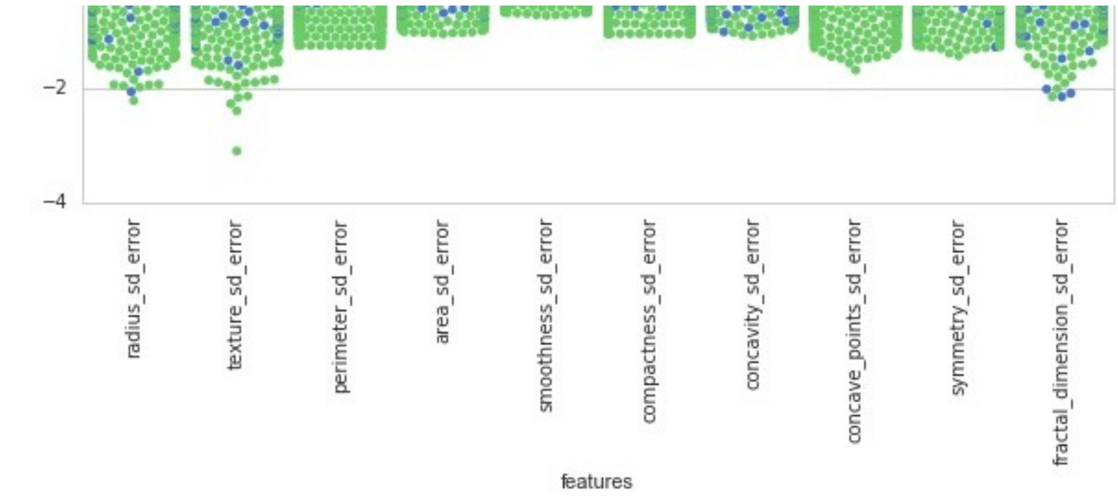
In [119]:

```
data = pd.concat([y,data_nor[sd_cols]],axis=1)
data = pd.melt(data,id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(10,10))
sns.swarmplot(x="features", y="value", hue="diagnosis", data=data)
plt.xticks(rotation=90)
```

Out[119]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), <a list of 10 Text xticklabel objects>)
```

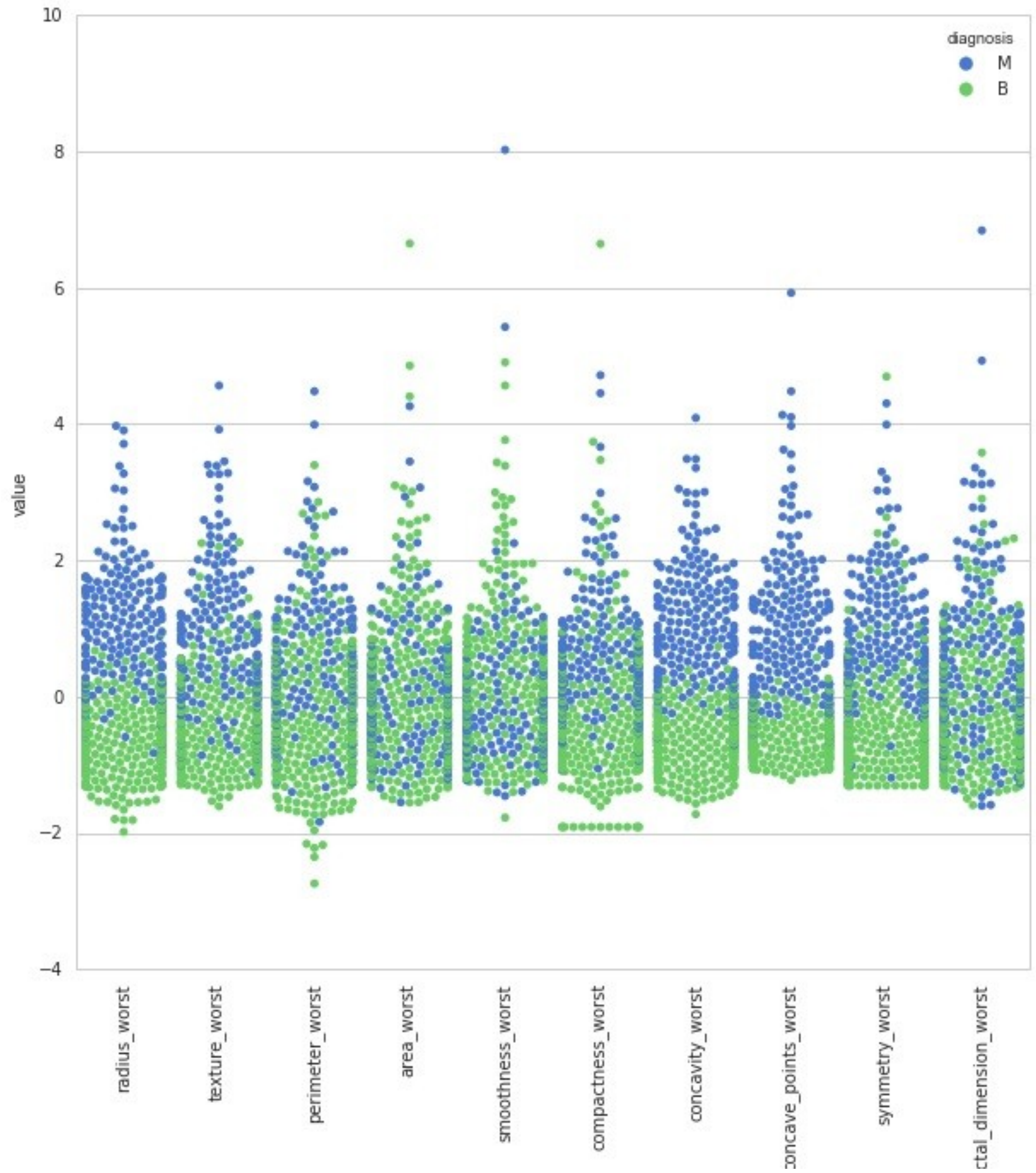




In [122]:

```
data = pd.concat([y,data_nor[worst_cols]],axis=1)
data = pd.melt(data,id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(10,10))
sns.swarmplot(x="features", y="value", hue="diagnosis", data=data)
toc = time.time()
plt.xticks(rotation=90)
print("swarm plot time: ", toc-tic ," s")
```

```
swarm plot time:  132.68085765838623  s
```



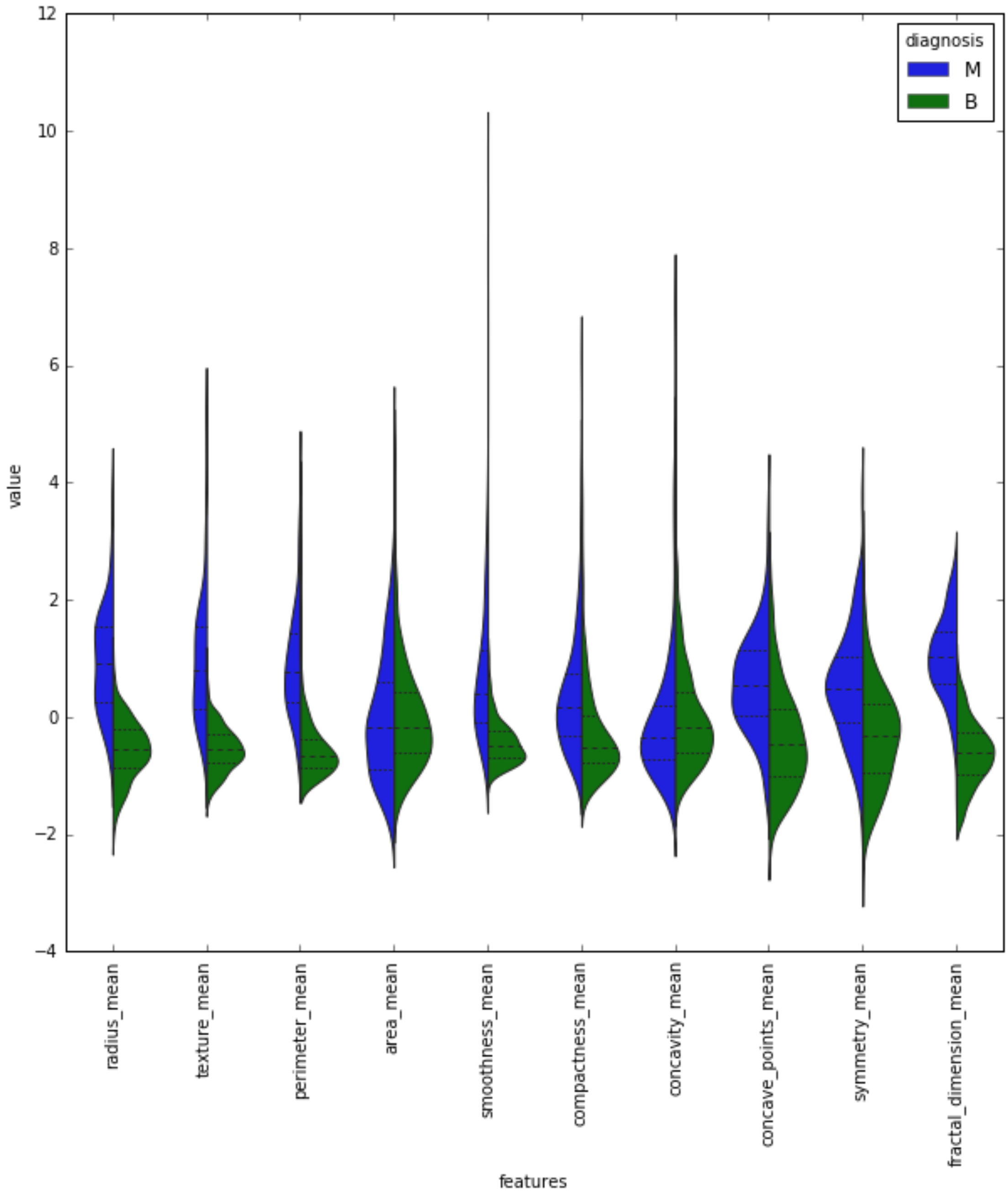
3 Plots Explanation: In the above three graphs, we can see variance more clearly. Let me ask you a question, in these three plots which feature looks like more evident in terms of classification? In my opinion concavity_worst in last swarm plot malignant and benign looks separately, not totaly but mostly. Hovewer, area_mean, area_sd_error and area_worst all three looks like malignant and benign are mixed so it is hard to classify by using this feature.

In [96]:

```
# All mean features
data_diag = y
data = pd.concat([y,data_nor[mean_cols]],axis=1)
data = pd.melt(data,id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(10,10))
sns.violinplot(x="features", y="value", hue="diagnosis", data=data,split=True, inner="qu
art")
plt.xticks(rotation=90)
```

Out[96]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), <a list of 10 Text xticklabel objects>)
```



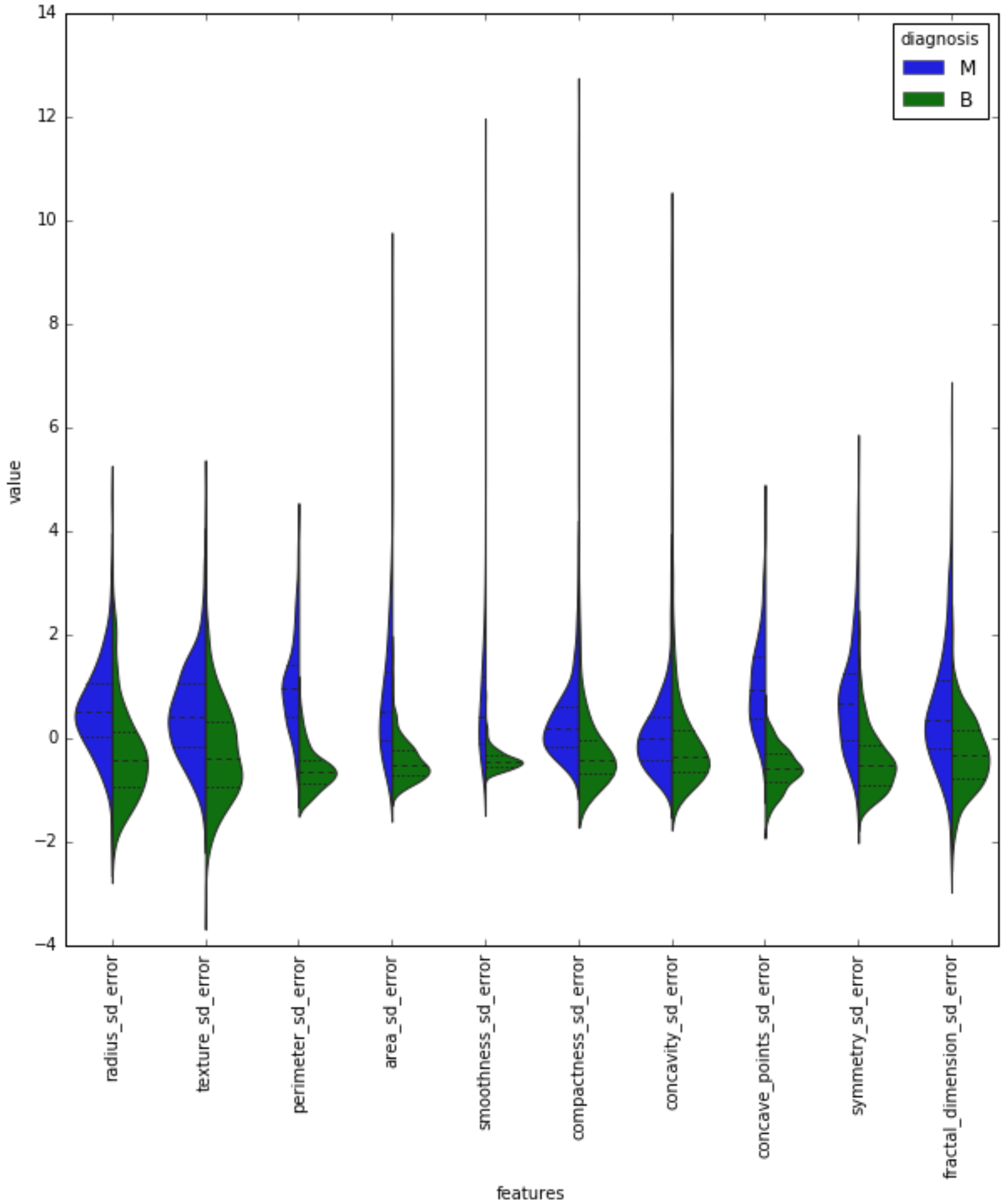
Plot Explanation: In texture_mean, perimeter_mean, concave_points_mean and symetry_mean features, median of the Malignant and Benign looks like separated so it can be good for classification. However, in area_mean feature, median of the Malignant and Benign does not looks like separated so it does not gives good information for classification.

In [97]:

```
data = pd.concat([y,data_nor[sd_cols]],axis=1)
data = pd.melt(data,id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(10,10))
sns.violinplot(x="features", y="value", hue="diagnosis", data=data,split=True, inner="qu
art")
plt.xticks(rotation=90)
```

Out[97]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), <a list of 10 Text xticklabel objects>)
```



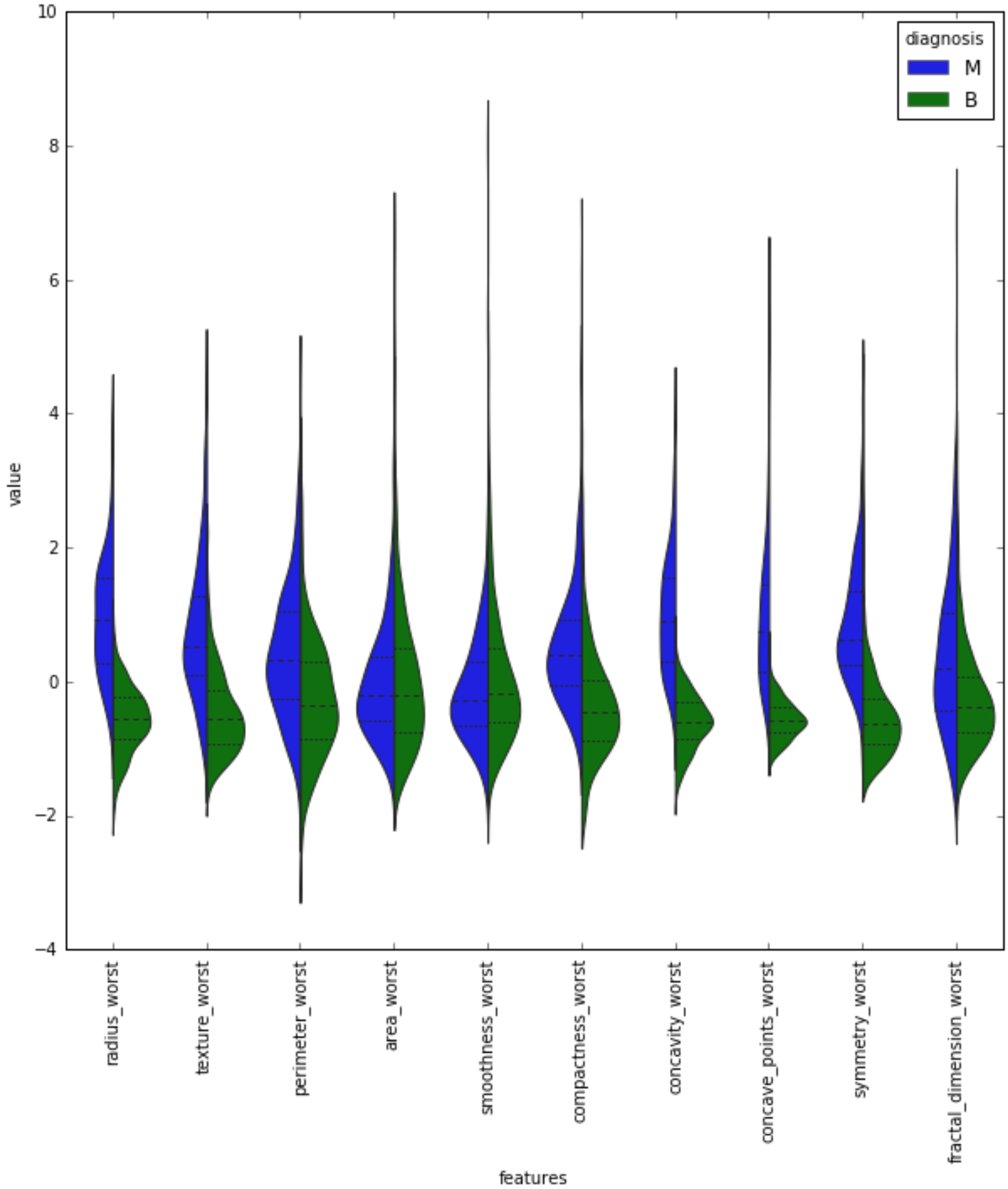
In [98]:

```
data = pd.concat([y,data_nor[worst_cols]],axis=1)
data = pd.melt(data,id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(10,10))
```

```
sns.violinplot(x="features", y="value", hue="diagnosis", data=data,split=True, inner="quart")
plt.xticks(rotation=90)
```

Out[98]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), <a list of 10 Text xticklabel objects>)
```



Plot Explanation: Lets interpret one more thing about plot above, variable of concavity_worst and concave_points_worst looks similar but how can we decide whether they are correlated with each other or not. It can not always be true but, if the features are correlated with each other then we can drop one of them.

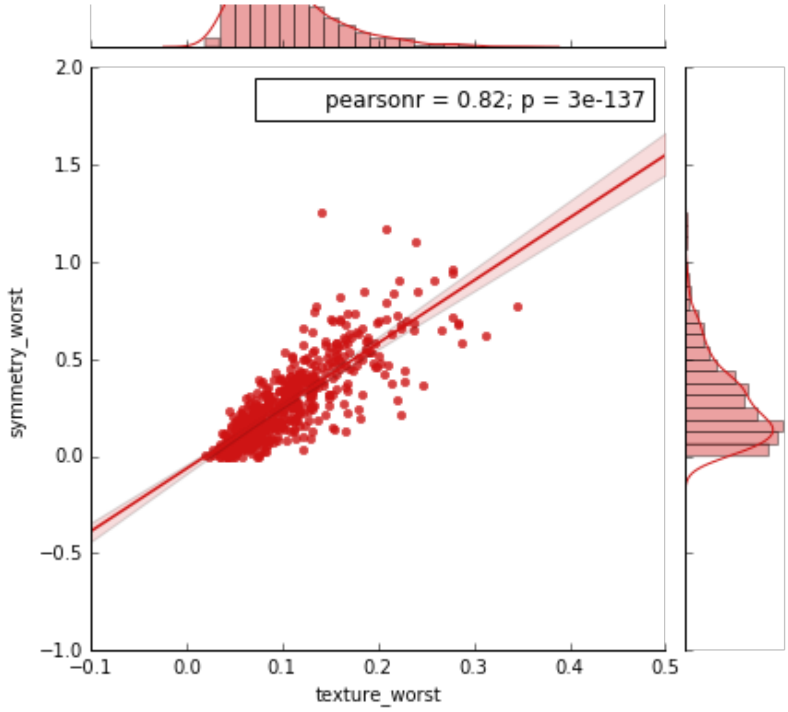
In order to compare two features deeper, lets use joint plot. In the joint plot below, it is really correlated. Pearsonr value is correlation value and 1 is the highest. Therefore, 0.82 is looks enough to say that they are correlated.

In [102]:

```
sns.jointplot(x.loc[:,'texture_worst'], x.loc[:,'symmetry_worst'], kind="regg", color="#ce1414")
```

Out[102]:

```
<seaborn.axisgrid.JointGrid at 0x13f43886550>
```



In last violin plot, concavity_worst, concave_points_worst, and radius_worst also looks similar. Let's plot the pair grid plot to see if they are correlated.

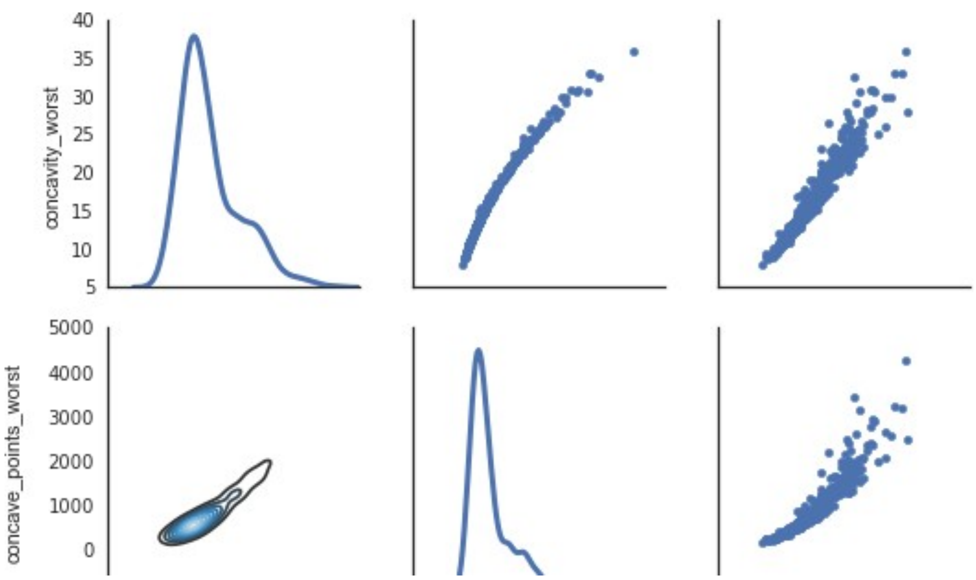
In [107]:

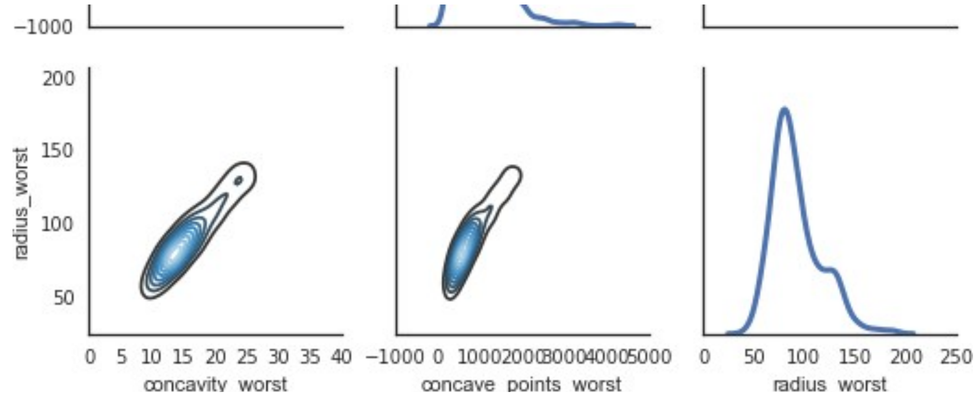
```
sns.set(style="white")
df = x.loc[:,['concavity_worst','concave_points_worst','radius_worst']]
g = sns.PairGrid(df, diag_sharey=False)
g.map_lower(sns.kdeplot, cmap="Blues_d")
g.map_upper(plt.scatter)
g.map_diag(sns.kdeplot, lw=3)
```

```
C:\Users\sana.rasheed\AppData\Local\Continuum\Anaconda3\lib\site-packages\matplotlib\axes
\_axes.py:519: UserWarning: No labelled objects found. Use label='...' kwarg on individua
l plots.
  warnings.warn("No labelled objects found. "
C:\Users\sana.rasheed\AppData\Local\Continuum\Anaconda3\lib\site-packages\matplotlib\axes
\_axes.py:519: UserWarning: No labelled objects found. Use label='...' kwarg on individua
l plots.
  warnings.warn("No labelled objects found. "
C:\Users\sana.rasheed\AppData\Local\Continuum\Anaconda3\lib\site-packages\matplotlib\axes
\_axes.py:519: UserWarning: No labelled objects found. Use label='...' kwarg on individua
l plots.
  warnings.warn("No labelled objects found. "
```

Out[107]:

```
<seaborn.axisgrid.PairGrid at 0x13f461d6048>
```





In above graph, we can observe that all three features are correlated.

Lets observe the correlation between all features and so we use this insights in feature selection for predictive model.

In [194]:

```
#correlation map
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(x.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

Out[194]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x13f4a04b470>
```

To look into correlation matrix easily, next 3 plots are grouped by Mean, SD_Error and Worst

In [193]:

```
#correlation map
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(x[mean_cols].corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

Out[193]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x13f4a0380b8>
```

```
#correlation map
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(x[sd_cols].corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

Out[195]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x13f4930b898>
```

In [196]:

```
#correlation map
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(x[worst_cols].corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

Out[196]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x13f4bd83f60>
```

Feature Selection and Classification

In this Section, we will use 2 feature selection techniques and will test them with 2 Classifiers.

Following Feature Selection Techniques will be used:

1. Feature Selection by using correlation Heatmap
2. Univariate feature selection

In Classifications, we will use following 2 Techniques to train our model and predict:

1. Random Forest
2. Support Vector Machine

1. Feature Selection by using Correlation Heatmap

As it can be seen in map heat figure radius_mean, radius_worst, texture_mean, and concavity_worst are correlated with each others so we will use only concavity_worst. If you ask how I choose concavity_worst as a feature to use, well actually there is no correct answer, I just look at swarm plots and concavity_worst looks like clear for me. We cannot make exact separation among other correlated features without trying. So lets find other correlated features and look accuracy with random forest classifier.

The area_sd_error & smoothness_sd_error, perimeter_sd_error, concave_points_sd_error, concave_points_worst, texture_worst, symmetry_worst are few other highly correlated variables.

In [232]:

```
drop_list1 = [
    'radius_mean', 'radius_worst', 'texture_mean', 'concavity_worst',
    'area_sd_error', 'smoothness_sd_error', 'perimeter_sd_error',
    'concave_points_sd_error', 'concave_points_worst', 'texture_worst', 'symmetry_worst'
]
x_1 = x.drop(drop_list1,axis = 1 )        # do not modify x, we will use it later
x_1.head()
```

Out[232]:

	radius_sd_error	texture_sd_error	perimeter_mean	perimeter_worst	area_mean	area_worst	smoothness_mean	smoothnes
0	10.38	0.11840	0.3001	0.2419	0.07871	0.9053	8.589	0
1	17.77	0.08474	0.0869	0.1812	0.05667	0.7339	3.398	0
2	21.25	0.10960	0.1974	0.2069	0.05999	0.7869	4.585	0
3	20.38	0.14250	0.2414	0.2597	0.09744	1.1560	3.445	0
4	14.34	0.10030	0.1980	0.1809	0.05883	0.7813	5.438	0

Well, we choose our features but did we choose correctly ? Lets use random forest and find accuracy according to chosen features.

1.1 Random Forest Classification

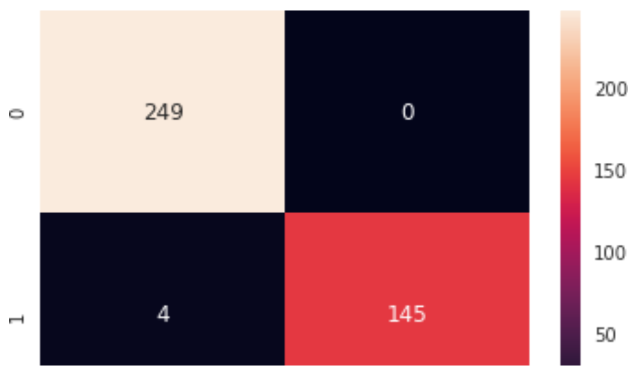
In [257]:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score,confusion_matrix
from sklearn.metrics import accuracy_score

# split data train 70 % and test 30 %
x_train, x_test, y_train, y_test = train_test_split(x_1, y, test_size=0.3, random_state=
42)

#random forest classifier with n_estimators=10 (default)
clf_rf = RandomForestClassifier(random_state=43)
clr_rf = clf_rf.fit(x_train,y_train)

ac = accuracy_score(y_train,clf_rf.predict(x_train))
print('Random Forsest Accuracy is Training Data: ',ac)
cm = confusion_matrix(y_train,clf_rf.predict(x_train))
sns.heatmap(cm,annot=True,fmt="d")
```

```
Random Forsest Accuracy is Training Data:  0.989949748744
```

Out[257]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x13f5039b2b0>
```



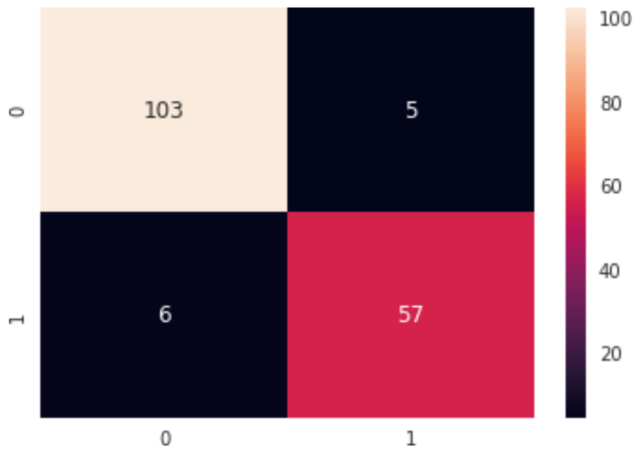
In [256]:

```
ac = accuracy_score(y_test,clf_rf.predict(x_test))
print('Random Forsest Accuracy on Test Data: ',ac)
cm = confusion_matrix(y_test,clf_rf.predict(x_test))
sns.heatmap(cm,annot=True,fmt="d")
```

```
Random Forsest Accuracy on Test Data:  0.93567251462
```

Out[256]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x13f5021ad30>
```



The accuracy on Training Data is 98.9% and 93.5% on Test Data. We can see in confusion matrix that it has made few wrong predictions. Not bad, right!...

About the Overfit

Moreover, Accuracy gap between training and testing dataset is not wide, so **our model isn't overfitting**. If our model does much better on the training set than on the test set, then we're likely overfitting. For example, it would be a big red flag if our model saw 99% accuracy on the training set but only 55% accuracy on the test set.

There are few techniques which helps to prevent Overfitting:

1. Cross-validation: In standard k-fold cross-validation, we partition the data into k subsets, called folds. Then, we iteratively train the algorithm on k-1 folds while using the remaining fold as the test set (called the "holdout fold")..
2. Train with more data:It won't work everytime, but training with more data can help algorithms detect the signal better. Of course, that's not always the case. If we just add more noisy data, this technique won't help.
3. Remove features: Some algorithms have built-in feature selection. For those that don't, we can manually improve their generalizability by removing irrelevant input features.
4. Early stopping: When we are training a learning algorithm iteratively, we can measure how well each iteration of the model performs. Up until a certain number of iterations, new iterations improve the model. After that point, however, the model's ability to generalize can weaken as it begins to overfit the training data. Early stopping refers stopping the training process before the learner passes that point.
5. Regularization: Regularization refers to a broad range of techniques for artificially forcing your model to be simpler. The method will depend on the type of learner you're using. For example, you could prune a decision tree, use dropout on a neural network, or add a penalty parameter to the cost function in regression. Oftentimes, the regularization method is a hyperparameter as well, which means it can be tuned through cross-validation.
6. Ensembling: Ensembles are machine learning methods for combining predictions from multiple separate models

Lets test these features with SVM classifier.

1.2 Support Vector Machine Classification

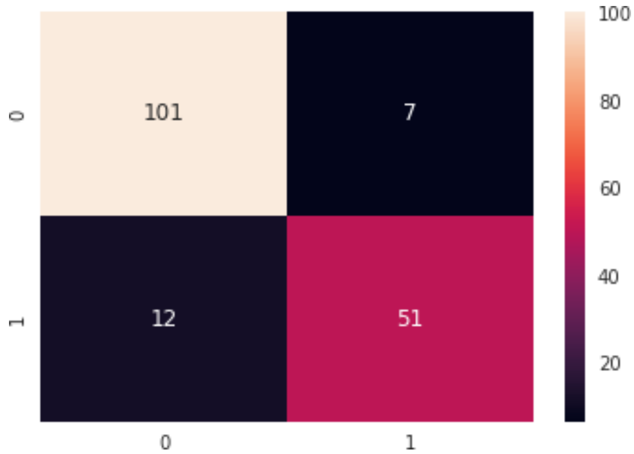
In [258]:

```
from sklearn import svm # for Support Vector Machine
from sklearn import metrics # for the check the error and accuracy of the model

model = svm.SVC()
model.fit(x_train,y_train)
#prediction=model.predict(x_test)
#metrics.accuracy_score(prediction,y_test)

ac = accuracy_score(y_test,model.predict(x_test))
print('Support Vector Machine Accuracy is: ',ac)
cm = confusion_matrix(y_test,model.predict(x_test))
sns.heatmap(cm,annot=True,fmt="d")
```

```
Support Vector Machine Accuracy is:  0.888888888889
```

Out[258]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x13f503c6c50>
```



Accuracy is 88.8%, not good as compare to Random Forest model.

Now, lets test other feature selection methods if we could find better results.

By considering the correlation matrix we can select our desired feature set for the model, but machine learning domain is also equipped with few feature selection algorithms to extract/compute best feature set for our model. Here, we will test one of them:

2. Univariate Feature Selection

In univariate feature selection, we will use SelectKBest that removes all but the k highest scoring features. http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html#sklearn.feature_selection.Select

In this method we need to choose how many features we will use. For example, will k (number of features) be 5 or 10 or 15? The answer is only try. We will not try all combinations but only choose k = 5 and find best 5 features for now.

In [259]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# split data train 70 % and test 30 %
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42
)

# find best scored 5 features
```

```
select_feature = SelectKBest(chi2, k=5).fit(x_train, y_train)
print('Score list:', select_feature.scores_)
print('Feature list:', x_train.columns)
```

```
Score list: [  1.77946492e+02   6.06916433e+01   1.34061092e+03   3.66899557e+04
   1.00015175e-01   3.41839493e+00   1.30547650e+01   7.09766457e+00
   1.95982847e-01   3.42575072e-04   2.45882967e+01   4.07131026e-02
   1.72696840e+02   6.12741067e+03   1.32470372e-03   3.74071521e-01
   6.92896719e-01   2.01587194e-01   1.39557806e-03   2.65927071e-03
   3.25782599e+02   1.16958562e+02   2.40512835e+03   7.50217341e+04
   2.63226314e-01   1.19077581e+01   2.58858117e+01   8.90751003e+00
   1.00635138e+00   1.23087347e-01]
Feature list: Index(['radius_mean', 'radius_sd_error', 'radius_worst', 'texture_mean',
       'texture_sd_error', 'texture_worst', 'perimeter_mean',
       'perimeter_sd_error', 'perimeter_worst', 'area_mean', 'area_sd_error',
       'area_worst', 'smoothness_mean', 'smoothness_sd_error',
       'smoothness_worst', 'compactness_mean', 'compactness_sd_error',
       'compactness_worst', 'concavity_mean', 'concavity_sd_error',
       'concavity_worst', 'concave_points_mean', 'concave_points_sd_error',
       'concave_points_worst', 'symmetry_mean', 'symmetry_sd_error',
       'symmetry_worst', 'fractal_dimension_mean',
       'fractal_dimension_sd_error', 'fractal_dimension_worst'],
      dtype='object')
```

The Top scored features are **fractal_dimension_mean, concave_points_worst, perimeter_sd_error, compactness_sd_error, and smoothness_sd_error**.

Let's build model on top of these 5 features.

2.1 Random Forest Classification

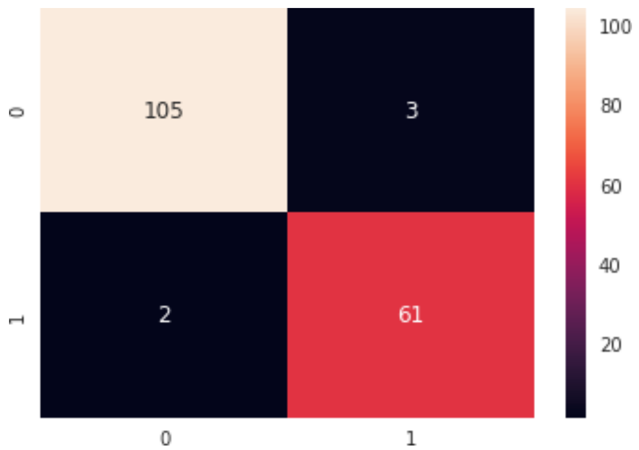
In [260]:

```
x_train_2 = select_feature.transform(x_train)
x_test_2 = select_feature.transform(x_test)
#random forest classifier with n_estimators=10 (default)
clf_rf_2 = RandomForestClassifier(n_estimators=100)
clr_rf_2 = clf_rf_2.fit(x_train_2,y_train)
ac_2 = accuracy_score(y_test,clf_rf_2.predict(x_test_2))
print('Random Forsest Accuracy is: ',ac_2)
cm_2 = confusion_matrix(y_test,clf_rf_2.predict(x_test_2))
sns.heatmap(cm_2,annot=True,fmt="d")
```

```
Random Forsest Accuracy is:  0.970760233918
```

Out[260]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x13f4fc3fb70>
```



These top 5 features are play significant role in the data set. Accuracy is 97% and as it can be seen in confusion matrix, we have few wrong predictions. If we set n_estimators=10 (default), then accuracy is 95% n_estimators=50, then accuracy is 96%

So, we also need to spend a good time just for Model Tuning .

2.2 Suport Vector Machine

In [261]:

```
from sklearn import svm # for Support Vector Machine
from sklearn import metrics # for the check the error and accuracy of the model

model = svm.SVC()
model.fit(x_train,y_train)

ac = accuracy_score(y_test,model.predict(x_test))
print('Support Vector Machine Accuracy is: ',ac)
cm = confusion_matrix(y_test,model.predict(x_test))
sns.heatmap(cm,annot=True,fmt="d")
```

```
Support Vector Machine Accuracy is:  0.631578947368
```

Out[261]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x13f51588ba8>
```

Let's do the little model tuning to gain some good accuracey.

In [262]:

```
model = svm.SVC(kernel = 'linear',C=.1, gamma=10, probability = True)
model.fit(x_train,y_train)

ac = accuracy_score(y_test,model.predict(x_test))
print('Support Vector Machine Accuracy is: ',ac)
cm = confusion_matrix(y_test,model.predict(x_test))
sns.heatmap(cm,annot=True,fmt="d")
```

```
Support Vector Machine Accuracy is:  0.964912280702
```