

import pandas as pd

import numpy as np

import sklearn

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

pd.read\_csv(<filename>)

df.to\_csv(<filename>)

df.describe()

df.info()

df.dropna()

df.reset\_index()

df.set\_value(row, column, new\_value)

df.select\_dtypes(exclude=["number","bool\_","object\_"])

df.loc[:,1:3] #select columns 1 to 3

train.fillna(train.mean(), inplace=True)

**Linear regression**

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

from sklearn.model\_selection import train\_test\_split

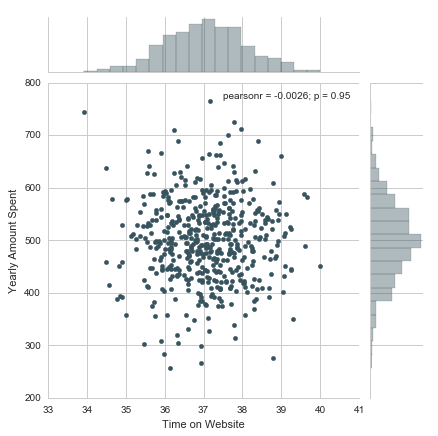
**Plots**

sns.set(rc={'figure.figsize':(11.7,8.27)}) #Set figure size

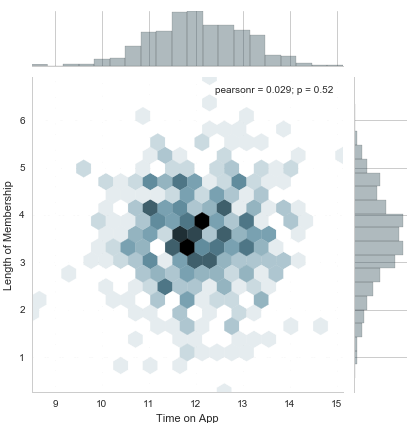
sns.set\_palette("GnBu\_d")

sns.set\_style('whitegrid')

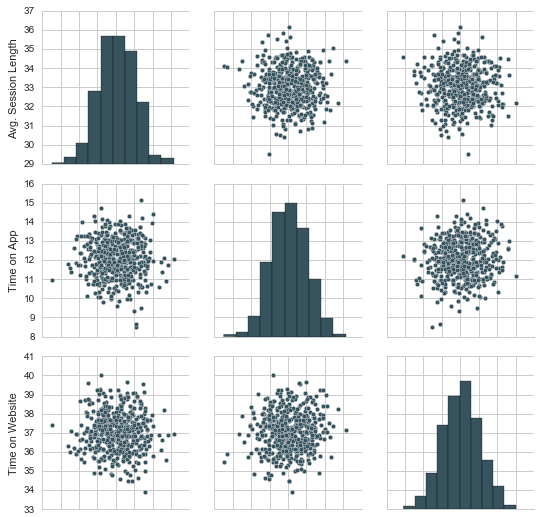
sns.jointplot(x='Time on Website',y='Yearly Amount Spent',data=df)



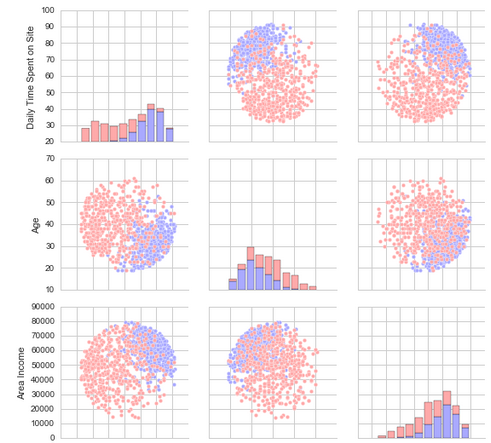
sns.jointplot(x='Time on App',y='Length of Membership',kind='hex',data=df)



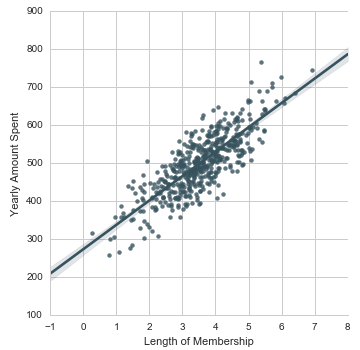
sns.pairplot(df)



sns.pairplot(df,hue='Clicked on Ad',palette='bwr')



sns.lmplot(x='Length of Membership',y='Yearly Amount Spent',data=customers)



**Train and test set**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=101)

**Fit the model**

lm = LinearRegression()

lm.fit(X\_train,y\_train)

print('Coefficients: \n', lm.coef\_)

**Predicting test data**

predictions = lm.predict(X\_test)

coefficients = pd.DataFrame(lm.coef\_,X.columns)

coefficients.columns = ['Coefficient']

coefficients

**Evaluate the model**

**Create a scatterplot of the real test values versus the predicted values.**

plt.scatter(y\_test,predictions)

plt.xlabel('Y Test')

plt.ylabel('Predicted Y')

**Calculate the Mean Absolute Error, Mean Squared Error, and the Root Mean Squared Error.**

print('MAE:', metrics.mean\_absolute\_error(y\_test, predictions))

print('MSE:', metrics.mean\_squared\_error(y\_test, predictions))

print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y\_test, predictions)))

**Plot a histogram of the residuals and make sure it looks normally distributed**

sns.distplot((y\_test-predictions),bins=50);

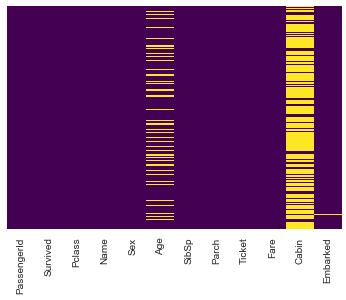
**Logistic regression**

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

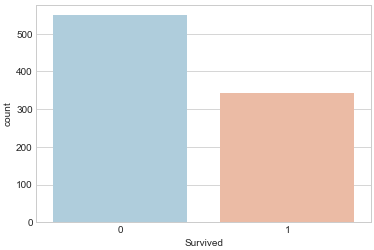
from sklearn.model\_selection import train\_test\_split

**Plots**

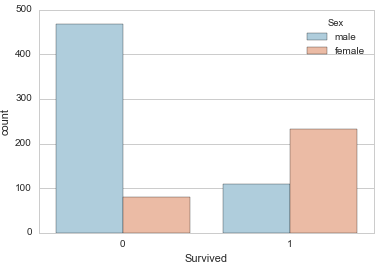
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis') #Missing data plot

sns.set\_style('whitegrid')

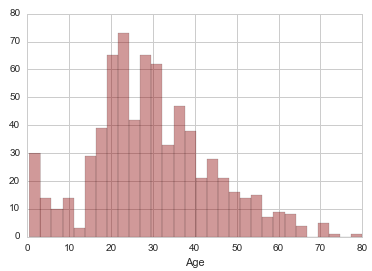
sns.countplot(x='Survived',data=df,palette='RdBu\_r')



sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu\_r')



sns.distplot(df['Age'].dropna(),kde=False,color='darkred',bins=30)



**Imputing**

We impute the missing age values. We do this by in this case using the average age of each of the 3 Pclasses.

def impute\_age(cols):

Age = cols[0]

Pclass = cols[1]

if pd.isnull(Age): #If the data for this row is missing

if Pclass == 1:

return 37 #Average age for Pclass=1

elif Pclass == 2:

return 29 #Average age for Pclass=2

else:

return 24#Average age for Pclass=3

else:

return Age

Apply the function

df['Age'] = df[['Age','Pclass']].apply(impute\_age,axis=1)

**Drop column**

df.drop('Cabin',axis=1,inplace=True)

**Make dataframe numerical**

Make dummy variables

sex = pd.get\_dummies(df['Sex'],drop\_first=True)

embark = pd.get\_dummies(df['Sex'],drop\_first=True)

Drop non-numerical columns

df.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)

**Concatenate the dataframe with the two new variables**

train = pd.concat([df,sex,embark],axis=1)

**Train and test set**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.drop('Survived',axis=1), df['Survived'], test\_size=0.30, random\_state=101)

**Fit the model**

logmodel = LogisticRegression()

logmodel.fit(X\_train,y\_train)

**Predicting test data**

predictions = logmodel.predict(X\_test)

**Evaluate the model**

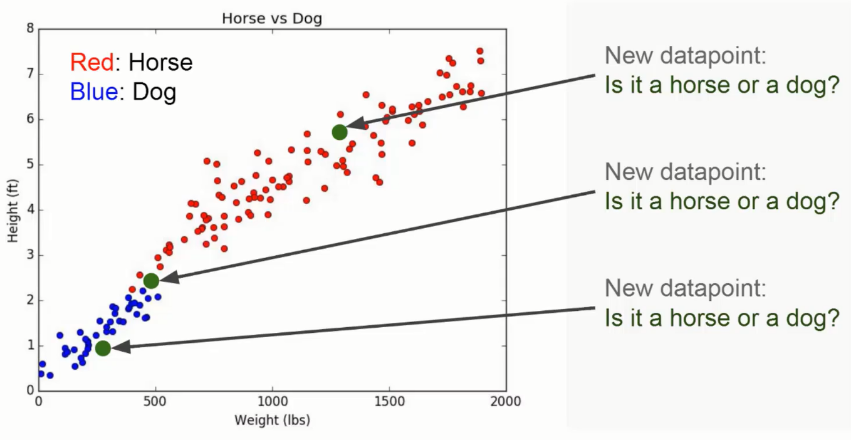
**print(classification\_report(y\_test,predictions))**

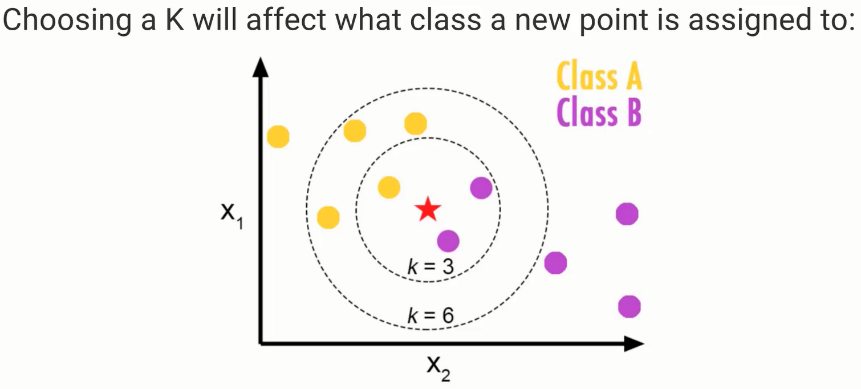
**K-Nearest Neighbours**

**from sklearn.neighbors import KNeighborsClassifier**

**from sklearn.metrics import classification\_report,confusion\_matrix**

**from sklearn.preprocessing import StandardScaler**





A higher K will lead to bigger radius of points that is looked at. Radius k=3 means the point belongs to Class B, radius k=6 means the point belongs to Class A.

**Standardize the Variables**

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have a much larger effect on the distance between the observations, and hence on the KNN classifier, than variables that are on a small scale.

scaler = StandardScaler()

scaler.fit(df.drop('TARGET CLASS',axis=1))

scaled\_features = scaler.transform(df.drop('TARGET CLASS',axis=1)) #Create arrays with the scaled features

df\_feat = pd.DataFrame(scaled\_features,columns=df.columns[:-1]) #Create a dataframe of the arrays and add column names (exclude the TARGET CLASS though).

**Train and test set**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(scaled\_features,df['TARGET CLASS'], test\_size=0.30)

**Fit the model**

knn = KNeighborsClassifier(n\_neighbors=1) #K=1

knn.fit(X\_train,y\_train)

pred = knn.predict(X\_test)

**Predicting test data**

print(confusion\_matrix(y\_test,pred))

print(classification\_report(y\_test,pred))

**Choosing a K Value**

error\_rate = []

for i in range(1,40):

knn = KNeighborsClassifier(n\_neighbors=i)

knn.fit(X\_train,y\_train)

pred\_i = knn.predict(X\_test)

error\_rate.append(np.mean(pred\_i != y\_test))

plt.figure(figsize=(10,6))

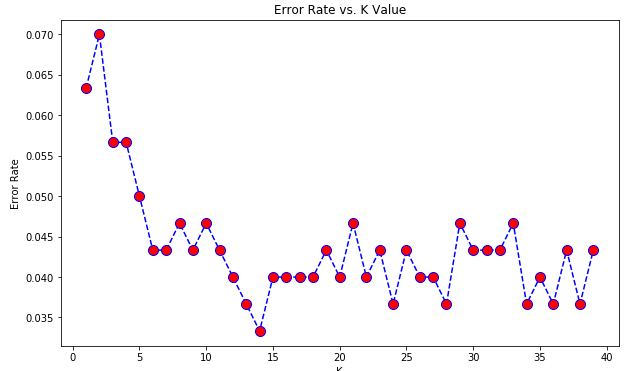
plt.plot(range(1,40),error\_rate,color='blue', linestyle='dashed', marker='o',

markerfacecolor='red', markersize=10)

plt.title('Error Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Error Rate')



Choose the k with the lowest error rate.

**Random forests**

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification\_report,confusion\_matrix

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import GridSearchCV

**Train and test set**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=101)

**Plots**

plt.figure(figsize=(10,6))

loans[loans['credit.policy']==1]['fico'].hist(alpha=0.5,color='blue',

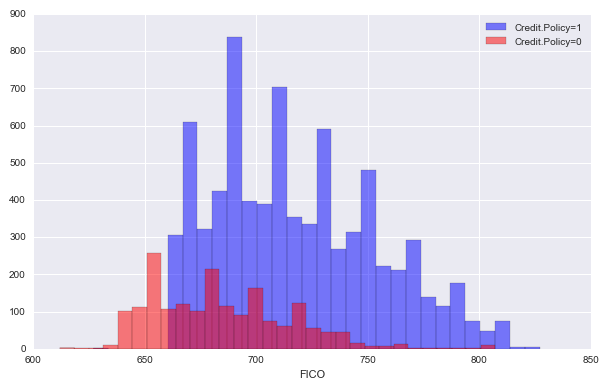
bins=30,label='Credit.Policy=1')

loans[loans['credit.policy']==0]['fico'].hist(alpha=0.5,color='red',

bins=30,label='Credit.Policy=0')

plt.legend()

plt.xlabel('FICO')



**Single decision tree**

dtree = DecisionTreeClassifier()

dtree.fit(X\_train,y\_train)

**Prediction and Evaluation**

predictions = dtree.predict(X\_test)

print(classification\_report(y\_test,predictions))

print(confusion\_matrix(y\_test,predictions))

**Random forests**

rfc = RandomForestClassifier(n\_estimators=100)

rfc.fit(X\_train, y\_train)

**Prediction and Evaluation**

rfc\_pred = rfc.predict(X\_test)

print(confusion\_matrix(y\_test,rfc\_pred))

print(classification\_report(y\_test,rfc\_pred))

print(rfc.get\_params()) #Get parameters

**Gridsearch**

param\_grid = {

'n\_estimators': [200, 350, 500],

'max\_features': ['auto', 'sqrt', 'log2'],

'max\_depth' : [4,5,6,7,8],

'criterion' :['gini', 'entropy']

}

CV\_rfc = GridSearchCV(estimator=rfc, param\_grid=param\_grid, cv= 5,verbose=3)

CV\_rfc.fit(X\_train, y\_train)

CV\_rfc.best\_params\_

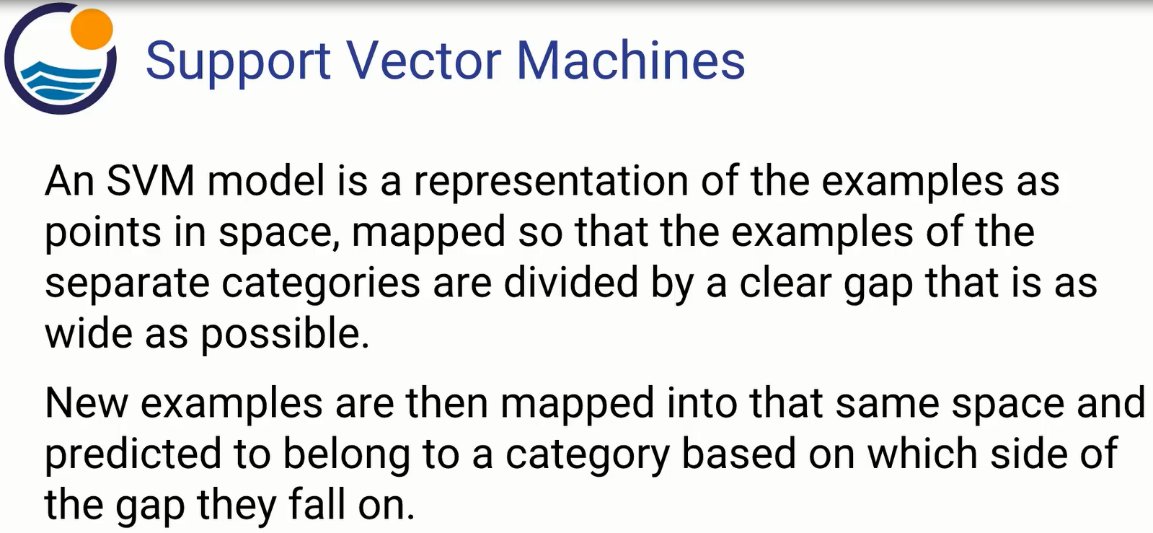
CV\_rfc\_pred = CV\_rfc.predict(test)

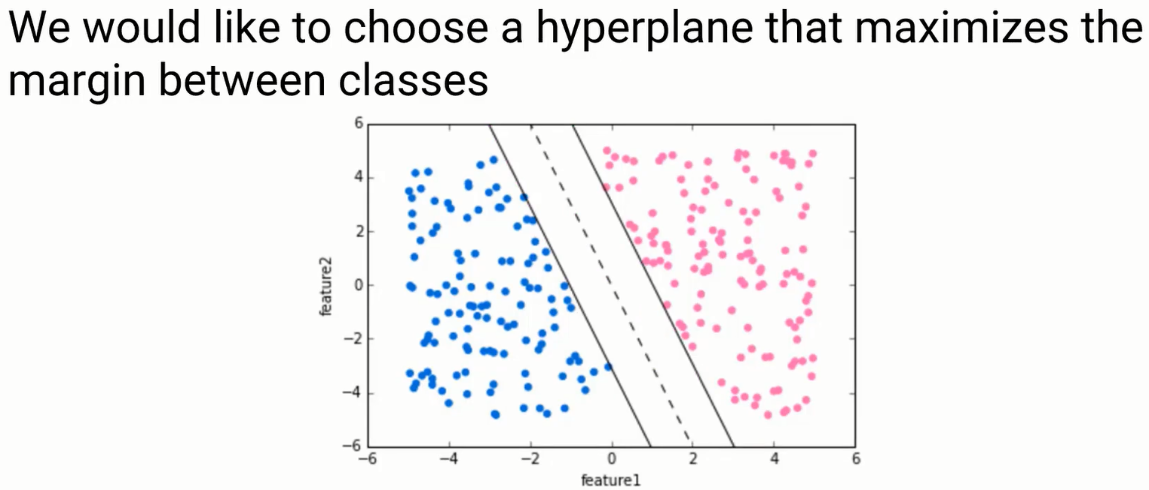
**Support vector machines**

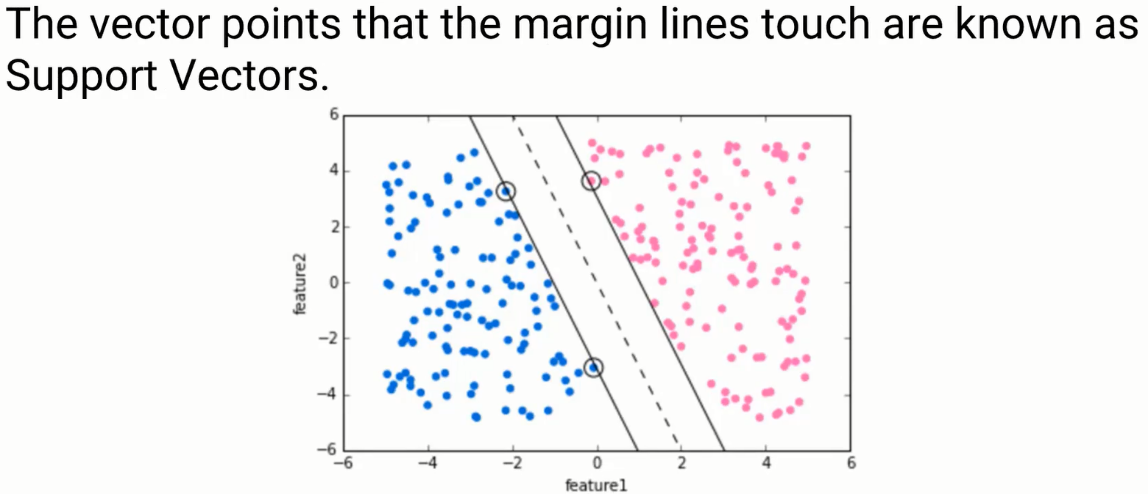
from sklearn.svm import SVC

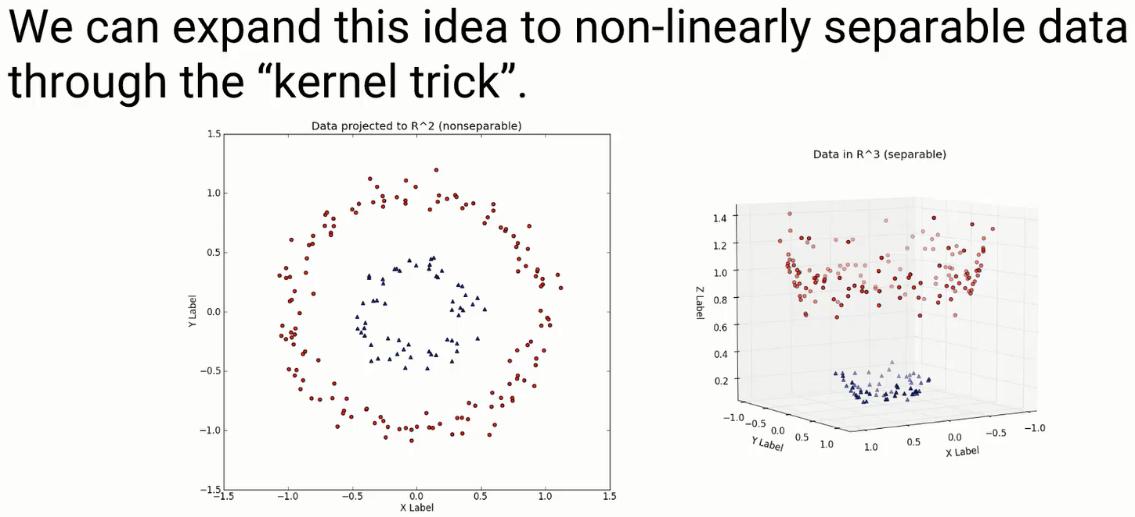
from sklearn.metrics import classification\_report,confusion\_matrix

from sklearn.model\_selection import GridSearchCV

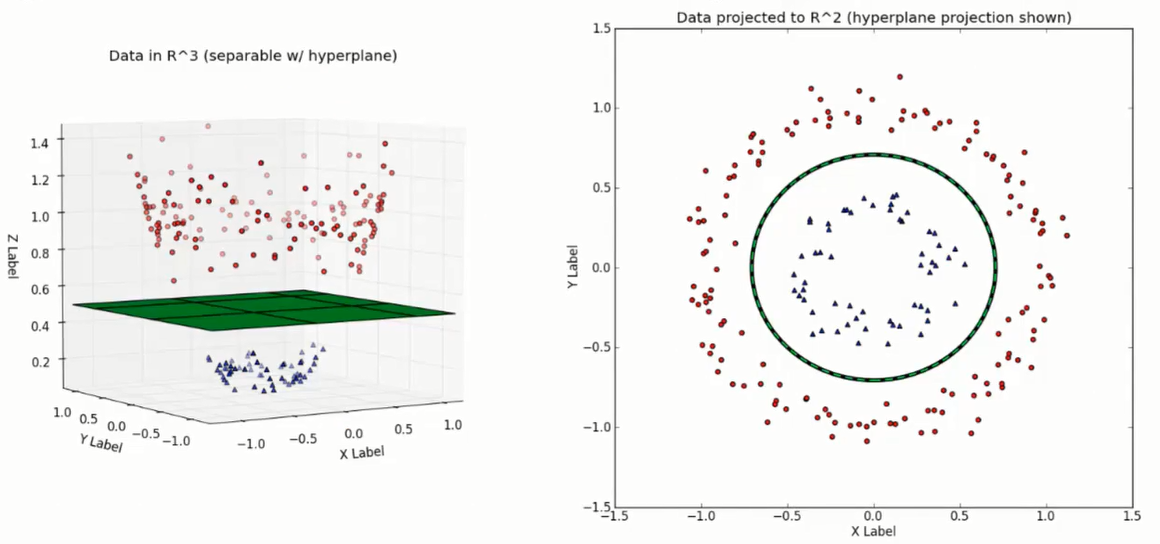






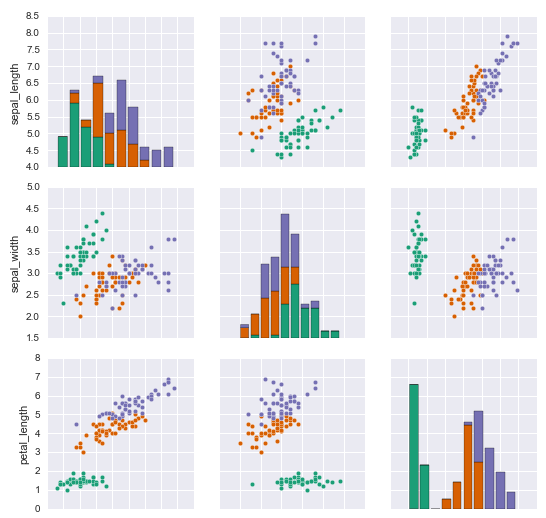


We add a third dimension that now seperates the data.



**Plots**

sns.pairplot(df,hue='species',palette='Dark2')



**Train and test set**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=101)

**Fit the model**

model = SVC()

model.fit(X\_train,y\_train)

**Predictions and Evaluations**

print(confusion\_matrix(y\_test,predictions))

print(classification\_report(y\_test,predictions))

**GridSearch**

GridSearchCV takes a dictionary that describes the parameters that should be tried and a model to train. The grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested.

param\_grid = {'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf']}

You should add refit=True and choose verbose to whatever number you want, higher the number, the more verbose (verbose just means the text output describing the process).

What fit does is a bit more involved then usual. First, it runs the same loop with cross-validation, to find the best parameter combination. Once it has the best combination, it runs fit again on all data passed to fit (without cross-validation), to built a single new model using the best parameter setting.

grid = GridSearchCV(SVC(),param\_grid,refit=True,verbose=3)

grid.fit(X\_train,y\_train)

You can inspect the best parameters found by GridSearchCV in the best\_params\_ attribute, and the best estimator in the best\_estimator\_ attribute:

grid.best\_params\_

grid.best\_estimator\_

**Predictions and Evaluations**

grid\_predictions = grid.predict(X\_test)

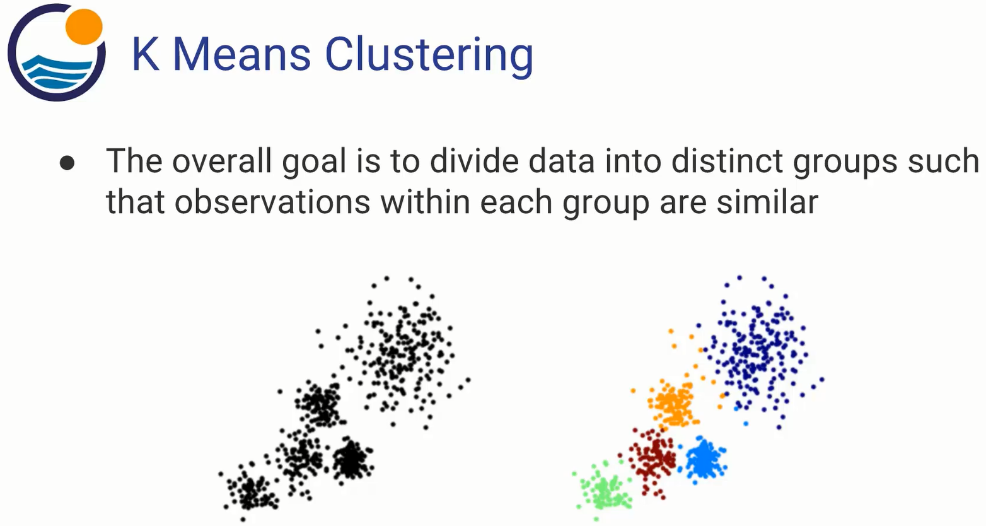
print(confusion\_matrix(y\_test,grid\_predictions))

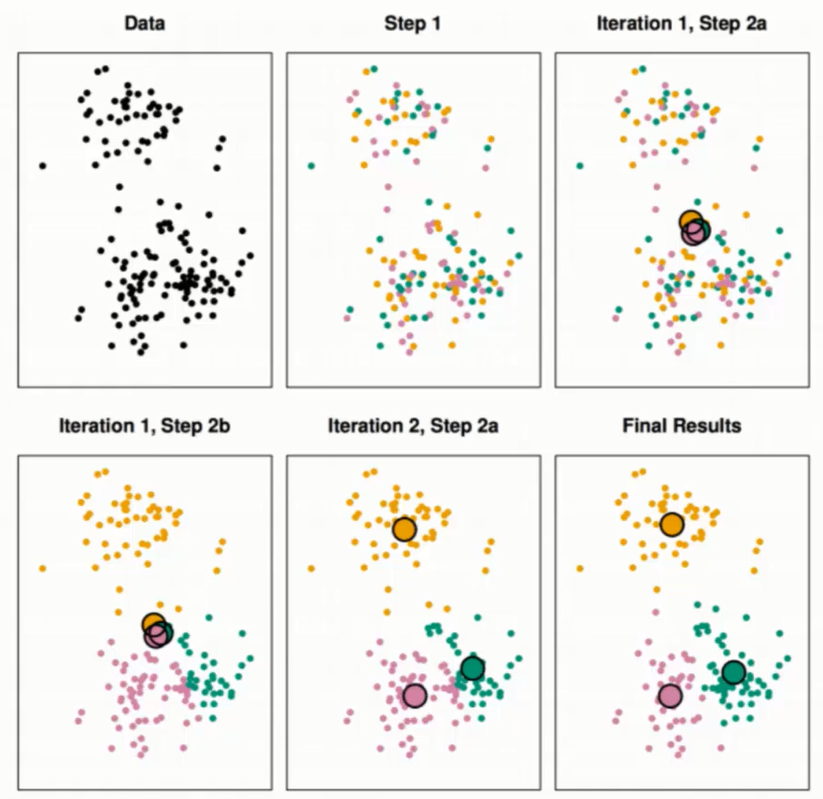
print(classification\_report(y\_test,grid\_predictions))

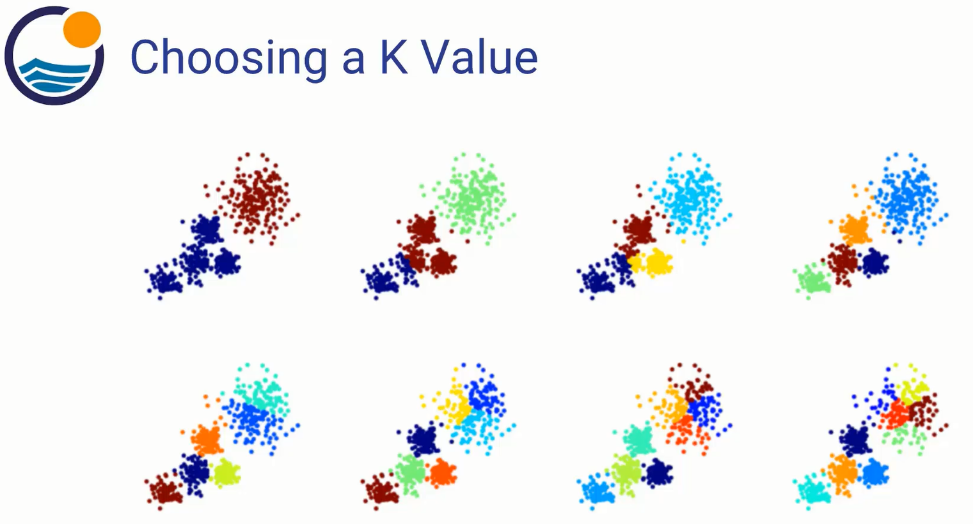
**K Means Clustering**

from sklearn.cluster import KMeans

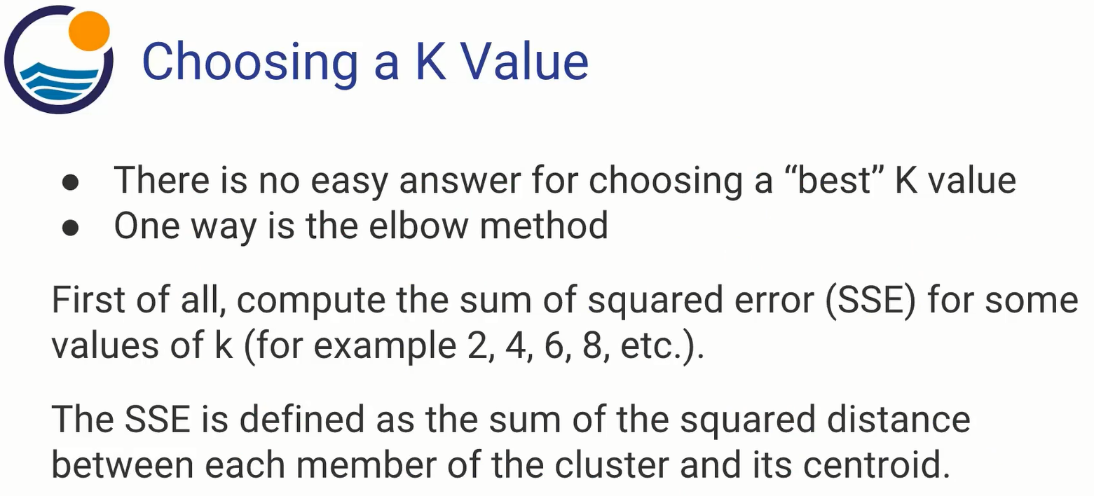
from sklearn.metrics import confusion\_matrix,classification\_report

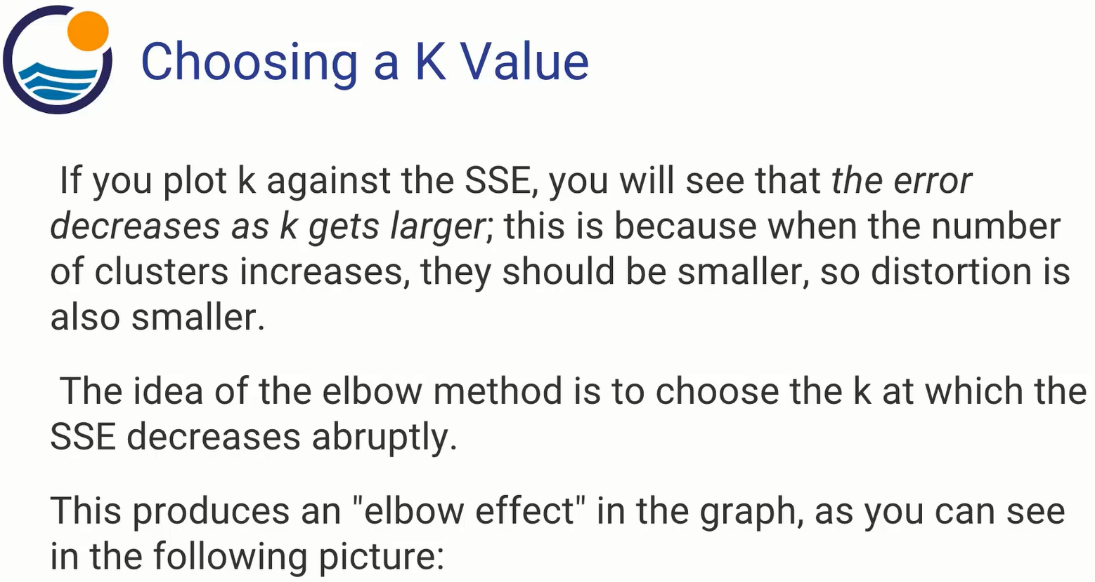


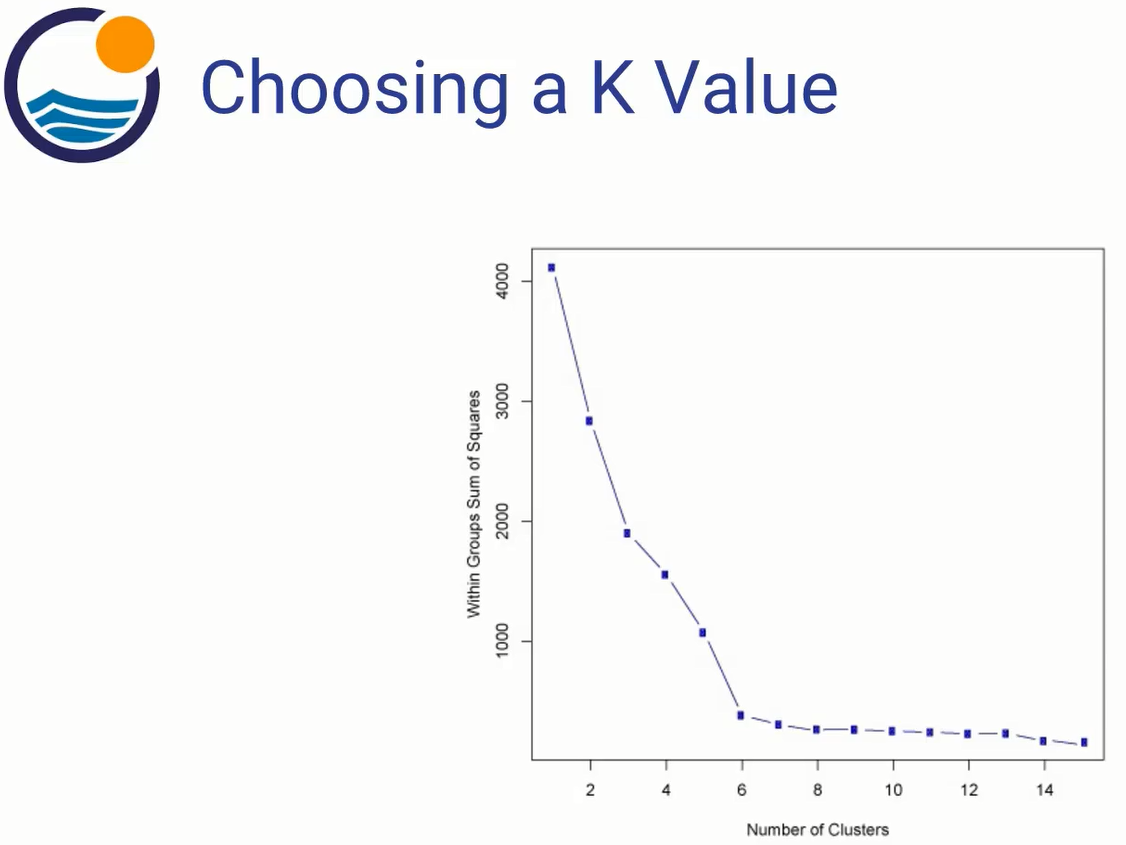




The K value decides how many clusters are created.





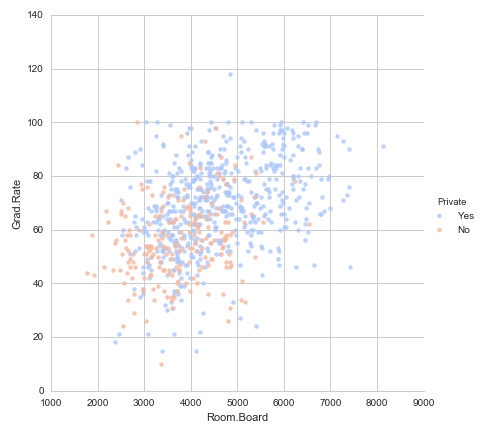


**Plots**

sns.set\_style('whitegrid')

sns.lmplot('Room.Board','Grad.Rate',data=df, hue='Private',

palette='coolwarm',size=6,aspect=1,fit\_reg=False)



**Fit the model**

kmeans = KMeans(n\_clusters=2) #Two clusters

kmeans.fit(df.drop('Private',axis=1)) #Drop the variable that we try to predict

kmeans.cluster\_centers\_ # Find the cluster centers

kmeans.labels\_ #Give the cluster for each datapoint

**Evaluate the model (given you have the luxury of having correct outcome, in this case ‘Private’.**

print(confusion\_matrix(df['Private'],kmeans.labels\_))

print(classification\_report(df['Private'],kmeans.labels\_))

**Principle component analysis**

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

**ALWAYS scale the data**

scaler = StandardScaler()

scaler.fit(df)

scaled\_data = scaler.transform(df)

**Fit the model**

pca = PCA(n\_components=2) #Specify the amount of components

pca.fit(scaled\_data)

**Apply the rotation and dimensionality reduction**

x\_pca = pca.transform(scaled\_data)

x\_pca.shape

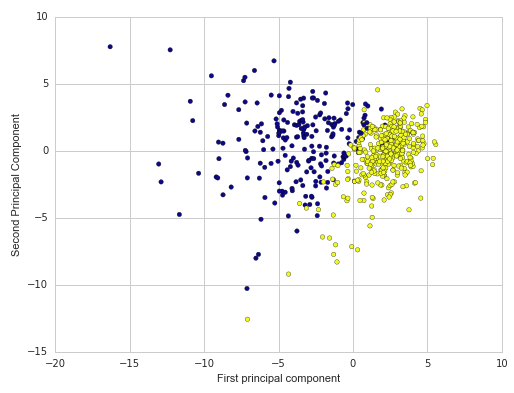
**Plot the components**

plt.figure(figsize=(8,6))

plt.scatter(x\_pca[:,0],x\_pca[:,1],c=cancer['target'],cmap='plasma')

plt.xlabel('First principal component')

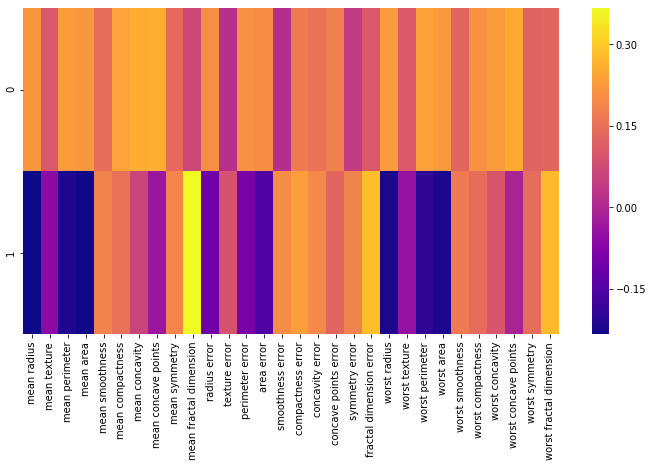
plt.ylabel('Second Principal Component')



**Interpreting the components**

pca.components\_ #Show the arrays of the two components

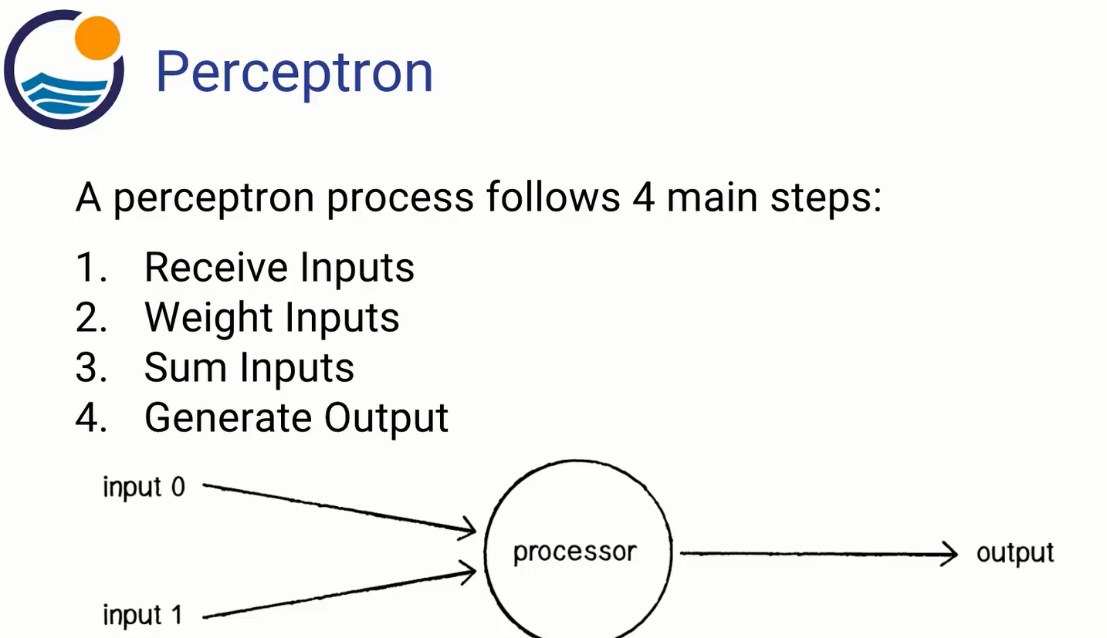
df\_comp = pd.DataFrame(pca.components\_,columns=cancer['feature\_names'])

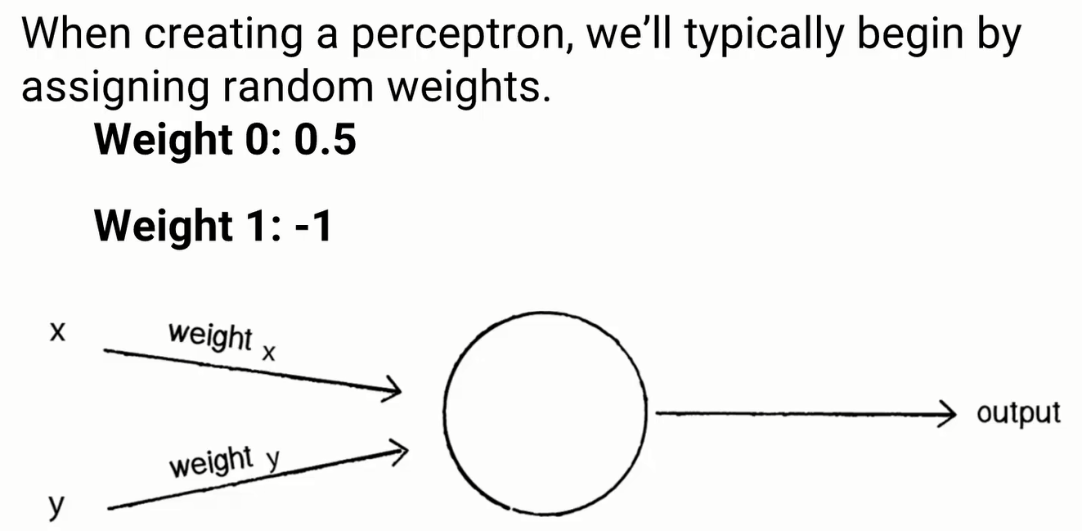


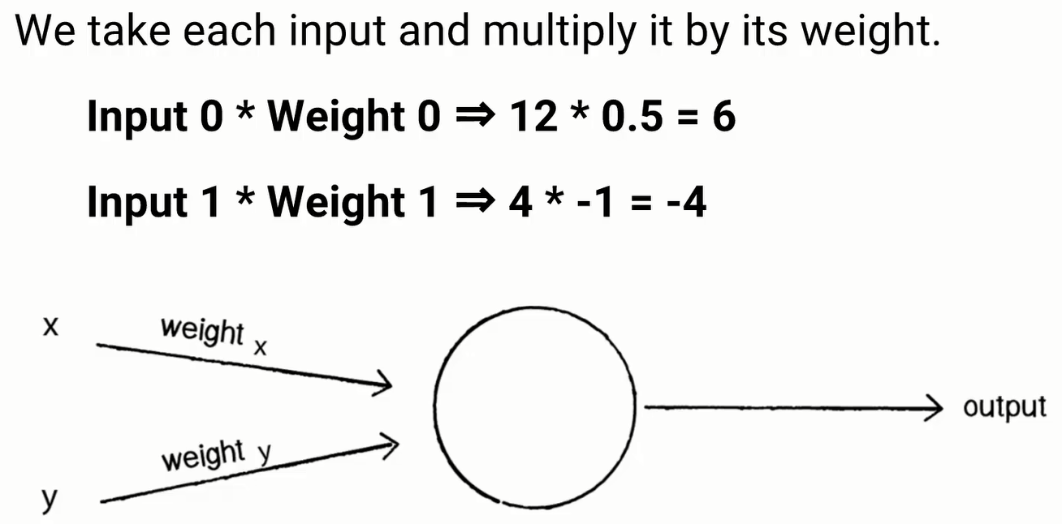
This heatmap and the color bar basically represent the correlation between the various feature and the principal component itself.

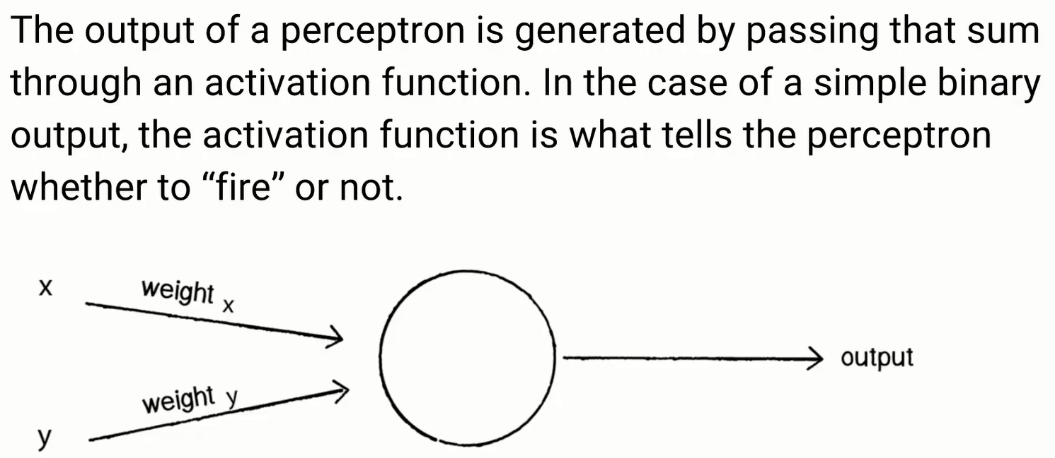
**Neural networks**

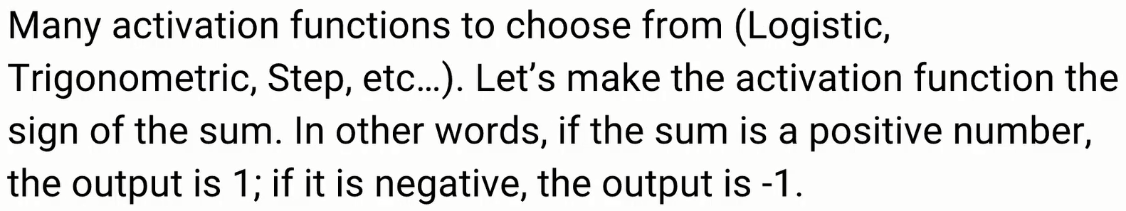
The simplest neural network is a perceptron.

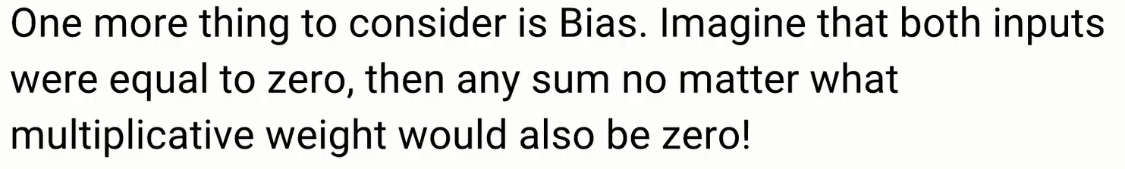


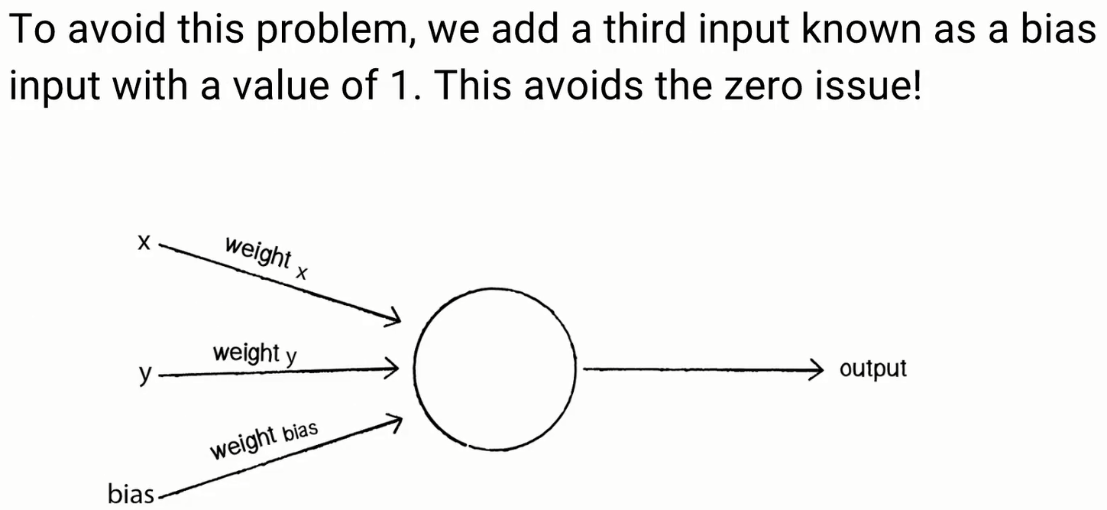


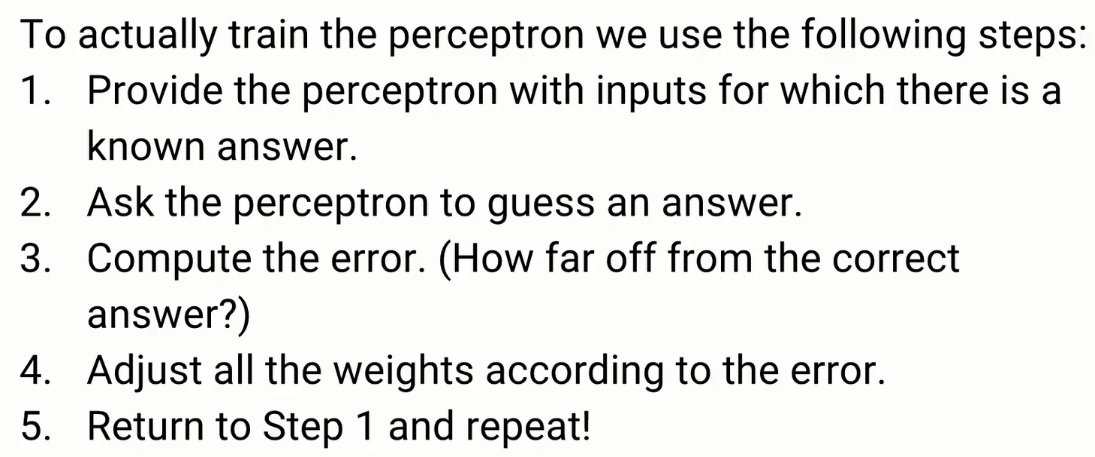


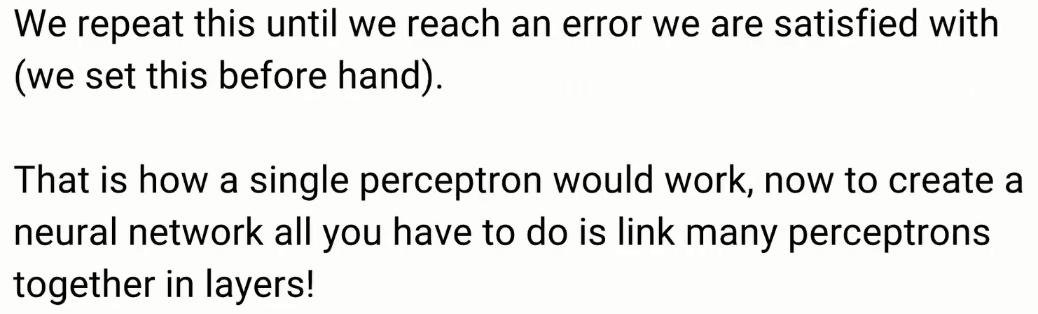


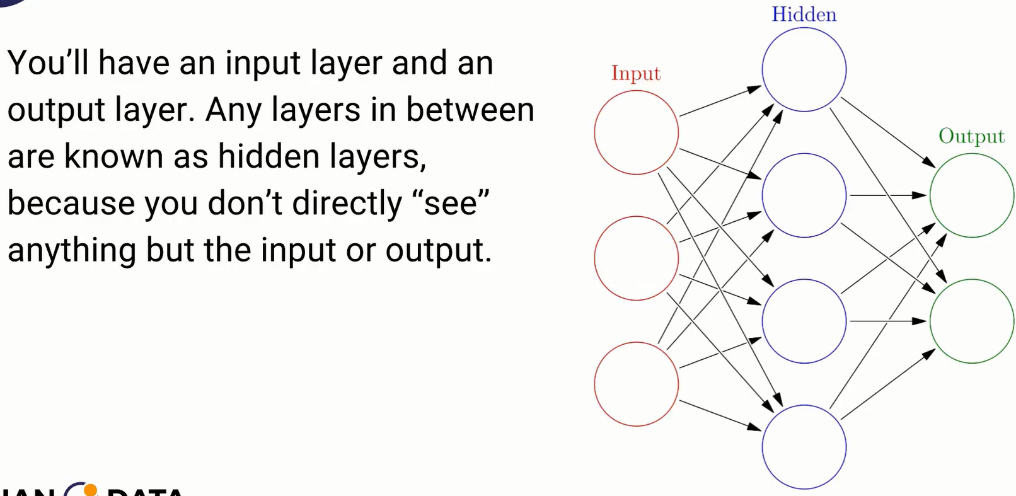


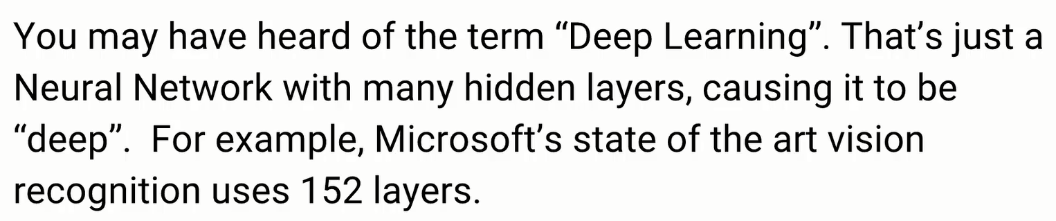












**TensorFlow**

