# Introduction

- HW Example

In [12]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

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
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

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

# Any results you write to the current directory are saved as output.
```

In [14]:

```
data.head()
```

Out[14]:

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

5 rows × 31 columns

In [15]:

```
data.describe()
```

Out[15]:

| | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_m |
|---|---|---|---|---|---|---|
| **count** | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000 |
| **mean** | 0.372583 | 14.127292 | 19.289649 | 91.969033 | 654.889104 | 0.096 |
| **std** | 0.483918 | 3.524049 | 4.301036 | 24.298981 | 351.914129 | 0.014 |
| **min** | 0.000000 | 6.981000 | 9.710000 | 43.790000 | 143.500000 | 0.052 |
| **25%** | 0.000000 | 11.700000 | 16.170000 | 75.170000 | 420.300000 | 0.086 |
| **50%** | 0.000000 | 13.370000 | 18.840000 | 86.240000 | 551.100000 | 0.095 |
| **75%** | 1.000000 | 15.780000 | 21.800000 | 104.100000 | 782.700000 | 0.105 |
| **max** | 1.000000 | 28.110000 | 39.280000 | 188.500000 | 2501.000000 | 0.163 |

8 rows × 31 columns

In [16]:

```
data.isnull().sum().sum()
```

Out[16]:

```
0
```

In [2]:

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

In [3]:

```
# %%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 [4]:

```
# %%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 [5]:

```
#%% 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 [6]:

```
#%% 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 [7]:

```
#%%# 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
```

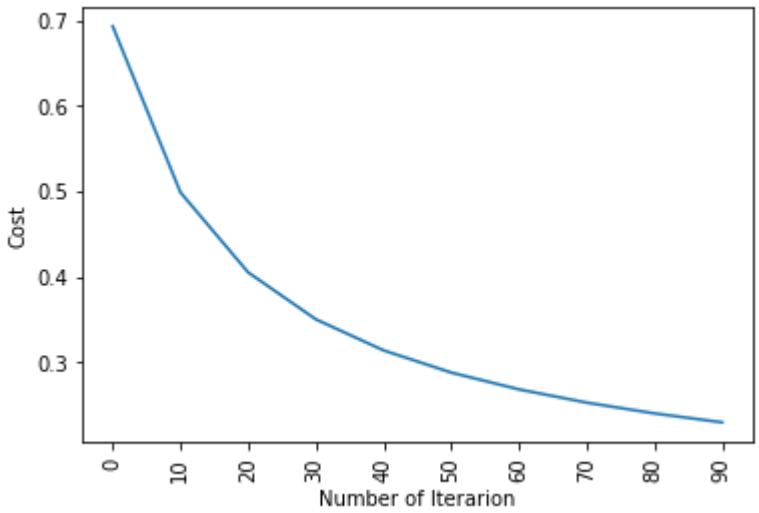
In [8]:

```
#%%  # 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 [9]:

```
# %%
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 [10]:

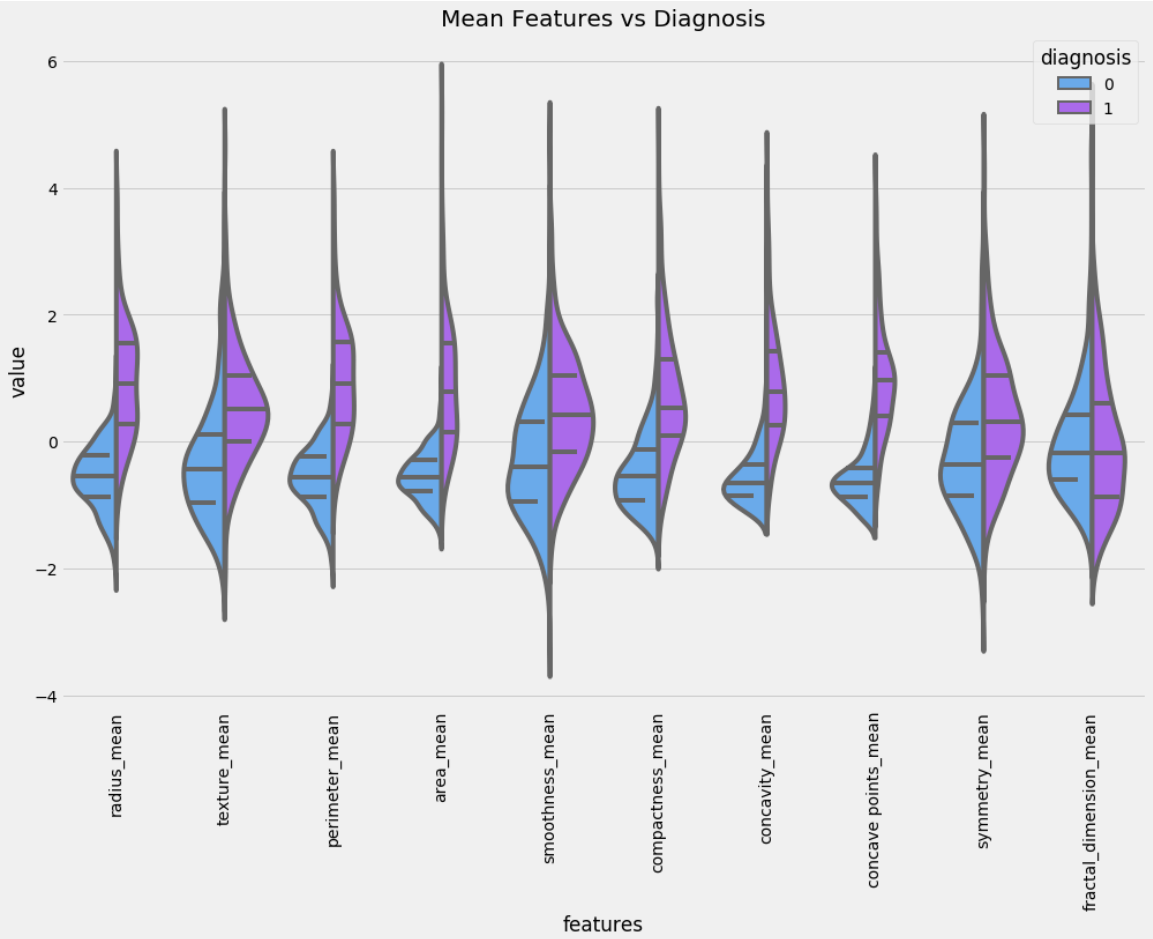
```
# sklearn
from sklearn import linear_model
logreg = linear_model.LogisticRegression(random_state = 42,max_iter= 150)
print("test accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_test.T, y_te
st.T)))
print("train accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_train.T,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

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

# Data Visualization

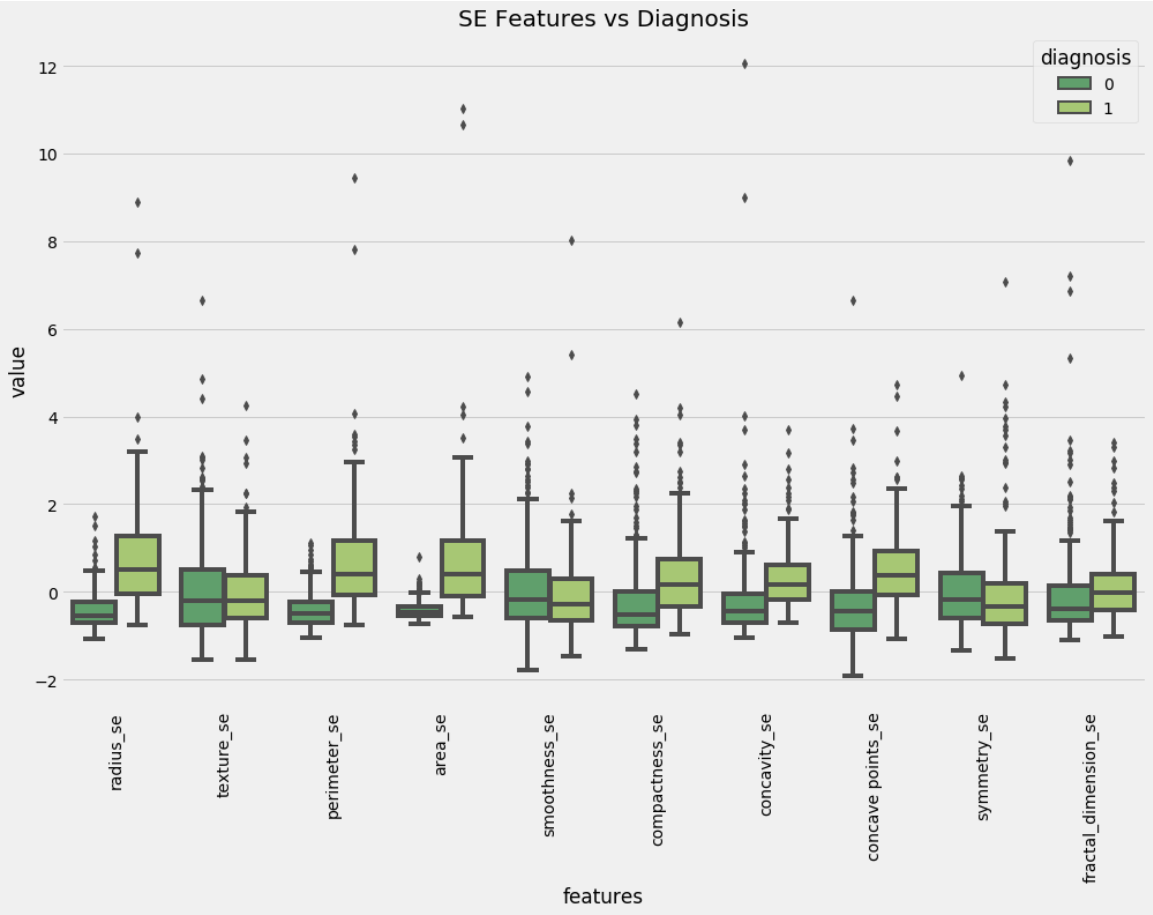
In [23]:

```
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="quart", palette = 'cool')
plt.title('Mean Features vs Diagnosis', fontsize = 20)
plt.xticks(rotation=90)
plt.show()
```



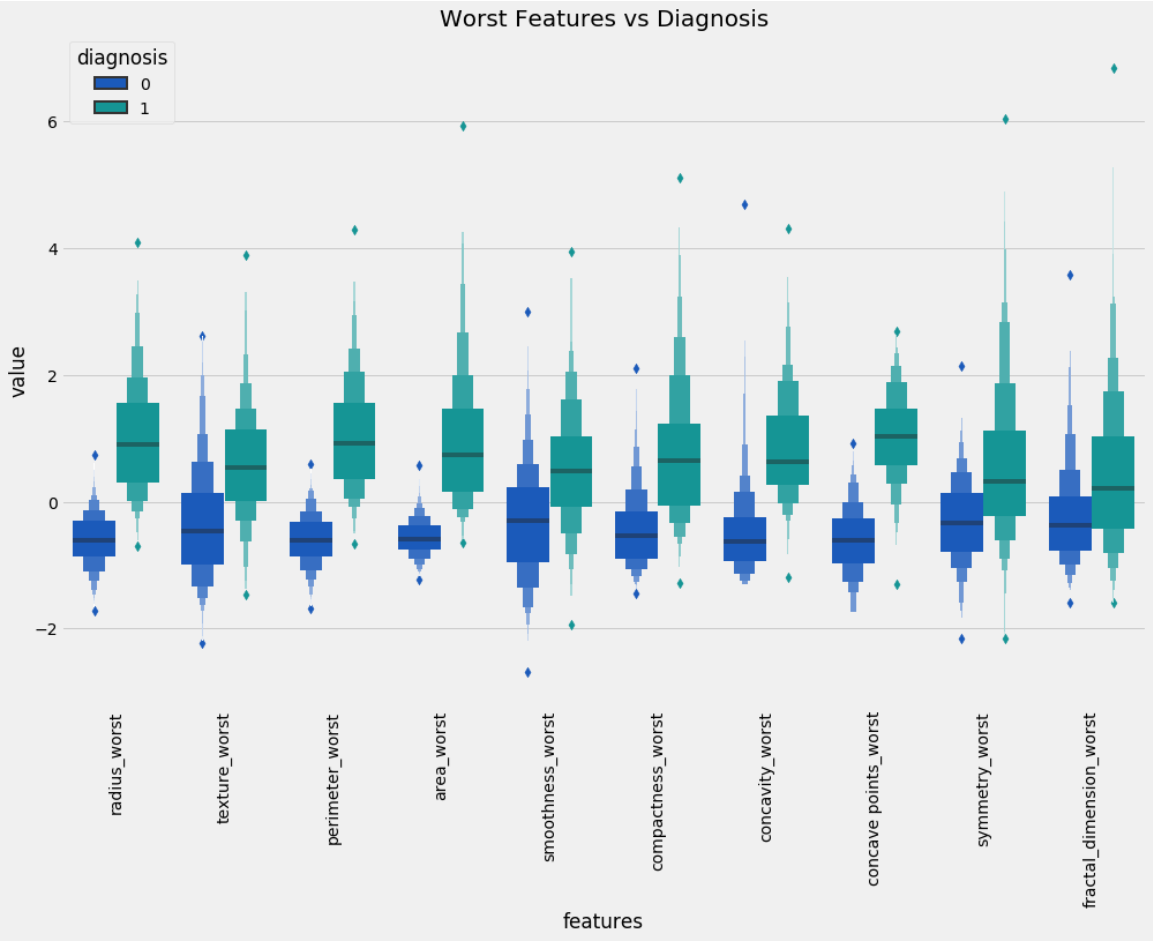
In [17]:

```
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()
```



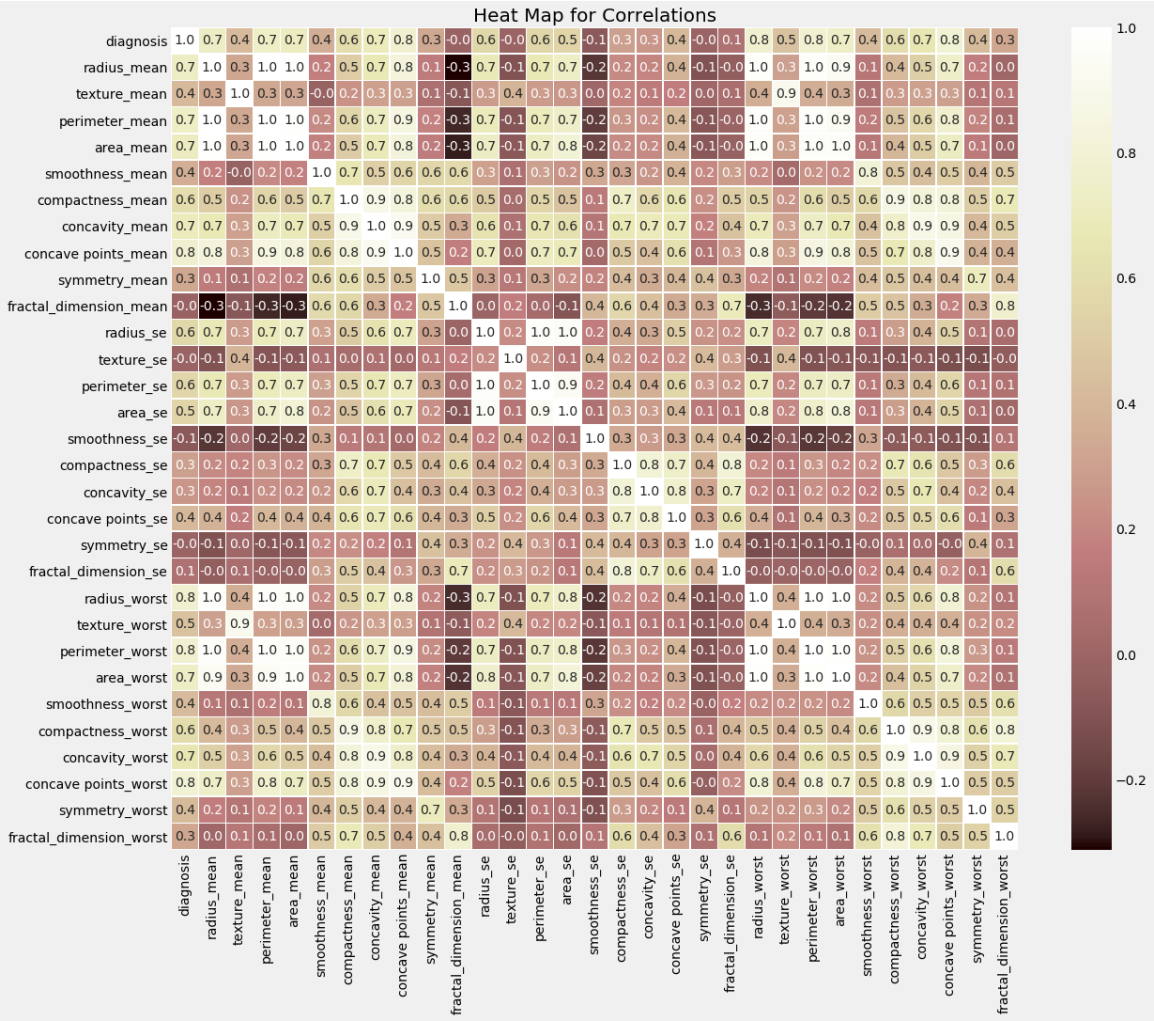
In [18]:

```
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()
```



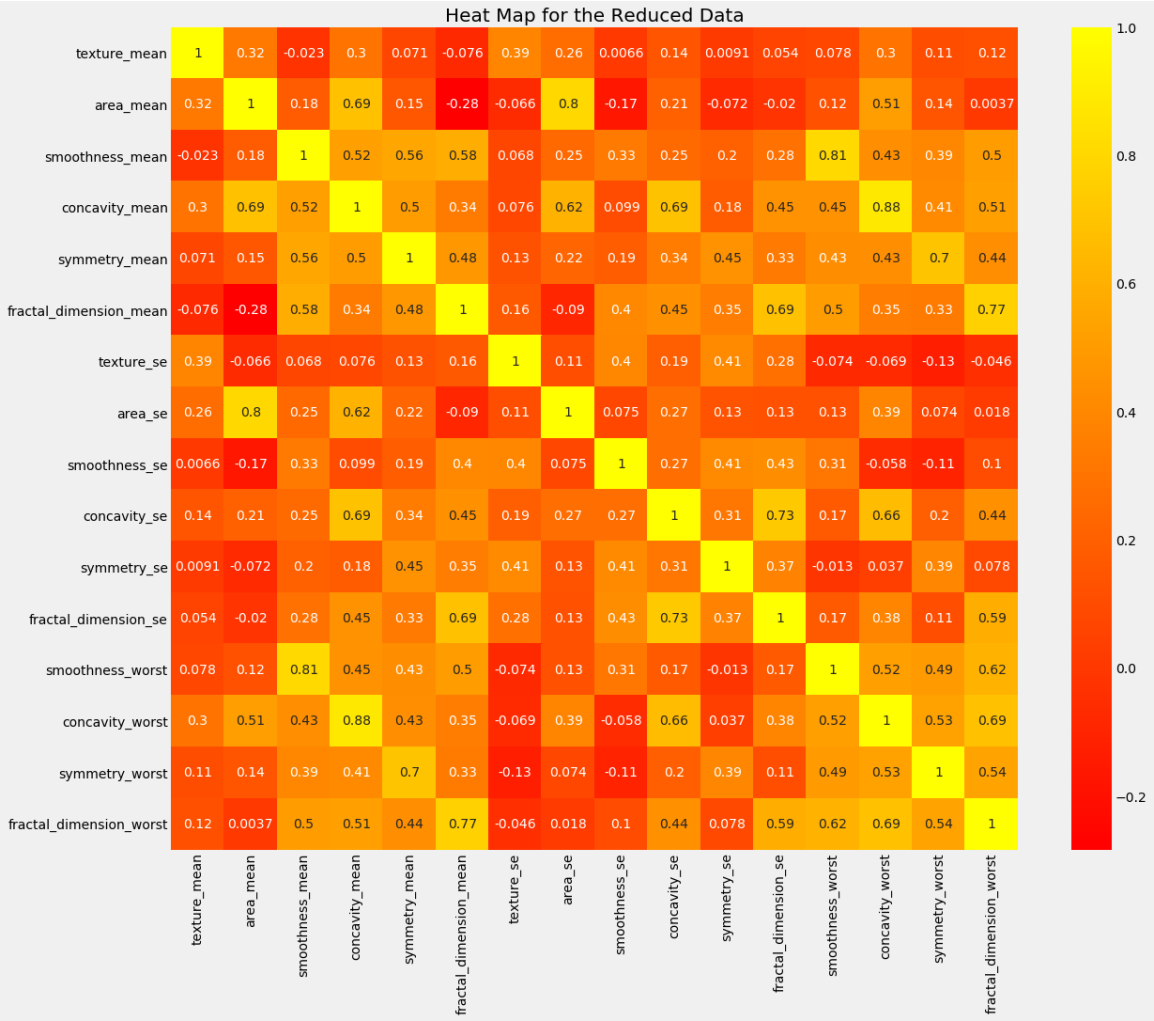
In [19]:

```
plt.rcParams['figure.figsize'] = (18, 15)

sns.heatmap(data.corr(), cmap = 'pink', annot = True, linewidths = 0.5, fmt = '.1f')
plt.title('Heat Map for Correlations', fontsize = 20)
plt.show()
```



In [20]:

```
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()
```




# Data preprocessing

In [24]:

```
# 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 [25]:

```
#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,)
```

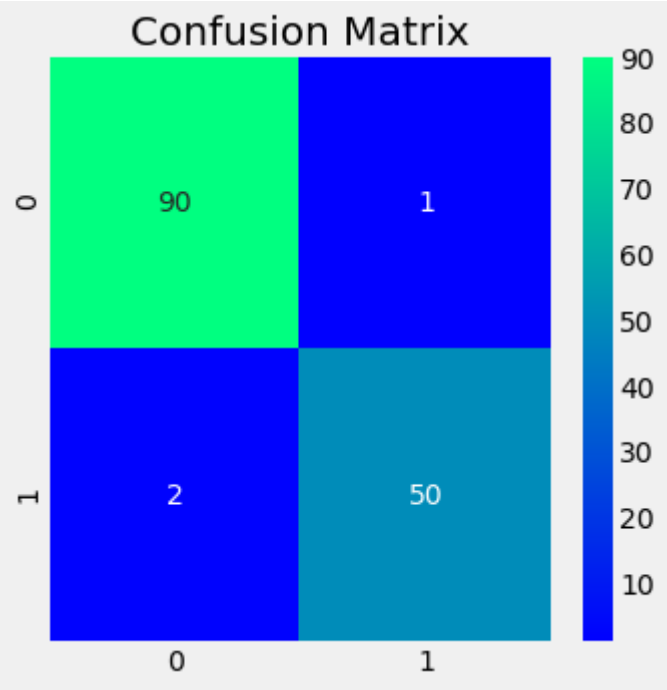
# Modelling

Random Forest

In [26]:

```
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 : 0.9790209790209791
              precision    recall  f1-score   support

           0       0.98      0.99      0.98        91
           1       0.98      0.96      0.97        52

    accuracy                           0.98       143
   macro avg       0.98      0.98      0.98       143
weighted avg       0.98      0.98      0.98       143
```




# Using RFECV

In [27]:

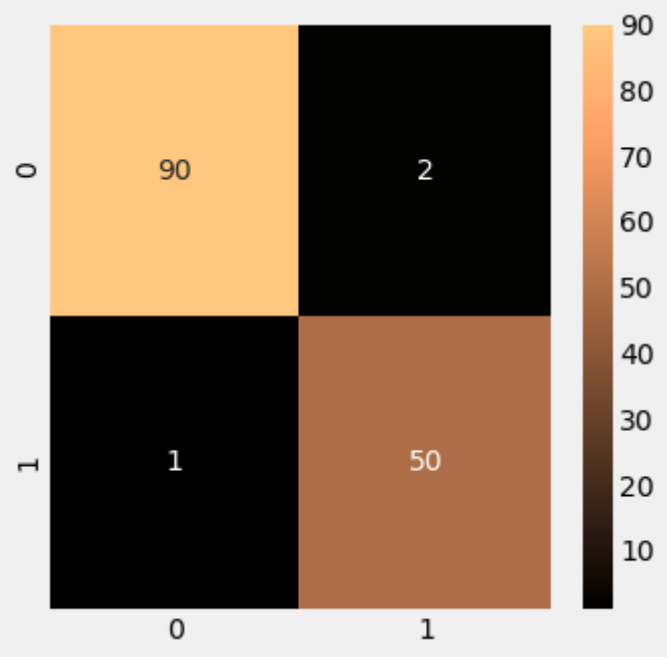
```
import warnings
warnings.filterwarnings('ignore')

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 : 15
Best features : Index(['radius_mean', 'texture_mean', 'perimeter_mean', 'a
rea_mean',
       'concavity_mean', 'concave points_mean', 'area_se', 'radius_worst',
       'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_wors
t',
       'concavity_worst', 'concave points_worst', 'symmetry_worst'],
      dtype='object')
```

In [28]:

```
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 : 0.9790209790209791
```




In [ ]: