Breast cancer

breast cancer Machine Learning model by - Tarun Sharma

In [1]:
```
# here we will import the libraries used for machine learning
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
from sklearn.linear_model import LogisticRegression # to apply the Logistic regression
from sklearn.model_selection import train_test_split # to split the data into two parts
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV# for tuning parameter
from sklearn.ensemble import RandomForestClassifier # for random forest classifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm # for Support Vector Machine
from sklearn import metrics # for the check the error and accuracy of the model
# Any results you write to the current directory are saved as output.
# dont worry about the error if its not working then insteda of model_selection we can use cross_validation
```

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

In [3]:
```
data.keys()  #
```

Out[3]:
```
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 [4]:
```
data.head()
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 33 columns

In [5]:
```
# remove id column
#data.drop('id', axis=1, inplace=True)
# remove Unnamed:32 Column
#data.drop('Unnamed: 32', axis=1, inplace=True)
```

In [6]:
```
data.head()
```

Out[6]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 33 columns

In [7]:
```
features_mean = list(data.columns[1:11])
features_se = list(data.columns[11:20])
features_worst = list(data.columns[21:31])
print(features_mean)
print(features_se)
print(features_worst)
```

```
['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_me
an', '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']
['fractal_dimension_se', 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst',
'compactness_worst', 'concavity_worst', 'concave points_worst', 'symmetry_worst']
```

In [8]:
```
# map the value of diagnosis M=1, B=0
data['diagnosis'] = data['diagnosis'].map({'M':1, 'B':0})
```

In [9]:
```
# give mean
data.describe()
```

Out[9]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	co
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	3.037183e+07	0.372583	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	
std	1.250206e+08	0.483918	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	
min	8.670000e+03	0.000000	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	
25%	8.692180e+05	0.000000	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	
50%	9.060240e+05	0.000000	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	
75%	8.813129e+06	1.000000	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	
max	9.113205e+08	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	

8 rows × 33 columns

In [10]:
```
# show data
data.head()
```

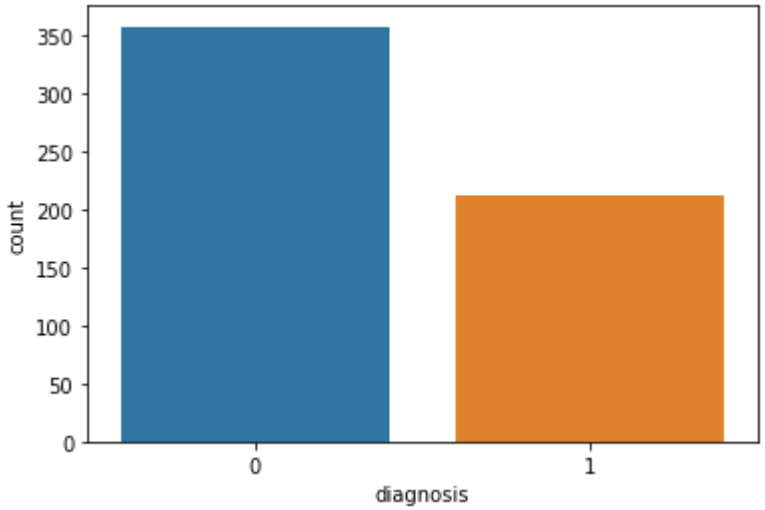
Out[10]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 33 columns

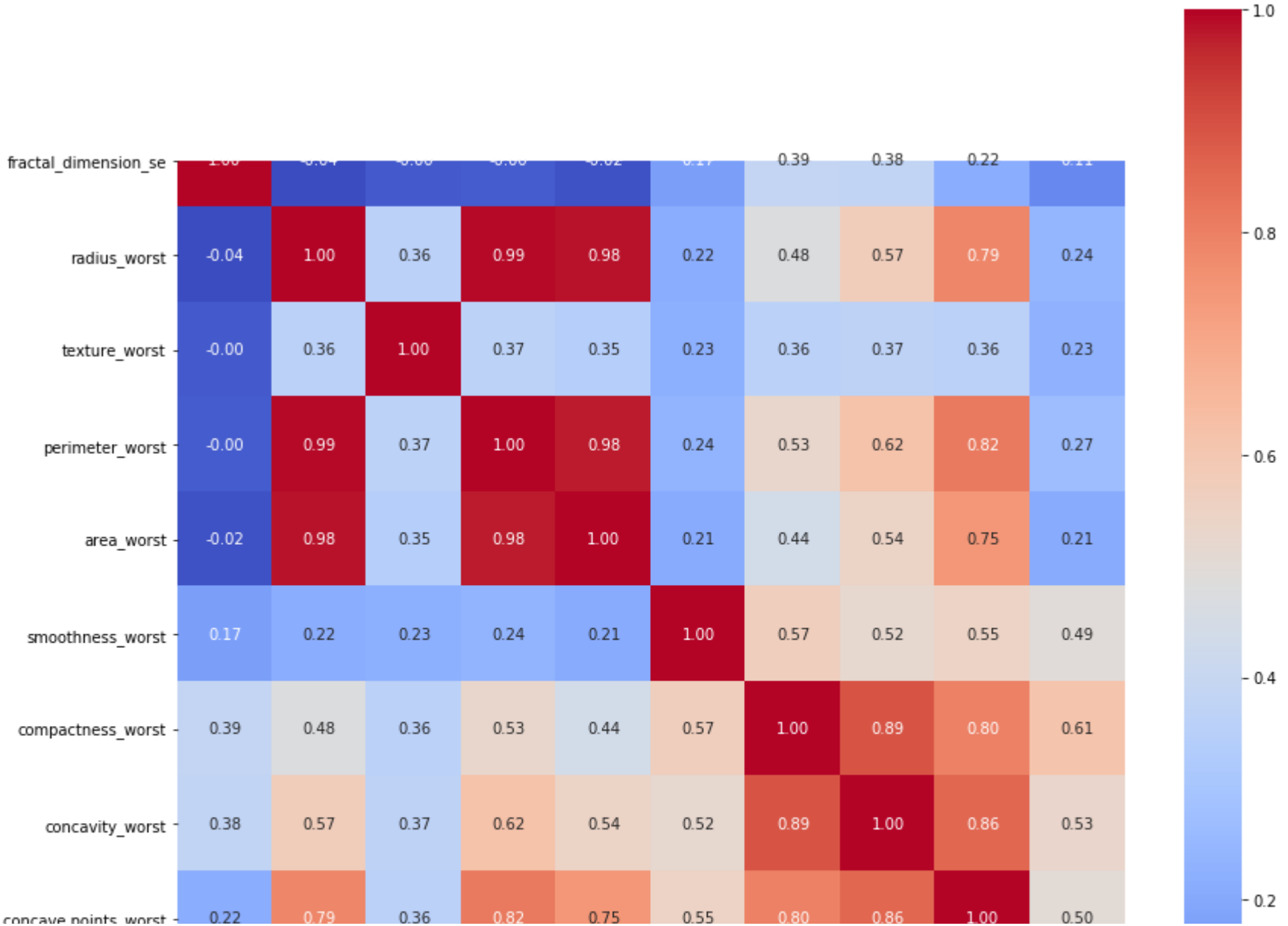
In [11]:
```
# using seaborn library we make graph of Diganosis column
sns.countplot(data['diagnosis'],label='Counts')
```

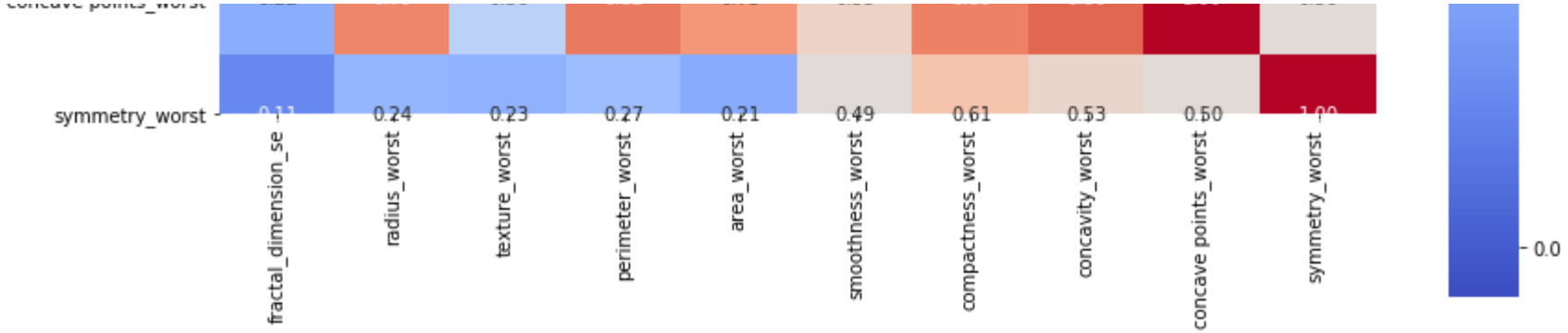
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1d679ff6688>



In [12]:
```
corr = data[features_worst].corr() # use to find coll
plt.figure(figsize=(14,14))
sns.heatmap(corr, cbar = True, square = True, annot= True, fmt= '.2f', annot_kws = {'size': 10}, xticklabels= fe
```

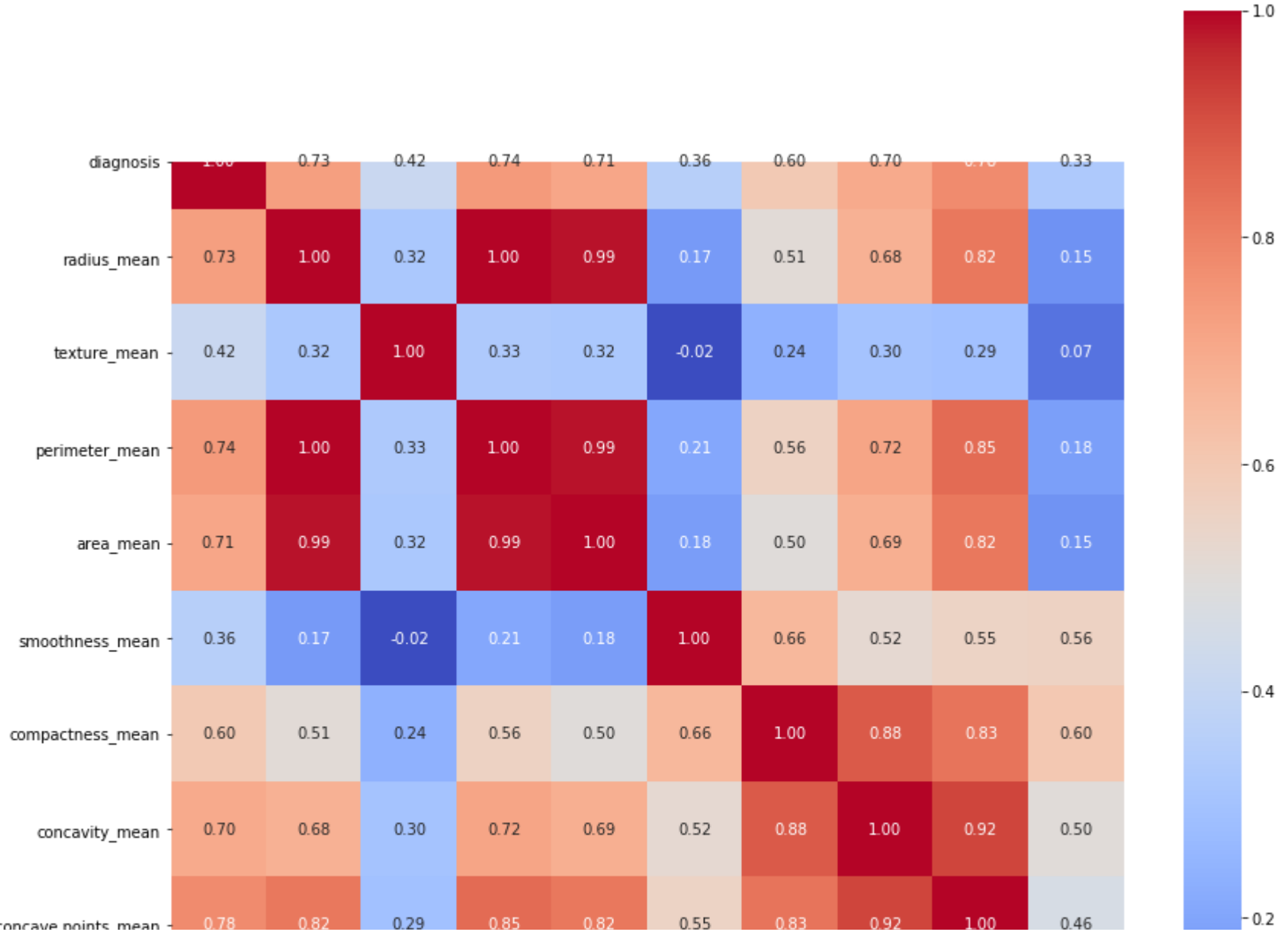
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1d67a2eb548>

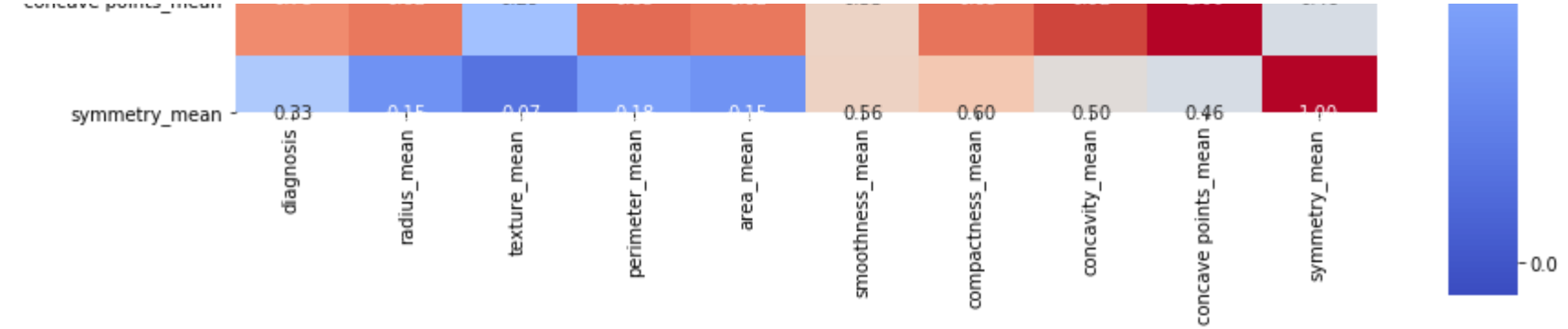



symmetry_worst
0.24
0.23
0.27
0.21
0.49
0.61
0.53
0.50
fractal_dimension_se
radius_worst
texture_worst
perimeter_worst
area_worst
smoothness_worst
compactness_worst
concavity_worst
concave points_worst
symmetry_worst
0.0

In [13]:
```
corr = data[features_mean].corr() # use to find coll
plt.figure(figsize=(14,14))
sns.heatmap(corr, cbar = True, square = True, annot= True, fmt= '.2f', annot_kws = {'size': 10}, xticklabels= fe
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1d67a93dec8>





In [14]:
```
# these variable use for prediction
prediction_var = ['texture_mean', 'perimeter_mean', 'smoothness_mean', 'compactness_mean', 'symmetry_mean']
```

In [15]:
```
# split data for training and testing purpose
train, test = train_test_split(data, test_size = 0.3)
# we can check their dimentions
print(train.shape)
print(test.shape)
```

```
(398, 33)
(171, 33)
```

In [16]:
```
# taking a training data
train_x = train[prediction_var]
# this is the output of our training data
train_y = train.diagnosis

# same we have to do for test

# taking a testing data
test_x = test[prediction_var]
# this is the output of our tasting data
test_y = test.diagnosis
```

RandomForestClassifier

In [59]:
```
''' Pre-Processing of data is done. Now Data is ready for classification Algorithmic Techiques. We can use any a

model = RandomForestClassifier(n_estimators=100) # a simple random forest model
```

In [60]:
```
model.fit(train_x, train_y) # now fir our model for training data
```

Out[60]:
```
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 [62]:
```
prediction = model.predict(test_x)  # predict for test data
prediction
# prediction will contain the predicted value by our model predicted values of dignosis
```

Out[62]:
```
array([1], dtype=int64)
```

In [20]:
```
metrics.accuracy_score(prediction, test_y)  # to check the accuracy
# here we will use accuracy measurement between our predicted value and our test output
```

Out[20]:
```
0.9239766081871345
```

Support Vector Classifier

In [21]:
```
# svm -> support  victor machine $ SVC() -> support victor classifire
model = svm.SVC()
model.fit(train_x, train_y)
prediction = model.predict(test_x)
metrics.accuracy_score(prediction, test_y)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: The default value of gamma
will change from 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicit
ly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[21]:
```
0.9064327485380117
```

In [22]:
```
prediction_var = features_mean # taking all features
```

Linear regression model

In [24]:
```
from sklearn import linear_model
```

In [25]:
```
model = linear_model.LinearRegression()
model.fit(train_x, train_y)
```

Out[25]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [52]:
```
pred = model.predict(test_x)
pred
```

Out[52]:
```
array([ 0.73551836,  0.6239801 ,  0.14278447,  0.18808503,  0.94568286,
        0.20856148,  0.74523756,  1.00244449,  0.09907784,  1.27923029,
        0.02875677,  0.2285594 , -0.10923779,  0.23228115,  0.49219128,
        0.11680444,  0.09214112, -0.18035148,  0.21873379,  0.14747194,
       -0.03045502,  0.44941615,  1.07334986,  0.26822047,  0.15741459,
       -0.01671552,  0.05860689,  0.05643039,  0.08003606,  0.01208026,
       -0.12691042,  0.53833193,  0.01847366,  0.20863071,  0.23770662,
        0.34481639,  0.82779694,  0.08472007, -0.26834088,  0.10087998,
        0.56379819, -0.26309405,  0.12537423,  0.95286059,  0.31120662,
        0.92779219,  0.49959637, -0.0463662 ,  0.16743787,  0.11562234,
        0.52102059,  0.7003929 ,  0.00223262,  0.12397521,  0.00677636,
        0.15636009, -0.23087677,  0.53618711,  0.33051315,  1.09193001,
        0.03189296,  0.03088805,  0.92973533,  0.60643714,  0.52890521,
        0.18514466,  0.11549495,  0.12816302,  0.35700503,  0.21599027,
        0.15191351,  0.18238183,  0.86266971,  0.23965605,  0.183941  ,
        1.78059878,  0.28129329,  1.05352203,  0.66119488,  1.04297487,
       -0.00273464,  0.73157229,  0.09748102, -0.040584  ,  0.58100475,
        0.08240199,  0.02988691,  0.89670472,  0.56248657, -0.15632295,
        0.29383455,  0.43235029,  0.78755316,  1.02559765,  1.22494714,
       -0.09967173,  0.17524506,  0.23552846,  0.45950369,  1.26230359,
        0.05093416,  0.03711505,  0.05540421,  0.44078744,  0.79580036,
       -0.09354941,  1.02118091,  0.82700365,  0.10974737,  0.27314509,
        0.27957629,  0.68675322,  0.71556441,  0.36020019,  0.8991227 ,
        0.38126353,  0.18382212,  0.47125345,  0.94962681,  0.80374016,
        0.80354866,  0.45556343,  0.37093529,  0.26299529,  0.15985387,
        0.13707825, -0.20605251,  0.2760116 ,  0.02999465,  0.08290128,
        0.24543631,  0.05996003,  0.84895728,  0.05726958,  0.20948086,
        0.75537846,  0.00888135,  0.08525165,  0.18228881,  0.36609158,
       -0.12891831, -0.10115366, -0.02631768,  0.38950482,  1.00216587,
       -0.13459022,  0.36743433,  0.56732347,  0.14235559, -0.07166133,
        0.12442691,  0.00807092,  0.10288115,  0.44562567,  0.10847565,
        0.51642755,  0.09850666, -0.01649626,  0.58961538,  0.20900568,
        0.54705357,  0.37756974, -0.06967953,  0.32347821,  0.38785794,
        0.88167401, -0.03809208,  0.62849473,  0.71926142,  0.31630998,
        0.20287027])
```

In [51]:
```
pre = model.predict(test_x[1:2])
pre[0]

if pre[0]< 0.5:
    print('report is nagative - 0')
else:
    print('report is positive - 1')
```

```
report is positive - 1
```

In [48]:
```
# array([0.73551836])
test_x[:3]
```

Out[48]:

	texture_mean	perimeter_mean	smoothness_mean	compactness_mean	symmetry_mean
479	19.51	109.80	0.10260	0.18930	0.2151
34	17.88	107.00	0.10400	0.15590	0.1998
395	17.18	89.75	0.08045	0.05361	0.1641

In [50]:
```
test_y[1:2]
```

Out[50]:
```
34    1
Name: diagnosis, dtype: int64
```

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