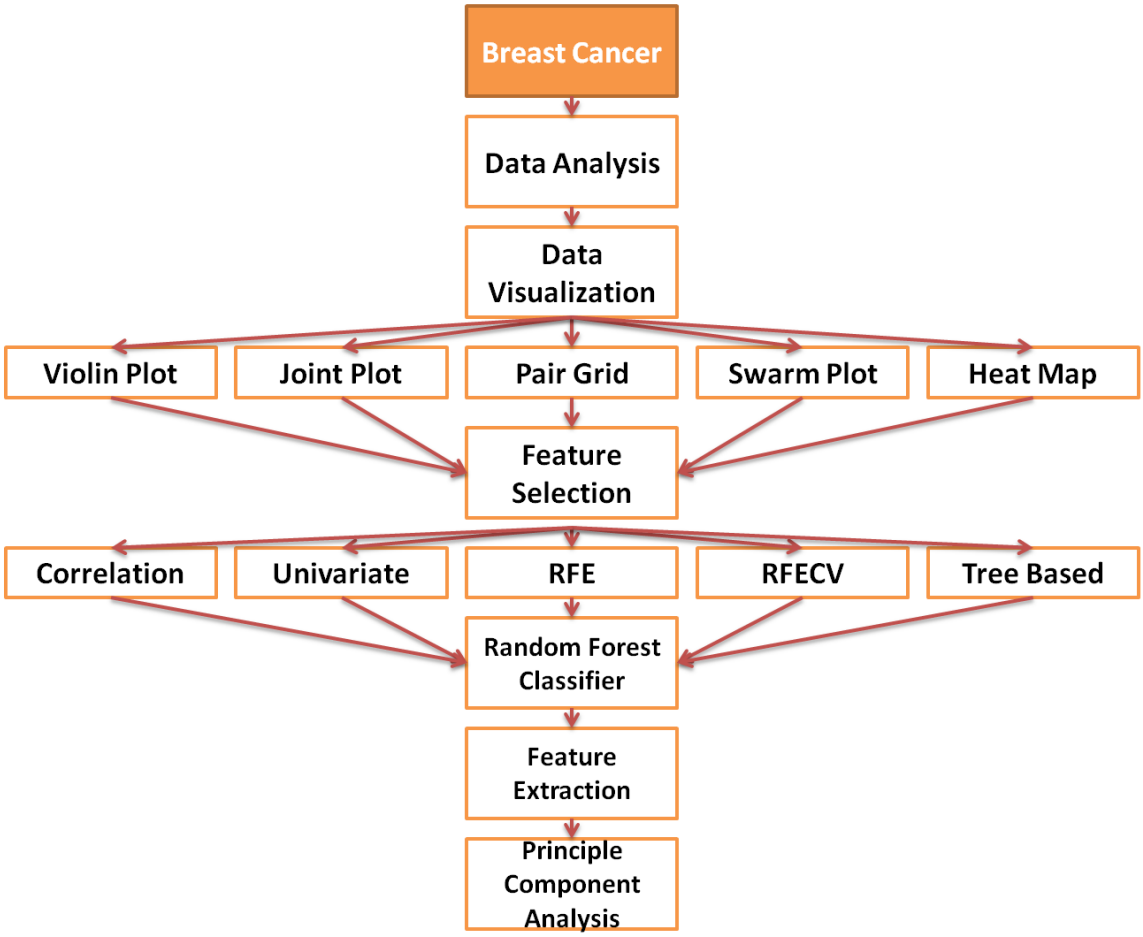
FEATURE SELECTION & DATA VISUALIZATION

What will you learn from this project?



Introduction

In this data analysis report, I usually focus on feature visualization and selection as a different from other kernels. Feature selection with correlation, univariate feature selection, recursive feature elimination, recursive feature elimination with cross validation and tree based feature selection methods are used with random forest classification. Apart from these, principle component analysis are used to observe number of components.

Enjoy your data analysis!!!

Data Analysis Content

Python Libraries

- In this section, we import used libraries during this kernel.

In [1]:

```
import numpy as np    # linear algebra
import pandas as pd     # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns   # data visualization library
sns.set()
import matplotlib.pyplot as plt
import time
#import warnings library
import warnings
# ignore all warnings
warnings.filterwarnings('ignore')
```

Data Content

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

- The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean 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

Read and Analyse Data

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

Before making anything like feature selection,feature extraction and classification, firstly we start with basic data analysis. Lets look at features of data.

In [3]:
```
data.head()  # head method show only first 5 rows
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cor
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

There are 4 things that take my attention

- There is an **id** that cannot be used for classificaiton
- **Diagnosis** is our class label
- **Unnamed: 32** feature includes NaN so we do not need it.
- I do not have any idea about other feature names actually I do not need because machine learning is awesome **:)**

Therefore, drop these unnecessary features. However do not forget this is not a feature selection.

In [4]:
```
# feature names as a list
col = data.columns       # .columns gives columns names in data
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 includes our labels and x includes our features
y = data.diagnosis                          # M or B
list = ['Unnamed: 32','id','diagnosis']
x = data.drop(list,axis = 1 )
x.head()
```

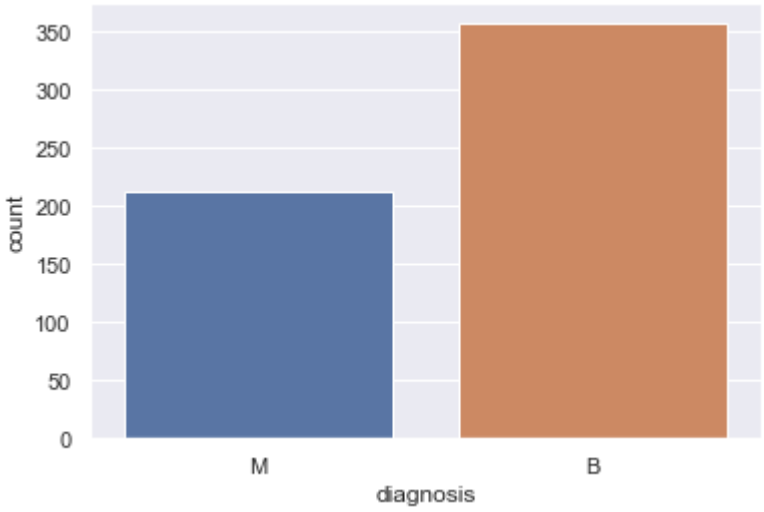
Out[5]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	conc
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

In [6]:
```
ax = sns.countplot(x=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
```



In [7]:
```
x.describe()
```

Out[7]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400

8 rows × 30 columns

Visualization

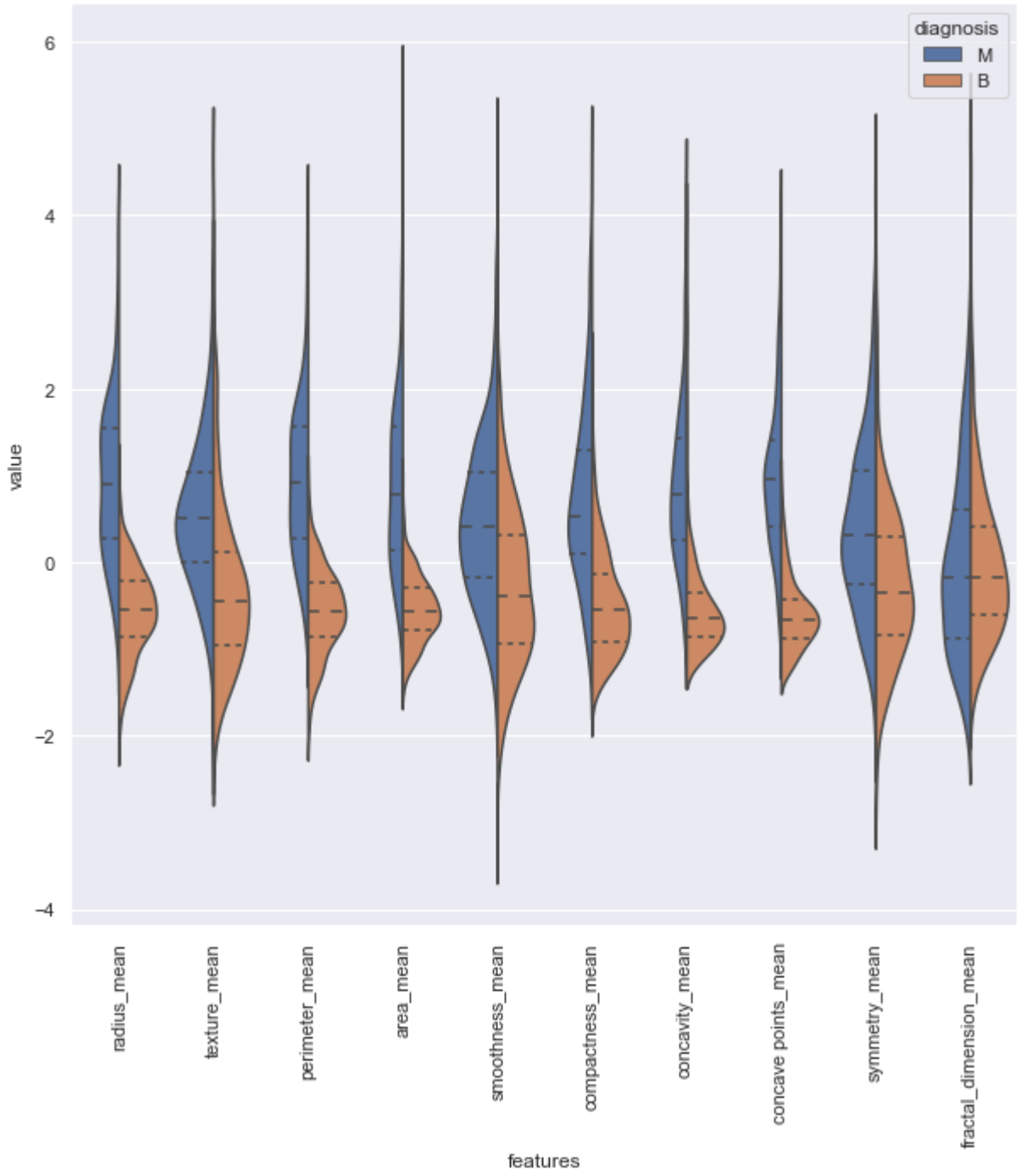
In order to visualizate data we are going to use seaborn plots that is not used in other kernels to inform you and for diversity of plots. What I use in real life is mostly violin plot and swarm plot. Do not forget we are not selecting feature, we are trying to know data like looking at the drink list at the pub door.

Before violin and swarm plot we need to normalization or standirdization. Because differences between values of features are very high to observe on plot. I plot features in 3 group and each group includes 10 features to observe better.

In [8]:
```
# first ten features
data_dia = y
```

```
data = x
data_n_2 = (data - data.mean()) / (data.std())              # standardization
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, inner="q
plt.xticks(rotation=90)
```

Out[8]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, 'radius_mean'),
  Text(1, 0, 'texture_mean'),
  Text(2, 0, 'perimeter_mean'),
  Text(3, 0, 'area_mean'),
  Text(4, 0, 'smoothness_mean'),
  Text(5, 0, 'compactness_mean'),
  Text(6, 0, 'concavity_mean'),
  Text(7, 0, 'concave points_mean'),
  Text(8, 0, 'symmetry_mean'),
  Text(9, 0, 'fractal_dimension_mean')])
```

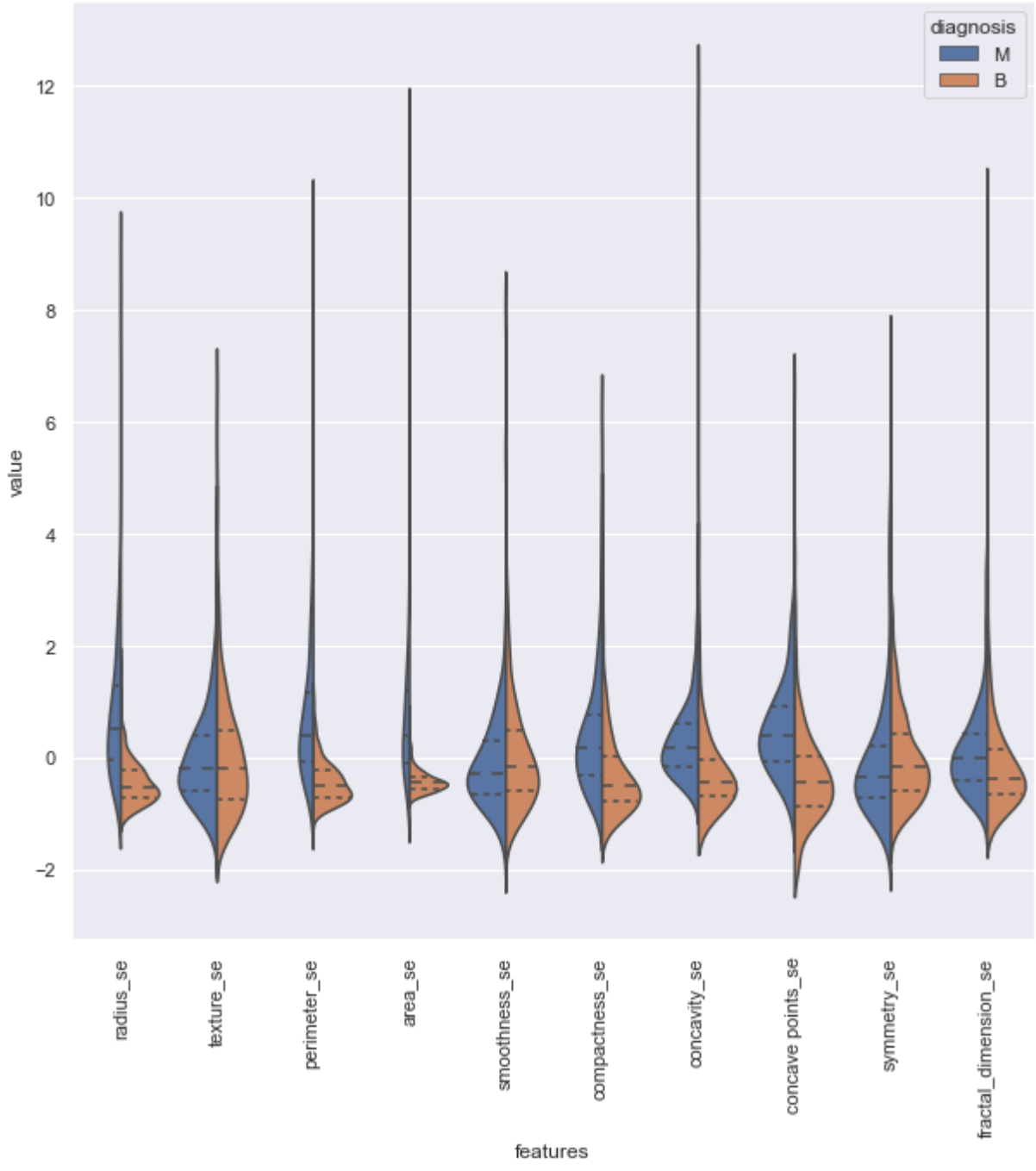


Lets interpret the plot above together. For example, in **texture_mean** feature, median of the *Malignant* and *Benign* looks like separated so it can be good for classification. However, in **fractal_dimension_mean** feature, median of the *Malignant* and *Benign* does not looks like separated so it does not gives good information for classification.

In [9]:

```
# Second ten features
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, inner="q
plt.xticks(rotation=90)
```

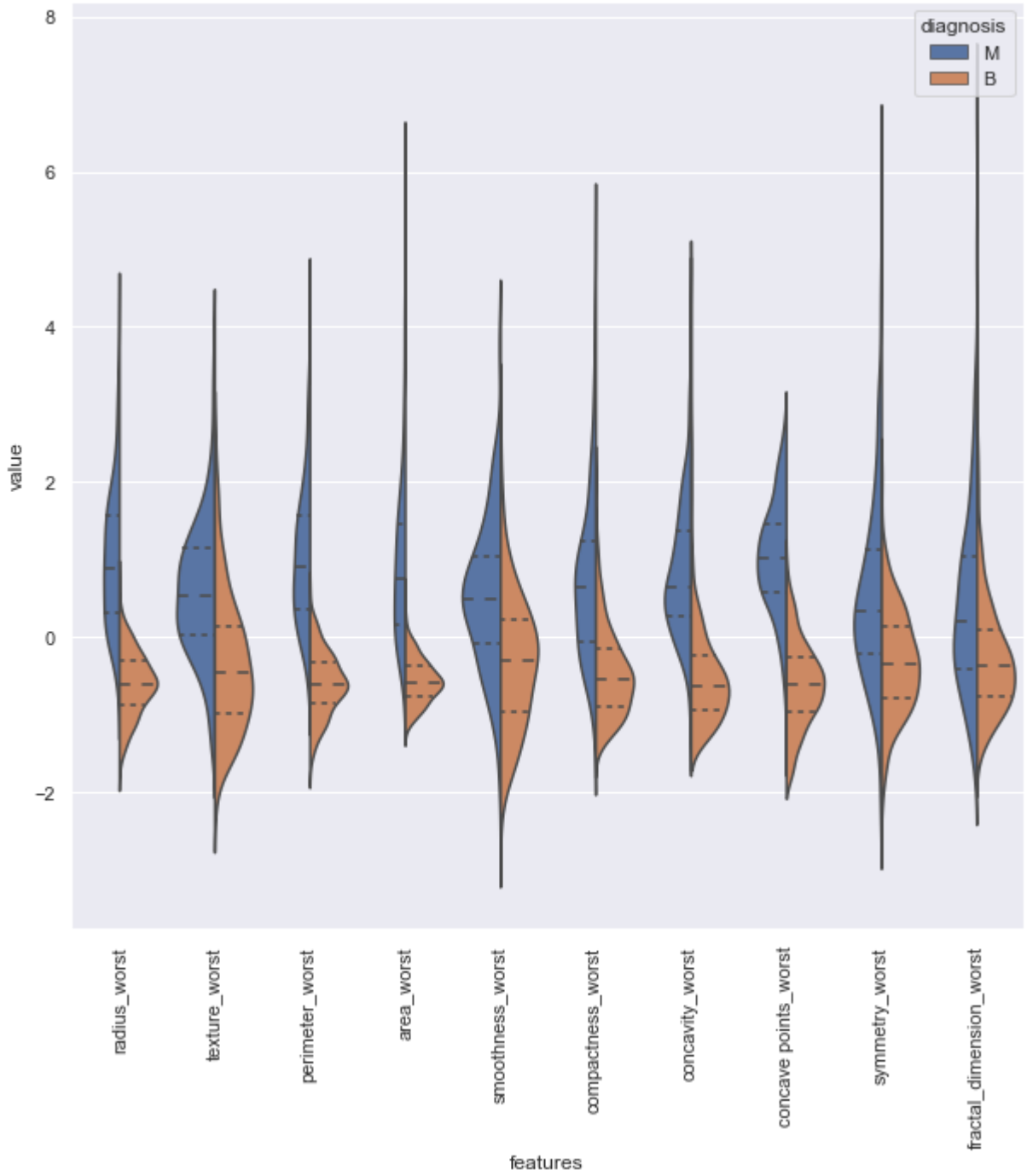
Out[9]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, 'radius_se'),
  Text(1, 0, 'texture_se'),
  Text(2, 0, 'perimeter_se'),
  Text(3, 0, 'area_se'),
  Text(4, 0, 'smoothness_se'),
  Text(5, 0, 'compactness_se'),
  Text(6, 0, 'concavity_se'),
  Text(7, 0, 'concave points_se'),
  Text(8, 0, 'symmetry_se'),
  Text(9, 0, 'fractal_dimension_se')])
```



In [10]:
```
# Second ten features
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, inner="q
plt.xticks(rotation=90)
```

Out[10]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, 'radius_worst'),
  Text(1, 0, 'texture_worst'),
  Text(2, 0, 'perimeter_worst'),
  Text(3, 0, 'area_worst'),
  Text(4, 0, 'smoothness_worst'),
  Text(5, 0, 'compactness_worst'),
  Text(6, 0, 'concavity_worst'),
  Text(7, 0, 'concave points_worst'),
  Text(8, 0, 'symmetry_worst'),
  Text(9, 0, 'fractal_dimension_worst')])
```

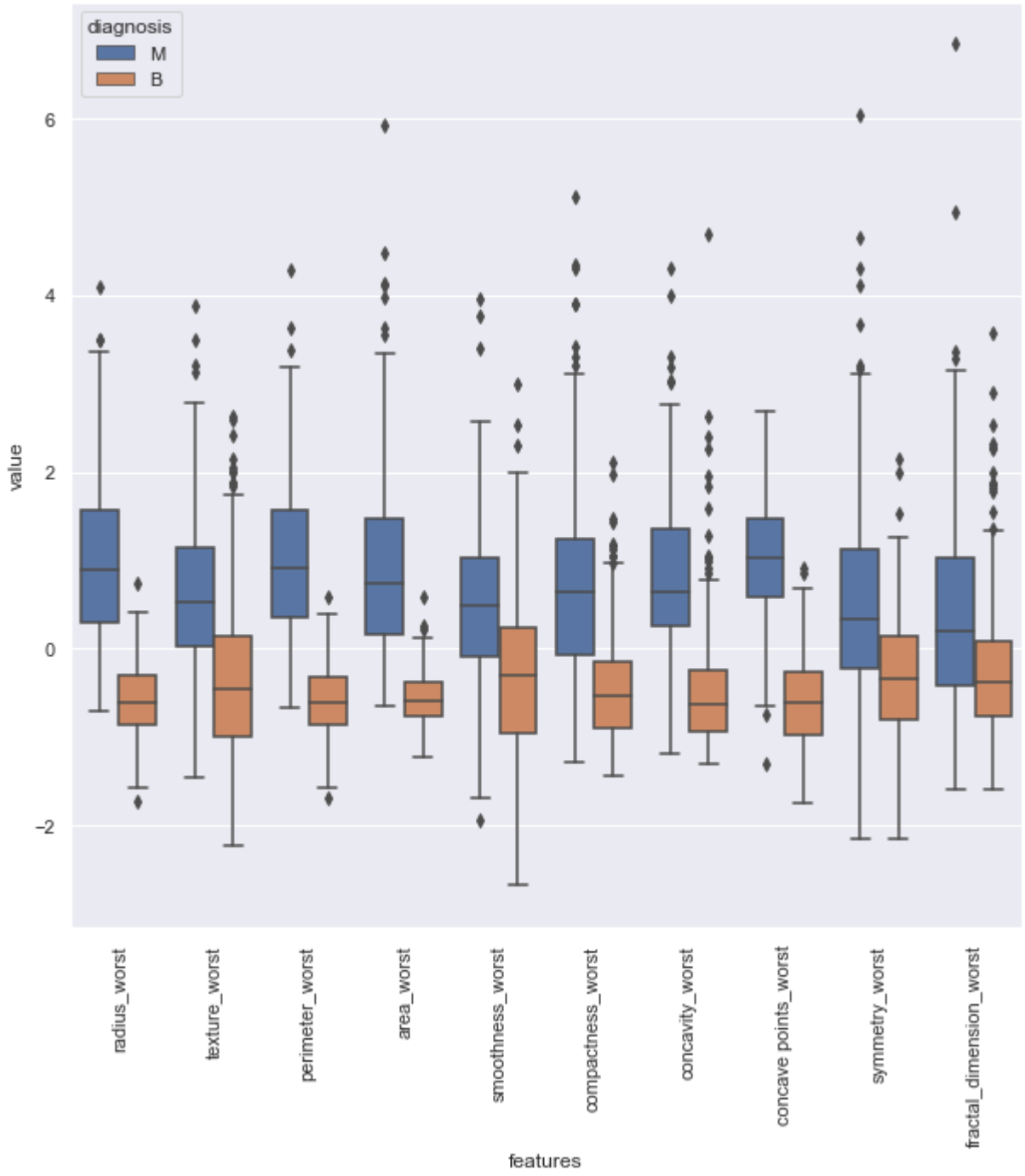


In [11]:

```
# As an alternative of violin plot, box plot can be used
# box plots are also useful in terms of seeing outliers
# I do not visualize all features with box plot
# In order to show you lets have an example of box plot
# If you want, you can visualize other features as well.
plt.figure(figsize=(10,10))
sns.boxplot(x="features", y="value", hue="diagnosis", data=data)
plt.xticks(rotation=90)
```

Out[11]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, 'radius_worst'),
  Text(1, 0, 'texture_worst'),
  Text(2, 0, 'perimeter_worst'),
  Text(3, 0, 'area_worst'),
  Text(4, 0, 'smoothness_worst'),
  Text(5, 0, 'compactness_worst'),
  Text(6, 0, 'concavity_worst'),
  Text(7, 0, 'concave points_worst'),
  Text(8, 0, 'symmetry_worst'),
  Text(9, 0, 'fractal_dimension_worst')])
```

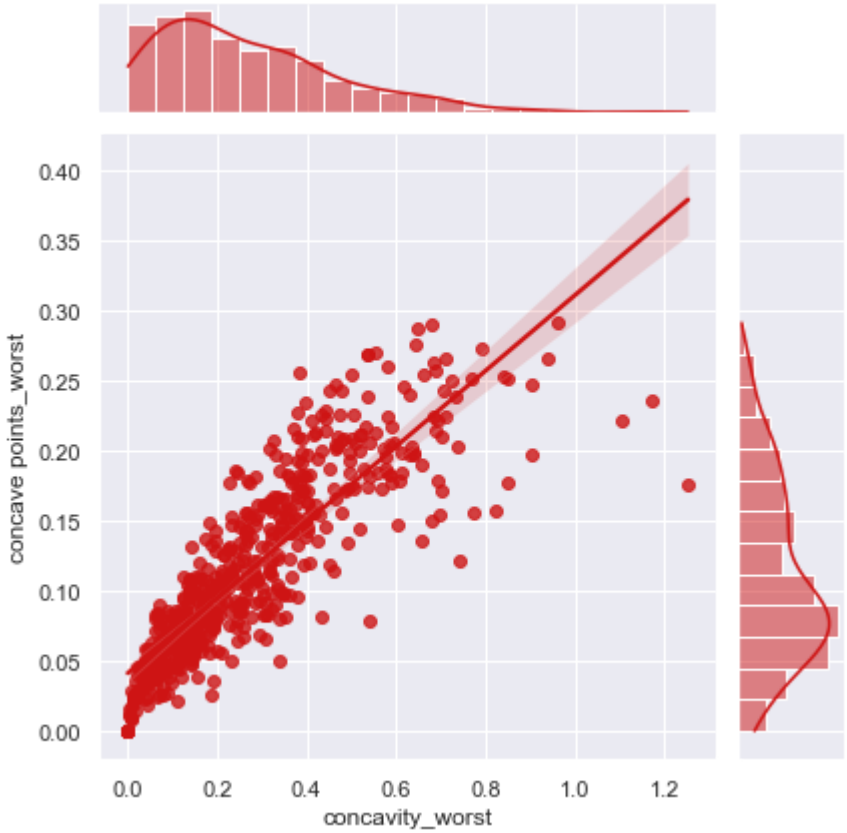


Lets interpret one more thing about plot above, variable of **concavity_worst** and **concave point_worst** looks like similar but how can we decide whether they are correlated with each other or not. (Not always true but, basically if the features are correlated with each other we can drop one of them)

In order to compare two features deeper, lets use joint plot. Look at this in joint plot below, it is really correlated. Pearsonr value is correlation value and 1 is the highest. Therefore, 0.86 is looks enough to say that they are correlated. Do not forget, we are not choosing features yet, we are just looking to have an idea about them.

In [12]:
```
sns.jointplot(x=x.loc[:,'concavity_worst'], y=x.loc[:,'concave points_worst'], kind="re
```

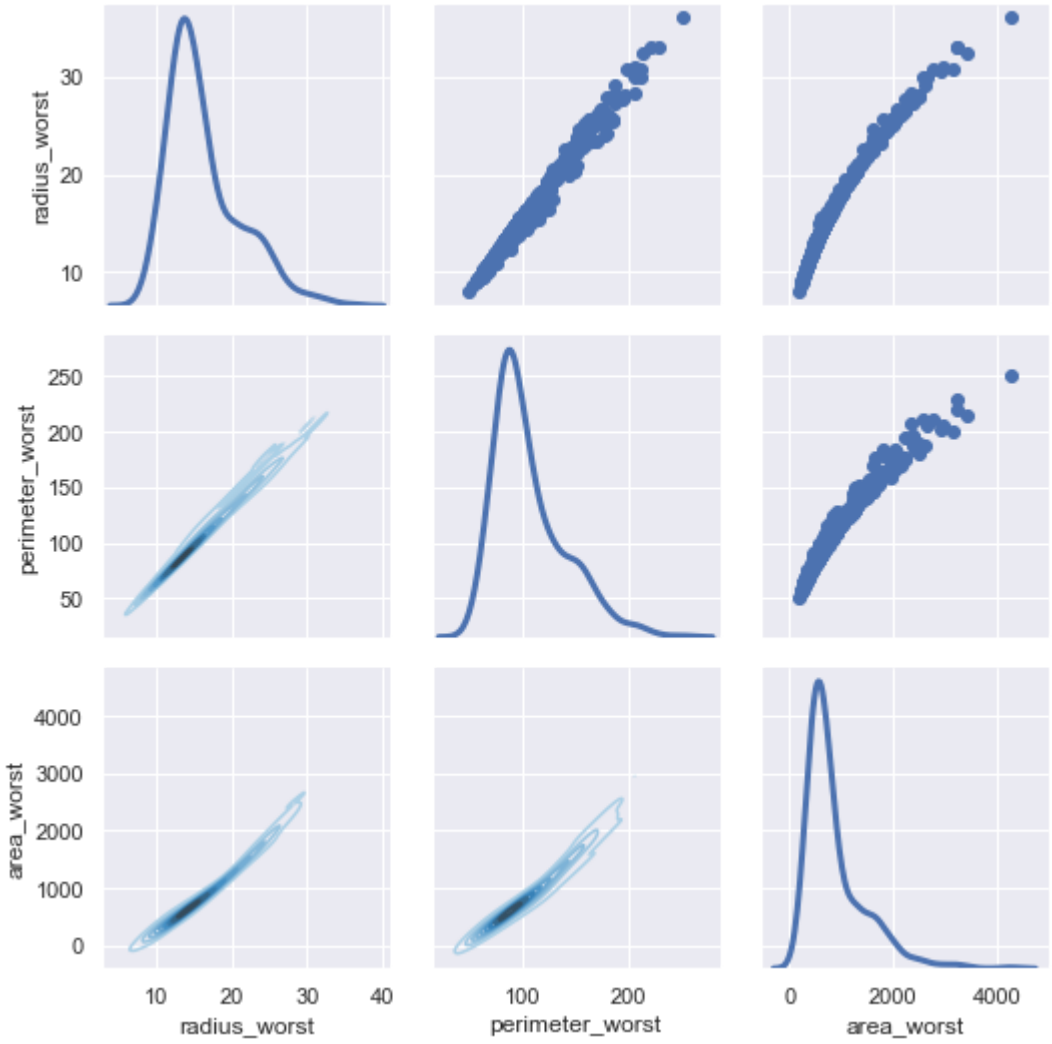
Out[12]: <seaborn.axisgrid.JointGrid at 0x17b70dca640>



What about three or more feauture comparision ? For this purpose we can use pair grid plot. Also it seems very cool :) And we discover one more thing **radius_worst**, **perimeter_worst** and **area_worst** are correlated as it can be seen pair grid plot. We definetely use these discoveries for feature selection.

In [13]:
```
# sns.set(style="white")
df = x.loc[:,['radius_worst','perimeter_worst','area_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)
```

Out[13]: <seaborn.axisgrid.PairGrid at 0x17b71310fa0>



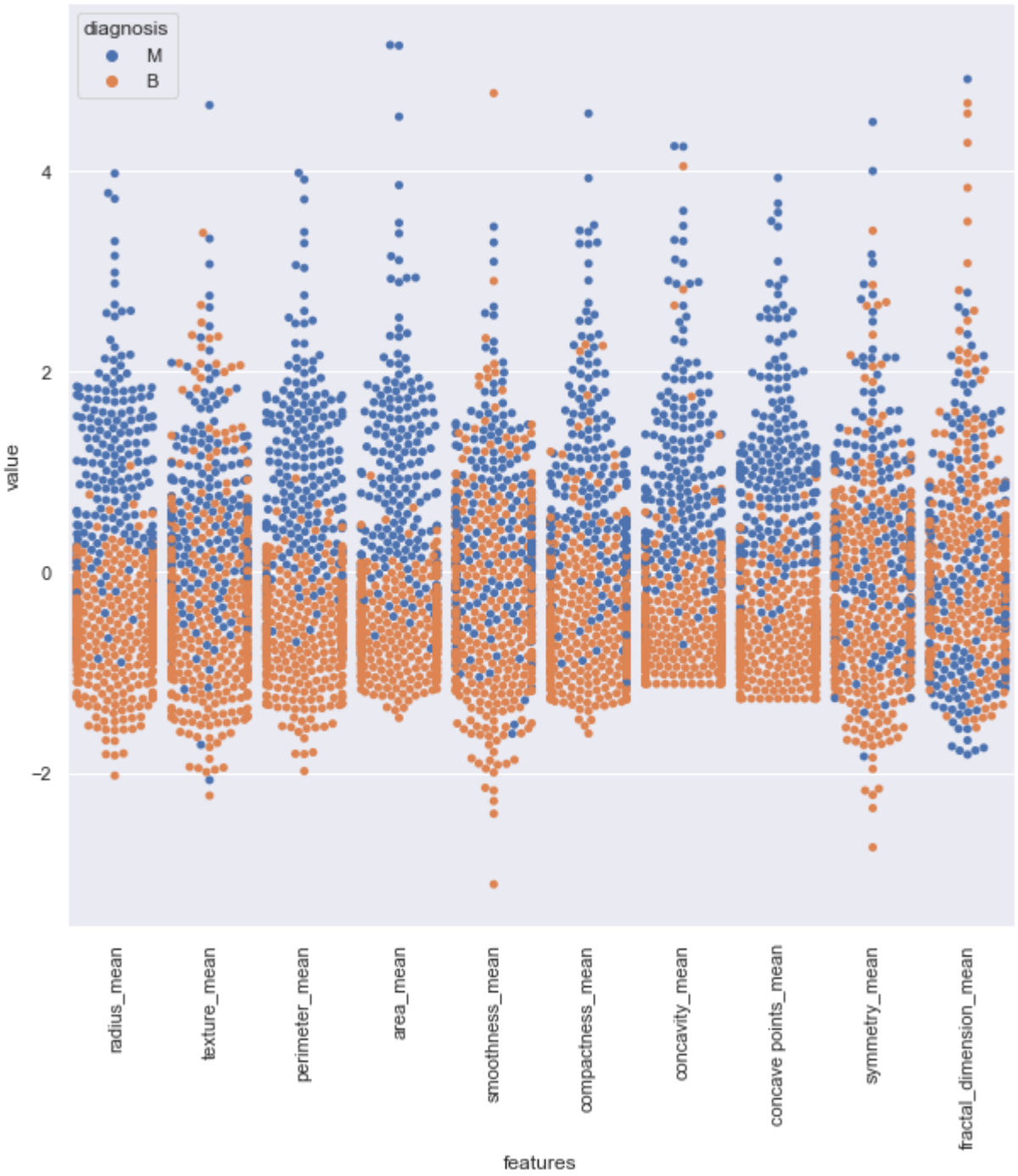
Up to this point, we make some comments and discoveries on data already. If you like what we did, I am sure swarm plot will open the pub's door :)

In swarm plot, I will do three part like violin plot not to make plot very complex appearance

In [14]:
```
# sns.set(style="whitegrid", palette="muted")
data_dia = y
data = x
data_n_2 = (data - data.mean()) / (data.std())              # standardization
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))
tic = time.time()
sns.swarmplot(x="features", y="value", hue="diagnosis", data=data)

plt.xticks(rotation=90)
```

Out[14]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [Text(0, 0, 'radius_mean'),
  Text(1, 0, 'texture_mean'),
  Text(2, 0, 'perimeter_mean'),
  Text(3, 0, 'area_mean'),
  Text(4, 0, 'smoothness_mean'),
  Text(5, 0, 'compactness_mean'),
```

```
 Text(6, 0, 'concavity_mean'),
 Text(7, 0, 'concave points_mean'),
 Text(8, 0, 'symmetry_mean'),
 Text(9, 0, 'fractal_dimension_mean')])
```



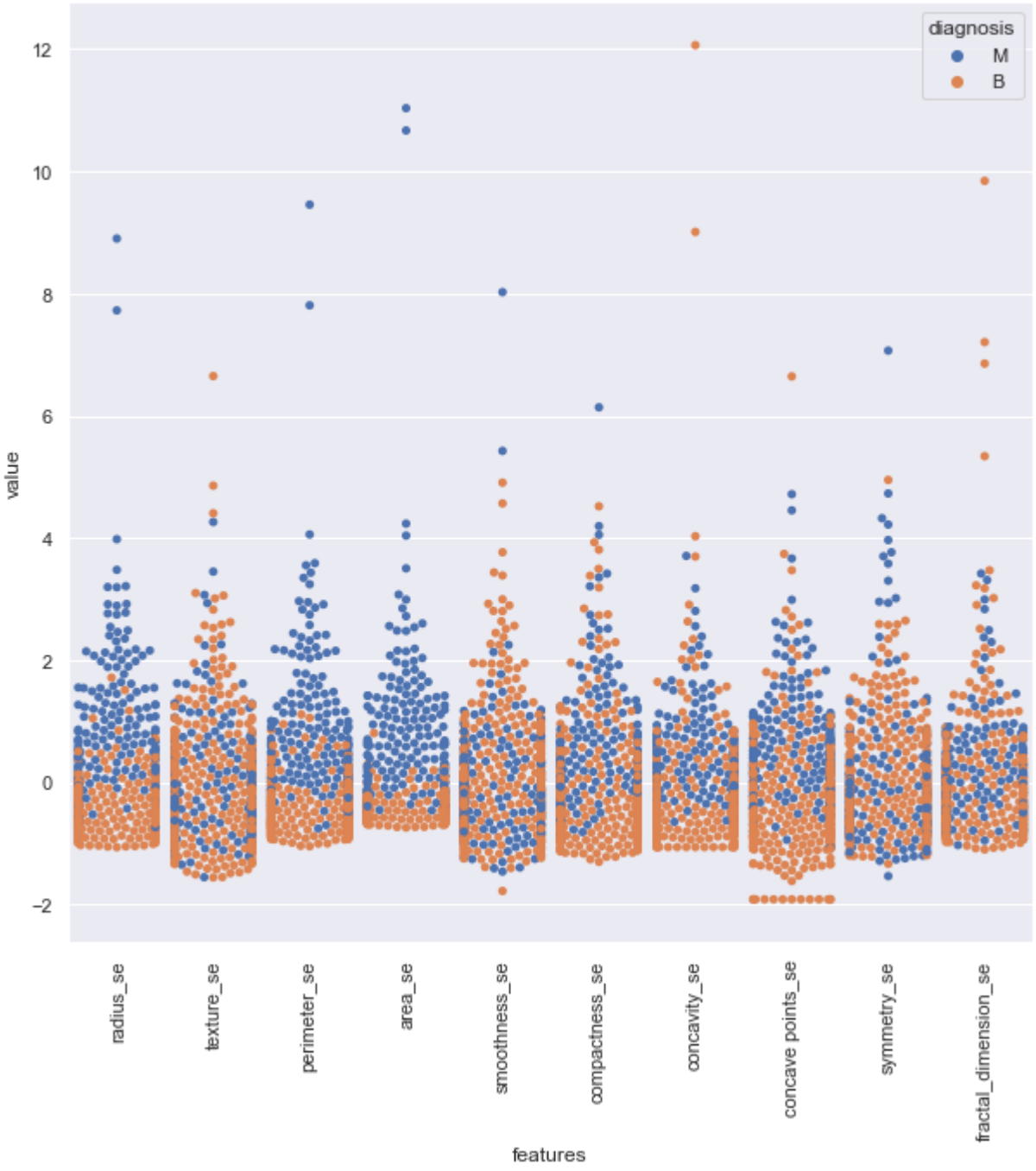
In [15]:

```
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=90)
```

Out[15]:

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [Text(0, 0, 'radius_se'),
  Text(1, 0, 'texture_se'),
```

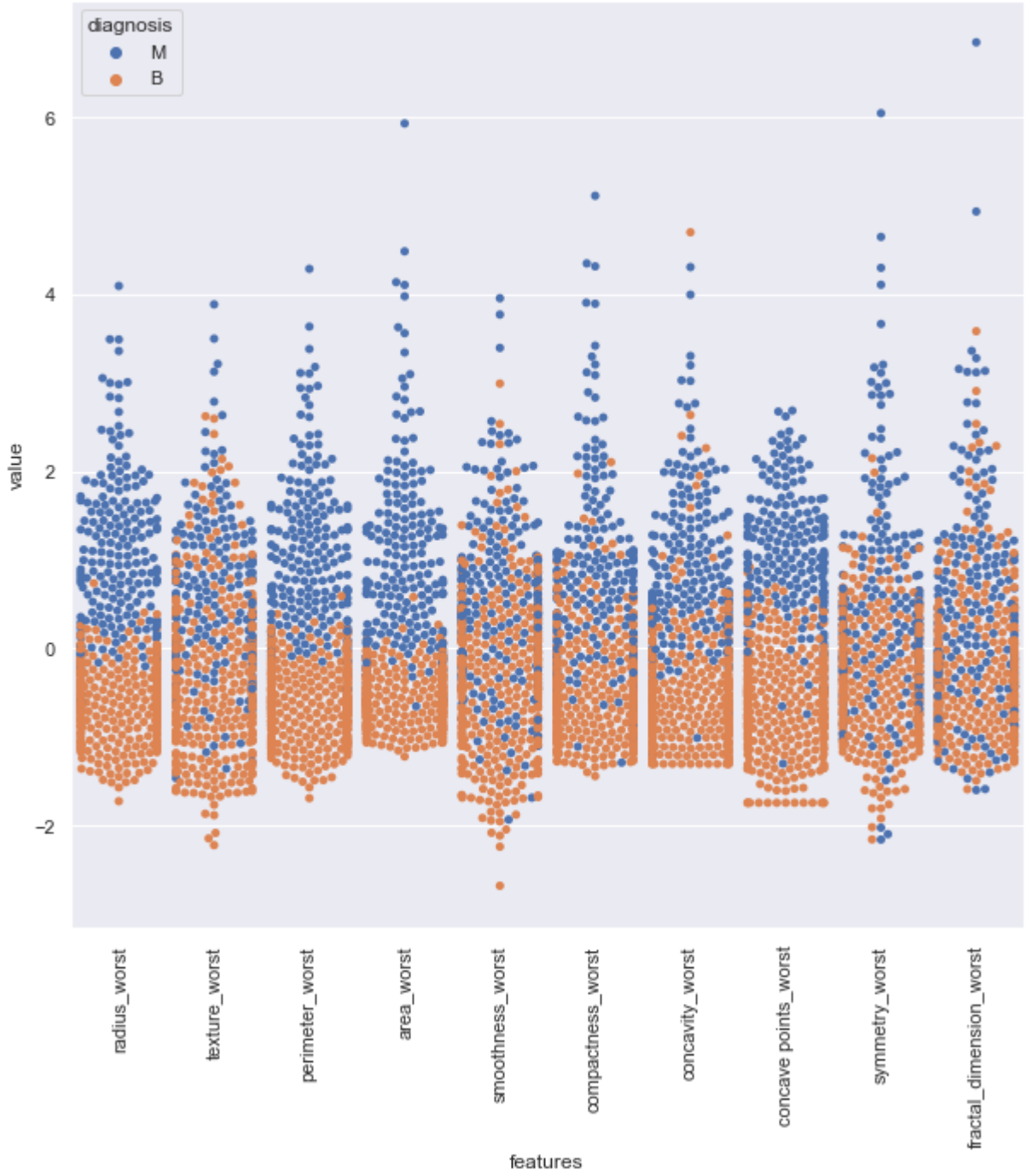
```
Text(2, 0, 'perimeter_se'),
Text(3, 0, 'area_se'),
Text(4, 0, 'smoothness_se'),
Text(5, 0, 'compactness_se'),
Text(6, 0, 'concavity_se'),
Text(7, 0, 'concave points_se'),
Text(8, 0, 'symmetry_se'),
Text(9, 0, 'fractal_dimension_se')])
```



In [16]:

```
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)
toc = time.time()
plt.xticks(rotation=90)
print("swarm plot time: ", toc-tic ," s")
```

```
swarm plot time:  53.97561526298523  s
```



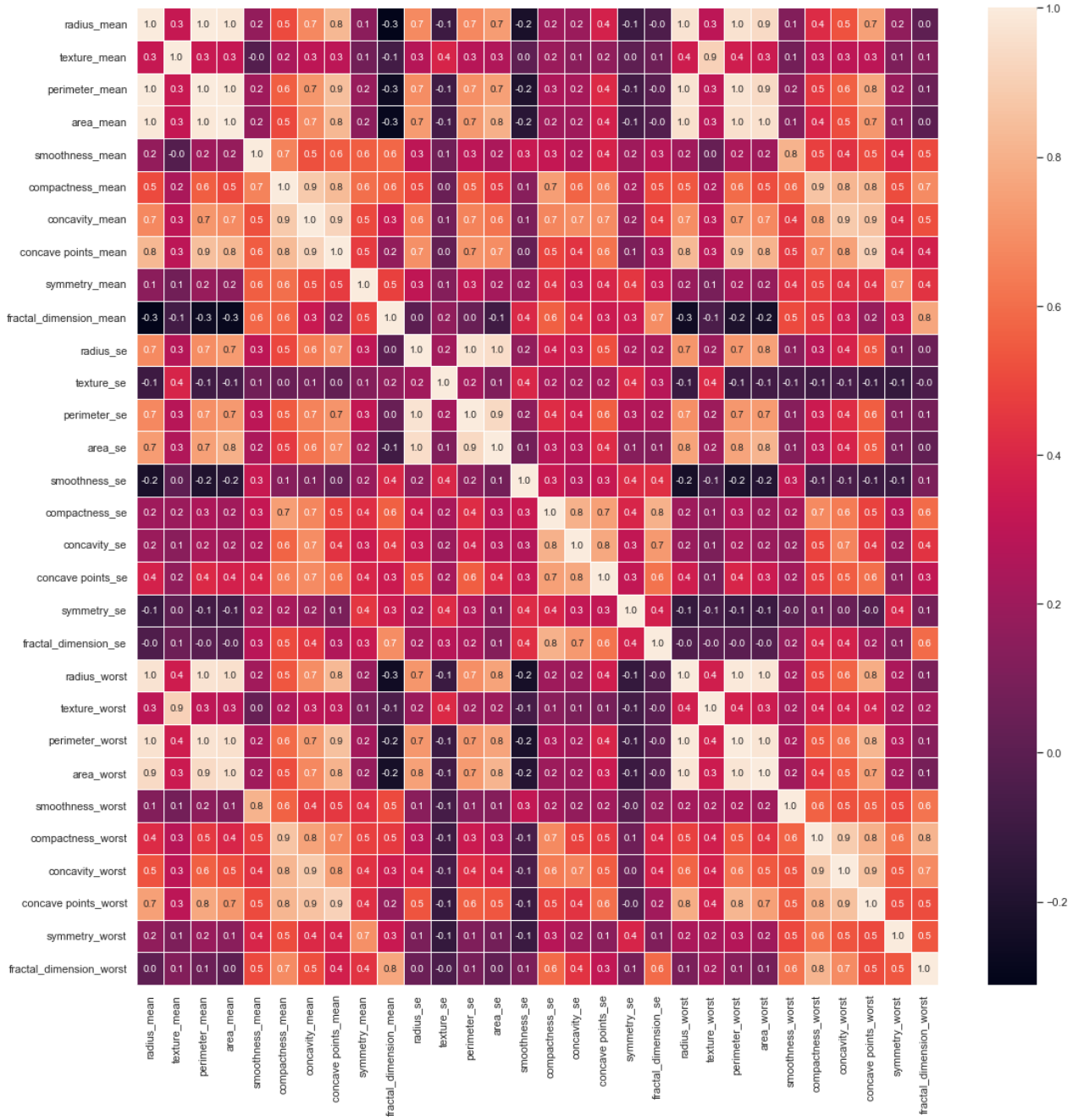
They looks cool right. And you can see variance more clear. Let me ask you a question, **in these three plots which feature looks like more clear in terms of classification.** In my opinion **area_worst** in last swarm plot looks like malignant and benign are seprated not totaly but mostly. Hovewer, **smoothness_se** in swarm plot 2 looks like malignant and benign are mixed so it is hard to classfy while using this feature.

What if we want to observe all correlation between features? Yes, you are right. The answer is heatmap that is old but powerful plot method.

In [17]:
```
#correlation map
f,ax = plt.subplots(figsize=(18, 18))
```

```
sns.heatmap(x.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

```
Out[17]: <AxesSubplot: >
```



Well, finaly we are in the pub and lets choose our drinks at feature selection part while using heatmap(correlation matrix).

Feature Selection and Random Forest Classification

Today our purpuse is to try new cocktails. For example, we are finaly in the pub and we want to drink different tastes. Therefore, we need to compare ingredients of drinks. If one of them includes lemon, after drinking it we need to eliminate other drinks which includes lemon so as to experience very different tastes.

In this part we will select feature with different methods that are feature selection with correlation, univariate feature selection, recursive feature elimination (RFE), recursive feature elimination with cross validation (RFECV) and tree based feature selection. We will use random forest classification in order to train our model and predict.

1) Feature selection with correlation and random forest classification

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

Compactness_mean, concavity_mean and concave points_mean are correlated with each other.Therefore I only choose **concavity_mean**. Apart from these, **radius_se, perimeter_se and area_se** are correlated and I only use **area_se**. **radius_worst, perimeter_worst and area_worst** are correlated so I use **area_worst**. **Compactness_worst, concavity_worst and concave points_worst** so I use **concavity_worst**. **Compactness_se, concavity_se and concave points_se** so I use **concavity_se**. **texture_mean and texture_worst are correlated** and I use **texture_mean**. **area_worst and area_mean** are correlated, I use **area_mean**.

In [18]:
```
drop_list1 = ['perimeter_mean','radius_mean','compactness_mean','concave points_mean','
x_1 = x.drop(drop_list1,axis = 1 )        # do not modify x, we will use it later
x_1.head()
```

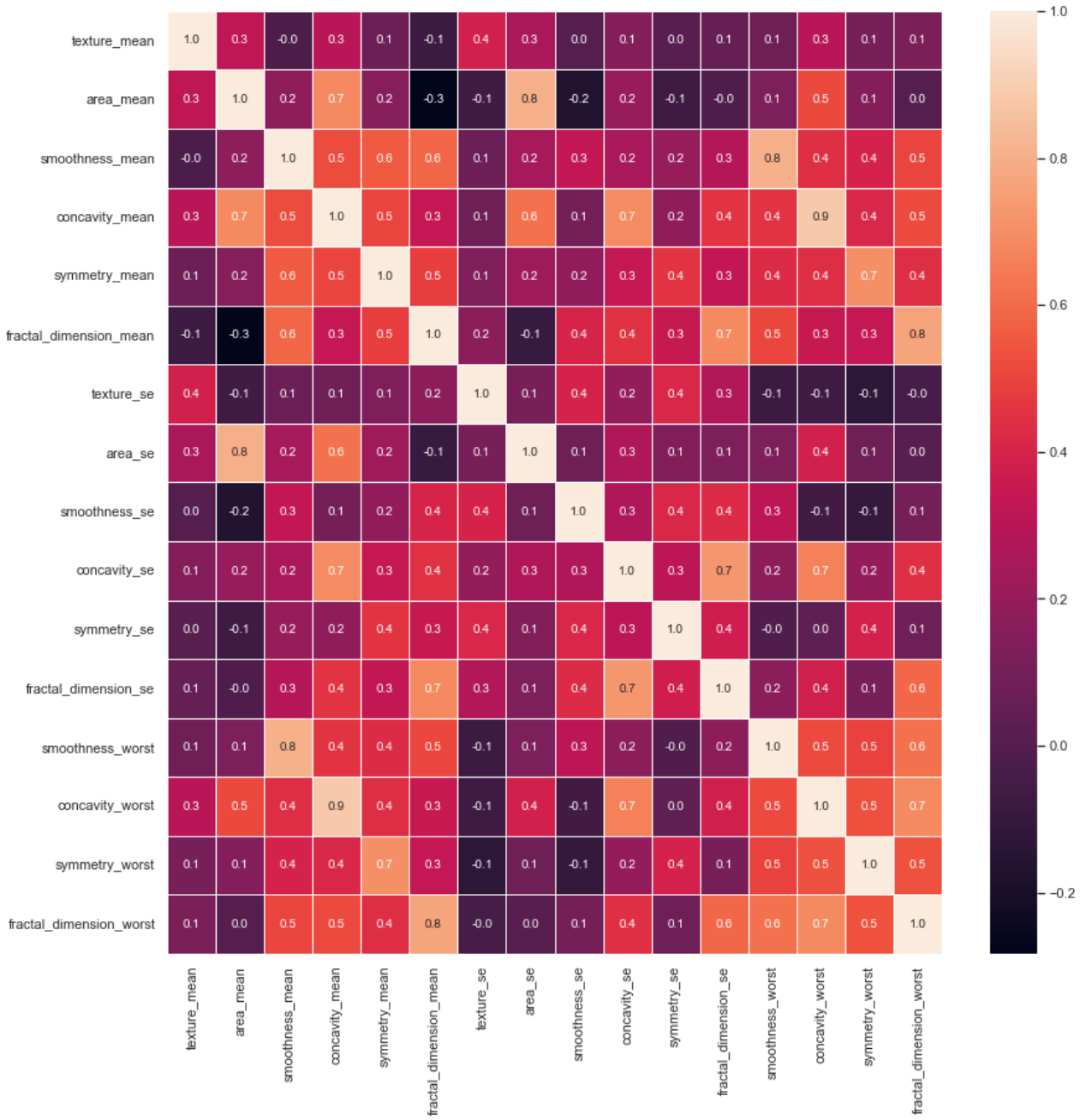
Out[18]:

	texture_mean	area_mean	smoothness_mean	concavity_mean	symmetry_mean	fractal_dimension_mea
0	10.38	1001.0	0.11840	0.3001	0.2419	0.0787
1	17.77	1326.0	0.08474	0.0869	0.1812	0.0566
2	21.25	1203.0	0.10960	0.1974	0.2069	0.0599
3	20.38	386.1	0.14250	0.2414	0.2597	0.0974
4	14.34	1297.0	0.10030	0.1980	0.1809	0.0588

After drop correlated features, as it can be seen in below correlation matrix, there are no more correlated features. Actually, I know and you see there is correlation value 0.9 but lets see together what happen if we do not drop it.

In [19]:
```
#correlation map
f,ax = plt.subplots(figsize=(14, 14))
sns.heatmap(x_1.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

Out[19]:
```
<AxesSubplot: >
```



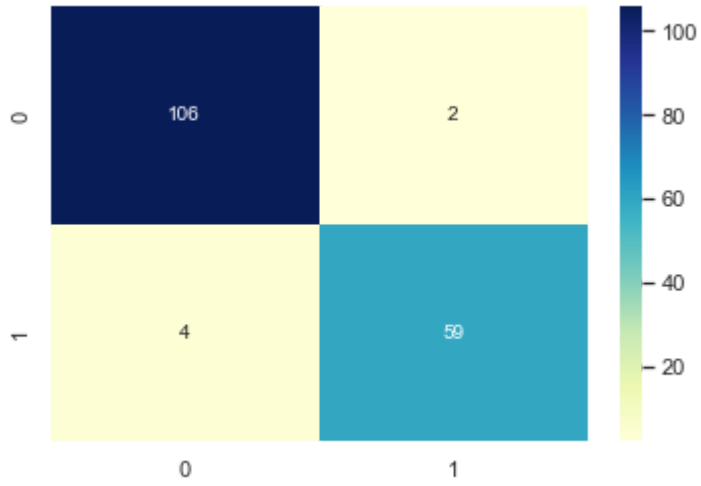
Well, we choose our features but **did we choose correctly ?** Lets use random forest and find accuracy according to chosen features.

In [20]:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# 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

#random forest classifier with n_estimators=10 (default)
clf_rf = RandomForestClassifier(random_state=43)
clr_rf = clf_rf.fit(x_train,y_train)
y_pred = clf_rf.predict(x_test)
cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(data=cm)
```

```
sns.heatmap(df_cm, annot=True, fmt='d', cmap='YlGnBu')

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           B       0.96      0.98      0.97       108
           M       0.97      0.94      0.95        63

    accuracy                           0.96       171
   macro avg       0.97      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```



2) Univariate feature selection and random forest classification

In univariate feature selection, we will use SelectKBest that removes all but the k highest scoring features.

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 trying or intuitively. I do not try all combinations but I only choose k = 5 and find best 5 features.

In [21]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# find best scored 5 features
select_feature = SelectKBest(chi2, k=5).fit(x_train, y_train)
```

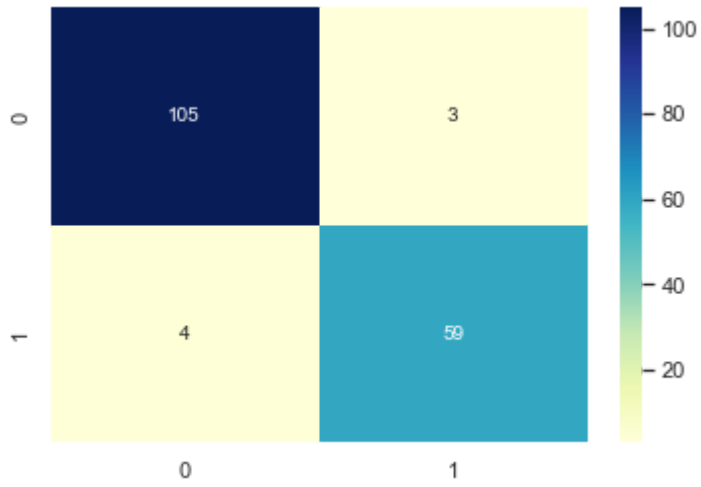
In [22]:
```
print('Score list:', select_feature.scores_)
print('Feature list:', x_train.columns)
```

```
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')
```

lets see what happens if we use only these best scored 5 feature.

In [23]:
```
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()
clr_rf_2 = clf_rf_2.fit(x_train_2,y_train)
y_pred = clf_rf_2.predict(x_test_2)
cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(data=cm)
sns.heatmap(df_cm, annot=True, fmt='d', cmap='YlGnBu')

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           B       0.96      0.97      0.97       108
           M       0.95      0.94      0.94        63

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171
```



3) Recursive feature elimination (RFE) with random forest

Basically, it uses one of the classification methods (random forest in our example), assign weights to each of features. Whose absolute weights are the smallest are pruned from the current set features. That procedure is recursively repeated on the pruned set until the desired number of features

Like previous method, we will use 5 features. However, which 5 features will we use ? We will choose them with RFE method.

In [24]:
```
from sklearn.feature_selection import RFE
# Create the RFE object and rank each pixel
clf_rf_3 = RandomForestClassifier()
rfe = RFE(estimator=clf_rf_3, n_features_to_select=5, step=1)
rfe = rfe.fit(x_train, y_train)
```

In [25]:
```
print('Chosen best 5 feature by rfe:',x_train.columns[rfe.support_])
```

```
Chosen best 5 feature by rfe: Index(['area_mean', 'concavity_mean', 'area_se', 'concavit
y_worst',
       'symmetry_worst'],
      dtype='object')
```

4) Recursive feature elimination with cross validation and random forest classification

Now we will not only **find best features** but we also find **how many features do we need** for best accuracy.

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

# The "accuracy" scoring is proportional to the number of correct classifications
clf_rf_4 = RandomForestClassifier()
rfecv = RFECV(estimator=clf_rf_4, step=1, cv=5,scoring='accuracy')   #5-fold cross-vali
rfecv = rfecv.fit(x_train, y_train)

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

```
Optimal number of features : 10
Best features : Index(['texture_mean', 'area_mean', 'smoothness_mean', 'concavity_mean',
       'fractal_dimension_mean', 'area_se', 'concavity_se', 'smoothness_worst',
       'concavity_worst', 'symmetry_worst'],
      dtype='object')
```

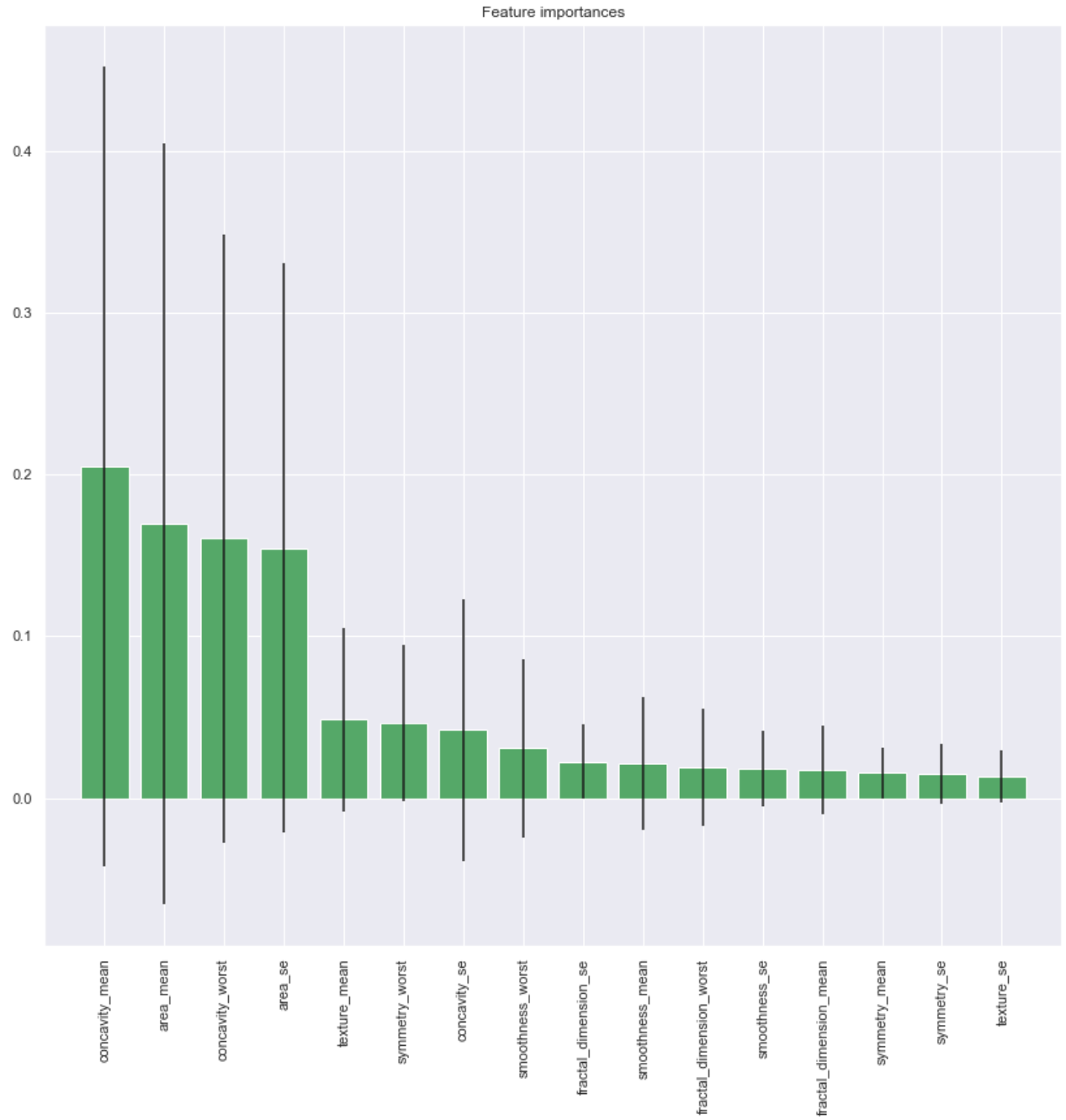
In [27]:
```
# Plot number of features VS. cross-validation scores
import matplotlib.pyplot as plt
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score of number of selected features")
plt.plot(range(1, len(rfecv.cv_results_["mean_test_score"]) + 1), rfecv.cv_results_["me
plt.show()
```

5) Tree based feature selection and random forest classification

In [28]:

```
clf_rf_5 = RandomForestClassifier()
clr_rf_5 = clf_rf_5.fit(x_train,y_train)
importances = clr_rf_5.feature_importances_
std = np.std([tree.feature_importances_ for tree in clf_rf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(x_train.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plot the feature importances of the forest

plt.figure(1, figsize=(14, 13))
plt.title("Feature importances")
plt.bar(range(x_train.shape[1]), importances[indices],
       color="g", yerr=std[indices], align="center")
plt.xticks(range(x_train.shape[1]), x_train.columns[indices],rotation=90)
plt.xlim([-1, x_train.shape[1]])
plt.show()
```

```
Feature ranking:
1. feature 3 (0.204729)
2. feature 1 (0.169547)
3. feature 13 (0.160635)
4. feature 7 (0.154687)
5. feature 0 (0.048466)
6. feature 14 (0.046354)
7. feature 9 (0.042033)
8. feature 12 (0.031078)
9. feature 11 (0.022403)
10. feature 2 (0.021307)
11. feature 15 (0.018985)
12. feature 8 (0.018327)
13. feature 5 (0.017113)
14. feature 4 (0.015578)
15. feature 10 (0.015281)
16. feature 6 (0.013477)
```



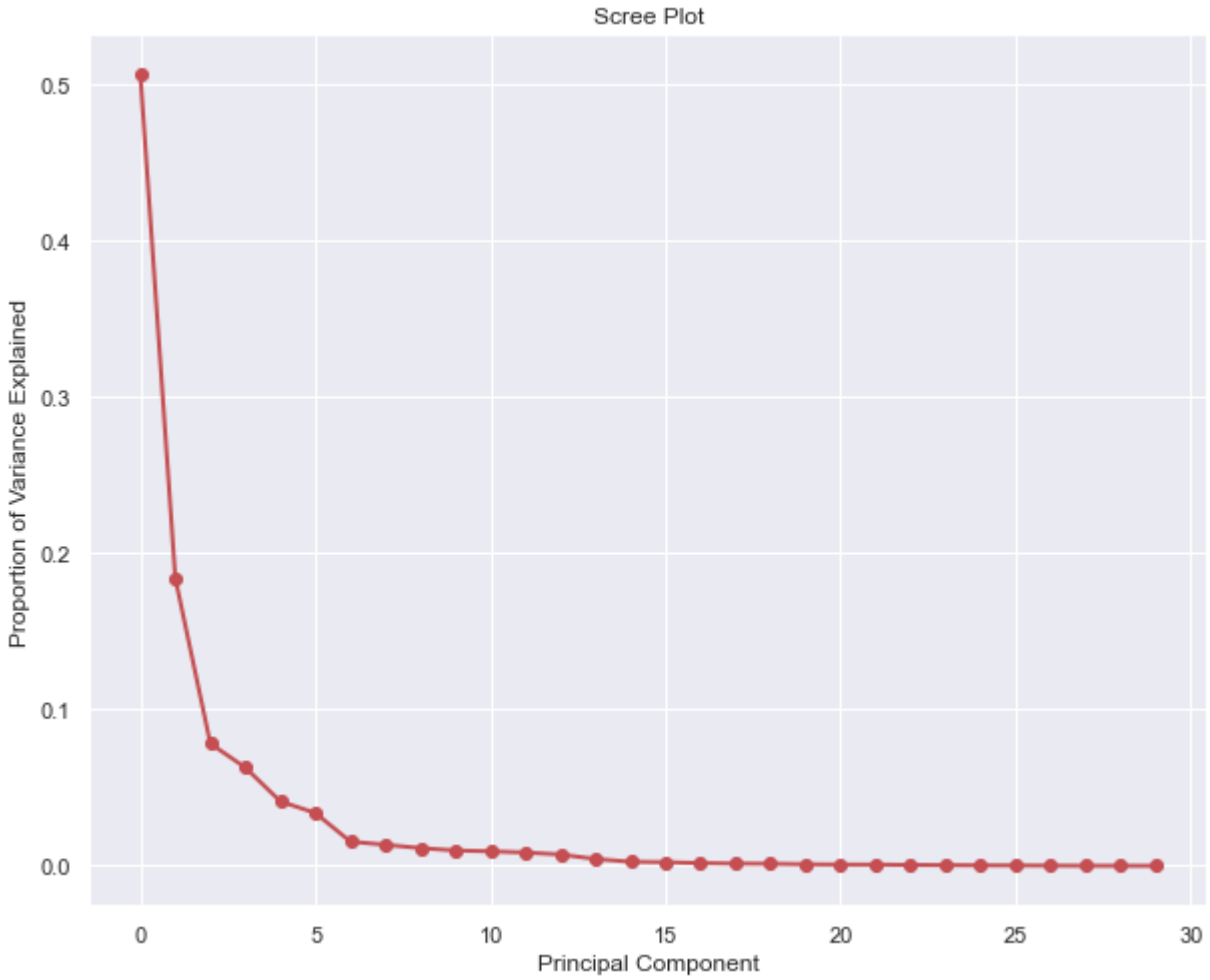
As you can seen in plot above, after 5 best features importance of features decrease. Therefore we can focus these 5 features. As I sad before, I give importance to understand features and find best of them.

Feature Extraction with PCA

In [29]:

```
# 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=4
#normalization
x_train_N = (x_train-x_train.mean())/(x_train.max()-x_train.min())
x_test_N = (x_test-x_test.mean())/(x_test.max()-x_test.min())

from sklearn.decomposition import PCA
pca = PCA()
```

```
pca.fit(x_train_N)

# Create scree plot
plt.figure(1, figsize=(10, 8))
plt.plot(pca.explained_variance_ratio_, 'ro-', linewidth=2)
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.show()
```



- According to variance ration, 6 component can be chosen.

In [31]:

```
# Split data into training and testing sets
# x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state

from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
pca = PCA(n_components=6)
X_train_pca = pca.fit_transform(x_train)
X_test_pca = pca.transform(x_test)
# Train a classification model on the transformed training data (e.g., using logistic r
# Train a logistic regression model on the transformed training data
clf = LogisticRegression()
clf.fit(X_train_pca, y_train)
# Use the trained model to make predictions on the transformed testing data.
# Make predictions on the transformed testing data
y_pred = clf.predict(X_test_pca)
```

```python
# Calculate confusion matrix
# Calculate a confusion matrix and other classification metrics
# (e.g., accuracy, precision, recall, F1 score) to evaluate the performance of the mode
cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(data=cm)
sns.heatmap(df_cm, annot=True, fmt='d', cmap='YlGnBu')

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           B       0.97      0.97      0.97       108
           M       0.95      0.95      0.95        63

    accuracy                           0.96       171
   macro avg       0.96      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```