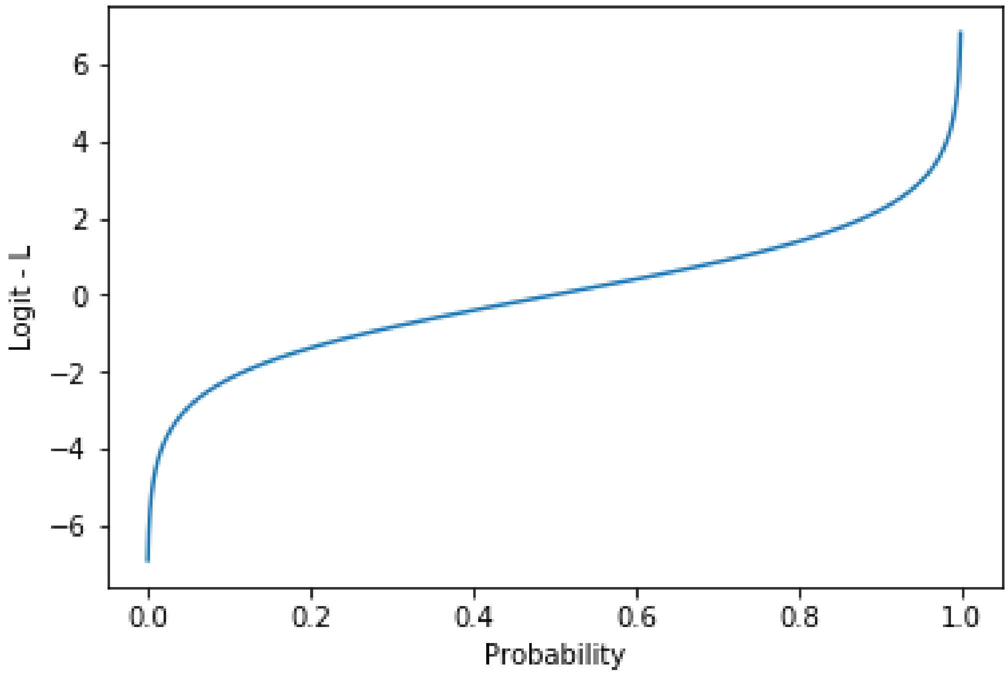
sklearn

October 22, 2019

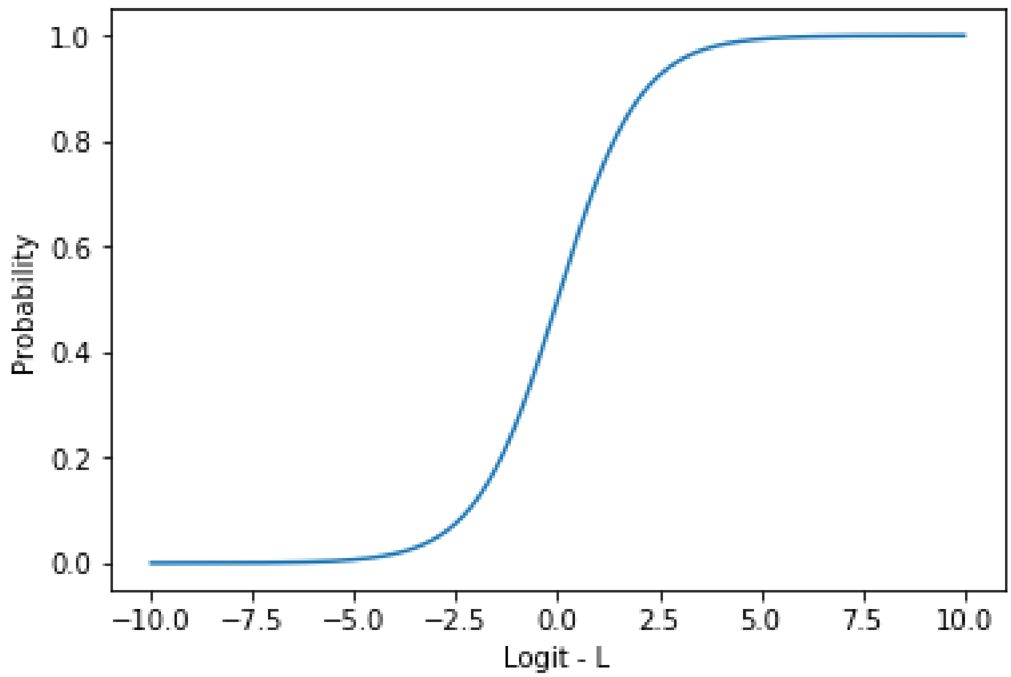
```
In [1]: %matplotlib inline
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: def logit(x):
            return np.log(x/(1-x))

In [3]: x=np.arange(0.001,0.999, 0.0001)
        y=[logit(n) for n in x]
        plt.plot(x,y)
        plt.xlabel("Probability")
        plt.ylabel("Logit - L")

Out[3]: Text(0, 0.5, 'Logit - L')
```



In [4]:
```
def sigmoid(x):
    return (1/(1+np.exp(-x)))
```

In [5]:
```
x=np.arange(-10,10,0.0001)
y=[sigmoid(n) for n in x]
plt.plot(x,y)
plt.xlabel("Logit - L")
plt.ylabel("Probability")
```

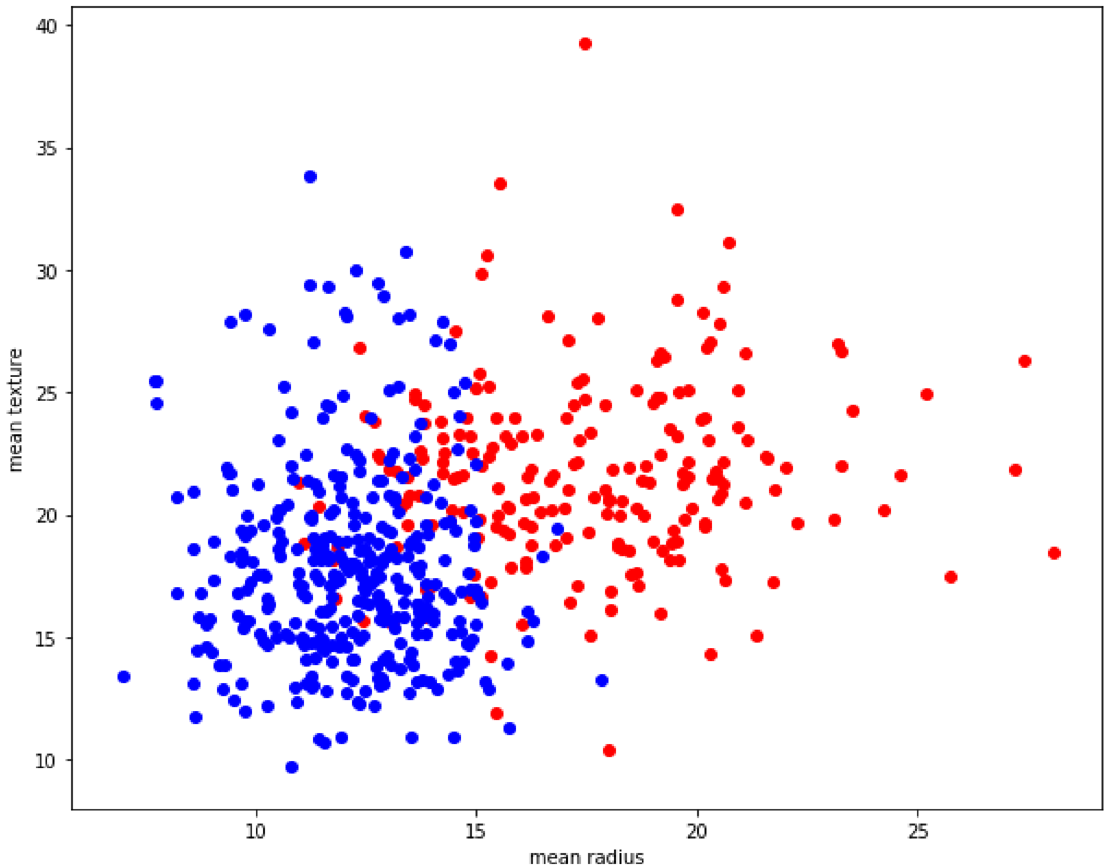
Out[5]:
```
Text(0, 0.5, 'Probability')
```



In [6]:
```
from sklearn.datasets import load_breast_cancer
cancer=load_breast_cancer()
```

In [7]:
```
X=[]
for target in range(2):
    X.append([[],[]])
    for i in range(len(cancer.data)):
        if cancer.target[i]==target:
            X[target][0].append(cancer.data[i][0])
            X[target][1].append(cancer.data[i][1])

colours =("r","b")
fig = plt.figure(figsize(10,8))
ax=fig.add_subplot(111)
```

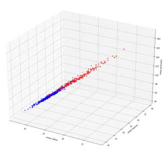
```
for target in range(2):
    ax.scatter(X[target][0],
               X[target][1],
               c=colours[target])

ax.set_xlabel("mean radius")
ax.set_ylabel("mean texture")
plt.show()
```



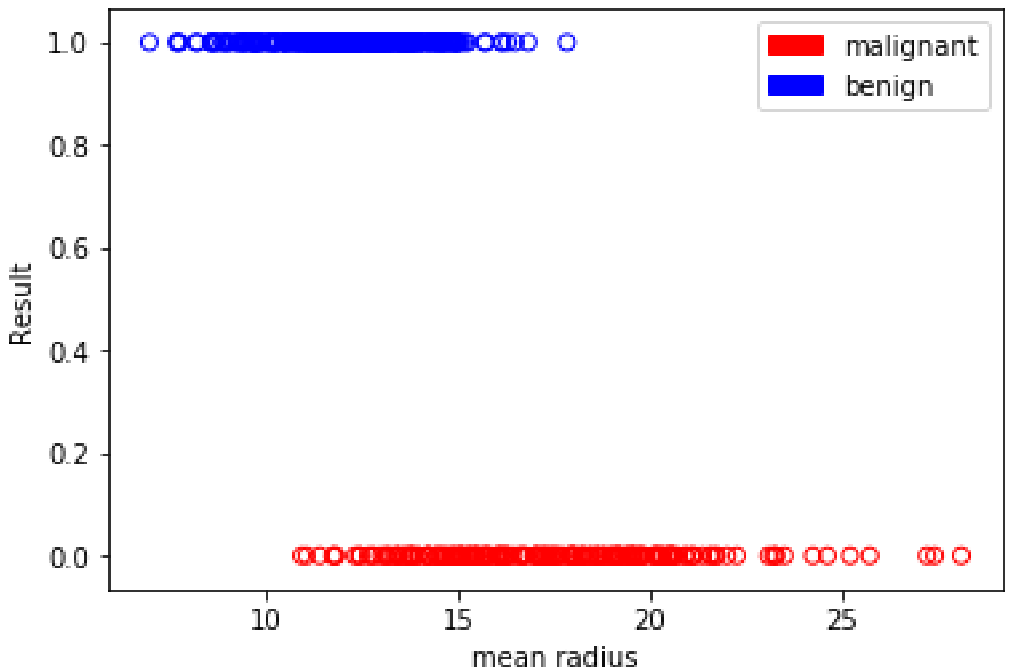
In [8]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.datasets import load_breast_cancer
```

In [9]:
```
cancer = load_breast_cancer()
X=[]
for target in range(2):
    X.append([[],[],[]])
    for i in range(len(cancer.data)):
        if cancer.target[i]==target:
            X[target][0].append(cancer.data[i][0])
```

```
            X[target][1].append(cancer.data[i][1])
            X[target][2].append(cancer.data[i][2])

colours = ("r","b")
fig=plt.figure(figsize=(18,15))
ax=fig.add_subplot(111, projection='3d')
for target in range(2):
    ax.scatter(X[target][0],
               X[target][1],
               X[target][2],
               c=colours[target])

ax.set_xlabel("mean radius")
ax.set_ylabel("mean texture")
ax.set_zlabel("mean perimeter")
plt.show()
```



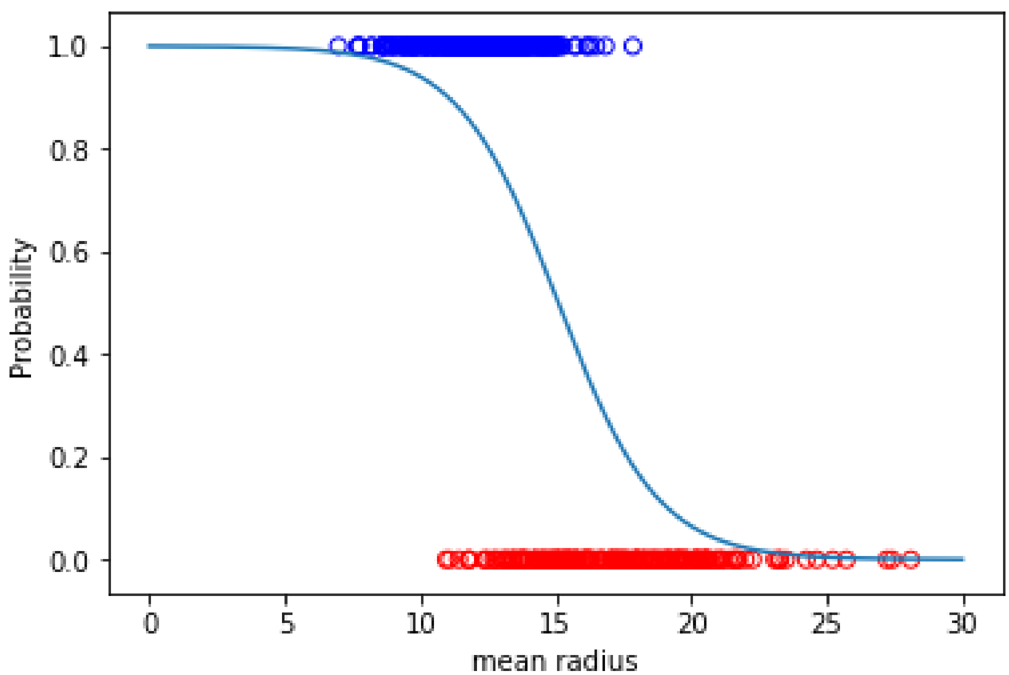
```
In [10]: %matplotlib inline
         import pandas as pd
         import matplotlib.pyplot as plt
         import matplotlib.patches as mpatches

         from sklearn.datasets import load_breast_cancer
         cancer = load_breast_cancer()
         x=cancer.data[:,0]#mean radius
         y=cancer.target#0:malignant, 1:benign
         colors = {0:'red', 1:'blue'}
         plt.scatter(x,y,
                    facecolors='none',
                    edgecolors=pd.DataFrame(cancer.target)[0].apply(lambda x:colors[x]),cmap=co
         plt.xlabel("mean radius")
         plt.ylabel("Result")

         red = mpatches.Patch(color='red', label='malignant')
         blue=mpatches.Patch(color='blue', label='benign')

         plt.legend(handles=[red, blue], loc=1)

Out[10]: <matplotlib.legend.Legend at 0x21fdd455ac8>
```




```
In [11]: from sklearn import linear_model
         import numpy as np

         log_regress = linear_model.LogisticRegression()

In [12]: # train the model
         log_regress.fit(X=np.array(x).reshape(len(x),1),y=y)

         # print trained model intercept
         print(log_regress.intercept_)
         print(log_regress.coef_)

[8.19393897]
[[-0.54291739]]


d:\dev\python\python36\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: D
  FutureWarning)


In [13]: # Plotting the Sigmoid Curve
         def sigmoid(x):
             return (1/(1+np.exp(-(log_regress.intercept_[0]+(log_regress.coef_[0][0]*x)))))

         x1=np.arange(0,30,0.01)
```

```
y1=[sigmoid(n) for n in x1]

plt.scatter(x,y,facecolors='none',
            edgecolors=pd.DataFrame(cancer.target)[0].apply(lambda x: colors[x]),
            cmap=colors)

plt.plot(x1,y1)
plt.xlabel("mean radius")
plt.ylabel('Probability')
```

Out[13]: Text(0, 0.5, 'Probability')



In [14]: `print(log_regress.predict_proba([[20]]))`

```
[[0.93489354 0.06510646]]
```

In [15]: `print(log_regress.predict([[20]])[0])`

```
0
```

In [16]: `print(log_regress.predict_proba([[8]]))`

```
[[0.02082411 0.97917589]]
```

```
In [17]: print(log_regress.predict([[8]])[0])
```

```
1
```

```
In [18]: # Training the model using all features
         from sklearn.datasets import load_breast_cancer
         cancer = load_breast_cancer()
```

```
In [19]: from sklearn.model_selection import train_test_split
         train_set, test_set, train_labels, test_labels = train_test_split(
                      cancer.data,
                      cancer.target,
                      test_size=0.25,
                      random_state=1,
                      stratify=cancer.target
         )
```

```
In [20]: from sklearn import linear_model
         x=train_set[:,0:30]
         y=train_labels
         log_regress=linear_model.LogisticRegression()
         log_regress.fit(X=x, y=y)
```

```
d:\dev\python\python36\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: I
  FutureWarning)
```

```
Out[20]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='warn',
                    n_jobs=None, penalty='l2', random_state=None, solver='warn',
                    tol=0.0001, verbose=0, warm_start=False)
```

```
In [21]: print(log_regress.intercept_)
```

```
[0.34532875]
```

```
In [22]: print(log_regress.coef_)
```

```
[[ 1.80111966e+00  2.55753177e-01 -3.76203243e-02 -5.88987979e-03
  -9.58049351e-02 -3.16746982e-01 -5.06749680e-01 -2.53240777e-01
  -2.26207024e-01 -1.03696078e-02  4.03711661e-03  9.76796186e-01
   2.02748087e-01 -1.22295425e-01 -8.25625028e-03 -1.41118624e-02
  -5.49936132e-02 -3.33054810e-02 -3.05731116e-02  1.13420163e-04
   1.62877492e+00 -4.35039273e-01 -1.50276583e-01 -2.32832527e-02
  -1.94406863e-01 -9.91538995e-01 -1.42903460e+00 -5.40825444e-01
  -6.28853082e-01 -9.04965298e-02]]
```

In [23]:
```
import pandas as pd
# get the predicted probablities and convert into a dataframe
preds_prob=pd.DataFrame(log_regress.predict_proba(X=test_set))
# assign column names to prediction
preds_prob.columns = ["Malignant", "Benign"]
# get the predicted class labels
preds = log_regress.predict(X=test_set)
preds_class=pd.DataFrame(preds)
preds_class.columns = ["Prediction"]
```

In [24]:
```
# actual diagnosis
original_result = pd.DataFrame(test_labels)
original_result.columns = ["Original Result"]
```

In [25]:
```
# merge the three dataframes into one
result = pd.concat([preds_prob, preds_class, original_result], axis=1)
print(result.head(10))
```

```
   Malignant        Benign  Prediction  Original Result
0   0.999812  1.881729e-04           0                0
1   0.998358  1.642333e-03           0                0
2   0.057984  9.420165e-01           1                1
3   1.000000  9.691544e-08           0                0
4   0.207299  7.927008e-01           1                0
5   0.001227  9.987728e-01           1                1
6   0.096810  9.031903e-01           1                1
7   0.007691  9.923086e-01           1                1
8   1.000000  7.828193e-11           0                0
9   0.057154  9.428460e-01           1                1
```

In [26]:
```
# generate table of predictions vs actual
print("Confusion Matrix")
print(pd.crosstab(preds, test_labels))
```

```
Confusion Matrix
col_0   0   1
row_0
0      48   3
1       5  87
```

In [27]:
```
from sklearn import metrics
# view the confusion matrix
print(metrics.confusion_matrix(y_true = test_labels, # True labels
                               y_pred = preds))      # Predicted labels
```

```
[[48  5]
 [ 3 87]]
```

```
In [28]: # Computing Accuracy, Recall, Precision, and Other Metrics
         print("Accuracy")
         print(log_regress.score(X=test_set,
                                 y=test_labels))
```

```
Accuracy
0.9440559440559441
```

```
In [29]: # view summary of common classification metrics
         print("---Metrics---")
         print(metrics.classification_report(
                 y_true = test_labels,
                 y_pred = preds
         ))
```

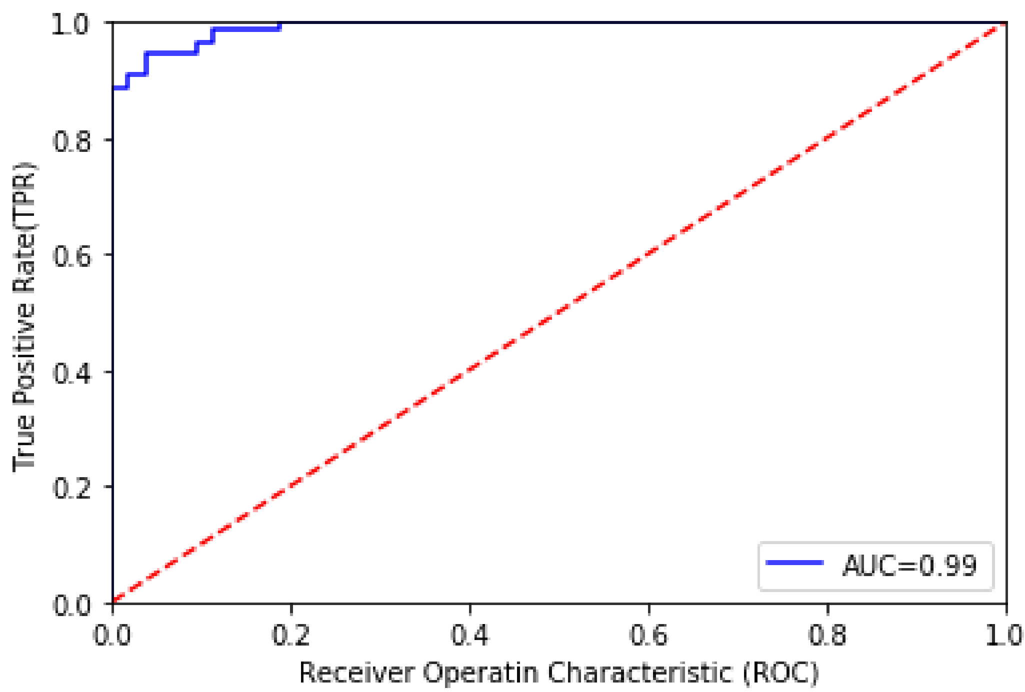
```
---Metrics---
              precision    recall  f1-score   support

           0       0.94      0.91      0.92        53
           1       0.95      0.97      0.96        90

   micro avg       0.94      0.94      0.94       143
   macro avg       0.94      0.94      0.94       143
weighted avg       0.94      0.94      0.94       143
```

```
In [31]: from sklearn.metrics import roc_curve, auc
         #--find the predicted probabilities using the test set
         probs = log_regress.predict_proba(test_set)
         preds = probs[:,1]

         #find the FPR, TPR, and threshold
         fpr, tpr, threshold = roc_curve(test_labels, preds)
         print(fpr)
         print(tpr)
         print(threshold)
```

```
[0.         0.         0.         0.01886792 0.01886792 0.03773585
 0.03773585 0.09433962 0.09433962 0.11320755 0.11320755 0.18867925
 0.18867925 1.        ]
[0.         0.01111111 0.88888889 0.88888889 0.91111111 0.91111111
 0.94444444 0.94444444 0.96666667 0.96666667 0.98888889 0.98888889
 1.         1.        ]
[1.99999109e+00 9.99991091e-01 9.36981948e-01 9.18023512e-01
 9.03190293e-01 8.58497024e-01 8.48205648e-01 5.43404089e-01
 5.25939874e-01 3.71991696e-01 2.71106136e-01 1.21481722e-01
 1.18623350e-01 1.30886736e-21]
```

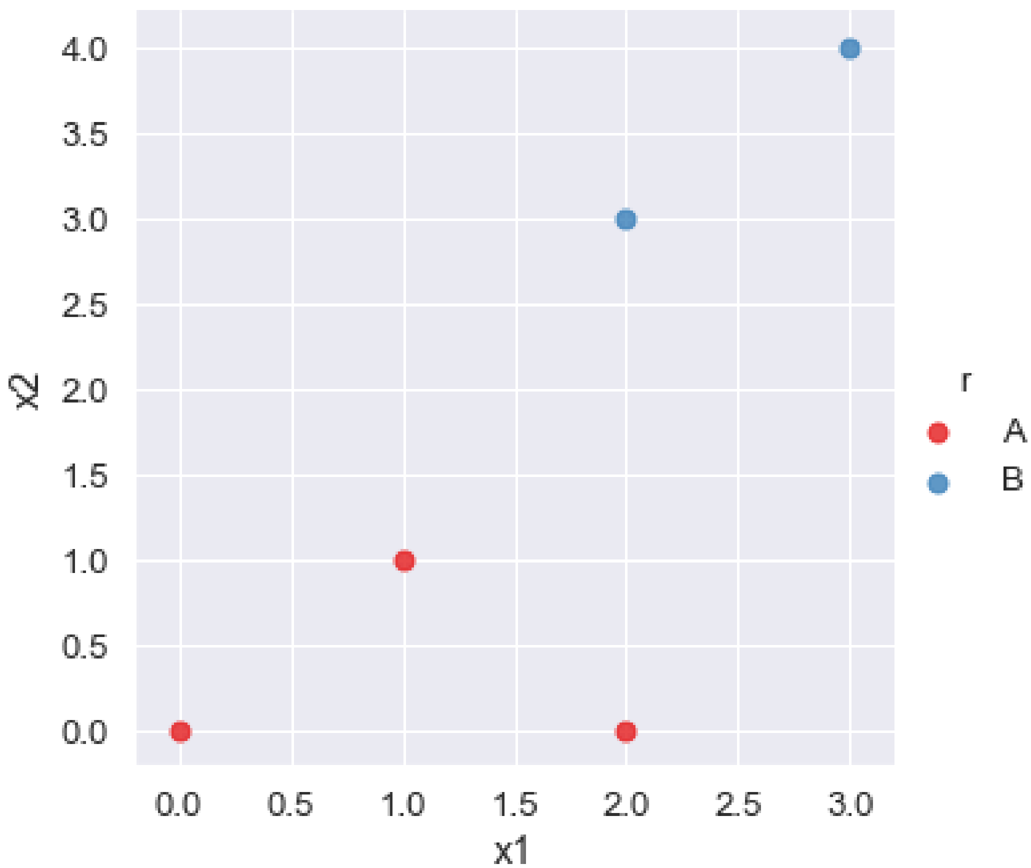
In [32]:
```
#find the area under the curve
roc_auc = auc(fpr, tpr)
```

In [33]:
```
import matplotlib.pyplot as plt
plt.plot(fpr, tpr, 'b', label = 'AUC=%0.2f' % roc_auc)
plt.plot([0,1], [0,1],'r--')
plt.xlim([0,1])
plt.ylim([0,1])
plt.ylabel('True Positive Rate(TPR)')
plt.xlabel('Receiver Operatin Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
```



In [34]:
```
%matplotlib inline
import pandas as pd
import numpy as np
import seaborn as sns; sns.set(font_scale=1.2)
import matplotlib.pyplot as plt
```

In [38]:
```
data=pd.read_csv('svm.csv')
sns.lmplot('x1', 'x2',
           data=data,
          hue='r',
          palette='Set1',
          fit_reg=False,
          scatter_kws={"s": 50});
```




```
In [39]: from sklearn import svm
         # Convertin the Columns as Matrices
         points = data[['x1', 'x2']].values
         result = data['r']

         clf = svm.SVC(kernel='linear')
         clf.fit(points, result)

         print('Vector of weights (w)=',clf.coef_[0])
         print('b=', clf.intercept_[0])
         print('Indices of support vectors = ', clf.support_)
         print('Support vectors = ', clf.support_vectors_)
         print('Number of support vectors for each class = ', clf.n_support_)
         print('Coefficients of the support vector in the decision function=', np.abs(clf.dual_
```

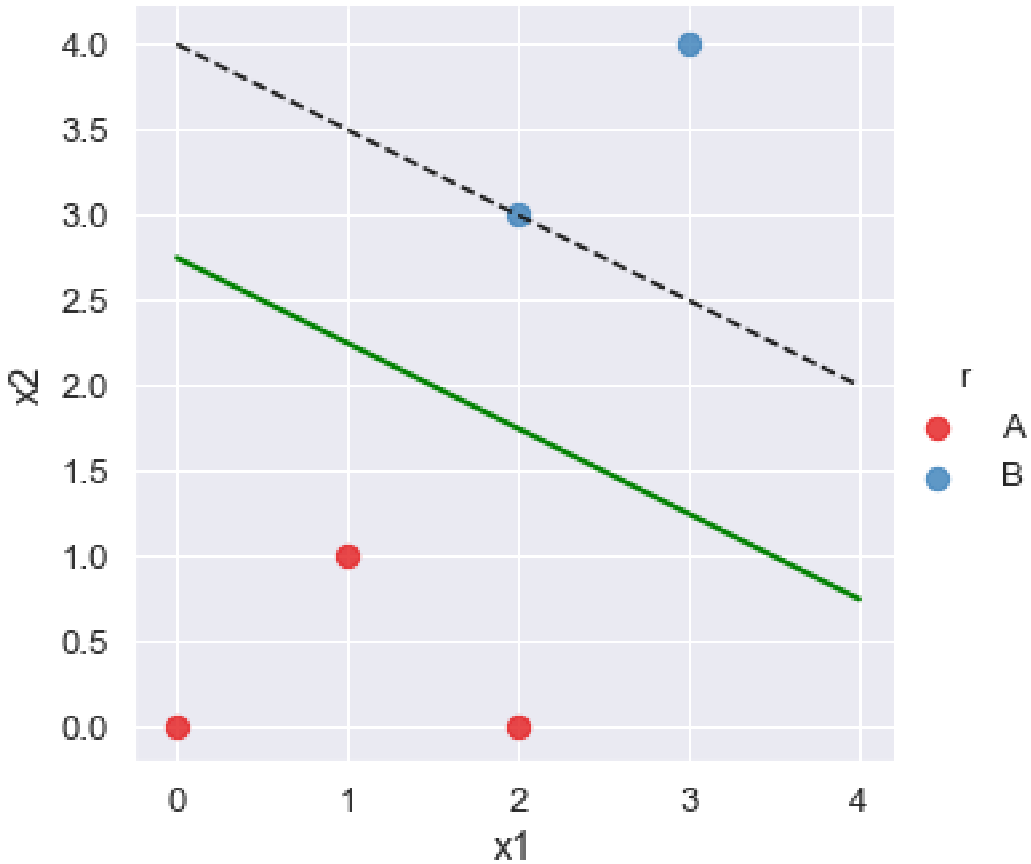
```
Vector of weights (w)= [0.4 0.8]
b= -2.2
Indices of support vectors =  [1 2]
Support vectors =  [[1. 1.]
```

```
 [2. 3.]]
Number of support vectors for each class =  [1 1]
Coefficients of the support vector in the decision function= [[0.4 0.4]]
```

In [41]:
```
# w is the vector of weights
w = clf.coef_[0]
# find the slope of the hyperplane
slope = -w[0]/w[1]
b = clf.intercept_[0]
# find the coordinates for the hyperplane
xx = np.linspace(0, 4)
yy = slope * xx - (b/w[1])
#plot the margins
s = clf.support_vectors_[0] # first support vector
yy_down = slope ** xx + (s[1] - slope * s[0])
s = clf.support_vectors_[-1] # first support vector
yy_up = slope * xx + (s[1] - slope * s[0])

#plot the points
sns.lmplot('x1', 'x2', data =data, hue='r', palette='Set1', fit_reg=False, scatter_kws
#plot the hyperplane
plt.plot(xx, yy, linewidth=2, color='green');

# plot the 2 margins
plt.plot(xx, yy_down, 'k--')
plt.plot(xx, yy_up, 'k--')
```

```
d:\dev\python\python36\lib\site-packages\ipykernel_launcher.py:11: RuntimeWarning: invalid valu
  # This is added back by InteractiveShellApp.init_path()
```

Out[41]:
```
[<matplotlib.lines.Line2D at 0x21fee52f710>]
```

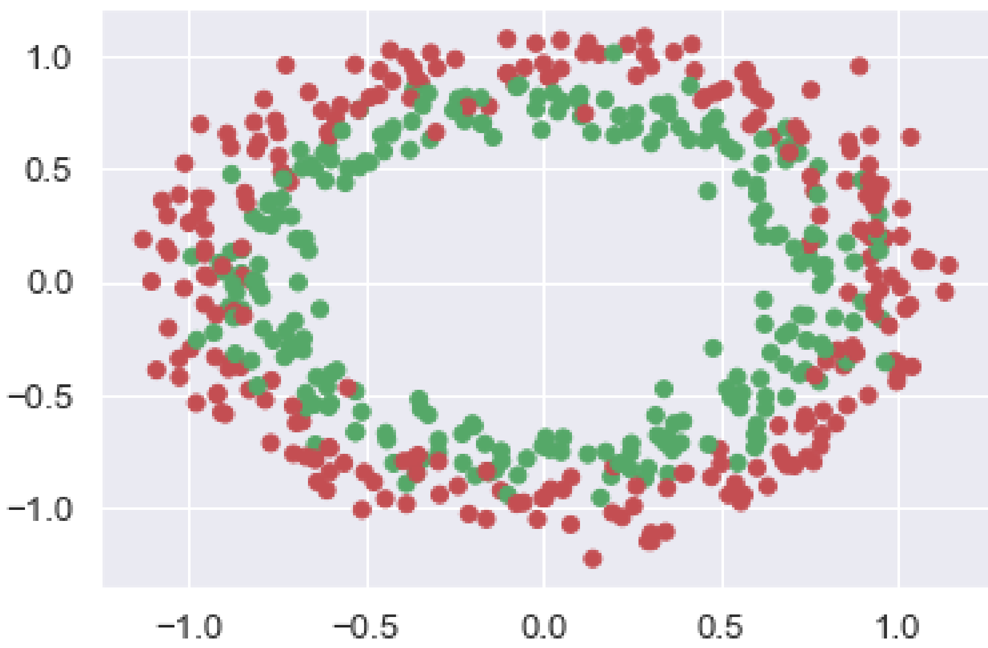


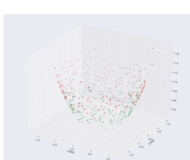

```
In [44]: print(clf.predict([[3,3]])[0])
         print(clf.predict([[4,0]])[0])
         print(clf.predict([[2,2]])[0])
         print(clf.predict([[1,2]])[0])
```

```
B
A
B
A
```

```
In [49]: # Adding a Third Dimension
         %matplotlib inline
         from mpl_toolkits.mplot3d import Axes3D
         import matplotlib.pyplot as plt
         import numpy as np
         from sklearn.datasets import make_circles
         #X is features and c is the class labels
         X, c = make_circles(n_samples = 500, noise = 0.09)
```

```
rgb = np.array(['r', 'g'])
plt.scatter(X[:, 0], X[:, 1], color=rgb[c])
plt.show()

fig = plt.figure(figsize=(18,15))
ax = fig.add_subplot(111, projection='3d')
z = X[:,0]**2 + X[:,1]**2
ax.scatter(X[:, 0], X[:, 1], z, color=rgb[c])
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.show()
```





```
In [53]: # combine X (x-axis, y-axis) and z into single ndarray
         features = np.concatenate((X,z.reshape(-1,1)), axis=1)

         # use SVM for training
```

```
from sklearn import svm

clf = svm.SVC(kernel = 'linear')
clf.fit(features, c)
```

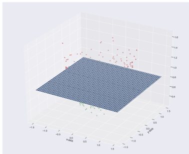
```
Out[53]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
In [55]: x3=lambda x,y: (-clf.intercept_[0]-clf.coef_[0][0]*x-clf.coef_[0][1]*y)/clf.coef_[0][
```

```
In [57]: tmp=np.linspace(-1.5,1.5,100)
         x,y=np.meshgrid(tmp, tmp)
```

```
In [59]: ax.plot_surface(x,y,x3(x,y))
         plt.show()
```

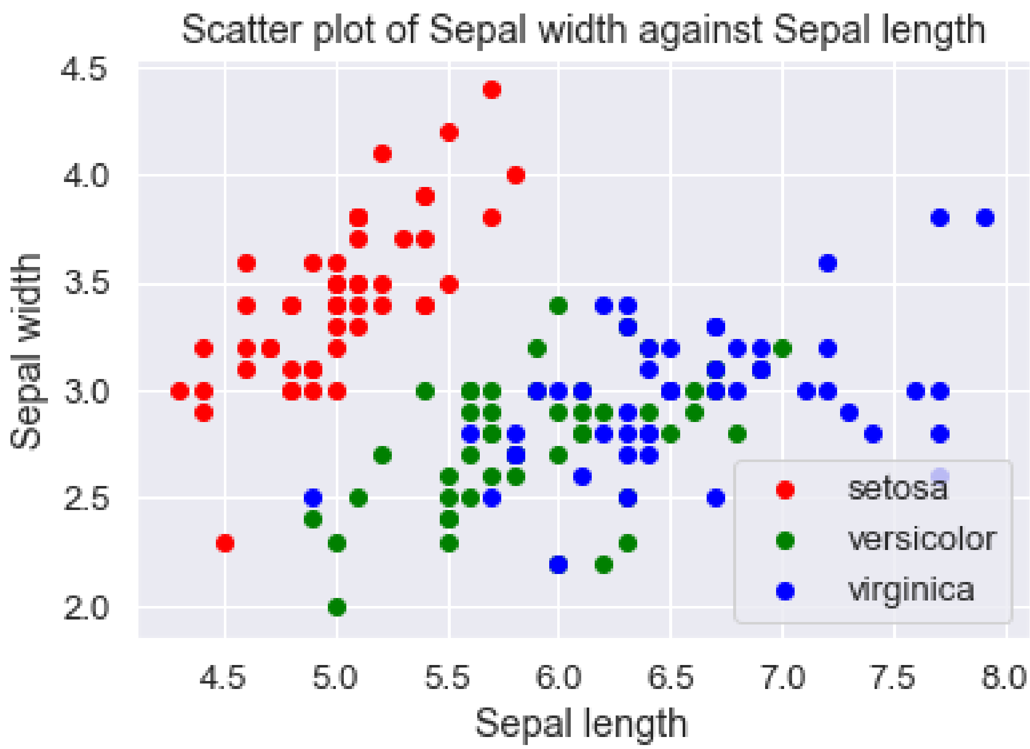
```
In [64]: from mpl_toolkits.mplot3d import Axes3D
         import matplotlib.pyplot as plt
         import numpy as np
         from sklearn.datasets import make_circles

         #--X is features and c is the class labels
         X, c = make_circles(n_samples=500, noise=0.09)
         z = X[:,0]**2+X[:,1]**2

         rgb = np.array(['r', 'g'])

         fig = plt.figure(figsize=(18,15))
         ax=fig.add_subplot(111, projection='3d')
         ax.scatter(X[:,0], X[:,1], z, color=rgb[c])
         plt.xlabel("x-axis")
         plt.ylabel("y-axis")

         # combine X (x-axis, y-axis) and z into single ndarray
         features = np.concatenate((X,z.reshape(-1,1)), axis=1)

         # use SVM for training
         from sklearn import svm

         clf = svm.SVC(kernel = 'linear')
         clf.fit(features, c)
         x3 = lambda x,y: (-clf.intercept_[0]-clf.coef_[0][0]*x-clf.coef_[0][1]*y)/clf.coef_[0
         tmp = np.linspace(-1.5,1.5,100)
         x,y=np.meshgrid(tmp, tmp)

         ax.plot_surface(x, y, x3(x,y))
         plt.show()
```



In [65]:
```
%matplotlib inline
import pandas as pd
import numpy as np
from sklearn import svm, datasets
import matplotlib.pyplot as plt

iris = datasets.load_iris()
print(iris.data[0:5])
print(iris.feature_names)
print(iris.target[0:5])
print(iris.target_names)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
[0 0 0 0 0]
['setosa' 'versicolor' 'virginica']
```

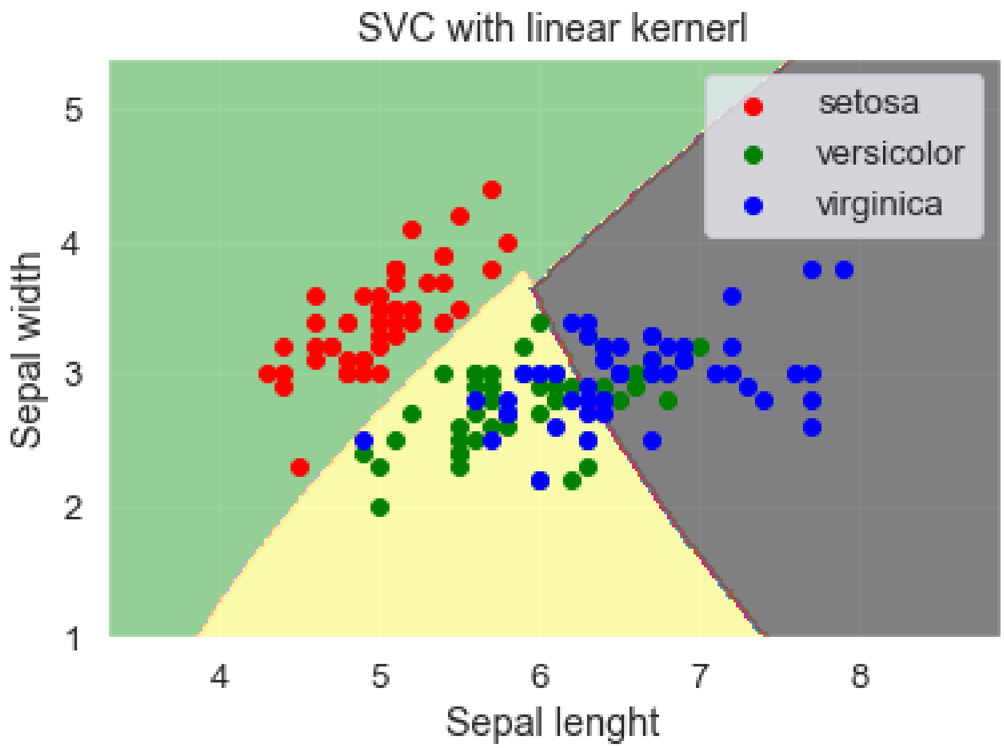
In [84]:
```
X = iris.data[:,:2] # take the first two features
y = iris.target

colors = ['red', 'green', 'blue']
for color, i, target in zip(colors, [0,1,2], iris.target_names):
    plt.scatter(X[y==i, 0], X[y==i, 1], color=color, label=target)

plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.legend(loc='best', shadow=False, scatterpoints=1)

plt.title('Scatter plot of Sepal width against Sepal length')
plt.show()

C = 1 # SVM regularization parameter
clf = svm.SVC(kernel='poly', degree=2,  C=C, gamma='auto').fit(X, y)
title = 'SVC with linear kernerl'
```



In [85]:
```
# min and max for the first feature
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max()+1
# min and max for the second feature
y_min, y_max = X[:, 1].min() -1, X[:, 1].max()+1
# step size in the mesh
h = (x_max / x_min)/100
# make predictions for each of the points in xx, yy
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h)
                    )
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# draw the result using a color plot
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Accent, alpha=0.8)

# plot the training points
colors = ['red', 'green', 'blue']
for color, i, target in zip(colors, [0,1,2], iris.target_names):
    plt.scatter(X[y==i, 0], X[y==i, 1], color=color, label=target)

plt.xlabel('Sepal lenght')
plt.ylabel('Sepal width')
```

```
        plt.title(title)
        plt.legend(loc='best', shadow=False, scatterpoints=1)
```

Out[85]: <matplotlib.legend.Legend at 0x21fee83e320>




```
In [76]: predictions = clf.predict(X)
         print(np.unique(predictions, return_counts=True))
```

```
(array([0, 1, 2]), array([50, 53, 47], dtype=int64))
```

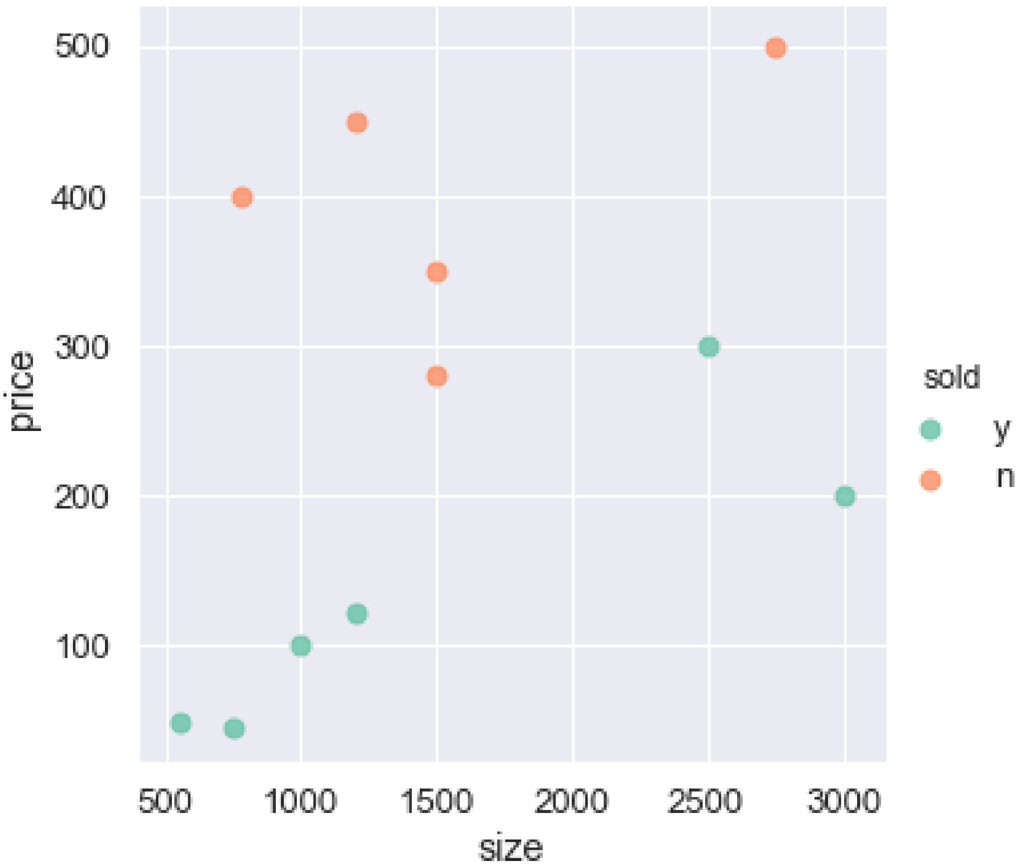
```
In [77]: # Using SVM for Real-Life Problems
```

```
In [86]: #matplotlib inline
         import pandas as pd
         import numpy as np
         from sklearn import svm
         import matplotlib.pyplot as plt
         import seaborn as sns; sns.set(font_scale=1.2)
```

```
In [87]: data = pd.read_csv('house_sizes_prices_svm.csv')
```

```
In [89]: sns.lmplot('size', 'price',
                    data=data,
```

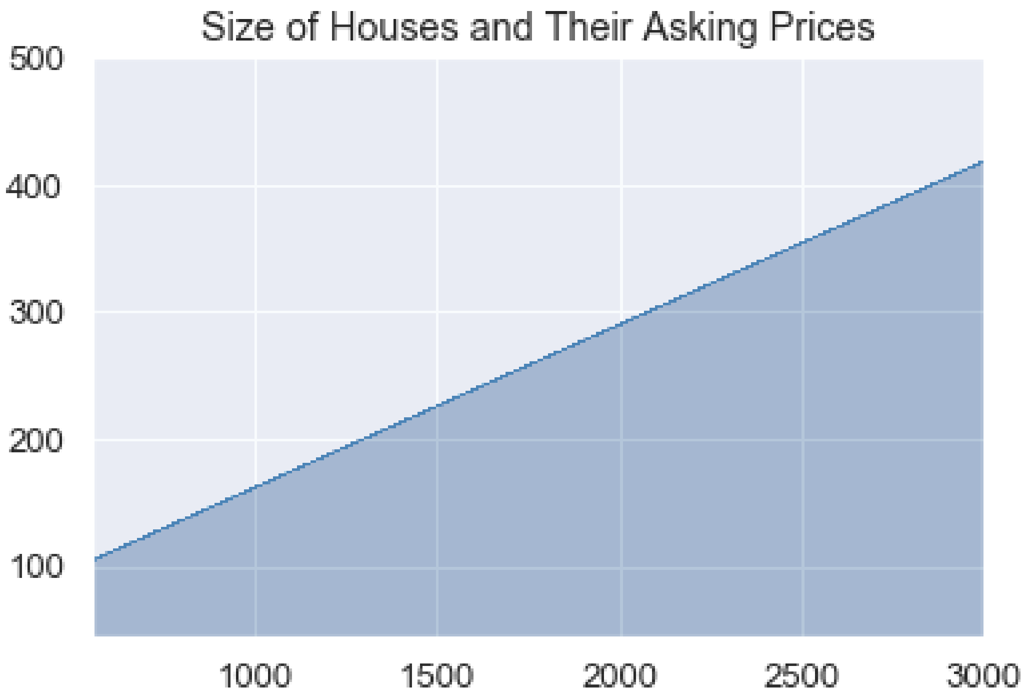
```
                 hue='sold',
                 palette='Set2',
                 fit_reg=False,
                 scatter_kws={"s": 50}
                );
```




```
In [90]: X = data[['size', 'price']].values
         y = np.where(data['sold']=='y', 1, 0) #1 for Y and 0 for N
         model = svm.SVC(kernel='linear').fit(X, y)

In [94]: x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
         y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
         h = (x_max / x_min) / 20
         xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                              np.arange(y_min, y_max, h)
                             )
         Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
         Z=Z.reshape(xx.shape)
         plt.contourf(xx, yy, Z, cmap=plt.cm.Blues, alpha=0.3)
```

```python
plt.xlabel=('Size of house')
plt.ylabel=('Asking price(100s)')
plt.title("Size of Houses and Their Asking Prices")
```

Out[94]: Text(0.5, 1.0, 'Size of Houses and Their Asking Prices')



In [ ]: