Visualization

In [2]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson
```

Descriptive Statistics

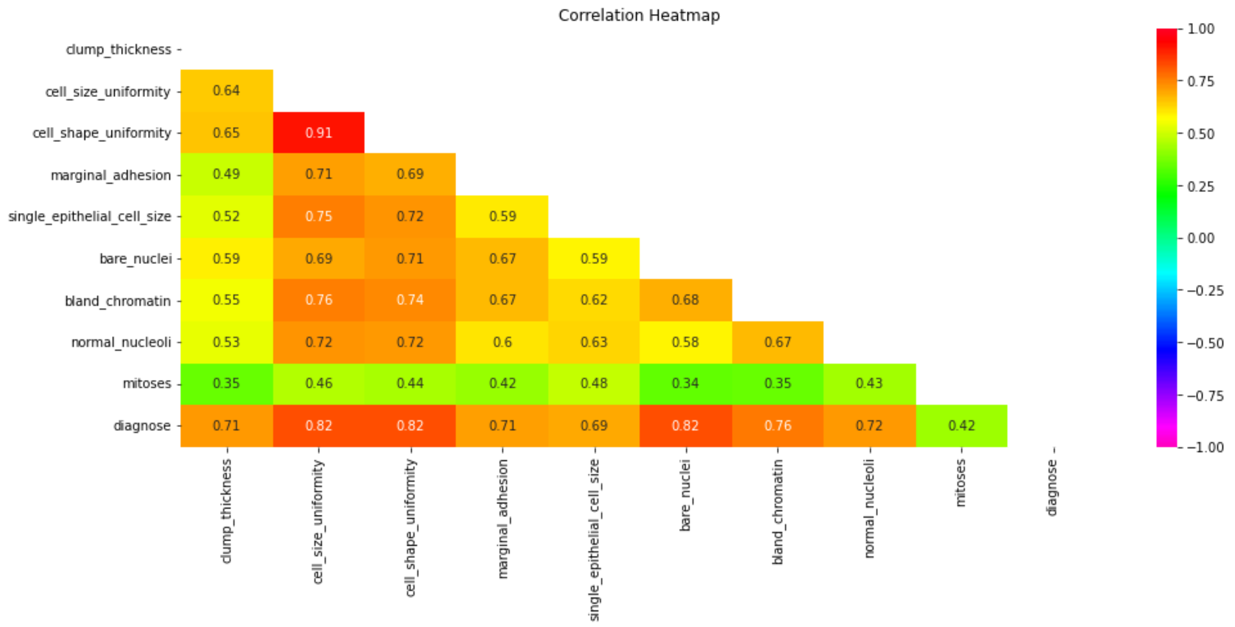
In [4]:
```
df=pd.read_csv('breast_cancer.csv')
df.describe()
```

Out[4]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_epithelia
count	683.000000	683.000000	683.000000	683.000000	6
mean	4.442167	3.150805	3.215227	2.830161	
std	2.820761	3.065145	2.988581	2.864562	
min	1.000000	1.000000	1.000000	1.000000	
25%	2.000000	1.000000	1.000000	1.000000	
50%	4.000000	1.000000	1.000000	1.000000	
75%	6.000000	5.000000	5.000000	4.000000	
max	10.000000	10.000000	10.000000	10.000000	

Correlation Heatmap

In [5]:
```
plt.figure(figsize=(16, 6))

# define the mask to set the values in the upper triangle to True
mask = np.triu(np.ones_like(df.corr(), dtype=bool))

heatmap = sns.heatmap(df.corr(), vmin=-1, vmax=1, mask=mask, annot=True, cmap='gist_ra

heatmap.set_title('Correlation Heatmap');
```


Correlation Heatmap
clump_thickness
cell_size_uniformity
cell_shape_uniformity
marginal_adhesion
single_epithelial_cell_size
bare_nuclei
bland_chromatin
normal_nucleoli
mitoses
diagnose
0.64
0.65 0.91
0.49 0.71 0.69
0.52 0.75 0.72 0.59
0.59 0.69 0.71 0.67 0.59
0.55 0.76 0.74 0.67 0.62 0.68
0.53 0.72 0.72 0.6 0.63 0.58 0.67
0.35 0.46 0.44 0.42 0.48 0.34 0.35 0.43
0.71 0.82 0.82 0.71 0.69 0.82 0.76 0.72 0.42
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

In [2]:

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
```

In [3]:

```
df=pd.read_csv(r'C:\Users\ajsru\OneDrive\Desktop\SCM516\breast_cancer.csv')
```

In [4]:

```
df.head(5)
```

Out[4]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_epithelial_cell_size	bare_nuclei	blan
0	5	1	1	1	2	1	
1	5	4	4	5	7	10	
2	3	1	1	1	2	2	
3	6	8	8	1	3	4	
4	4	1	1	3	2	1	

In [5]:

```
y=df.loc[:,'diagnose']
```

In [6]:

```
X=df.drop('diagnose', axis=1)
```

In [7]:

```
Xtrain, Xtest, Ytrain, Ytest=train_test_split(X,y, test_size=0.3, random_state=0)
```

In [8]:

```
Xtrain;
```

In [9]:

```
type(Xtrain)
```

Out[9]:

```
pandas.core.frame.DataFrame
```

In [10]:

```
np.shape(Xtrain)
```

Out[10]:

```
(478, 9)
```

In [11]:

```
model=GaussianNB()
```

In [12]:

```
model.fit(Xtrain,Ytrain)
```

Out[12]:

```
▾ GaussianNB
GaussianNB()
```

In [13]:

```
Xtest
```

Out[13]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_epithelial_cell_size	bare_nuclei	bla
113	1	1	1	1	2	5	
378	3	1	1	1	2	1	
303	5	5	5	2	5	10	
504	4	7	8	3	4	10	
301	1	1	1	1	2	1	
...	...	...	...	...	...	...	
521	5	1	1	1	2	1	
647	1	1	3	1	2	1	
503	3	1	2	2	2	1	
498	3	1	1	1	1	1	
293	10	8	4	4	4	10	

205 rows × 9 columns

In [14]:

```
ypred=model.predict(Xtest)
```

In [15]:

```
ypred
```

Out[15]:

```
array([0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1], dtype=int64)
```

In [16]:

```
Ytest
```

Out[16]:

```
113    0
378    0
303    1
504    1
301    0
      ..
```

```
521    0
647    0
503    0
498    0
293    1
Name: diagnose, Length: 205, dtype: int64
```

In [17]:

```
model.score(Xtrain,Ytrain)
```

Out[17]:

```
0.9707112970711297
```

In [18]:

```
model.score(Xtest,Ytest)
```

Out[18]:

```
0.9463414634146341
```

In [19]:

```
cm=confusion_matrix(ypred, Ytest)
cm
```

Out[19]:

```
array([[121,   2],
       [  9,  73]], dtype=int64)
```

In [20]:

```
print(classification_report(ypred, Ytest))
```

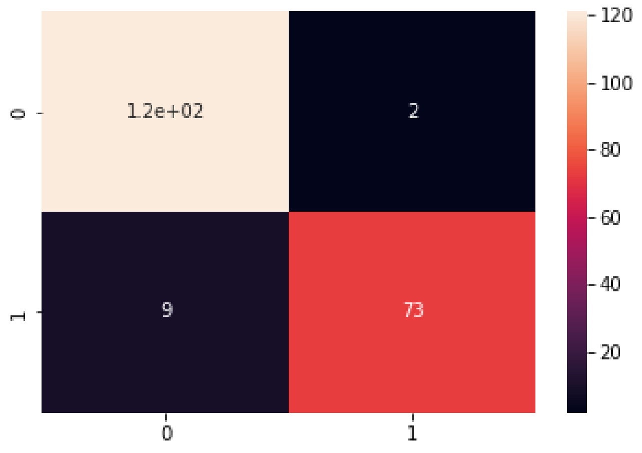
```
              precision    recall  f1-score   support

           0       0.93      0.98      0.96       123
           1       0.97      0.89      0.93        82

    accuracy                           0.95       205
   macro avg       0.95      0.94      0.94       205
weighted avg       0.95      0.95      0.95       205
```

In [21]:

```
sns.heatmap(cm, annot=True)
```

Out[21]:

```
<AxesSubplot:>
```

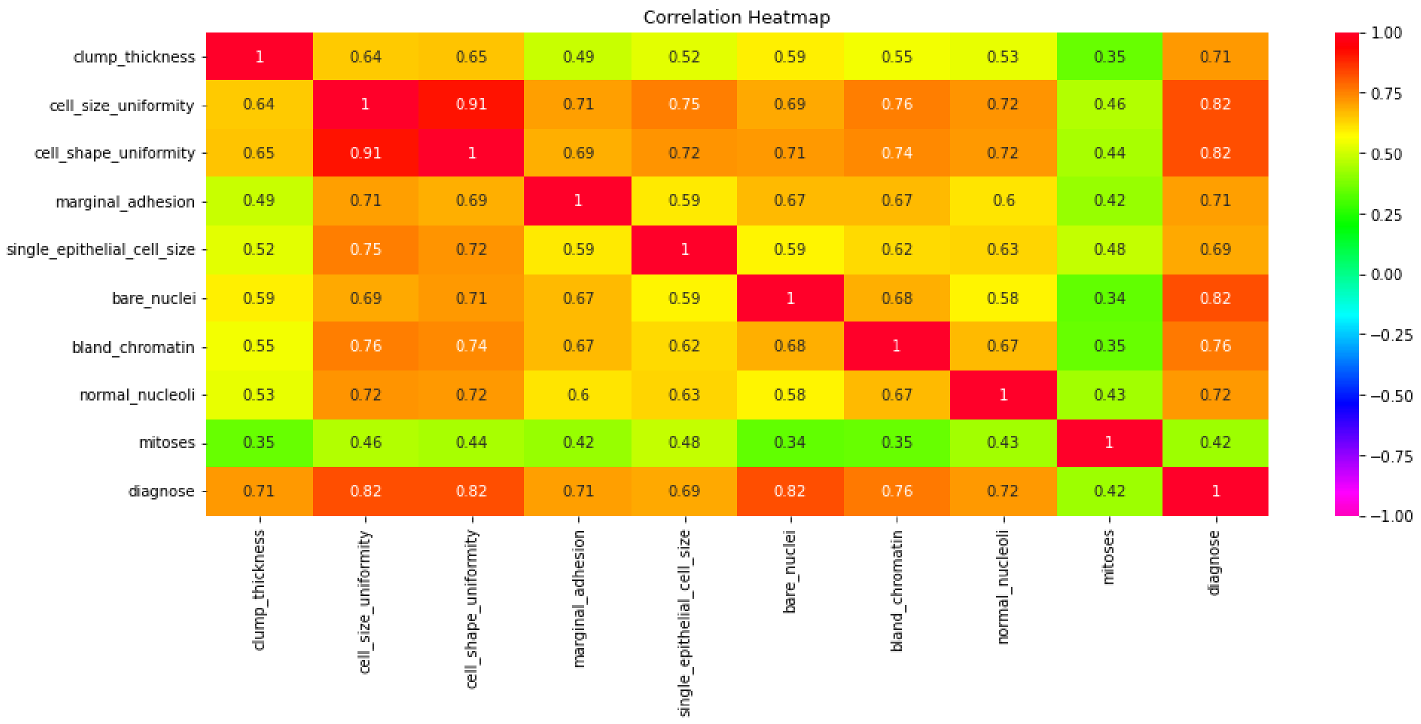


In [22]:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

In [23]:

```
plt.figure(figsize=(16, 6))

heatmap = sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True, cmap='gist_rainbow_r')

heatmap.set_title('Correlation Heatmap');
```

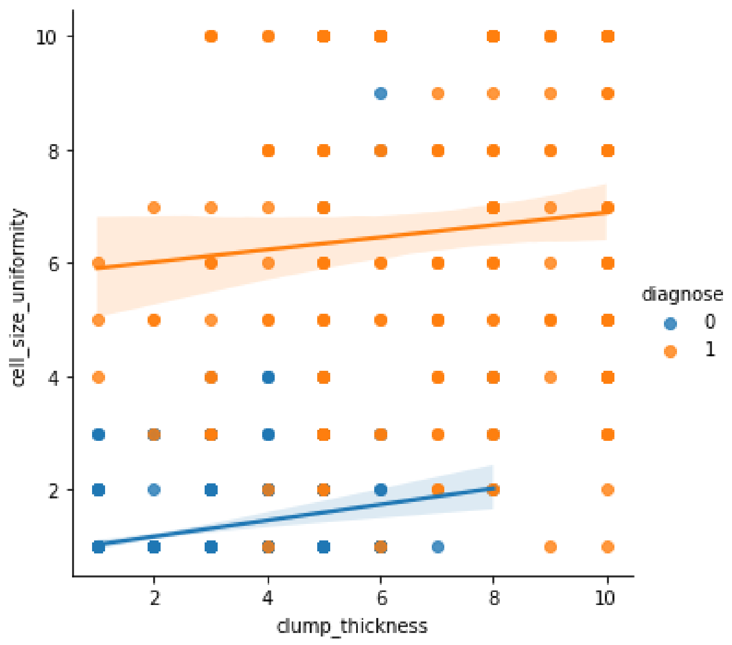


In [24]:

```
sns.lmplot(data=df, x='clump_thickness', y='cell_size_uniformity', hue='diagnose')
```

Out[24]:

```
<seaborn.axisgrid.FacetGrid at 0x1ed51e2a9a0>
```

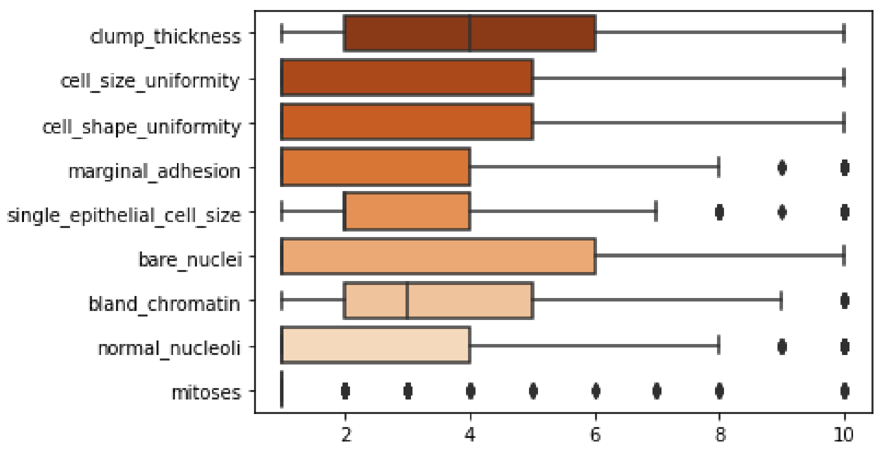


In [25]:

```
sns.boxplot(data=X, orient='h', palette='Oranges_r')
```

Out[25]:

```
<AxesSubplot:>
```

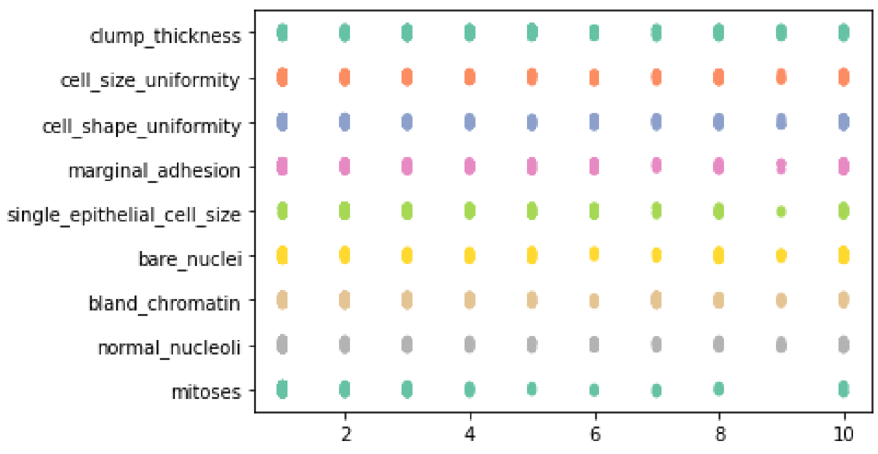


In [26]:

```
sns.stripplot(data=X, orient='h', palette='Set2')
```

Out[26]:

```
<AxesSubplot:>
```

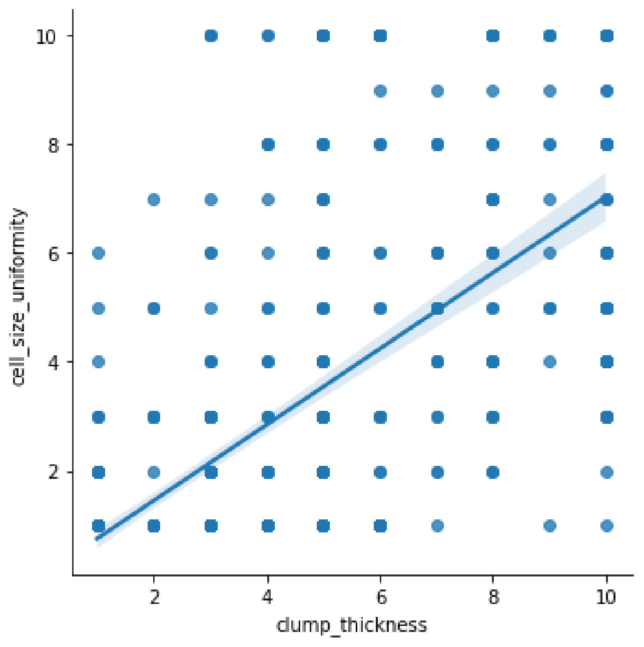


In [27]:

```
sns.lmplot(data=df, x='clump_thickness', y='cell_size_uniformity',palette='Set3')
```

Out[27]:

```
<seaborn.axisgrid.FacetGrid at 0x1ed51fb4f10>
```

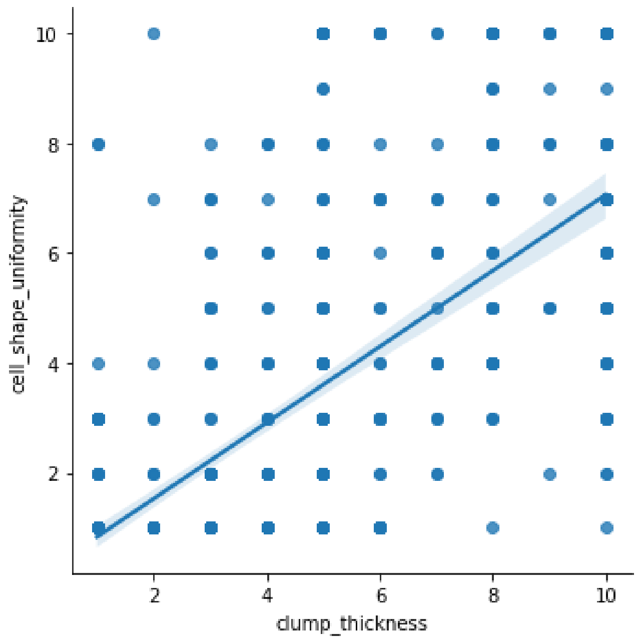


In [28]:

```
sns.lmplot(data=df, x='clump_thickness', y='cell_shape_uniformity')
```

Out[28]:

```
<seaborn.axisgrid.FacetGrid at 0x1ed523a7850>
```

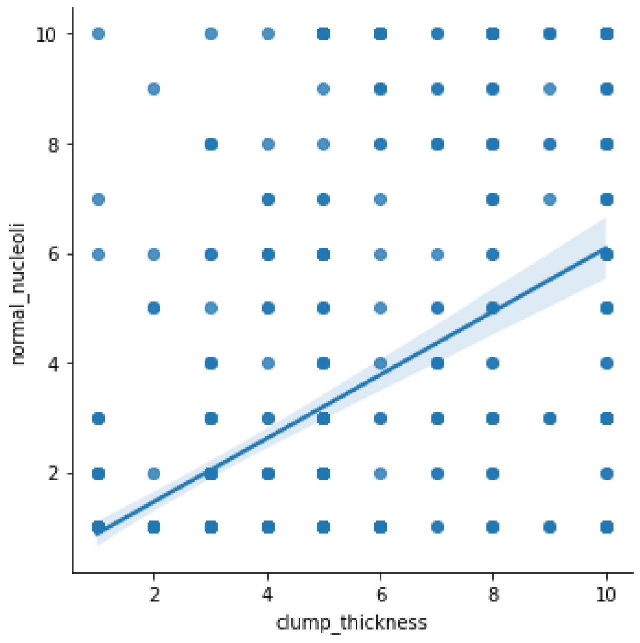


In [29]:

```
sns.lmplot(data=df, x='clump_thickness', y='normal_nucleoli')
```

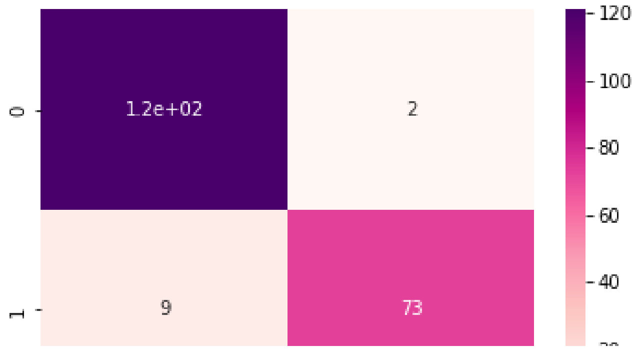
Out[29]:

```
<seaborn.axisgrid.FacetGrid at 0x1ed51506eb0>
```



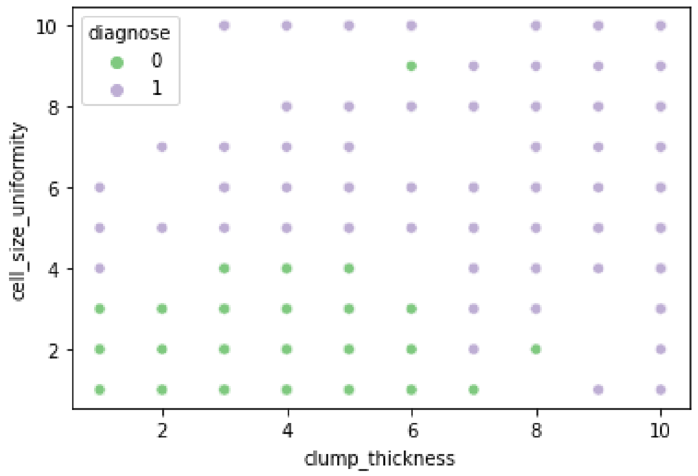
In [30]:

```
sns.heatmap(cm, annot=True, cmap='RdPu');
```



In [35]:

```
import seaborn as sns

sns.scatterplot(x=X.iloc[:,0], y=X.iloc[:,1], hue=y, palette='Accent');
```



In [32]:

```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\ajsru\anaconda3\lib\site-packages (1.6
.2)
Requirement already satisfied: numpy in c:\users\ajsru\anaconda3\lib\site-packages (from
xgboost) (1.21.5)
Requirement already satisfied: scipy in c:\users\ajsru\anaconda3\lib\site-packages (from
xgboost) (1.7.3)
```

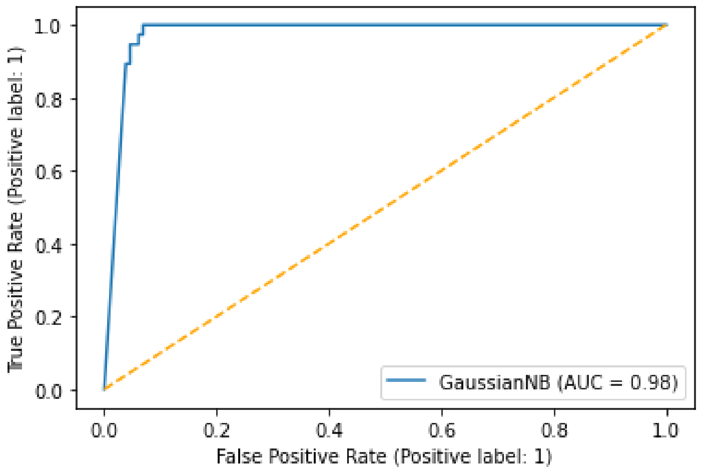
In [33]:

```
from sklearn.metrics import confusion_matrix,classification_report, plot_roc_curve,roc_cu
rve, auc
```

In [34]:

```
disp=plot_roc_curve(model, Xtest, Ytest)
plt.plot([0,1], [0,1], color='orange', linestyle='--');
```

```
C:\Users\ajsru\AppData\Roaming\Python\Python39\site-packages\sklearn\utils\deprecation.py
:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve
` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`s
klearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDispla
y.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```



In [ ]:

```

```

KNN Classification Model

Creating Training and Testing Models

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn import neighbors, datasets
from matplotlib.colors import ListedColormap
from sklearn.inspection import DecisionBoundaryDisplay
```

In [3]:
```
df=pd.read_csv('breast_cancer.csv')
```

In [4]:
```
df.head(5)
```

Out[4]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_epithelial_cel
0	5	1	1	1	
1	5	4	4	5	
2	3	1	1	1	
3	6	8	8	1	
4	4	1	1	3	

In [5]:
```
X=df.drop('diagnose', axis=1)
```

In [6]:
```
X1=df.iloc[:,0:4]
```

In [7]:
```
X1.head()
```

Out[7]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion
0	5	1	1	1
1	5	4	4	5
2	3	1	1	1
3	6	8	8	1
4	4	1	1	3

In [8]:
```
xlabel=X1.columns[0]
ylabel=X1.columns[1]
```

In [9]:
```
y=df['diagnose']
```

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
```

In [11]:
```
# Data standardization

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
# Fit only on X_train
scaler.fit(X_train)

# Scale both X_train and X_test
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [12]:
```
model=KNeighborsClassifier(n_neighbors=5)   # k=5
```

In [13]:
```
model.fit(X_train, y_train)
```

Out[13]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [14]:
```
ypred=model.predict(X_test)
```

In [15]:
```
ypred
```

Out[15]:
```
array([0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1], dtype=int64)
```

Accuracy for Training Data

In [16]:
```
model.score(X_train, y_train)
```

Out[16]:
```
0.9790794979079498
```

Accuracy for Testing Data

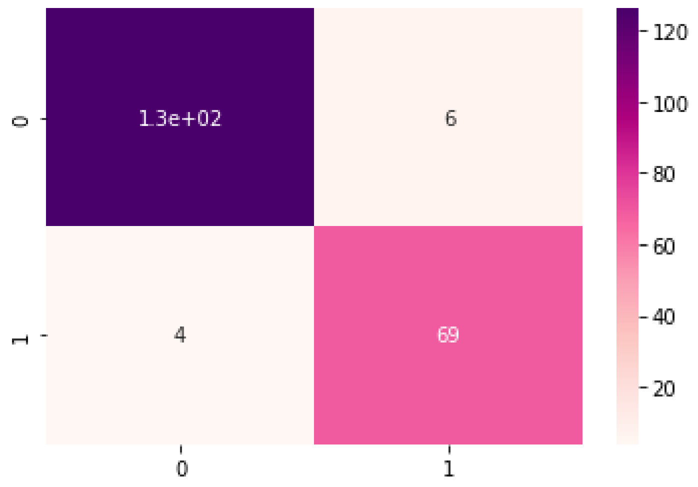
In [17]:
```
model.score(X_test, y_test)
```

Out[17]:
```
0.9512195121951219
```

Confusion Matrix

In [18]:
```
cm=confusion_matrix(ypred, y_test)
```

In [19]:
```
sns.heatmap(cm, annot=True, cmap='RdPu');
```



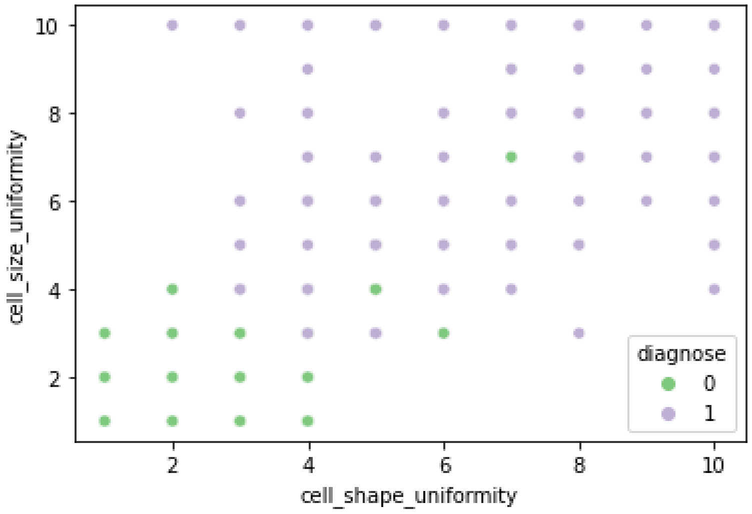
Classification Report

In [20]:
```
print(classification_report(ypred, y_test))
```

```
              precision    recall  f1-score   support

           0       0.97      0.95      0.96       132
           1       0.92      0.95      0.93        73

    accuracy                           0.95       205
   macro avg       0.94      0.95      0.95       205
weighted avg       0.95      0.95      0.95       205
```

Scatter Plot - Cell_shape_uniformity vs Cell_size_uniformity

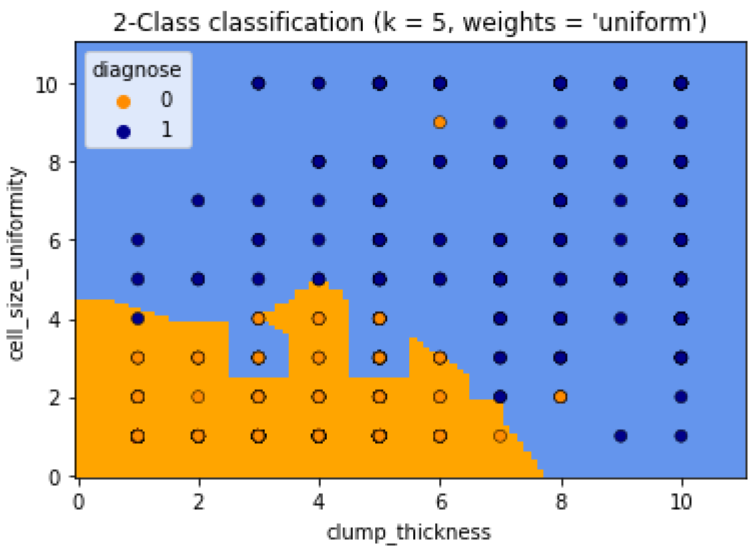
In [21]:
```
sns.scatterplot(data=df, x="cell_shape_uniformity", y="cell_size_uniformity", hue="dia
```

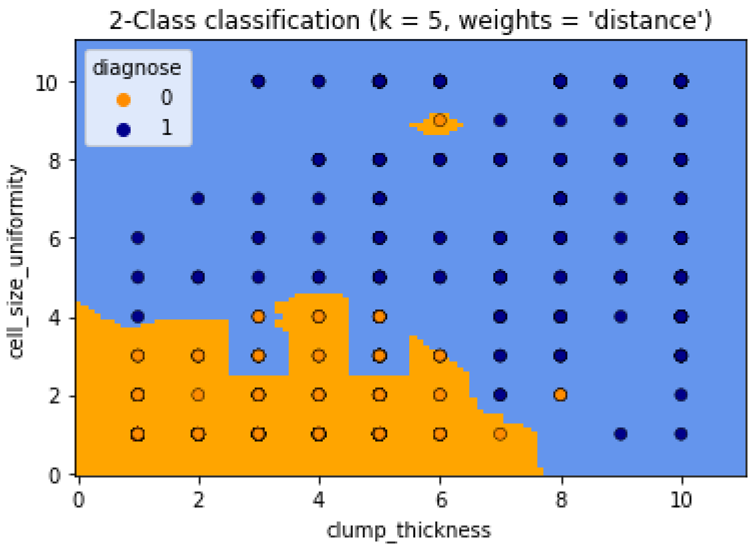


Plotting decision boundry when considering two features (Clump Thickness and Cell size uniformity)

In [22]:
```
cmap_light = ListedColormap(["orange", "cornflowerblue"])
cmap_bold = ["darkorange", "darkblue"]
```

In [23]:
```
n_neighbors = 5

X=df.iloc[:,0:2]
y=df['diagnose']


cmap_light = ListedColormap(["orange", "cornflowerblue"])
cmap_bold = ["darkorange", "darkblue"]


for weights in ["uniform", "distance"]:
    # we create an instance of Neighbours Classifier and fit the data.
    model = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    model.fit(X, y)

    _, ax= plt.subplots()
    DecisionBoundaryDisplay.from_estimator(model,X,cmap=cmap_light,ax=ax,response_meth
        xlabel=X.columns[0],
        ylabel=X.columns[1],
        shading="auto")

 # Plot also the training points'

    sns.scatterplot(
        x=X.iloc[:,0],
        y=X.iloc[:,1],
        hue=y,
        palette=cmap_bold,
        alpha=1.0,
        edgecolor="black",
       )

    plt.title(
          "2-Class classification (k = %i, weights = '%s')" % (n_neighbors, weights));
```





Random Forest

In [30]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
import seaborn as sns
```

In [11]:
```
df = pd.read_csv(r'C:\Users\91735\Downloads\breast_cancer.csv')
```

In [12]:
```
df.describe()
```

Out[12]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_epithelia
count	683.000000	683.000000	683.000000	683.000000	6
mean	4.442167	3.150805	3.215227	2.830161	
std	2.820761	3.065145	2.988581	2.864562	
min	1.000000	1.000000	1.000000	1.000000	
25%	2.000000	1.000000	1.000000	1.000000	
50%	4.000000	1.000000	1.000000	1.000000	
75%	6.000000	5.000000	5.000000	4.000000	
max	10.000000	10.000000	10.000000	10.000000	

In [3]:
```
x = df.drop('diagnose',axis = 1)
y = df['diagnose']
```

In [4]:
```
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size = 0.3,random_state=0)
```

In [18]:
```
model = RandomForestClassifier(random_state = 1,n_estimators = 2000)
```

In [19]:
```
model.fit(x,y)
```

Out[19]:
```
▾ RandomForestClassifier
RandomForestClassifier(n_estimators=2000, random_state=1)
```

In [21]:
```
I = model.feature_importances_
```

In [22]:
```
I
```

Out[22]:
```
array([0.04786191, 0.27018697, 0.22232477, 0.02821389, 0.08151435,
       0.16604124, 0.10599434, 0.07083451, 0.00702803])
```

In [ ]:

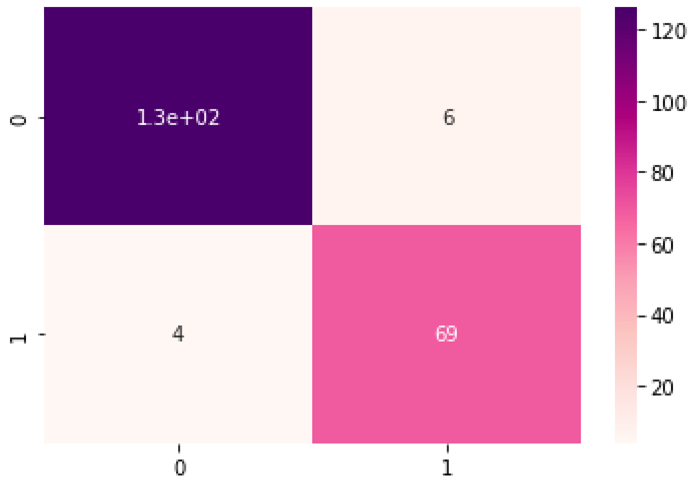
```
In [6]: model.fit(xtrain,ytrain)
        model.score(xtrain,ytrain)

Out[6]: 1.0
```

```
In [7]: model.score(xtest,ytest)

Out[7]: 0.9512195121951219
```

```
In [8]: ypred = model.predict(xtest)
```

```
In [28]: cm = confusion_matrix(ypred,ytest)
         cm

Out[28]: array([[126,   6],
                [  4,  69]], dtype=int64)
```

```
In [31]: sns.heatmap(cm, annot = True,  cmap = 'RdPu')

Out[31]: <AxesSubplot:>
```



```
In [10]: print(classification_report(ypred,ytest))

              precision    recall  f1-score   support

           0       0.97      0.95      0.96       132
           1       0.92      0.95      0.93        73

    accuracy                           0.95       205
   macro avg       0.94      0.95      0.95       205
weighted avg       0.95      0.95      0.95       205
```

Logistics Regression

In [20]:
```
# Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns
from mlxtend.plotting import plot_decision_regions
from sklearn.decomposition import PCA
```

In [21]:
```
# Loading dataset
df = pd.read_csv(r'breast_cancer.csv')

df.describe()
```

Out[21]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_epithelia
count	683.000000	683.000000	683.000000	683.000000	6
mean	4.442167	3.150805	3.215227	2.830161	
std	2.820761	3.065145	2.988581	2.864562	
min	1.000000	1.000000	1.000000	1.000000	
25%	2.000000	1.000000	1.000000	1.000000	
50%	4.000000	1.000000	1.000000	1.000000	
75%	6.000000	5.000000	5.000000	4.000000	
max	10.000000	10.000000	10.000000	10.000000	

In [22]:
```
# Previewing data real quick
df.head(10)
```

Out[22]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_epithelial_cel
0	5	1	1	1	
1	5	4	4	5	
2	3	1	1	1	
3	6	8	8	1	
4	4	1	1	3	
5	8	10	10	8	
6	1	1	1	1	
7	2	1	2	1	
8	2	1	1	1	
9	4	2	1	1	

In [23]:
```
# Defining x variables and y variable
x = df.iloc[:,0:9].values
y = df.iloc[:,9].values
```

In [24]:
```
# Splitting data for training and testing
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.3, random_state =
```

In [25]:
```
# Initializing the model
lrmodel = LogisticRegression(solver='newton-cg')
```

In [26]:
```
# Fitting our training data on the model
lrmodel.fit(xtrain, ytrain)
```

Out[26]:
```
▾            LogisticRegression
LogisticRegression(solver='newton-cg')
```

In [27]:
```
# Checking intercept of our regression model
lrmodel.intercept_
```

Out[27]:
```
array([-9.17796322])
```

In [28]:
```
# Checking the coefficients of our regression model
lrmodel.coef_
```

Out[28]:
```
array([[0.3943735 , 0.14430425, 0.21099364, 0.19743802, 0.22348085,
        0.46479598, 0.31518682, 0.24472243, 0.18561027]])
```

In [29]:
```
# Predicting our testing data
lrprediction = lrmodel.predict(xtest)
```

In [30]:
```
# Checking accuracy of our model by comparing it with training data
trscore = lrmodel.score(xtrain, ytrain)
```

In [31]:
```
# Checking accuracy of our model by comparing it with testing data
lrscore = lrmodel.score(xtest, ytest)
```

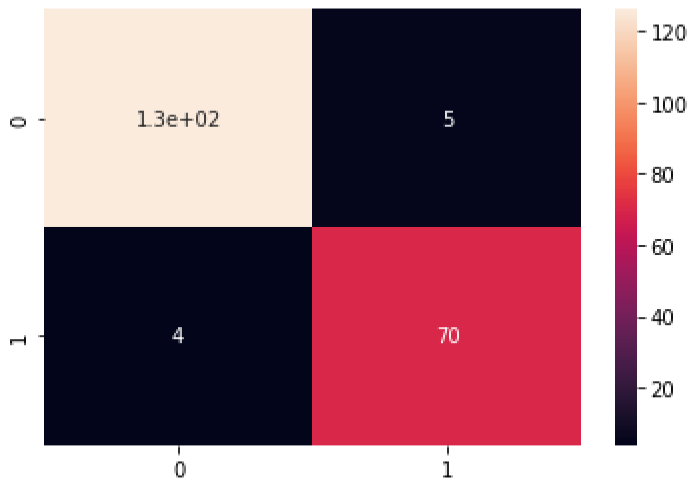
In [32]:
```
# Our model has an accuracy of 95.6 %
print('Training Accuracy = ', trscore * 100, '%')
print('Testing Accuracy = ', lrscore * 100, '%')
```

```
Training Accuracy =  97.48953974895397 %
Testing Accuracy =  95.60975609756098 %
```

In [33]:
```
confusionmatrix = confusion_matrix(lrprediction, ytest)
```

In [34]:
```
sns.heatmap(confusionmatrix, annot = True)
```

Out[34]:
```
<AxesSubplot:>
```



In [35]:
```
print(classification_report(lrprediction, ytest))
```

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       131
           1       0.93      0.95      0.94        74

    accuracy                           0.96       205
   macro avg       0.95      0.95      0.95       205
weighted avg       0.96      0.96      0.96       205
```

In [36]:
```
df
```

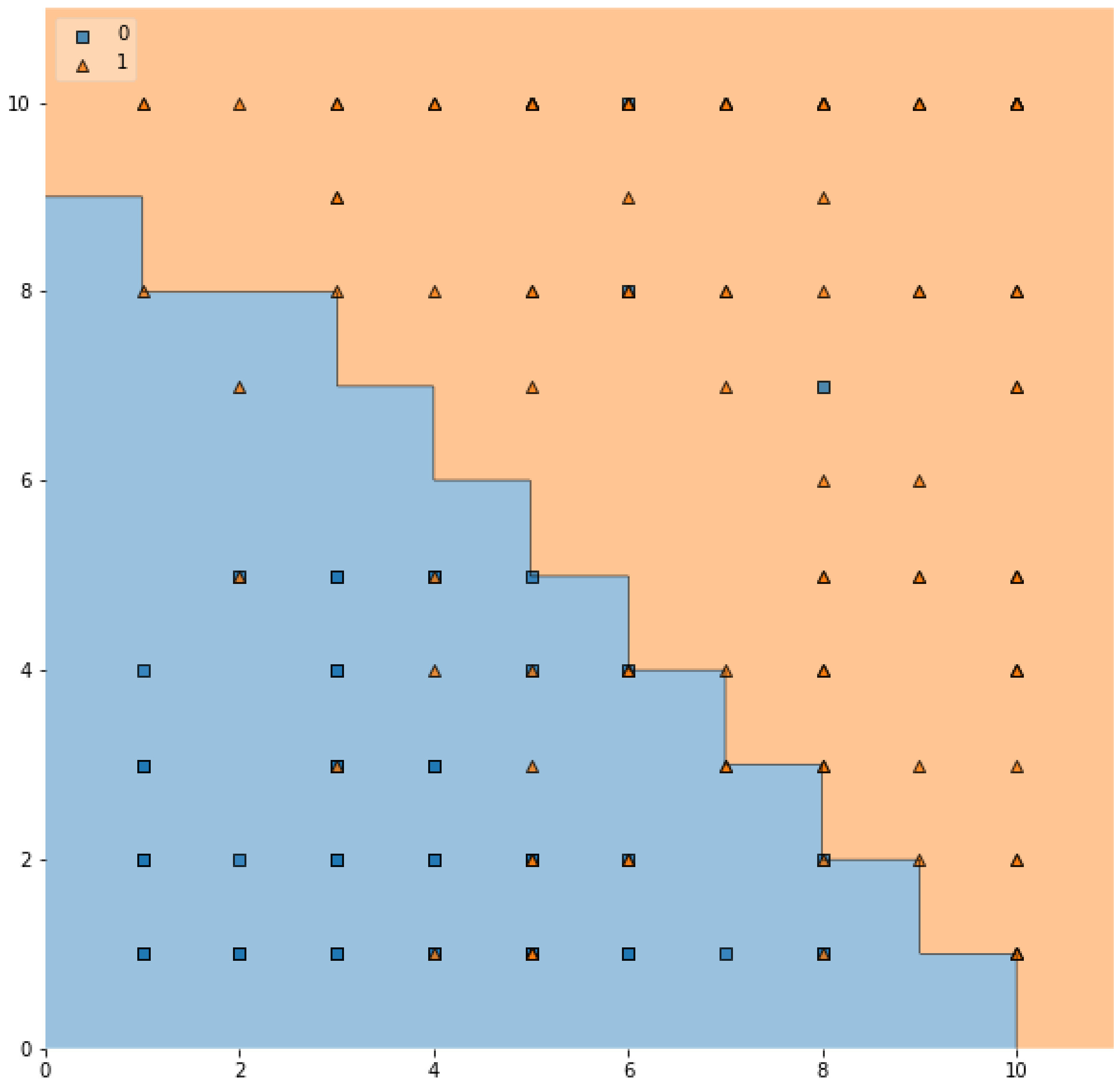
Out[36]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_epithelial_
0	5	1	1	1	
1	5	4	4	5	
2	3	1	1	1	
3	6	8	8	1	
4	4	1	1	3	
...	...	...	...	...	
678	3	1	1	1	
679	2	1	1	1	
680	5	10	10	3	
681	4	8	6	4	
682	4	8	8	5	

683 rows × 10 columns

In [40]:
```
# Creating Decision Boundaries between features with high coefficient (clump thickness

dbx = df.iloc[:,[0,5]].values
dby = df['diagnose'].values

dbxtrain, dbxtest, dbytrain, dbytest = train_test_split(dbx, dby, test_size=0.3, rando

lr = LogisticRegression(solver='newton-cg',random_state=0)

lr.fit(dbxtrain, dbytrain);

fig8, ax = plt.subplots(figsize=(10, 10))
fig8 = plot_decision_regions(dbxtrain, dbytrain, clf=lr, legend=2);

plt.show
```

Out[40]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



0
1
0
2
4
6
8
10

ROC curve

In [2]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\kjosep14\anaconda3\lib\site-packag
es (1.6.2)
Requirement already satisfied: scipy in c:\users\kjosep14\anaconda3\lib\site-packages
(from xgboost) (1.7.3)
Requirement already satisfied: numpy in c:\users\kjosep14\anaconda3\lib\site-packages
(from xgboost) (1.21.5)
```

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, plot_roc_curve, r
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import xgboost as xgb
import shap
```

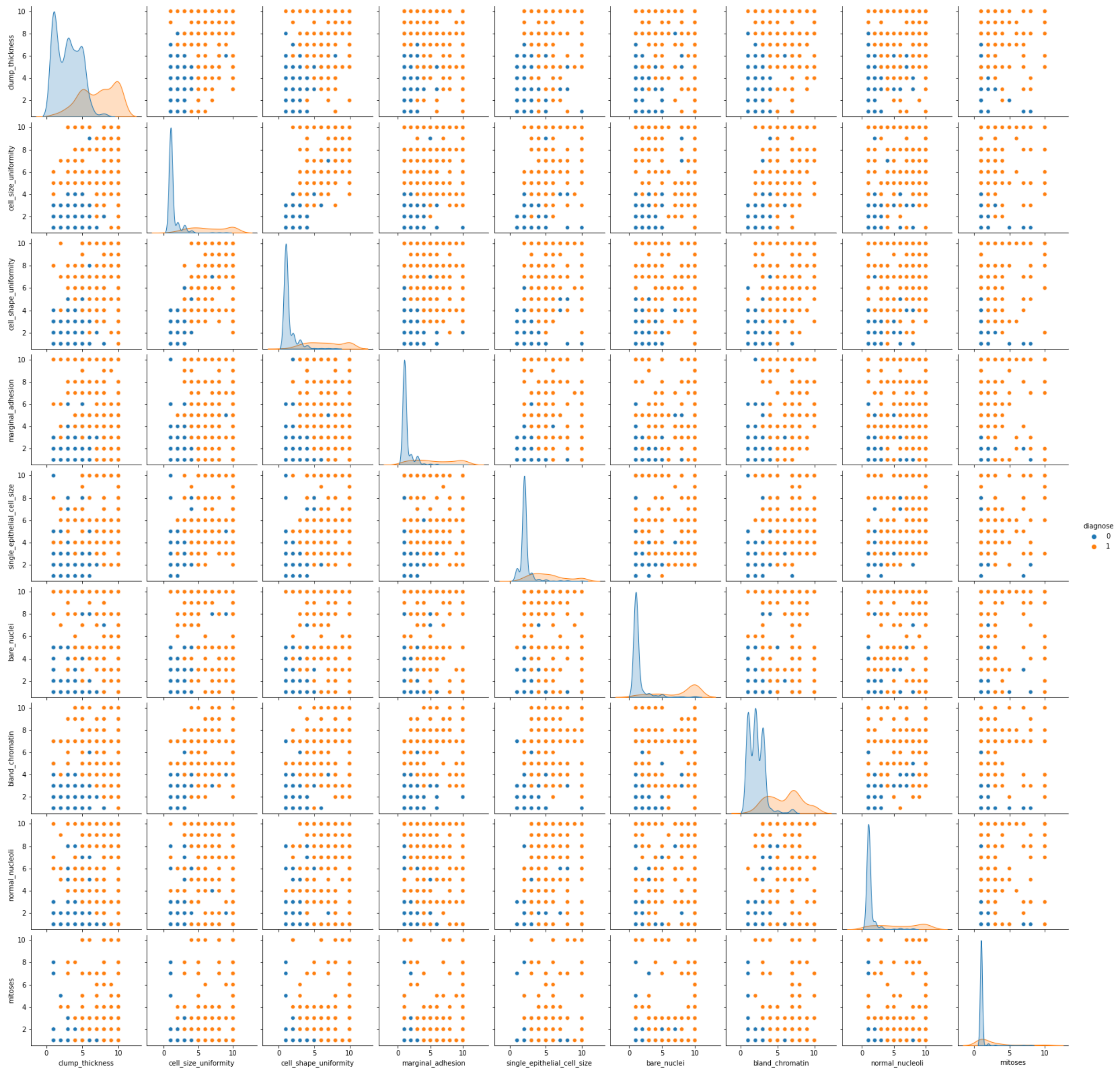
In [6]:
```
df=pd.read_csv('breast_cancer.csv')
```

In [7]:
```
df.head(3)
```

Out[7]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_epithelial_cel
0	5	1	1	1	
1	5	4	4	5	
2	3	1	1	1	

In [8]:
```
sns.pairplot(df, hue='diagnose');    # pairplot of dataset, scatter plot which represer
                                    # each pair of features
```



In [9]:
```python
X=df.drop('diagnose', axis=1)
y=df['diagnose']
```

In [10]:
```python
# split data into testing and training

Xtrain, Xtest, ytrain, ytest=train_test_split(X,y, test_size=0.3, random_state=0)
```

In [11]:
```python
# Standardaize data

scaler=StandardScaler()
scaler.fit(Xtrain)
Xtrain=scaler.transform(Xtrain)
Xtest=scaler.transform(Xtest)
```

In [50]:
```python
# define all models

model1 = LogisticRegression(random_state=1,solver='newton-cg',multi_class='multinomial
model2 = RandomForestClassifier(random_state=1, n_estimators=2000)
model3 = GaussianNB()
model4 = KNeighborsClassifier(n_neighbors=5)
```

In [51]:
```
# fit the model

model1.fit(Xtrain, ytrain)
model2.fit(Xtrain, ytrain)
model3.fit(Xtrain, ytrain)
model4.fit(Xtrain, ytrain);
```

In [52]:
```
# predict class (y) for testing data (Xtest)

ypred1=model1.predict(Xtest)
ypred2=model2.predict(Xtest)
ypred3=model3.predict(Xtest)
ypred4=model4.predict(Xtest)
```

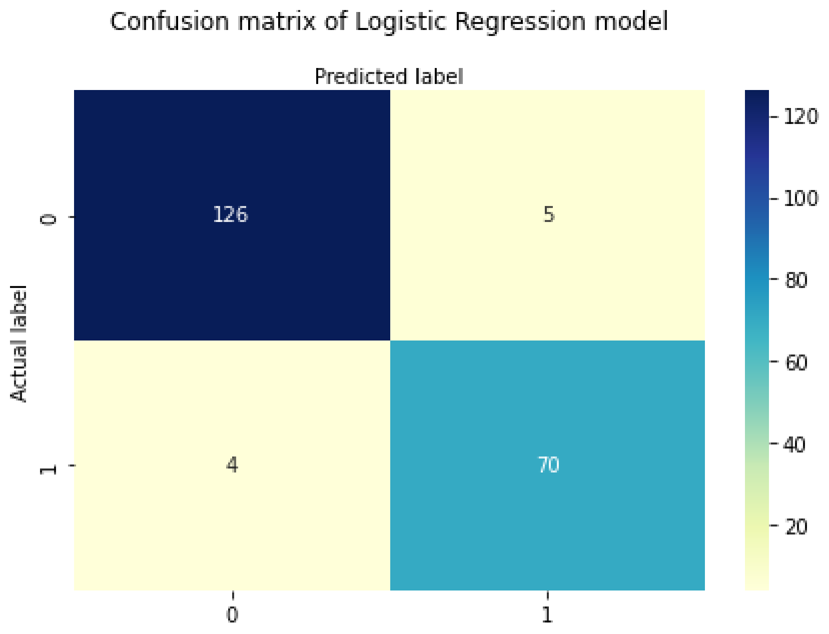
In [53]:
```
ypred6;
```

In [54]:
```
print("Accuracy of Logistic Regression is: ", model1.score(Xtrain, ytrain))
print("Accuracy of Random Forest is: ", model2.score(Xtrain, ytrain))
print("Accuracy of Gaussaian NB is: ", model3.score(Xtrain, ytrain))
print("Accuracy of KNN is: ", model6.score(Xtrain, ytrain))
```

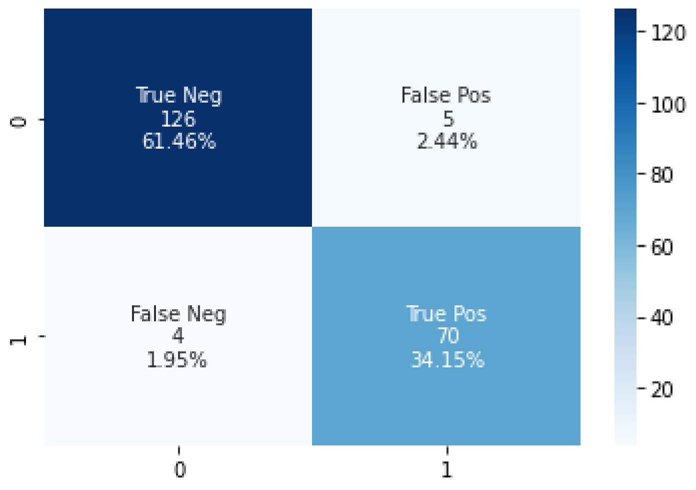
```
Accuracy of Logistic Regression is:  0.9769874476987448
Accuracy of Random Forest is:  1.0
Accuracy of Gaussaian NB is:  0.9707112970711297
Accuracy of KNN is:  0.9790794979079498
```

In [56]:
```
# calculate confusion matrix

cm1=confusion_matrix(ypred1, ytest)
cm2=confusion_matrix(ypred2, ytest)
cm3=confusion_matrix(ypred3, ytest)
cm4=confusion_matrix(ypred4, ytest)
```

In [57]:
```
# plot confusion matrix

class_names=[0,1] # name of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(cm1, annot=True, cmap="YlGnBu" ,fmt='g')   # cm1 is the confusion matrix f
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix of Logistic Regression model', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label');
```

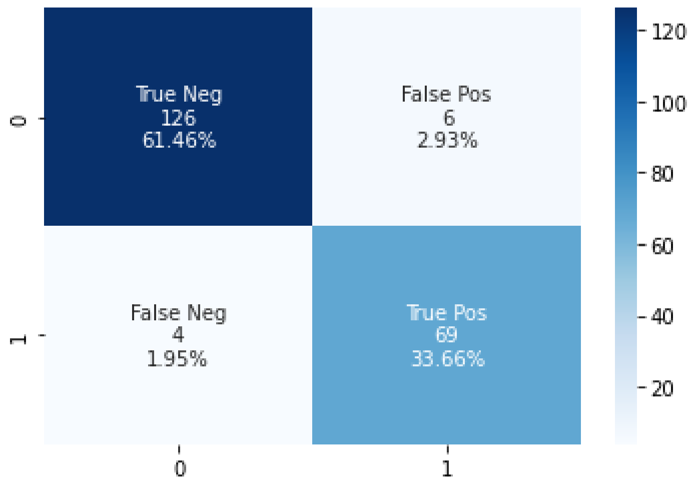


In [58]:
```
# Confusion matrix for model 1 ---- cm1 is the confusion matrix for model 1 (Logistic

group_names = ['True Neg','False Pos','False Neg','True Pos']

group_counts = ["{0:0.0f}".format(value) for value in
                cm1.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cm1.flatten()/np.sum(cm1)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]

labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cm1, annot=labels, fmt='', cmap='Blues');
```

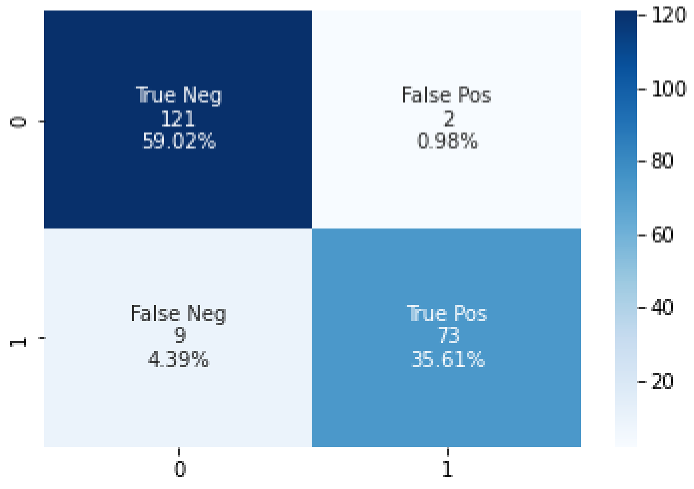


In [59]:
```
# confusion matrix for model 2 (Random Forest)

group_names = ['True Neg','False Pos','False Neg','True Pos']

group_counts = ["{0:0.0f}".format(value) for value in
                cm2.flatten()]
```

```
group_percentages = ["{0:.2%}".format(value) for value in
                     cm2.flatten()/np.sum(cm2)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]

labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cm2, annot=labels, fmt='', cmap='Blues');
```



In [46]:
```
# confusion matrix for model 3 (NB)

group_names = ['True Neg','False Pos','False Neg','True Pos']

group_counts = ["{0:0.0f}".format(value) for value in
                cm3.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cm3.flatten()/np.sum(cm3)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]

labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cm3, annot=labels, fmt='', cmap='Blues');
```

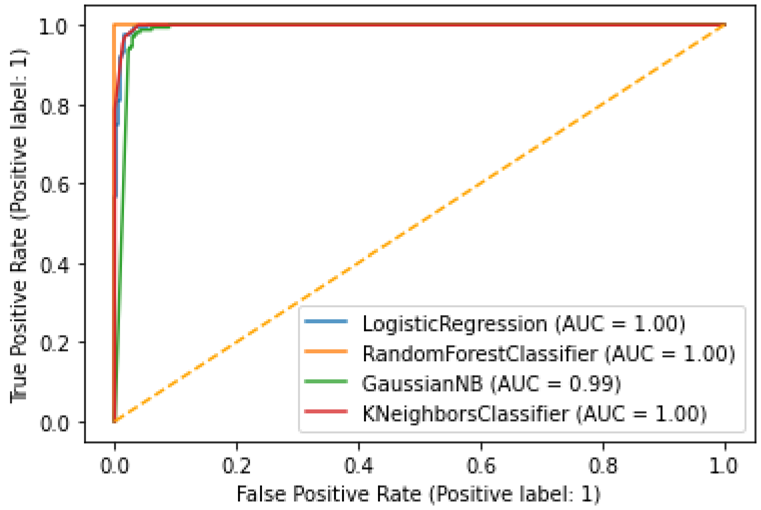


In [60]:
```
print(classification_report(ypred1, ytest))  # first model (logistic Regression)
```

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       131
           1       0.93      0.95      0.94        74

    accuracy                           0.96       205
   macro avg       0.95      0.95      0.95       205
weighted avg       0.96      0.96      0.96       205
```

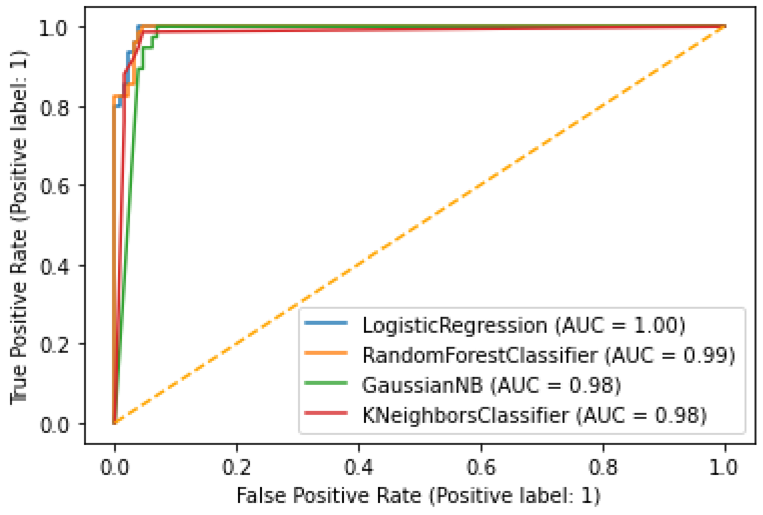
In [61]:
```
### plot ROC curve for Xtrain and ytrain

disp=plot_roc_curve(model1, Xtrain, ytrain)
plot_roc_curve(model2, Xtrain, ytrain, ax=disp.ax_)
plot_roc_curve(model3, Xtrain, ytrain, ax=disp.ax_)
plot_roc_curve(model4, Xtrain, ytrain, ax=disp.ax_)
plt.plot([0,1], [0,1], color='orange', linestyle='--');
```

```
Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated
in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metri
cs.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_e
stimator`.
Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated
in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metri
cs.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_e
stimator`.
Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated
in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metri
cs.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_e
stimator`.
Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated
in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metri
cs.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_e
stimator`.
```



In [62]:
```
### Plot ROC Curve for Xtest, ytest

disp=plot_roc_curve(model1, Xtest, ytest)
plot_roc_curve(model2, Xtest, ytest, ax=disp.ax_)
plot_roc_curve(model3, Xtest, ytest, ax=disp.ax_)
plot_roc_curve(model4, Xtest, ytest, ax=disp.ax_)
plt.plot([0,1], [0,1], color='orange', linestyle='--');
```

```
Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated
in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metri
cs.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_e
stimator`.
Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated
in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metri
cs.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_e
stimator`.
Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated
in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metri
cs.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_e
stimator`.
Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated
in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metri
cs.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_e
stimator`.
```



Feature importance using Random Forest

In [64]:
```
rf=RandomForestClassifier(random_state=1, n_estimators=100)
```

In [65]:
```
rf.fit(X,y)
```

Out[65]:
```
▾        RandomForestClassifier
RandomForestClassifier(random_state=1)
```

In [66]:
```
I=rf.feature_importances_
```

In [67]:
```
I
```

Out[67]:
```
array([0.05398005, 0.2516356 , 0.2464734 , 0.02746694, 0.10317641,
       0.13508699, 0.11311992, 0.06109489, 0.00796582])
```

In [68]:
```
X[0:2]
```

Out[68]:

	clump_thickness	cell_size_uniformity	cell_shape_uniformity	marginal_adhesion	single_epithelial_cel
0	5	1	1	1	
1	5	4	4	5	

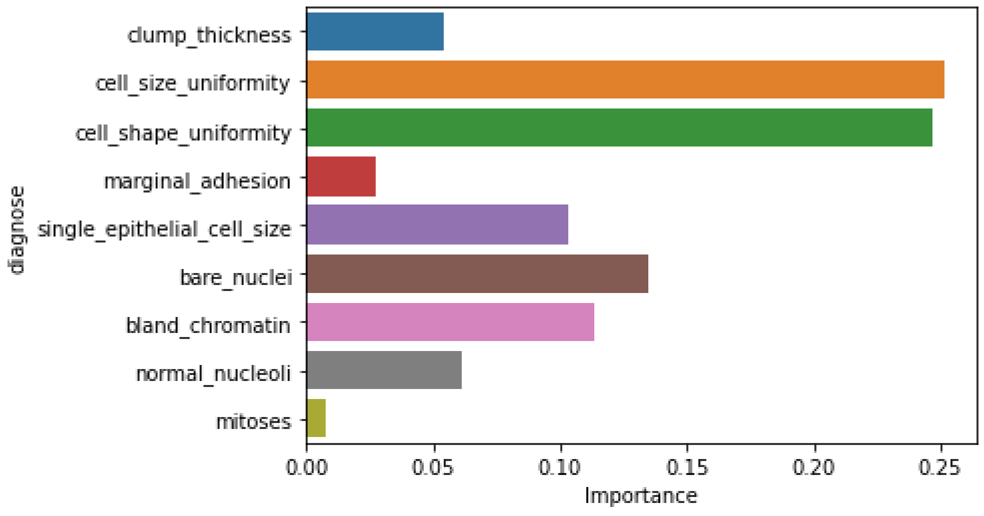
In [69]:
```
df2=pd.DataFrame({
    "Importance": [0.05398005, 0.2516356 , 0.2464734 , 0.02746694, 0.10317641,
       0.13508699, 0.11311992, 0.06109489, 0.00796582],
    "diagnose": ["clump_thickness", "cell_size_uniformity", "cell_shape_uniformity", '
                "mitoses"] })
```

In [70]:
```
df2
```

Out[70]:

	Importance	diagnose
0	0.053980	clump_thickness
1	0.251636	cell_size_uniformity
2	0.246473	cell_shape_uniformity
3	0.027467	marginal_adhesion
4	0.103176	single_epithelial_cell_size
5	0.135087	bare_nuclei
6	0.113120	bland_chromatin
7	0.061095	normal_nucleoli
8	0.007966	mitoses

In [71]:
```
sns.barplot(data=df2, y="diagnose", x="Importance");
```


clump_thickness
cell_size_uniformity
cell_shape_uniformity
marginal_adhesion
single_epithelial_cell_size
bare_nuclei
bland_chromatin
normal_nucleoli
mitoses
diagnose
Importance