In [48]:
```
import pandas as pd
df=pd.read_csv("breast-cancer-data.csv",header=0)
#print(df.head())
df.drop("Unnamed: 32",axis=1,inplace=True)
#df.info()
```

In [49]:
```
#print(df.describe())
df.drop("id",axis=1,inplace=True)
features_mean= list(df.columns[1:11])

features_se= list(df.columns[11:20])
features_worst=list(df.columns[21:31])
print(features_mean)
```

```
['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_me
an', 'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean']
```

In [50]:
```
df['diagnosis']=df['diagnosis'].map({'M':1,'B':0})
#print(df['diagnosis'])
```

In [4]:
```
df.describe()
```

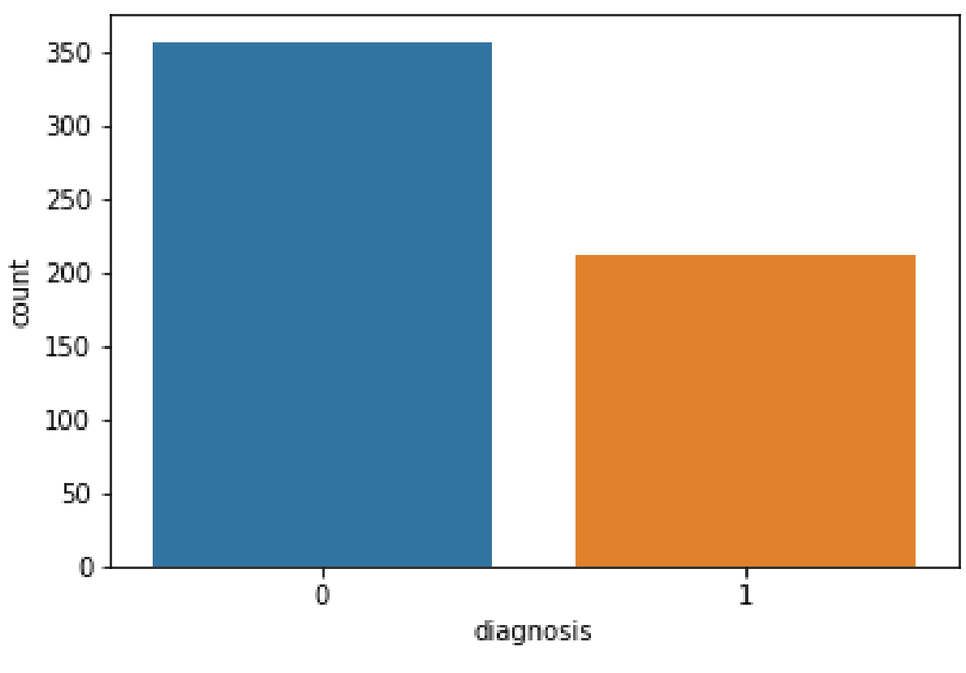
Out[4]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con points_n
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.00
mean	0.372583	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.04
std	0.483918	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.03
min	0.000000	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.00
25%	0.000000	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.02
50%	0.000000	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.03
75%	1.000000	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.07
max	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.20

8 rows × 31 columns

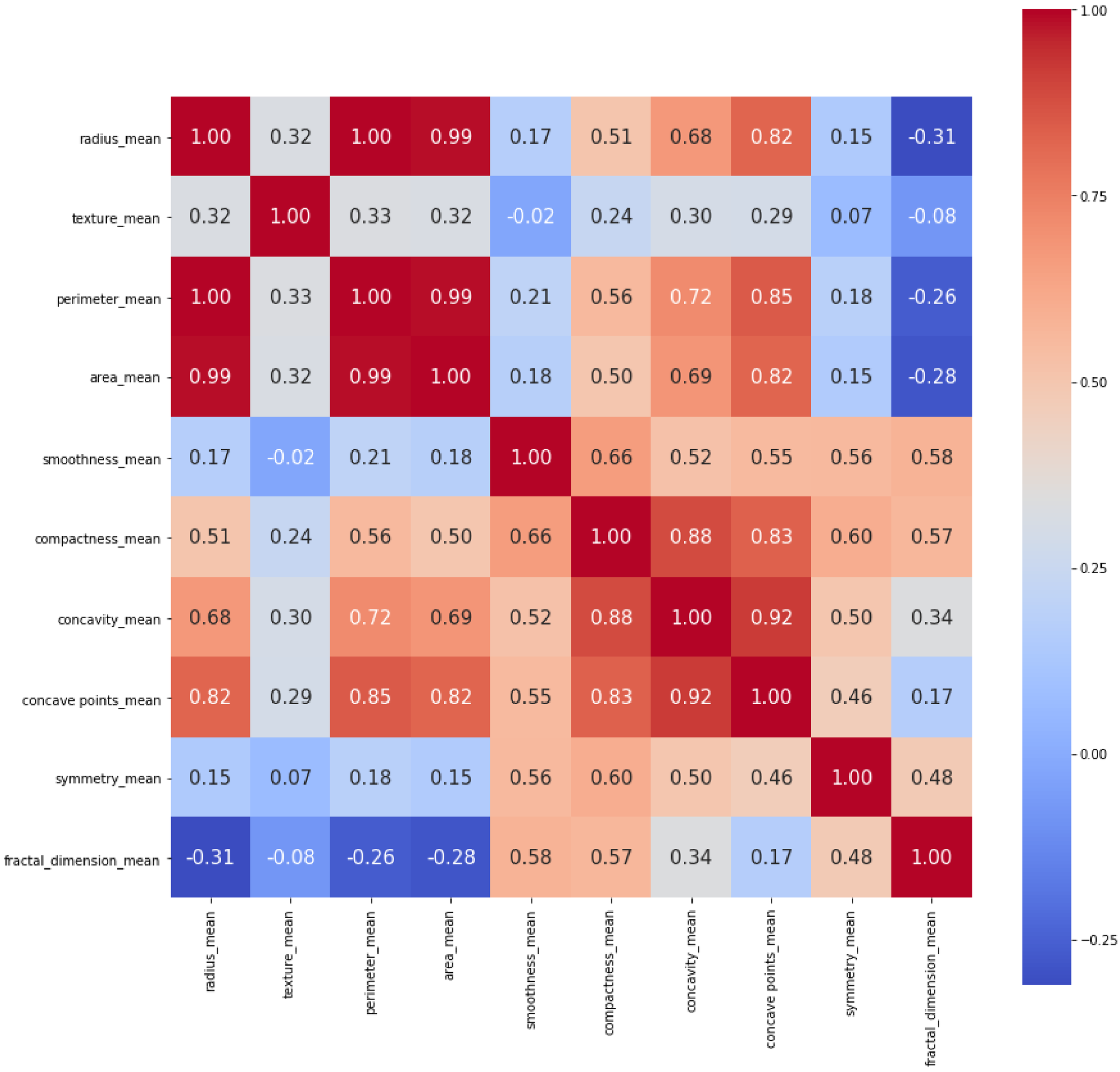
In [5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.countplot(df['diagnosis'],label="Count")
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x1e908698f60>



In [6]:
```
corr = df[features_mean].corr() # .corr is used for find corelation
plt.figure(figsize=(14,14))
sns.heatmap(corr, cbar = True,  square = True, annot=True, fmt= '.2f',annot_kws={'size': 15},
            xticklabels= features_mean, yticklabels= features_mean,
            cmap= 'coolwarm')
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1e9089a53c8>



In [31]:
```
from sklearn.model_selection import train_test_split
prediction_var = ['texture_mean','perimeter_mean','smoothness_mean','compactness_mean','symmetry_mean']
train, test = train_test_split(df, test_size = 0.3)
print(train.shape)
print(test.shape)
train_X = train[prediction_var]
train_y=train.diagnosis
test_X= test[prediction_var]
test_y =test.diagnosis
```

```
(398, 31)
(171, 31)
```

In [21]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
model=RandomForestClassifier(n_estimators=100)
model.fit(train_X,train_y)
```

Out[21]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [22]:
```
prediction=model.predict(test_X)
```

In [23]:
```
metrics.accuracy_score(prediction,test_y)
```

Out[23]:
```
0.9473684210526315
```

In [24]:
```
prediction_var = features_mean
train_X= train[prediction_var]
train_y= train.diagnosis
test_X = test[prediction_var]
test_y = test.diagnosis
```

In [52]:
```
model=RandomForestClassifier(n_estimators=10000)
```

In [26]:
```
model.fit(train_X,train_y)
prediction = model.predict(test_X)
metrics.accuracy_score(prediction,test_y)
```

Out[26]:
```
0.9532163742690059
```

In [27]:
```
from sklearn.svm import SVC
from sklearn.model_selection import KFold
import numpy as np
#from sklearn.model_selection import cross_val_scores
```

In [28]:
```
def classification_model(model,data,prediction_input,output):
    model.fit(data[prediction_input],data[output])
    predictions = model.predict(data[prediction_input])
    accuracy = metrics.accuracy_score(predictions,data[output])
    print("Accuracy : %s" % "{0:.3%}".format(accuracy))
    kf = KFold(n_splits=5)
    print(kf.get_n_splits(data))
    error = []
    for train, test in kf.split(data):
        train_X = (data[prediction_input].iloc[train,:])
        train_y = data[output].iloc[train]
        model.fit(train_X, train_y)
        test_X=data[prediction_input].iloc[test,:]
        test_y=data[output].iloc[test]
        error.append(model.score(test_X,test_y))
        print("Cross-Validation Score : %s" % "{0:.3%}".format(np.mean(error)))
```

In [32]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
prediction_var =features_mean
outcome_var= "diagnosis"
classification_model(model,df,prediction_var,outcome_var)
```

```
Accuracy : 100.000%
5
Cross-Validation Score : 88.596%
Cross-Validation Score : 89.035%
Cross-Validation Score : 91.228%
Cross-Validation Score : 91.886%
Cross-Validation Score : 91.739%
```

In [33]:
```
model = RandomForestClassifier(n_estimators=100)
classification_model(model,df,prediction_var,outcome_var)
```

```
Accuracy : 100.000%
5
Cross-Validation Score : 88.596%
Cross-Validation Score : 90.789%
Cross-Validation Score : 93.275%
Cross-Validation Score : 93.860%
Cross-Validation Score : 94.380%
```

In [34]:
```
model = RandomForestClassifier(n_estimators=100,criterion='entropy',max_depth=40)
classification_model(model,df,prediction_var,outcome_var)
```

```
Accuracy : 100.000%
5
Cross-Validation Score : 87.719%
Cross-Validation Score : 90.789%
Cross-Validation Score : 92.982%
Cross-Validation Score : 93.860%
Cross-Validation Score : 94.026%
```

In [35]:
```
from sklearn.model_selection import GridSearchCV
data_X= df.iloc[:,2:]
data_y= df["diagnosis"]
#print(data_X)
#print(data_y)
#print(train_X)
#print(train_Y)
```

In [36]:
```
def Classification_model_gridsearchCV(model,param_grid,data_X,data_y):
    clf = GridSearchCV(model,param_grid,cv=10,scoring="accuracy")
    clf.fit(data_X,data_y)
    print("The best parameter found on development set is :")
    print(clf.best_params_)
    print("the bset estimator is ")
    print(clf.best_estimator_)
    print("The best score is ")
    print(clf.best_score_)
```

In [37]:
```
param_grid = {'max_features': ['auto', 'sqrt', 'log2'],
              'min_samples_split': [2,3,4,5,6,7,8,9,10],
              'min_samples_leaf':[2,3,4,5,6,7,8,9,10] }
model= DecisionTreeClassifier()
Classification_model_gridsearchCV(model,param_grid,data_X,data_y)
```

```
The best parameter found on development set is :
{'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 3}
the bset estimator is
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features='sqrt', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=4, min_samples_split=3,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
The best score is
0.9507908611599297

C:\Users\hp\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:841: DeprecationWarning: The default of the `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

In [45]:
```
from sklearn import svm
model=svm.SVC()
param_grid = [
              {'C': [1, 10, 100, 1000],
               'kernel': ['linear']
              },
              {'C': [1, 10, 100, 1000],
               'gamma': [0.001, 0.0001],
               'kernel': ['rbf']
              },
 ]
Classification_model_gridsearchCV(model,param_grid,data_X,data_y)
```

```
The best parameter found on development set is :
{'C': 10, 'kernel': 'linear'}
the bset estimator is
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
The best score is
0.961335676625659
```

In [46]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
#print(df['diagnosis'])
X=df.iloc[:,2:]
y=df.iloc[:,0:1]
```

In [47]:
```
#from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size=0.30, random_state=1)
```

In [41]:
```
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('pca', PCA(n_components=2)),
                    ('clf', LogisticRegression(random_state=1))])
pipe_lr.fit(X_train, y_train)
print('Test Accuracy: %.3f' % pipe_lr.score(X_test, y_test))
```

```
Test Accuracy: 0.947

C:\Users\hp\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be
changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\hp\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```