Introduction In this problem we have to use 30 different columns and we have to predict the Stage of Breast Cancer M (Malignant) and B (Bengin) This analysis has been done using Basic Machine Learning Algorithm with detailed explanation Attribute Information:

1) ID number

2) Diagnosis (M = malignant, B = benign)

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

a) radius (mean of distances from center to points on the perimeter)

b) texture (standard deviation of gray-scale values)

c) perimeter

d) area

e) smoothness (local variation in radius lengths)

f) compactness (perimeter^2 / area - 1.0)

g). concavity (severity of concave portions of the contour)

h). concave points (number of concave portions of the contour)

i). symmetry

j). fractal dimension ("coastline approximation" - 1)

3) Here 3- 32 are divided into three parts first is Mean (3-13), Stranded Error(13-23) and Worst(23-32) and each contain 10 parameter (radius, texture,area, perimeter, smoothness,compactness,concavity,concave points,symmetry and fractal dimension)

Here Mean means the means of the all cells, standard Error of all cell and worst means the worst cell

In [80]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
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

# machine learning
from sklearn.preprocessing import StandardScaler

from scipy import stats
plt.style.use("ggplot")
import warnings
warnings.filterwarnings("ignore")
from scipy import stats


import sklearn.linear_model as skl_lm
from sklearn import preprocessing
from sklearn import neighbors
from sklearn.metrics import confusion_matrix, classification_report, precision_score
from sklearn.model_selection import train_test_split


import statsmodels.api as sm
import statsmodels.formula.api as smf


# initialize some package settings
sns.set(style="whitegrid", color_codes=True, font_scale=1.3)

%matplotlib inline

import os
#%% import dataset
df = pd.read_csv("data.csv",encoding='latin1')
df.drop(['Unnamed: 32',"id"], axis=1, inplace=True)
df.diagnosis = [1 if each == "M" else 0 for each in data.diagnosis]
y = df.diagnosis.values
x_df = data.drop(['diagnosis'], axis=1)
```

In [47]:

```
df.head()
```

Out[47]:

| | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | c |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | |
| **1** | 0 | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | |
| **2** | 0 | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | |
| **3** | 0 | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | |
| **4** | 0 | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | |

5 rows × 31 columns

In [48]:

```
df.describe()
```

Out[48]:

| | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_me |
|---|---|---|---|---|---|---|
| **count** | 569.0 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.0000 |
| **mean** | 0.0 | 14.127292 | 19.289649 | 91.969033 | 654.889104 | 0.0963 |
| **std** | 0.0 | 3.524049 | 4.301036 | 24.298981 | 351.914129 | 0.0140 |
| **min** | 0.0 | 6.981000 | 9.710000 | 43.790000 | 143.500000 | 0.0526 |
| **25%** | 0.0 | 11.700000 | 16.170000 | 75.170000 | 420.300000 | 0.0863 |
| **50%** | 0.0 | 13.370000 | 18.840000 | 86.240000 | 551.100000 | 0.0958 |
| **75%** | 0.0 | 15.780000 | 21.800000 | 104.100000 | 782.700000 | 0.1053 |
| **max** | 0.0 | 28.110000 | 39.280000 | 188.500000 | 2501.000000 | 0.1634 |

8 rows × 31 columns

In [49]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   diagnosis                569 non-null    int64
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

# DATA VISUALIZATION

In [67]:

```
benign, malignant = data['diagnosis'].value_counts()
print('Number of cells labeled Benign: ', benign)
print('Number of cells labeled Malignant : ', malignant)
print('')
print('% of cells labeled Benign', round(benign / len(df) * 100, 2), '%')
print('% of cells labeled Malignant', round(malignant / len(df) * 100, 2), '%')
```

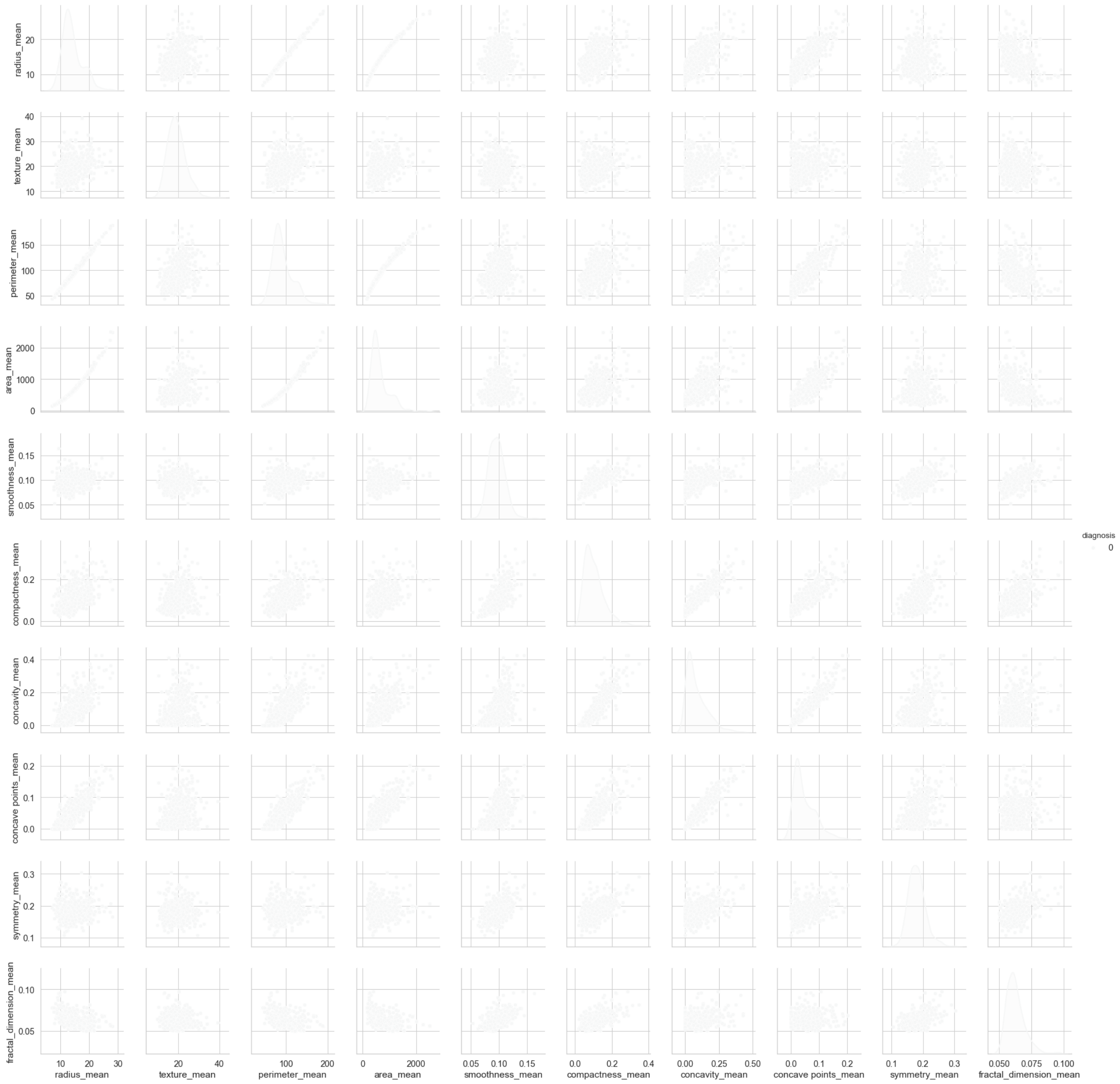
```
Number of cells labeled Benign:  357
Number of cells labeled Malignant :  212

% of cells labeled Benign 62.74 %
% of cells labeled Malignant 37.26 %
```

In [68]:

```
# generate a scatter plot matrix with the "mean" columns
cols = ['diagnosis',
        'radius_mean',
        'texture_mean',
        'perimeter_mean',
        'area_mean',
        'smoothness_mean',
        'compactness_mean',
        'concavity_mean',
        'concave points_mean',
        'symmetry_mean',
        'fractal_dimension_mean']

sns.pairplot(data=df[cols], hue='diagnosis', palette='RdBu')
```

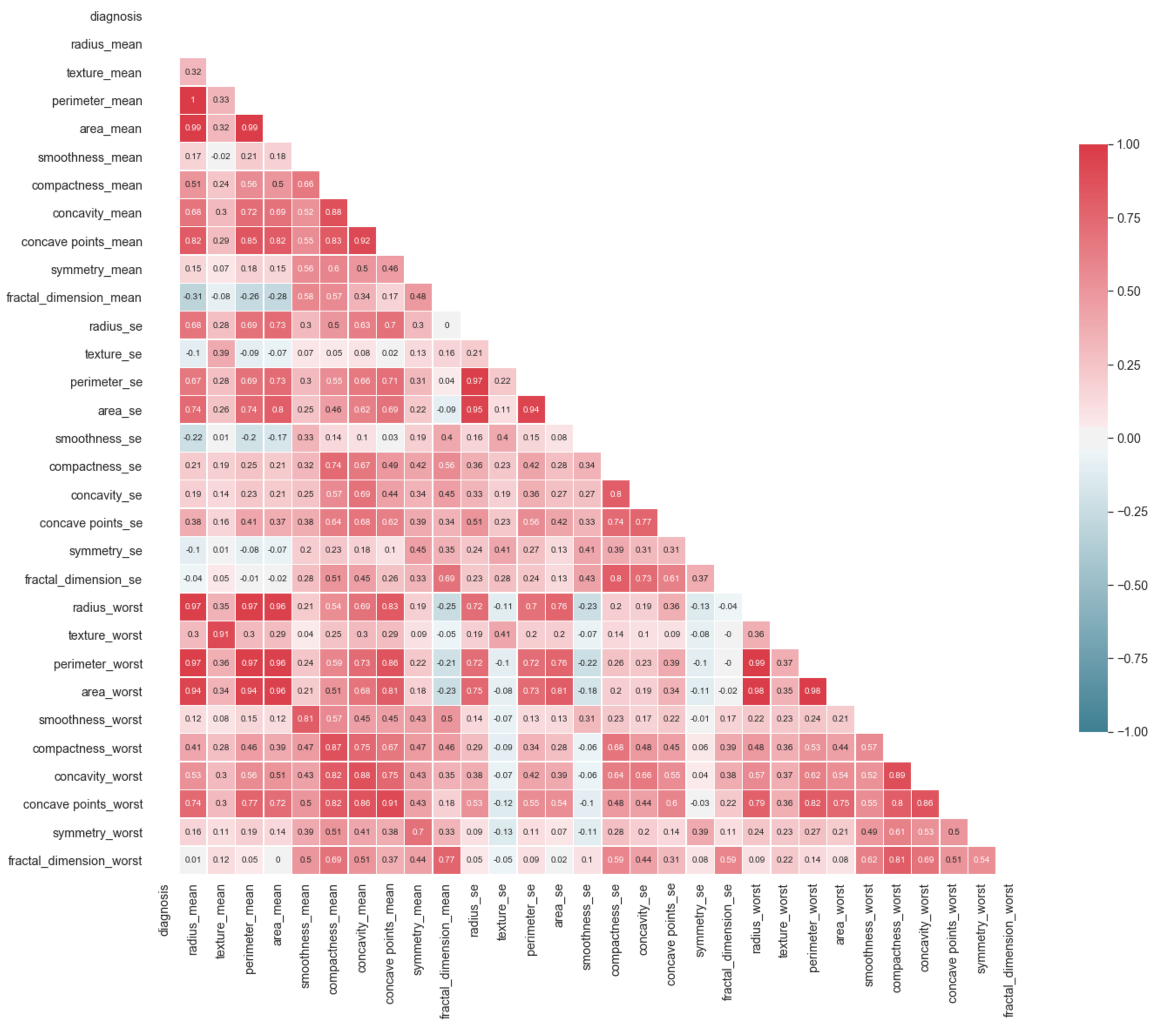
Out[68]:

```
<seaborn.axisgrid.PairGrid at 0x273e2b16e08>
```


radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
diagnosis
0

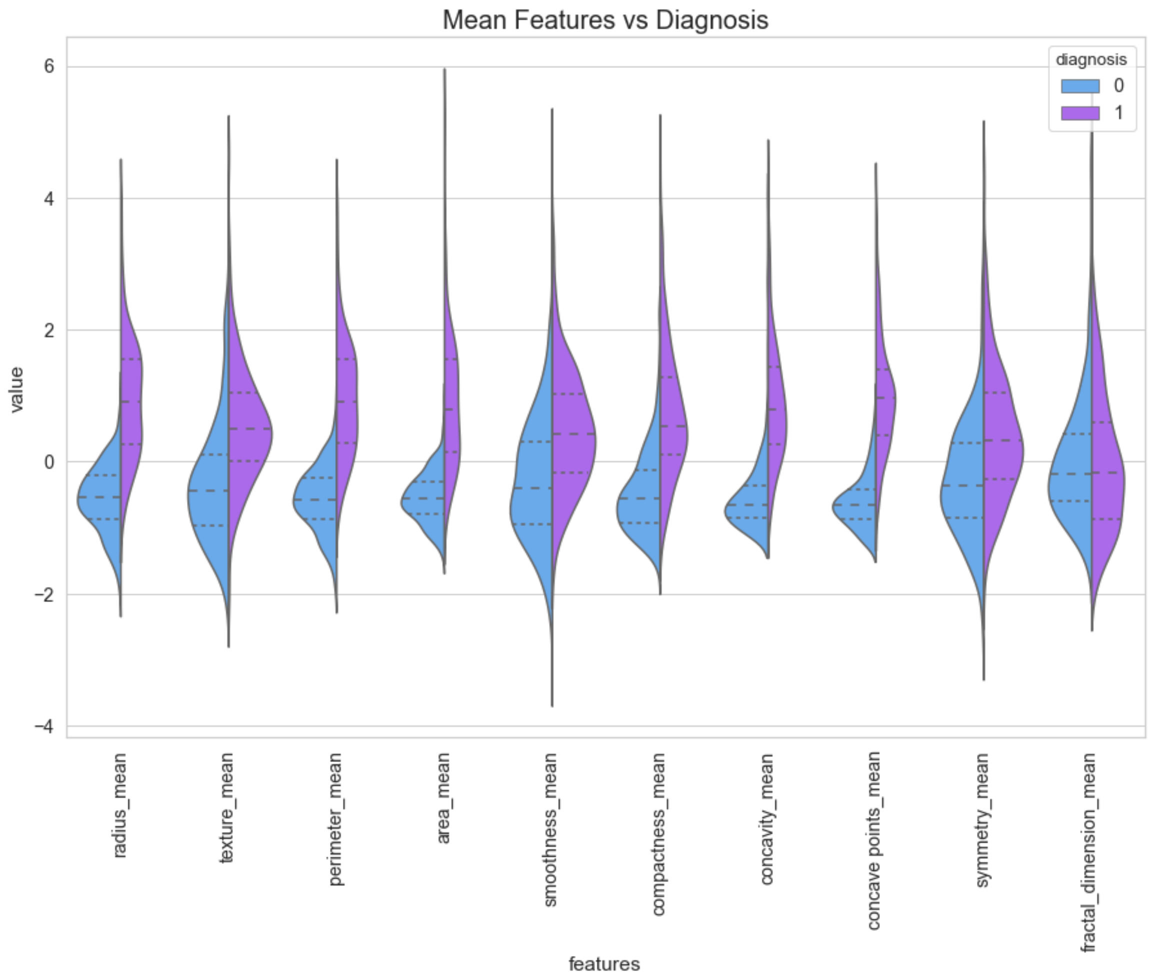
In [84]:

```python
# Draw the heatmap again, with the new correlation matrix
corr = df.corr().round(2)
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

f, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)
plt.tight_layout()
```



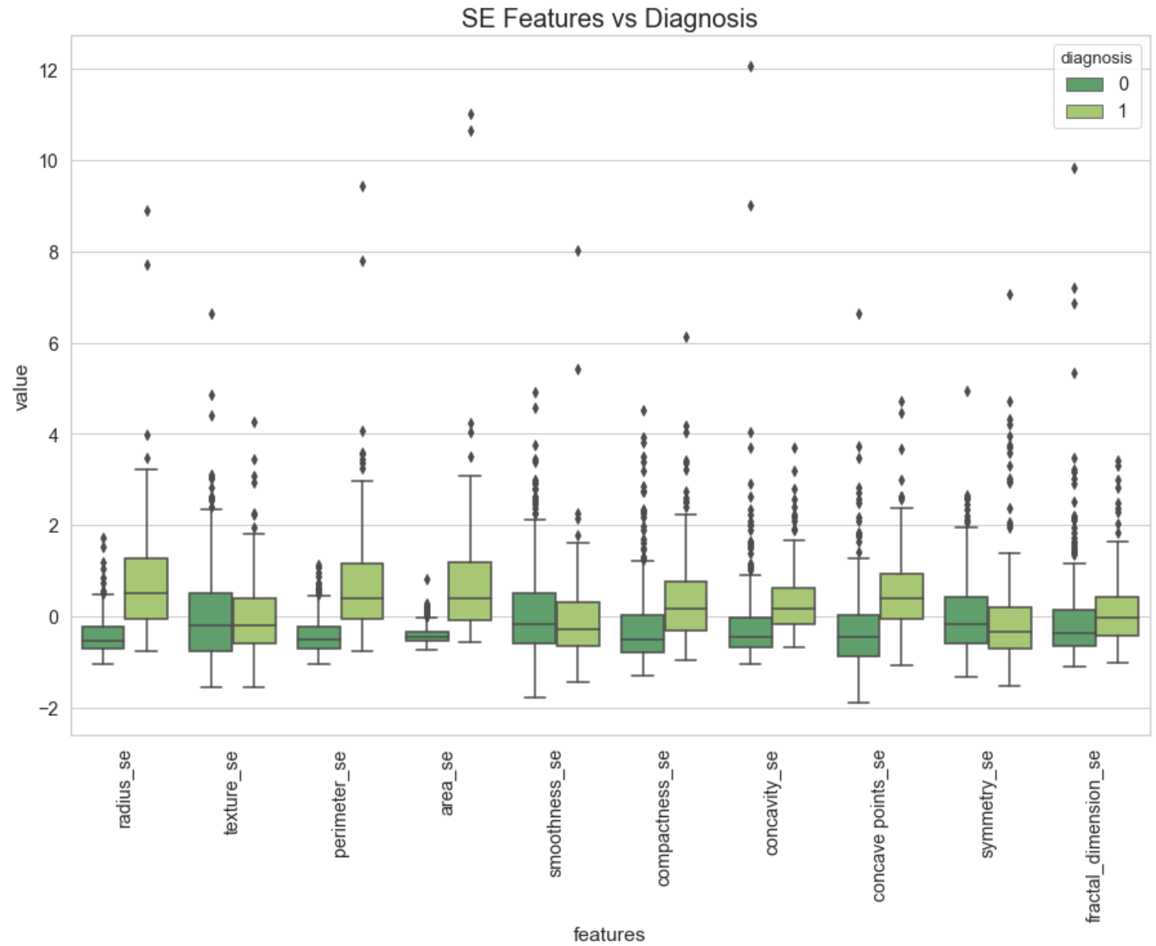
In [85]:

```
y = data['diagnosis']
x = data.drop('diagnosis', axis = 1)

x = (x - x.mean()) / (x.std())
df = pd.concat([y, x.iloc[:,0:10]], axis=1)
df = pd.melt(df, id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(15, 10))
sns.violinplot(x="features", y="value", hue="diagnosis", data=df,split=True, inner="qua
rt", palette = 'cool')
plt.title('Mean Features vs Diagnosis', fontsize = 20)
plt.xticks(rotation=90)
plt.show()
```



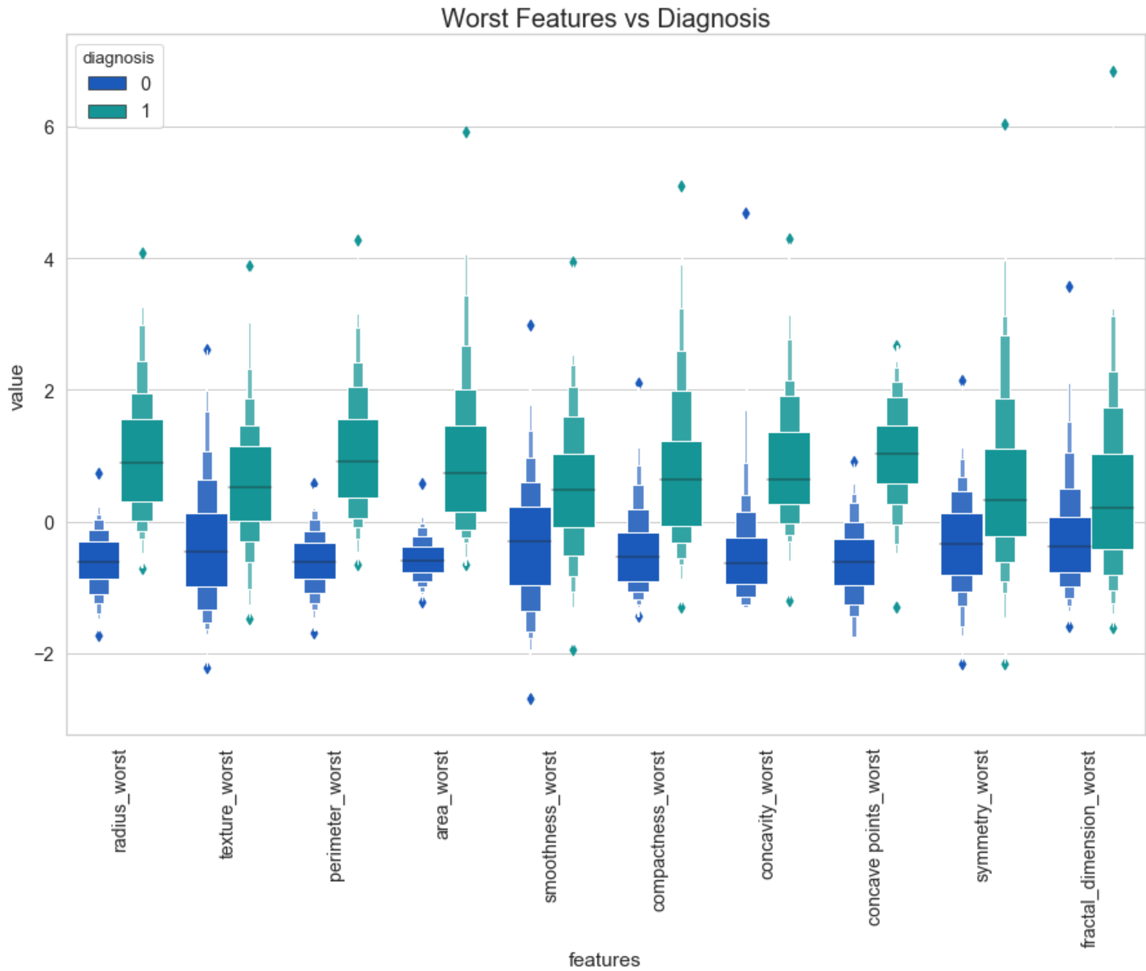
In [86]:

```
y = data['diagnosis']
x = data.drop('diagnosis', axis = 1)

x = (x - x.mean()) / (x.std())
df = pd.concat([y, x.iloc[:,10:20]], axis=1)
df = pd.melt(df, id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(15, 10))
sns.boxplot(x="features", y="value", hue="diagnosis", data=df, palette = 'summer')
plt.title('SE Features vs Diagnosis', fontsize = 20)
plt.xticks(rotation=90)
plt.show()
```



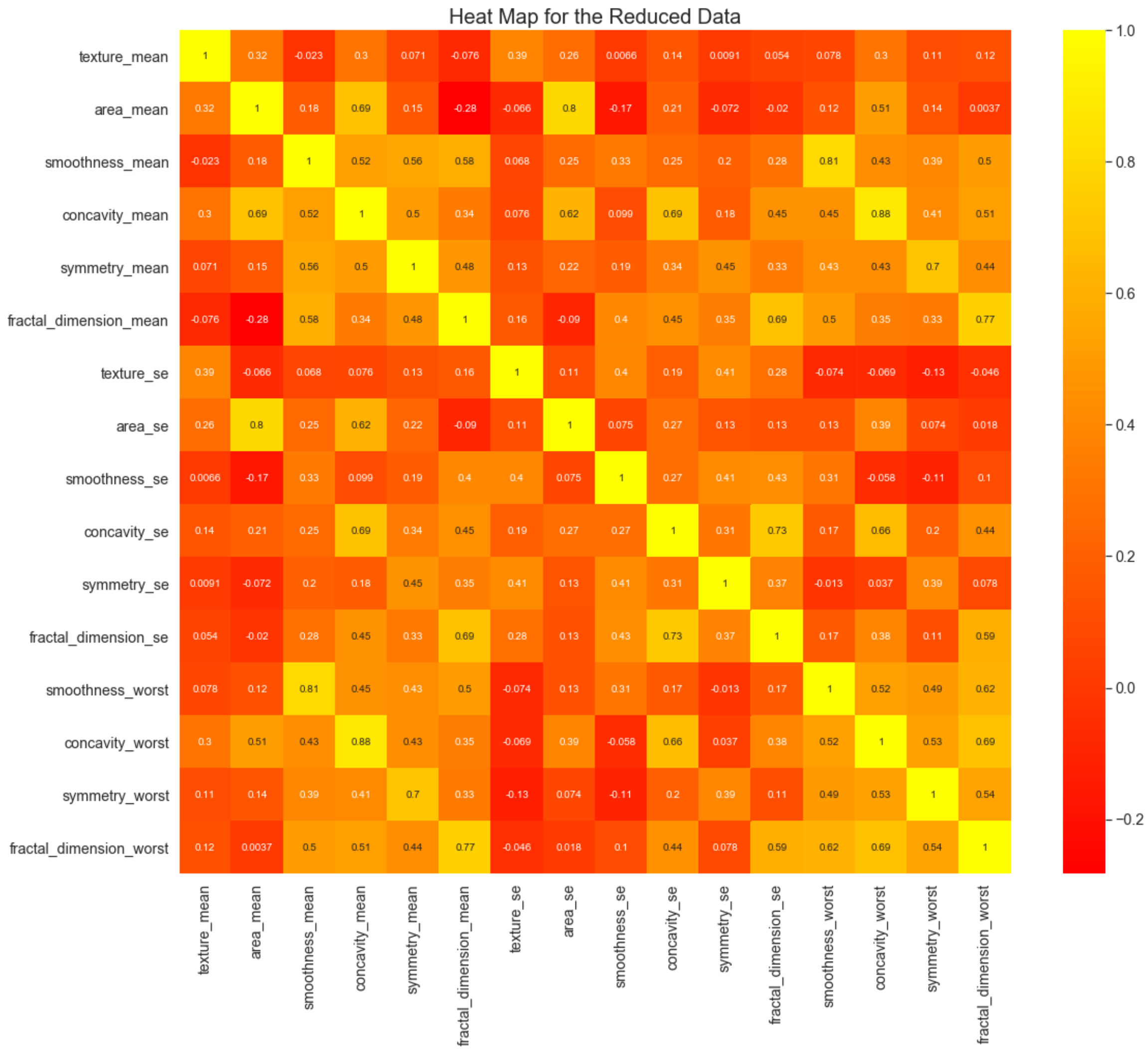
In [87]:

```
y = data['diagnosis']
x = data.drop('diagnosis', axis = 1)

x = (x - x.mean()) / (x.std())
df = pd.concat([y, x.iloc[:,20:30]], axis=1)
df = pd.melt(df, id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(15, 10))
sns.boxenplot(x="features", y="value", hue="diagnosis", data=df, palette = 'winter')
plt.title('Worst Features vs Diagnosis', fontsize = 20)
plt.xticks(rotation=90)
plt.show()
```



In [88]:

```
list_to_delete = ['perimeter_mean','radius_mean','compactness_mean','concave points_mea
n',
                  'radius_se','perimeter_se','radius_worst','perimeter_worst','compactn
ess_worst',
                  'concave points_worst','compactness_se','concave points_se','texture_
worst','area_worst']
x = x.drop(list_to_delete, axis = 1)

plt.rcParams['figure.figsize'] = (18, 15)
sns.heatmap(x.corr(), annot = True, cmap = 'autumn')
plt.title('Heat Map for the Reduced Data', fontsize = 20)
plt.show()
```




## 1- Prediction Accuracy

In [51]:

```
#we select x,y axis and we normalize our data
y = df.diagnosis.values
x_df = data.drop("diagnosis",axis=1)
x = (x_data-np.min(x_data))/(np.max(x_data)-np.min(x_data))
```

In [6]:

```
#we separate train and test data with sklearn selection model
#You can thnk this x_train for learn and y_train is answer of x_train and finally we te
sting our data with x_test andy_test
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y)
```

In [7]:

```
print("xtrain:{}".format((x_train).shape))
print("y_train:{}".format((y_train).shape))
print("xtest:{}".format((x_test).shape))
print("ytest:{}".format((y_test).shape))
```

```
xtrain:(426, 30)
y_train:(426,)
xtest:(143, 30)
ytest:(143,)
```

In [8]:

```
#We did
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression(max_iter = 200)
lgr.fit(x_train,y_train)
print("our accuracy is:{}".format(lgr.score(x_test,y_test)))
```

```
our accuracy is:0.972027972027972
```

In [9]:

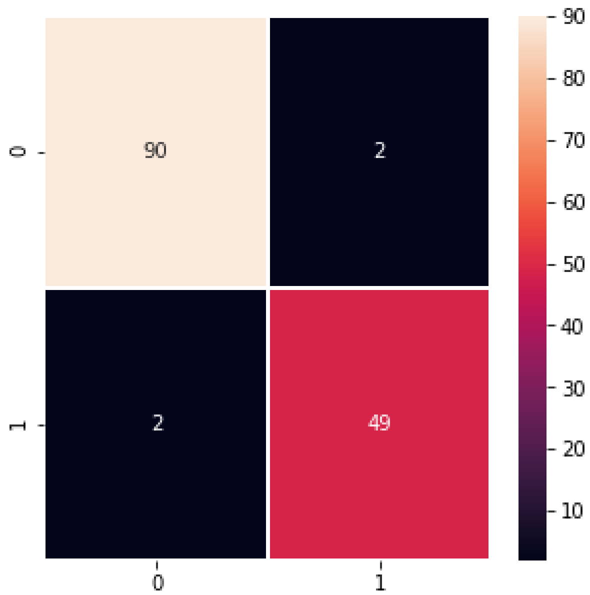
```
#We can evaluate our model so and we have y_predict and y_true(y_test)
from sklearn.metrics import confusion_matrix
y_true = y_test
y_pred = lgr.predict(x_test) #Predict data for eveluating
cm = confusion_matrix(y_true,y_pred)
```

In [10]:

```
#We draw heatmap for showing confusion matrix
import matplotlib.pyplot as plt
import seaborn as sns
f,ax = plt.subplots(figsize = (5,5))
sns.heatmap(cm,annot = True,linewidth = 1,fmt =".0f",ax = ax)
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x273e1cdd9c8>
```




## 2- Prediction Accuracy ( using gradient ,cost ,iteration)

In [52]:

```
x = (x_data -np.min(x_data))/(np.max(x_data)-np.min(x_data)).values
```

In [26]:

```
# %%train test split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.15, random_state=
42)

x_train = x_train.T
x_test = x_test.T
y_train = y_train.T
y_test = y_test.T

print("x train: ",x_train.shape)
print("x test: ",x_test.shape)
print("y train: ",y_train.shape)
print("y test: ",y_test.shape)
```

```
x train:  (30, 483)
x test:  (30, 86)
y train:  (483,)
y test:  (86,)
```

In [27]:

```
# %%initialize
# lets initialize parameters
# So what we need is dimension 4096 that is number of pixels as a parameter for our ini
tialize method(def)
def initialize_weights_and_bias(dimension):
    w = np.full((dimension,1),0.01)
    b = 0.0
    return w, b
```

In [28]:

```
#%% sigmoid
# calculation of z
#z = np.dot(w.T,x_train)+b
def sigmoid(z):
    y_head = 1/(1+np.exp(-z))
    return y_head
#y_head = sigmoid(5)
```

In [29]:

```
#%% forward and backward
# In backward propagation we will use y_head that found in forward progation
# Therefore instead of writing backward propagation method, lets combine forward propag
ation and backward propagation
def forward_backward_propagation(w,b,x_train,y_train):
    # forward propagation
    z = np.dot(w.T,x_train) + b
    y_head = sigmoid(z)
    loss = -y_train*np.log(y_head)-(1-y_train)*np.log(1-y_head)
    cost = (np.sum(loss))/x_train.shape[1]      # x_train.shape[1]  is for scaling
    # backward propagation
    derivative_weight = (np.dot(x_train,((y_head-y_train).T)))/x_train.shape[1] # x_tra
in.shape[1]  is for scaling
    derivative_bias = np.sum(y_head-y_train)/x_train.shape[1]                 # x_trai
n.shape[1]  is for scaling
    gradients = {"derivative_weight": derivative_weight,"derivative_bias": derivative_b
ias}
    return cost,gradients
```

In [30]:

```
#%%# Updating(learning) parameters
def update(w, b, x_train, y_train, learning_rate,number_of_iterarion):
    cost_list = []
    cost_list2 = []
    index = []
    # updating(learning) parameters is number_of_iterarion times
    for i in range(number_of_iterarion):
        # make forward and backward propagation and find cost and gradients
        cost,gradients = forward_backward_propagation(w,b,x_train,y_train)
        cost_list.append(cost)
        # lets update
        w = w - learning_rate * gradients["derivative_weight"]
        b = b - learning_rate * gradients["derivative_bias"]
        if i % 10 == 0:
            cost_list2.append(cost)
            index.append(i)
            print ("Cost after iteration %i: %f" %(i, cost))
    # we update(learn) parameters weights and bias
    parameters = {"weight": w,"bias": b}
    plt.plot(index,cost_list2)
    plt.xticks(index,rotation='vertical')
    plt.xlabel("Number of Iterarion")
    plt.ylabel("Cost")
    plt.show()
    return parameters, gradients, cost_list
```

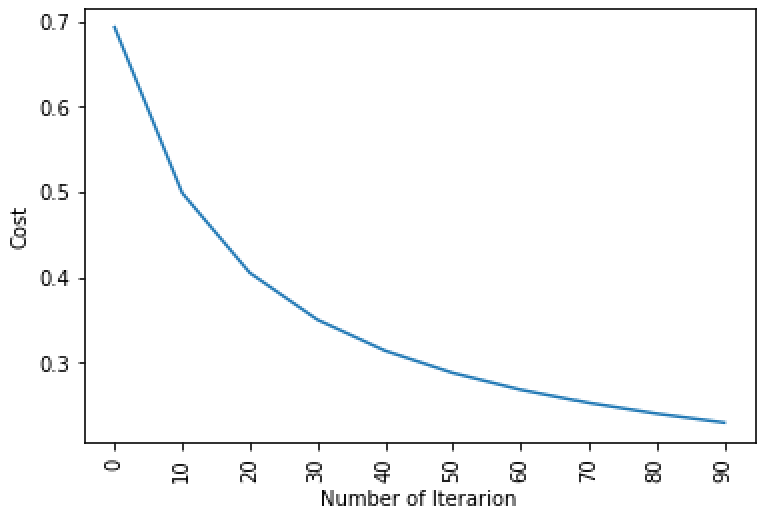
In [31]:

```
#%%  # prediction
def predict(w,b,x_test):
    # x_test is a input for forward propagation
    z = sigmoid(np.dot(w.T,x_test)+b)
    Y_prediction = np.zeros((1,x_test.shape[1]))
    # if z is bigger than 0.5, our prediction is sign one (y_head=1),
    # if z is smaller than 0.5, our prediction is sign zero (y_head=0),
    for i in range(z.shape[1]):
        if z[0,i]<= 0.5:
            Y_prediction[0,i] = 0
        else:
            Y_prediction[0,i] = 1

    return Y_prediction
# predict(parameters["weight"],parameters["bias"],x_test)
```

In [32]:

```
# %%
def logistic_regression(x_train, y_train, x_test, y_test, learning_rate ,  num_iteratio
ns):
    # initialize
    dimension =  x_train.shape[0]  # that is 4096
    w,b = initialize_weights_and_bias(dimension)
    # do not change learning rate
    parameters, gradients, cost_list = update(w, b, x_train, y_train, learning_rate,num
_iterations)

    y_prediction_test = predict(parameters["weight"],parameters["bias"],x_test)
    y_prediction_train = predict(parameters["weight"],parameters["bias"],x_train)

    # Print train/test Errors
    print("train accuracy: {} %".format(100 - np.mean(np.abs(y_prediction_train - y_tra
in)) * 100))
    print("test accuracy: {} %".format(100 - np.mean(np.abs(y_prediction_test - y_test
)) * 100))

logistic_regression(x_train, y_train, x_test, y_test,learning_rate = 1, num_iterations
= 100)
```

```
Cost after iteration 0: 0.692836
Cost after iteration 10: 0.498576
Cost after iteration 20: 0.404996
Cost after iteration 30: 0.350059
Cost after iteration 40: 0.313747
Cost after iteration 50: 0.287767
Cost after iteration 60: 0.268114
Cost after iteration 70: 0.252627
Cost after iteration 80: 0.240036
Cost after iteration 90: 0.229543
```




```
train accuracy: 94.40993788819875 %
test accuracy: 94.18604651162791 %
```

In [33]:

```
# sklearn
from sklearn import linear_model
logreg = linear_model.LogisticRegression(random_state = 42,max_iter= 150)
print("test accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_test.T, y_te
st.T)))
print("train accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_train.T, y_
train.T)))
```

```
test accuracy: 0.9767441860465116
train accuracy: 0.968944099378882
```

# 3- Prediction Accuracy

In [70]:

```
# label encoding of the dependent variable
# importing label encoder
from sklearn.preprocessing import LabelEncoder
# performing label encoding
le = LabelEncoder()
y= le.fit_transform(y)
```

In [71]:

```
#splitting the dataset into training and testing sets

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_stat
e = 16)

print("Shape of x_train :", x_train.shape)
print("Shape of y_train :", y_train.shape)
print("Shape of x_test :", x_test.shape)
print("Shape of y_test :", y_test.shape)
```

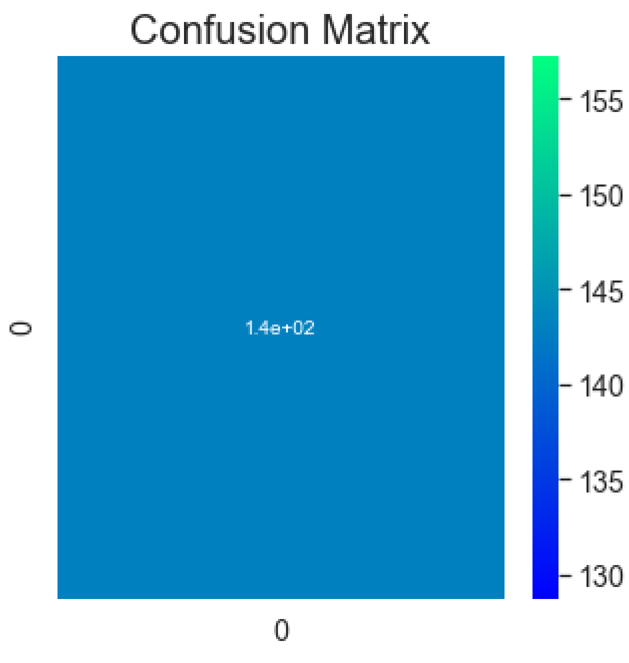
```
Shape of x_train : (426, 30)
Shape of y_train : (426,)
Shape of x_test : (143, 30)
Shape of y_test : (143,)
```

Rain Forest

In [89]:

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

# creating a model
model = RandomForestClassifier(n_estimators = 400, max_depth = 10)

# feeding the training set into the model
model.fit(x_train, y_train)

# predicting the test set results
y_pred = model.predict(x_test)

# Calculating the accuracies
print("Training accuracy :", model.score(x_train, y_train))
print("Testing accuarcy :", model.score(x_test, y_test))

# classification report
cr = classification_report(y_test, y_pred)
print(cr)

# confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm, annot = True, cmap = 'winter')
plt.title('Confusion Matrix', fontsize = 20)
plt.show()
```

```
Training accuracy : 1.0
Testing accuarcy : 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       143

    accuracy                           1.00       143
   macro avg       1.00      1.00      1.00       143
weighted avg       1.00      1.00      1.00       143
```



In [76]:

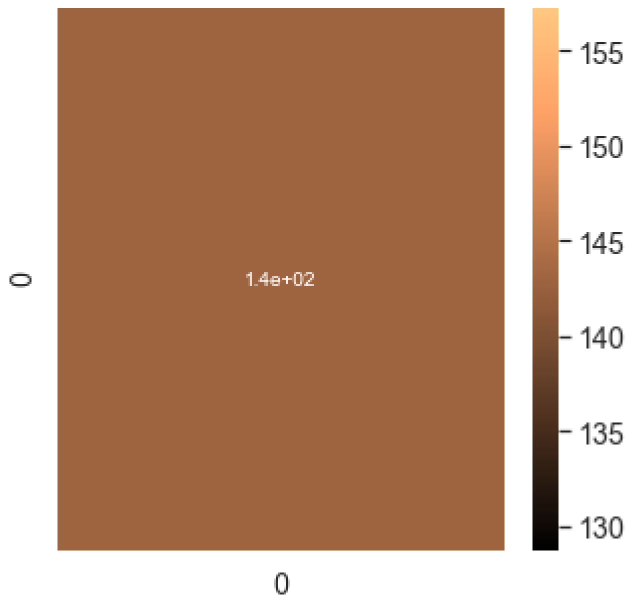
```
import warnings
warnings.filterwarnings('ignore')

from sklearn.feature_selection import RFECV

# The "accuracy" scoring is proportional to the number of correct classifications
model = RandomForestClassifier()
rfecv = RFECV(estimator = model, step = 1, cv = 5, scoring = 'accuracy')
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 : 1
Best features : Index(['fractal_dimension_worst'], dtype='object')
```

In [77]:

```
y_pred = rfecv.predict(x_test)

print("Training Accuracy :", rfecv.score(x_train, y_train))
print("Testing Accuracy :", rfecv.score(x_test, y_test))

cm = confusion_matrix(y_pred, y_test)
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm, annot = True, cmap = 'copper')
plt.show()
```

```
Training Accuracy : 1.0
Testing Accuracy : 1.0
```




In [ ]: