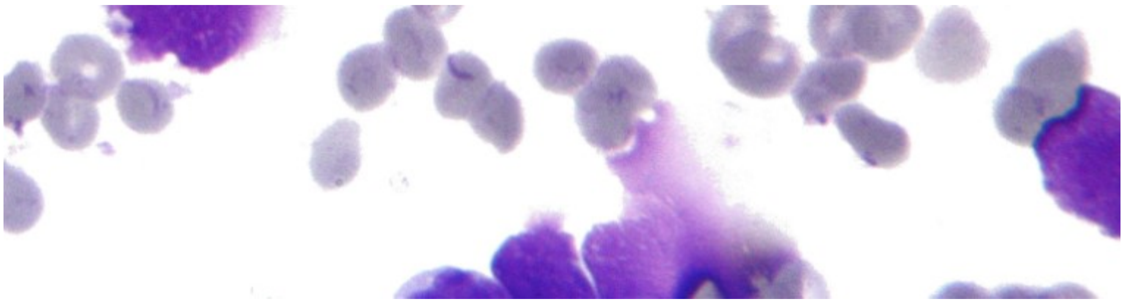
Data Visalization with Seaborn (Part 2): Fe Selection and Classification



This project is second in a series focused on data visualization with Seaborn. Yo Project: Exploratory Data Analysis with Seaborn (https://www.coursera.org/lear analysis-seaborn/) on Coursera.

About the Dataset:

The Breast Cancer Diagnostic data (https://archive.ics.uci.edu/ml/datasets /Breast+Cancer+Wisconsin+%28Diagnostic%29) is available on the UCI Machi Repository. This database is also available through the UW CS ftp server (http:/ /math-prog/cpo-dataset/machine-learn/cancer/WDBC/).

Features are computed from a digitized image of a fine needle aspirate (FNA) o They describe characteristics of the cell nuclei present in the image. n the 3-di that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Progra Discrimination of Two Linearly Inseparable Sets", Optimization Methods and So 23-34].

Attribute Information:

- ID number
- Diagnosis (M = malignant, B = benign) 3-32)

Ten real-valued features are computed for each cell nucleus:

1. radius (mean of distances from center to points on the perimeter)
2. texture (standard deviation of gray-scale values)
3. perimeter
4. area
5. smoothness (local variation in radius lengths)
6. compactness (perimeter^2 / area - 1.0)
7. concavity (severity of concave portions of the contour)
8. concave points (number of concave portions of the contour)
9. symmetry
10. fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest valu
features were computed for each image, resulting in 30 features. For instance,
Radius, field 13 is Radius SE, field 23 is Worst Radius.

All feature values are recoded with four significant digits.

Missing attribute values: none

Class distribution: 357 benign, 212 malignant

Task 1: Loading Libraries and Data

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('seaborn')
import time
```

In [2]:
```
data = pd.read_csv('data/data.csv')
```

Exploratory Data Analysis

Separate Target from Features

In [3]:
```
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smootl
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030

5 rows × 33 columns

In [4]:
```
col = data.columns
print(col)
```

```
 Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
        'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
        'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
        'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
        'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
        'fractal_dimension_se', 'radius_worst', 'texture_worst',
        'perimeter_worst', 'area_worst', 'smoothness_worst',
        'compactness_worst', 'concavity_worst', 'concave points_worst',
        'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
       dtype='object')
```

In [5]:
```
y = data.diagnosis
drop_cols = ['Unnamed: 32','id','diagnosis']
x = data.drop(drop_cols,axis = 1 )
x.head()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compac
0	17.99	10.38	122.80	1001.0	0.11840	0.27760
1	20.57	17.77	132.90	1326.0	0.08474	0.07864
2	19.69	21.25	130.00	1203.0	0.10960	0.15990
3	11.42	20.38	77.58	386.1	0.14250	0.28390
4	20.29	14.34	135.10	1297.0	0.10030	0.13280

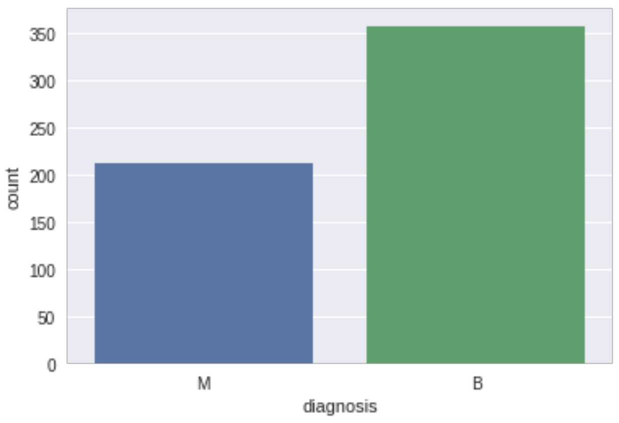
5 rows × 30 columns

Plot Diagnosis Distributions

In [6]:

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

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



In [7]:

```
x.describe()
```

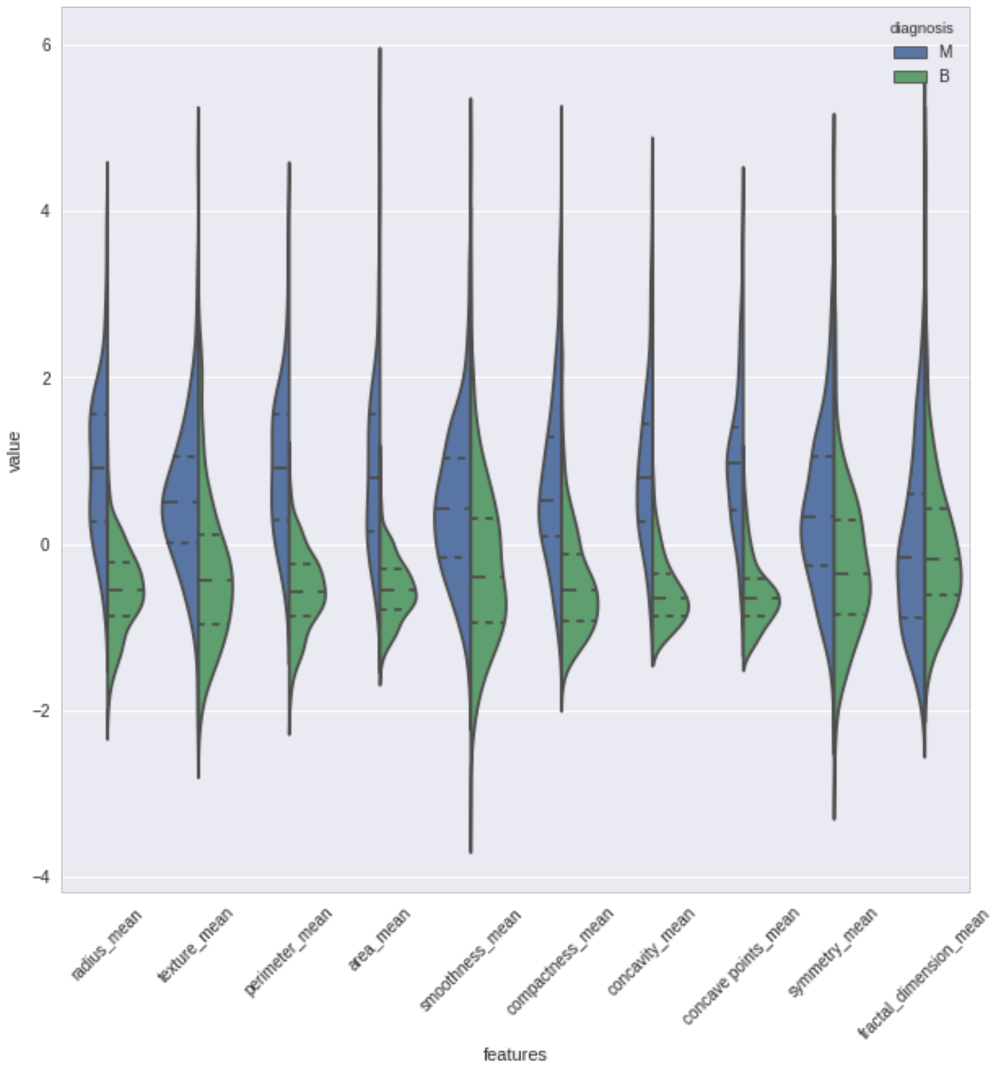
	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	con
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.10
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.05
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.01
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.06
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.09
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.13
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.34

8 rows × 30 columns

Data Visualization

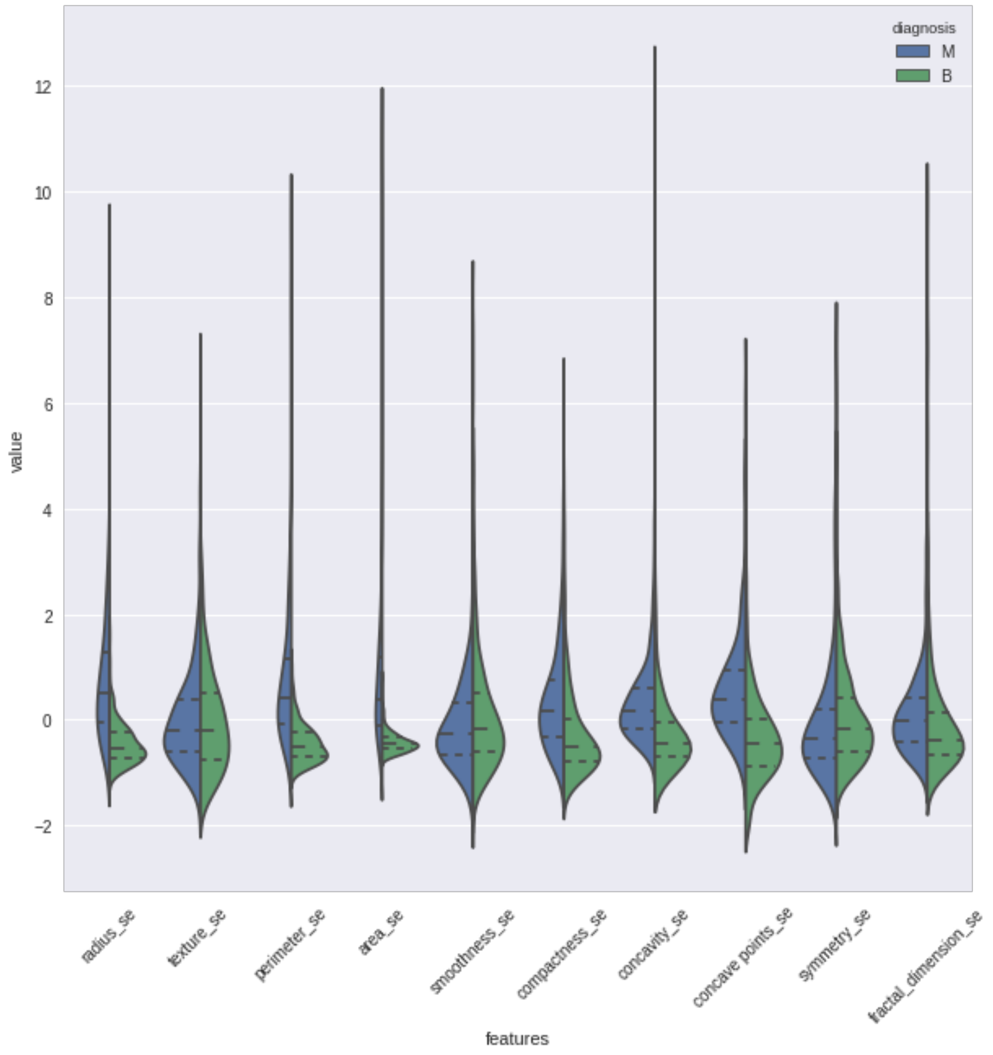
Visualizing Standardized Data with Seaborn

In [8]:
```
data_dia = y
data = x
data_n_2 = (data - data.mean()) / (data.std())
data = pd.concat([y,data_n_2.iloc[:,0:10]],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, :
plt.xticks(rotation=45);
```



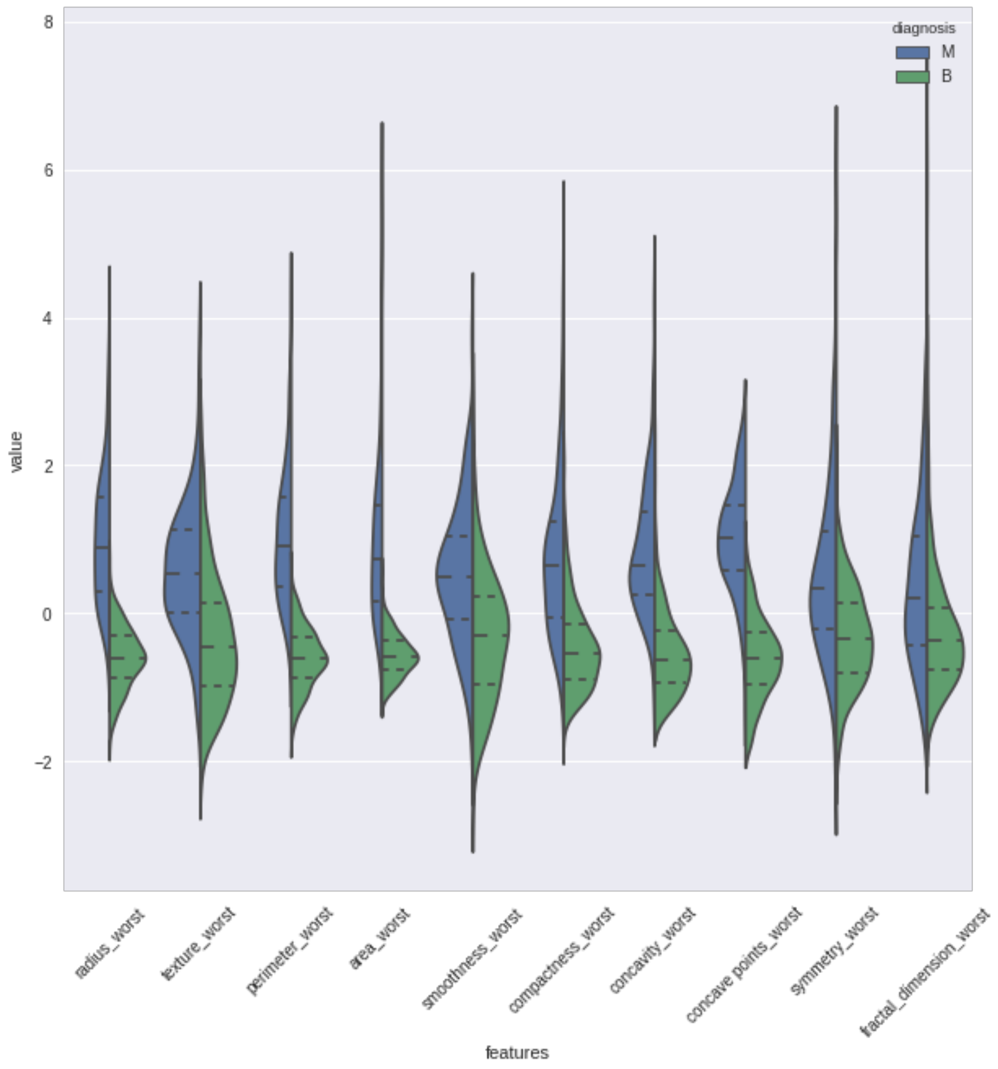
Violin Plots and Box Plots

```
In [9]: data = pd.concat([y,data_n_2.iloc[:,10:20]],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, :
        plt.xticks(rotation=45);
```

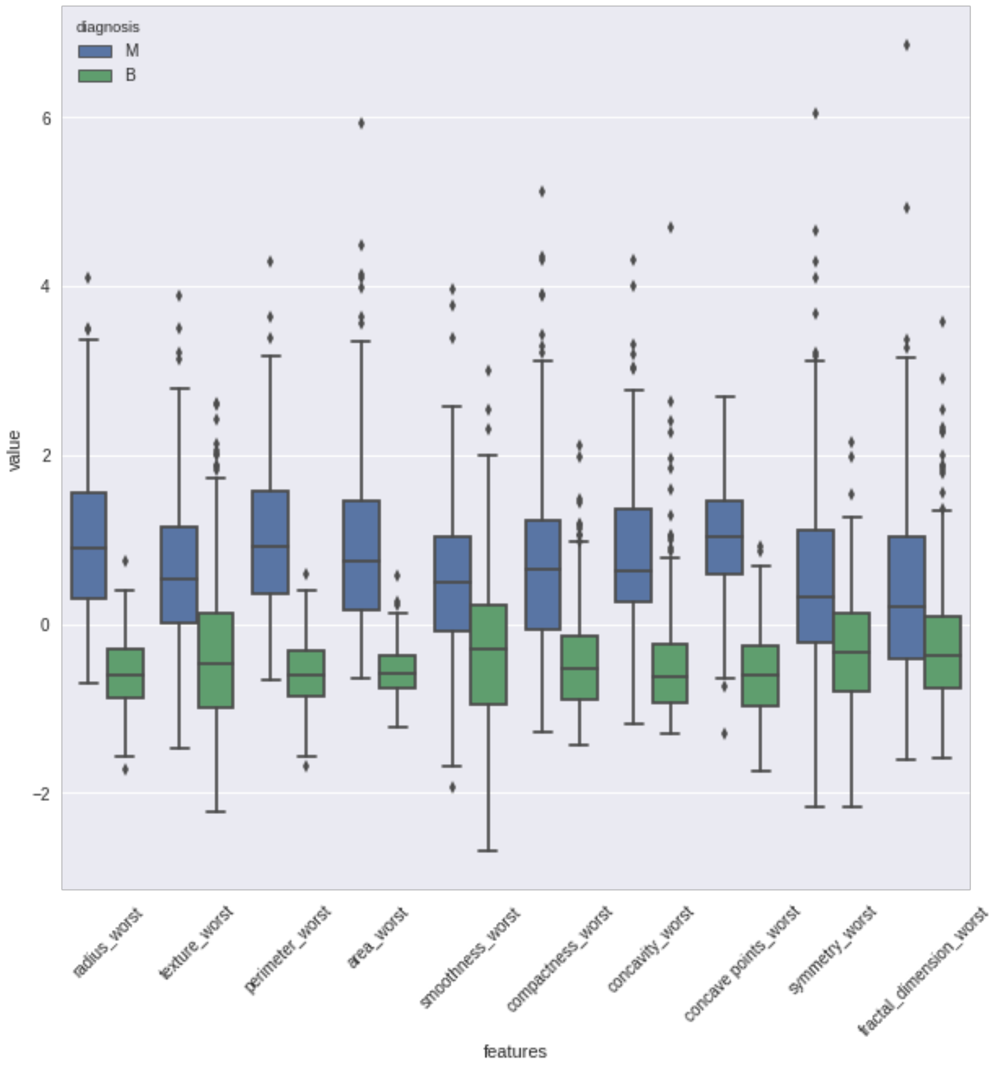


In [10]:

```
data = pd.concat([y,data_n_2.iloc[:,20:31]],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, :
plt.xticks(rotation=45);
```

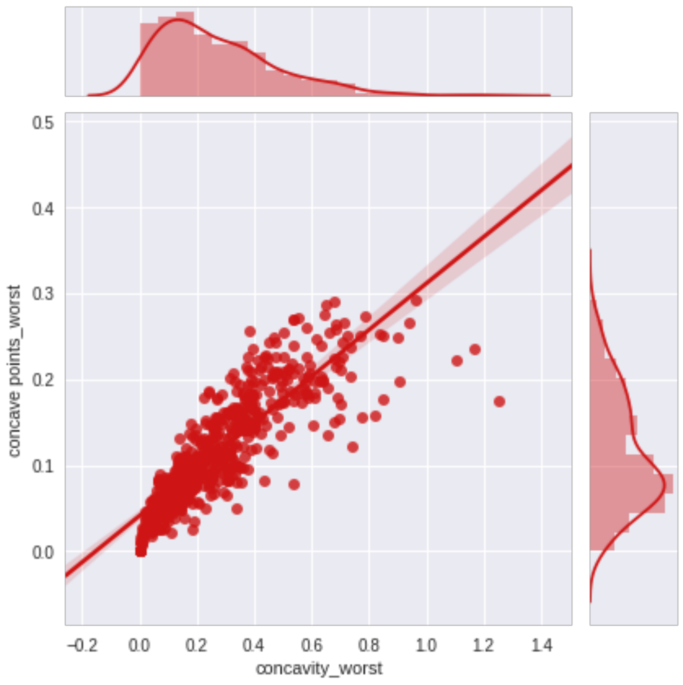


In [11]:
```
plt.figure(figsize=(10,10))
sns.boxplot(x="features", y="value", hue="diagnosis", data=data)
plt.xticks(rotation=45);
```



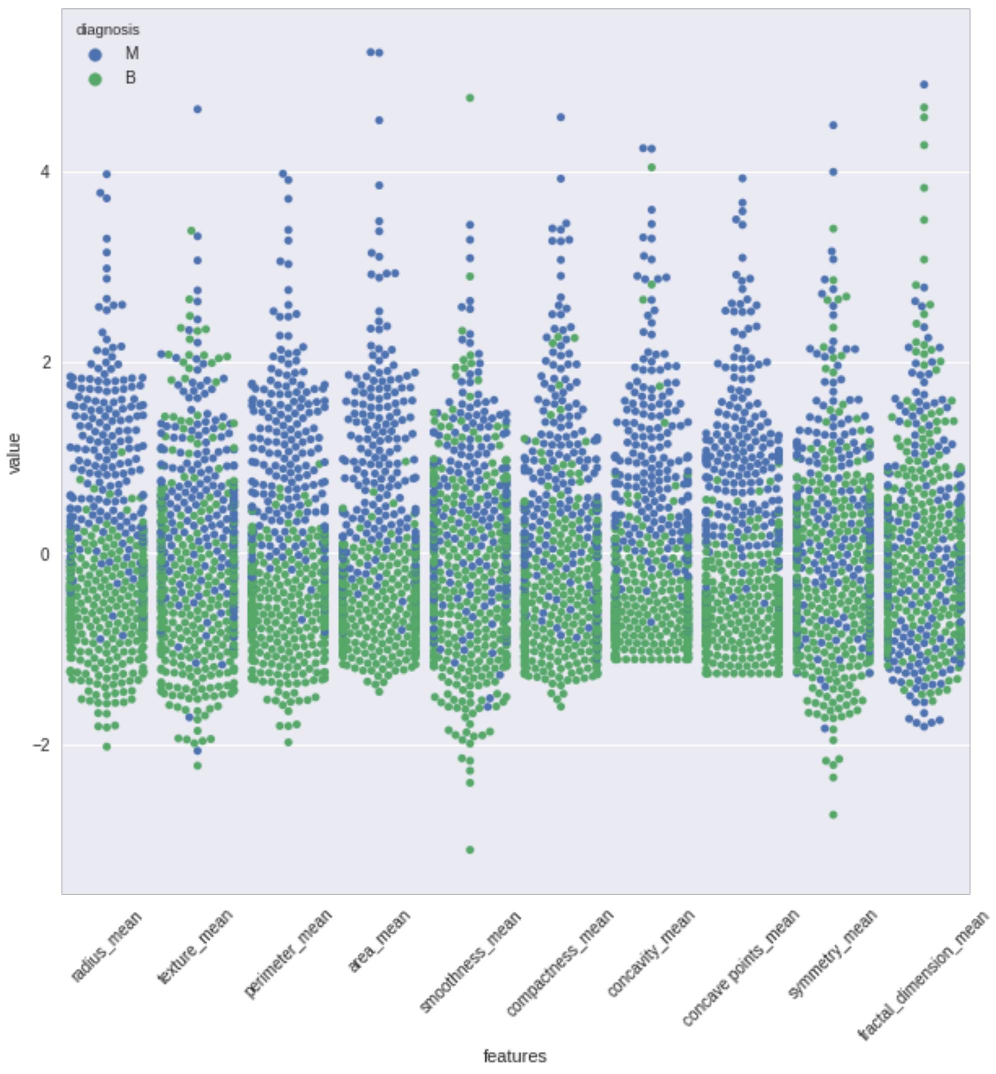
Using Joint Plots for Feature Comparison

In [12]:
```
sns.jointplot(x.loc[:,'concavity_worst'],
              x.loc[:,'concave points_worst'],
              kind="regg",
              color="#ce1414");
```

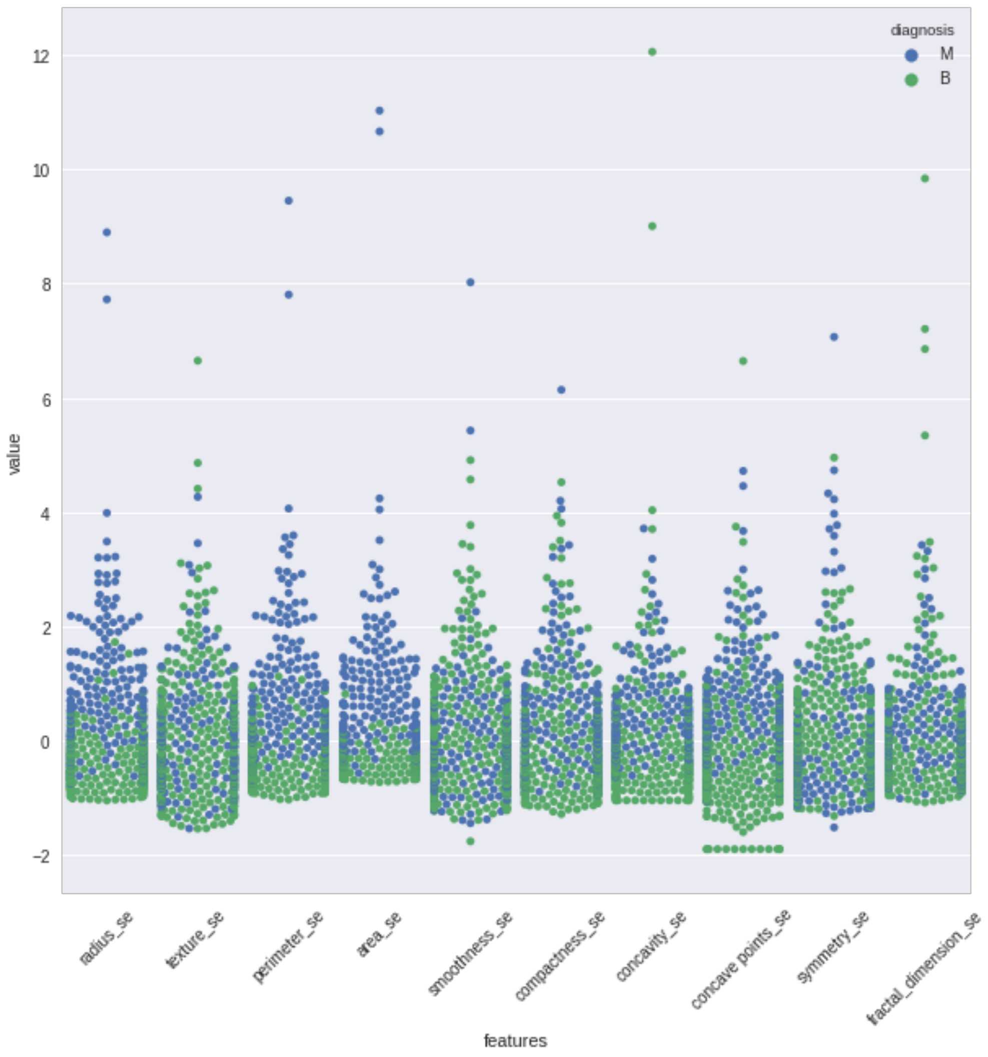


Observing the Distribution of Values and their Variance
Plots

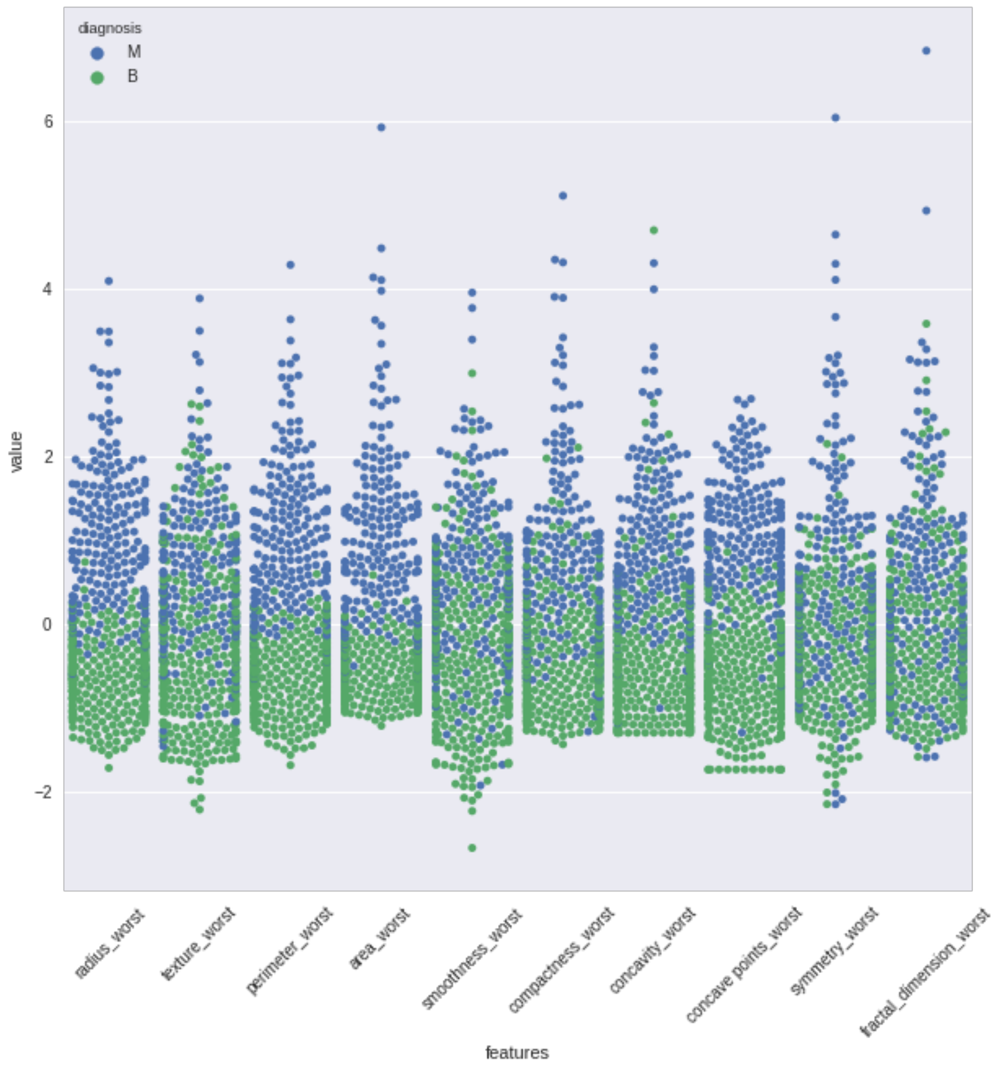
In [13]:
```
#sns.set(style="whitegrid", palette="muted")
data_dia = y
data = x
data_n_2 = (data - data.mean()) / (data.std())
data = pd.concat([y,data_n_2.iloc[:,0:10]],axis=1)
data = pd.melt(data,id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(10,10))
sns.swarmplot(x="features", y="value", hue="diagnosis", data=data)
plt.xticks(rotation=45);
```



In [14]:
```
data = pd.concat([y,data_n_2.iloc[:,10:20]],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=45);
```

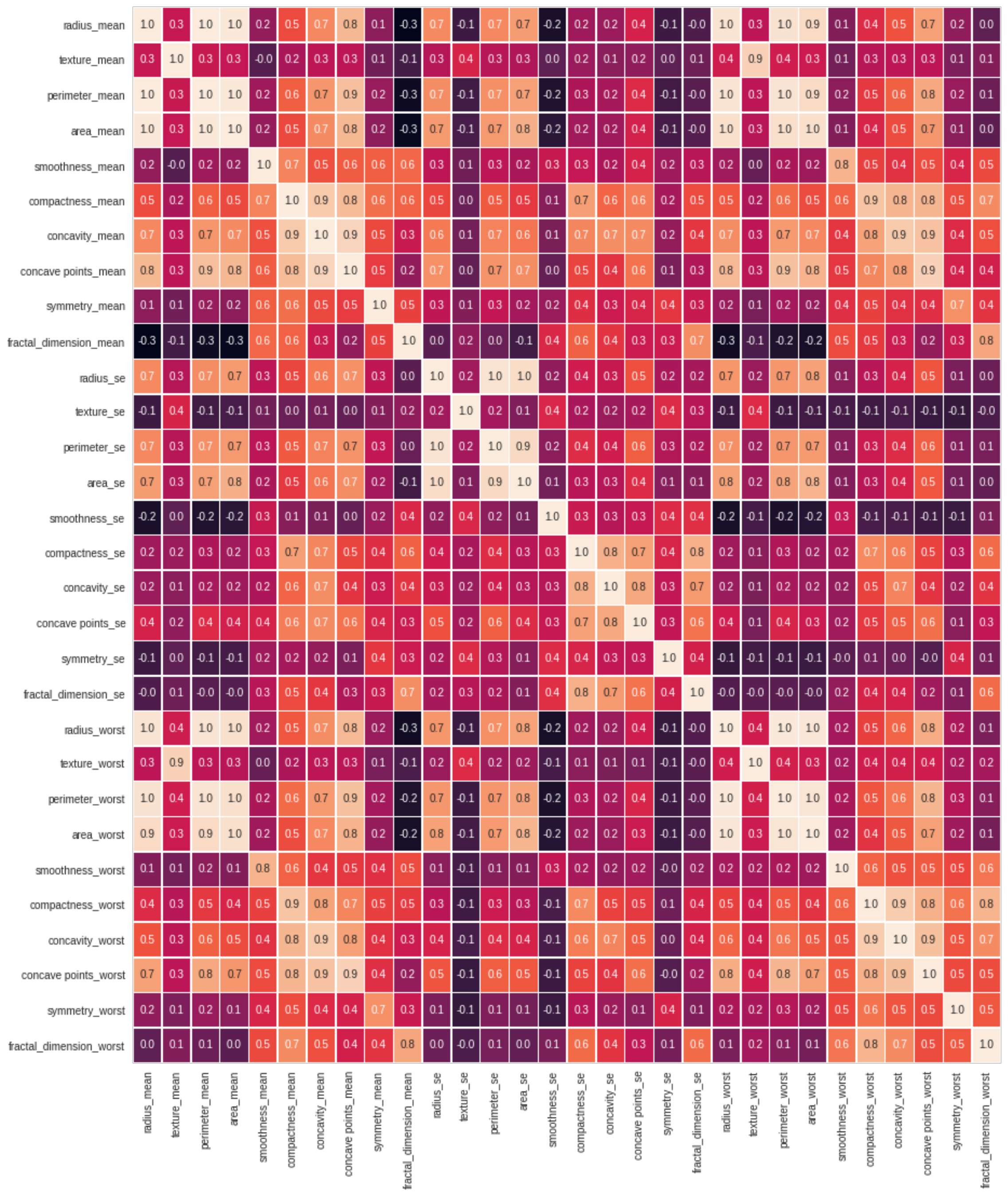


In [15]:
```
data = pd.concat([y,data_n_2.iloc[:,20:31]],axis=1)
data = pd.melt(data,id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(10,10))
sns.swarmplot(x="features", y="value", hue="diagnosis", data=data)
plt.xticks(rotation=45);
```



Observing all Pair-wise Correlations

In [16]:
```python
#correlation map
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(x.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax);
```



Task 2: Dropping Correlated Columns from Feature Matri

Note: If you are starting the notebook from this task, you can run cells from all in the kernel by going to the top menu and Kernel > Restart and Run All\n",

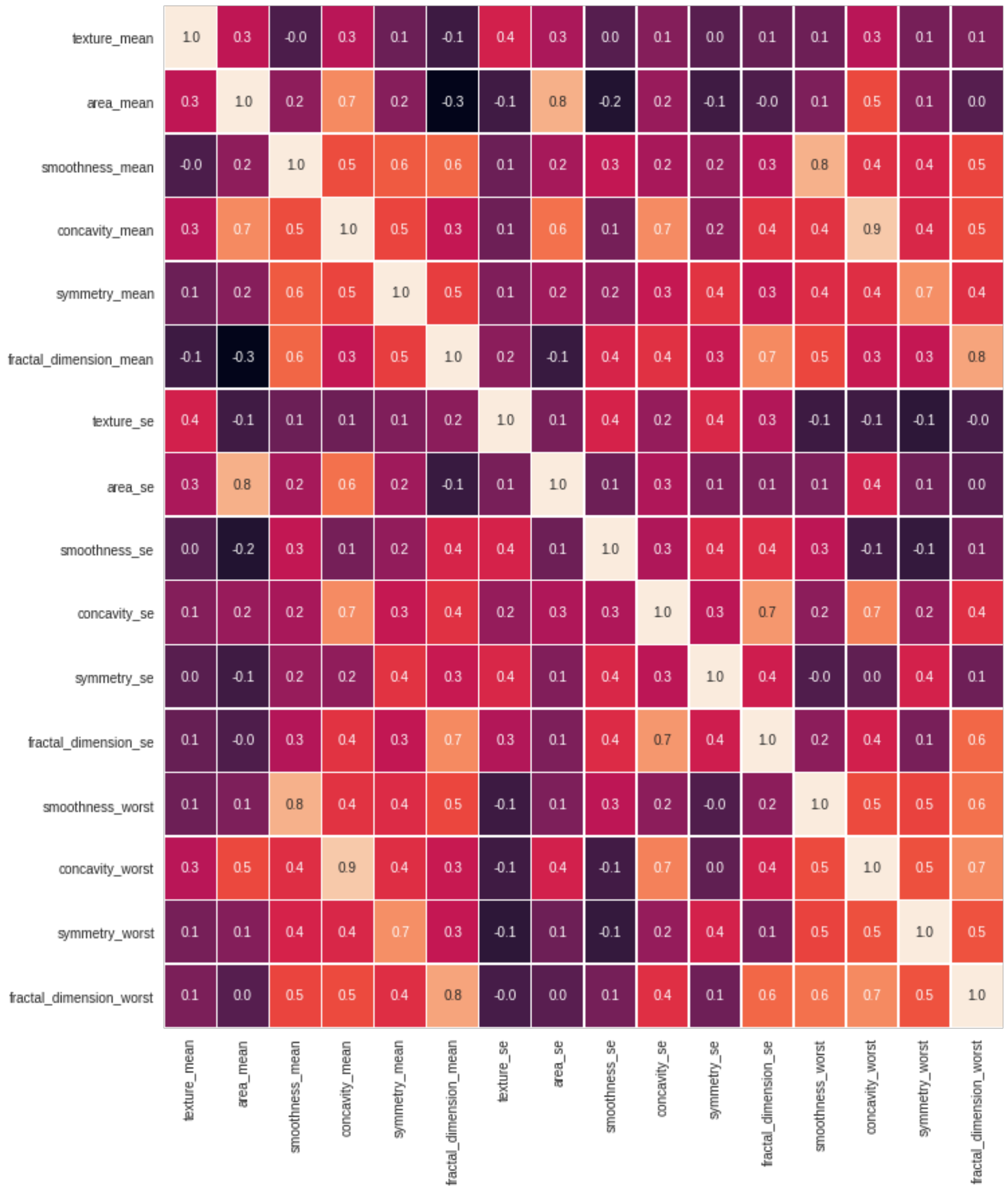
In [17]:

```
drop_cols=['perimeter_mean','radius_mean','compactness_mean',
              'concave points_mean','radius_se','perimeter_se',
              'radius_worst','perimeter_worst','compactness_worst',
              'concave points_worst','compactness_se','concave points_se',
              'texture_worst','area_worst']
df=x.drop(drop_cols, axis=1)
df.head()
```

	texture_mean	area_mean	smoothness_mean	concavity_mean	symmetry_mean	frac
0	10.38	1001.0	0.11840	0.3001	0.2419	0.078
1	17.77	1326.0	0.08474	0.0869	0.1812	0.056
2	21.25	1203.0	0.10960	0.1974	0.2069	0.059
3	20.38	386.1	0.14250	0.2414	0.2597	0.097
4	14.34	1297.0	0.10030	0.1980	0.1809	0.058

```
In [18]:  f,ax= plt.subplots(figsize=(14,14))
          sns.heatmap(df.corr(), annot=True, linewidth=.5, fmt='.1f', ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6cf3f7d748>
```



Task 3: Classification using XGBoost (minimal feature se

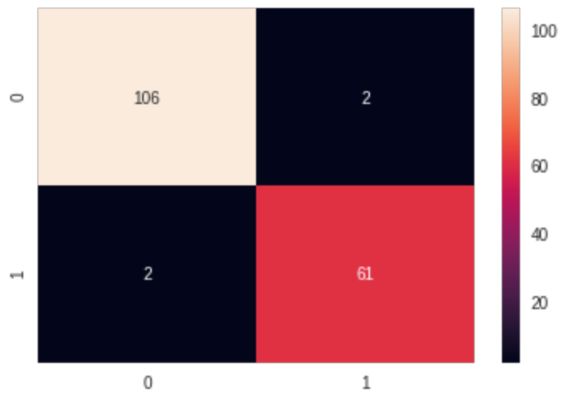
```
In [19]:  from sklearn.model_selection import train_test_split
          import xgboost as xgb
          from sklearn.metrics import f1_score,confusion_matrix
          from sklearn.metrics import accuracy_score
```

In [21]:
```
X_train,X_test,y_train,y_test=train_test_split(df, y, test_size=0.3, random_sta

clf_1=xgb.XGBClassifier(random_state=42)
clf_1=clf_1.fit(X_train,y_train)
```

In [22]:
```
print('Accuracy is: ', accuracy_score(y_test,clf_1.predict(X_test)))
cm= confusion_matrix(y_test, clf_1.predict(X_test))
sns.heatmap(cm, annot=True, fmt='d');
```

```
Accuracy is:  0.9766081871345029
```



Task 4: Univariate Feature Selection and XGBoost

In [23]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [24]:
```
select_feature=SelectKBest(chi2, k=10).fit(X_train,y_train)

print('Score List: ',select_feature.scores_)
print('Feature List: ', X_train.columns)
```

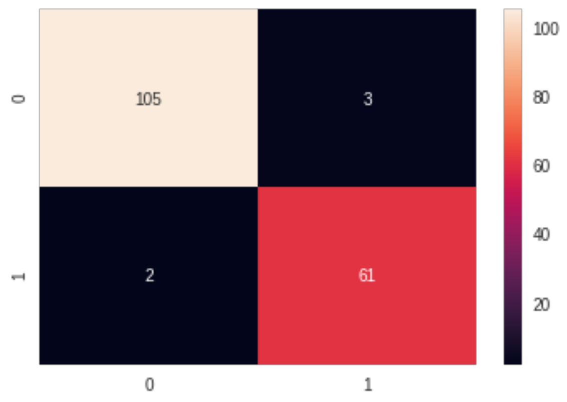
```
Score List:  [6.06916433e+01 3.66899557e+04 1.00015175e-01 1.30547650e+01
 1.95982847e-01 3.42575072e-04 4.07131026e-02 6.12741067e+03
 1.32470372e-03 6.92896719e-01 1.39557806e-03 2.65927071e-03
 2.63226314e-01 2.58858117e+01 1.00635138e+00 1.23087347e-01]
Feature List:  Index(['texture_mean', 'area_mean', 'smoothness_mean', 'concavity_mean',
       'symmetry_mean', 'fractal_dimension_mean', 'texture_se', 'area_se',
       'smoothness_se', 'concavity_se', 'symmetry_se', 'fractal_dimension_se',
       'smoothness_worst', 'concavity_worst', 'symmetry_worst',
       'fractal_dimension_worst'],
      dtype='object')
```

In [25]:
```
X_train_2= select_feature.transform(X_train)
X_test_2=select_feature.transform(X_test)

clf_2= xgb.XGBClassifier().fit(X_train_2,y_train)

print('Accuracy is : ',accuracy_score(y_test, clf_2.predict(X_test_2)))
cm_2= confusion_matrix(y_test, clf_2.predict(X_test_2))
sns.heatmap(cm_2, annot=True, fmt='d');
```

```
Accuracy is :  0.9707602339181286
```



Task 5: Recursive Feature Elimination with Cross-Validati

In [26]:
```
from sklearn.feature_selection import RFECV

clf_3=xgb.XGBClassifier()
rfecv=RFECV(estimator=clf_3, step=1, cv=5, scoring='accuracy', n_jobs=-1).fit(X_

print('Optimal number of features: ',rfecv.n_features_)
print('Best Features: ',X_train.columns[rfecv.support_])
```

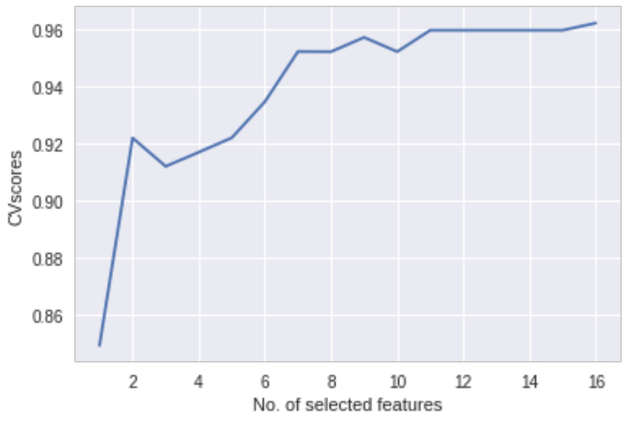
```
Optimal number of features:  16
Best Features:  Index(['texture_mean', 'area_mean', 'smoothness_mean', 'concavity_mean',
       'symmetry_mean', 'fractal_dimension_mean', 'texture_se', 'area_se',
       'smoothness_se', 'concavity_se', 'symmetry_se', 'fractal_dimension_se',
       'smoothness_worst', 'concavity_worst', 'symmetry_worst',
       'fractal_dimension_worst'],
      dtype='object')
```

In [27]:
```
print('Accuracy is: ',accuracy_score(y_test, rfecv.predict(X_test)))
```

```
Accuracy is:  0.9766081871345029
```

In [28]:
```
num_features= [i for i in range(1,len(rfecv.grid_scores_) + 1)]
cv_scores= rfecv.grid_scores_
ax=sns.lineplot(x=num_features, y=cv_scores)
ax.set(xlabel='No. of selected features', ylabel='CVscores')
```

```
[Text(0, 0.5, 'CVscores'), Text(0.5, 0, 'No. of selected features')]
```



Task 6: Feature Extraction using Principal Component Ar

In [30]:
```
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.3, random_state=

X_train_norm=(X_train -X_train.mean()) / (X_train.max() - X_train.min())
X_test_norm=(X_test- X_test.mean()) / (X_test.max() - X_test.min())
```

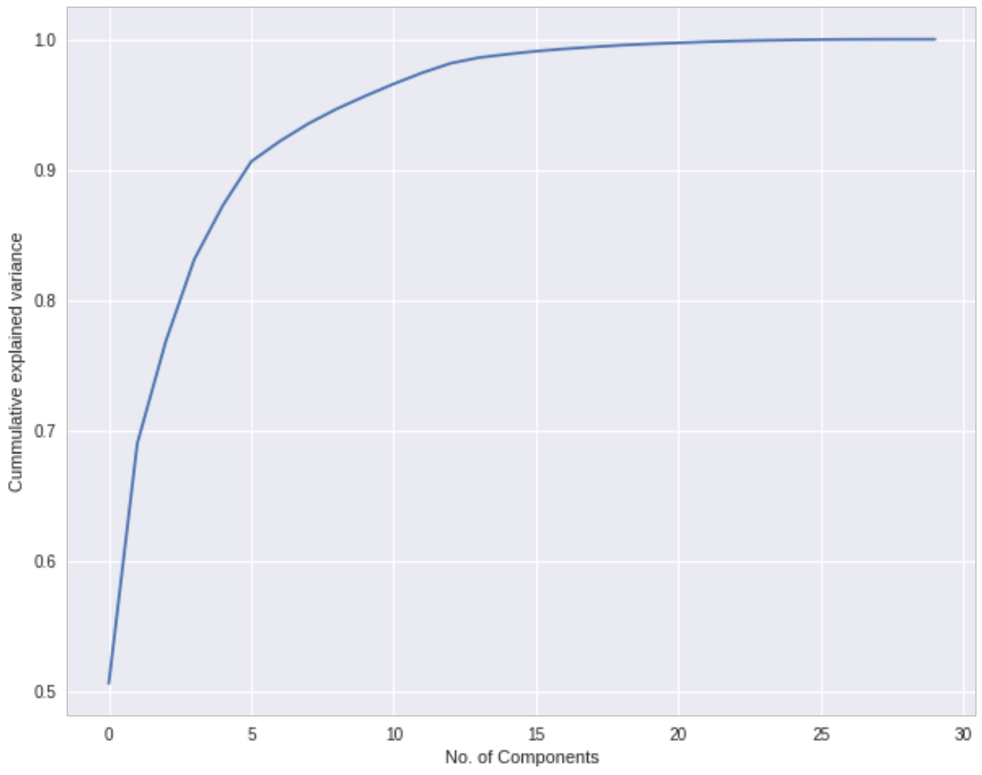
In [31]:

```
from sklearn.decomposition import PCA

pca=PCA()
pca.fit(X_train_norm)

plt.figure(1, figsize=(10,8))
sns.lineplot(data=np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('No. of Components')#linear transformation of the features(to low di
plt.ylabel('Cummulative explained variance')
```

```
Text(0, 0.5, 'Cummulative explained variance')
```



In [ ]: