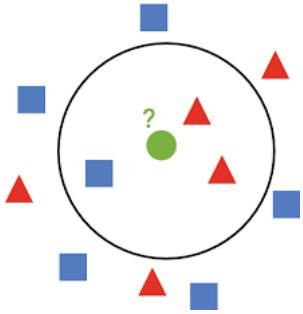
KNN ALGORITHM

- knn is a supervised classification algorithm.
- this is how knn works:



  - blue and red points are train data witch have label.
  - the green point is a new case that we want to predict its label.
  - first we find k nearest points(in this case k = 3)
  - find the label that repeats most among 3 nearest points and this is the prediction (in this case red)

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

read data set from csv file and return it as a data frame

- this dataset is about two kind of tumors and their specifications
  - tumors can be malignant(m) or Benign(b)
- this is a classification problem

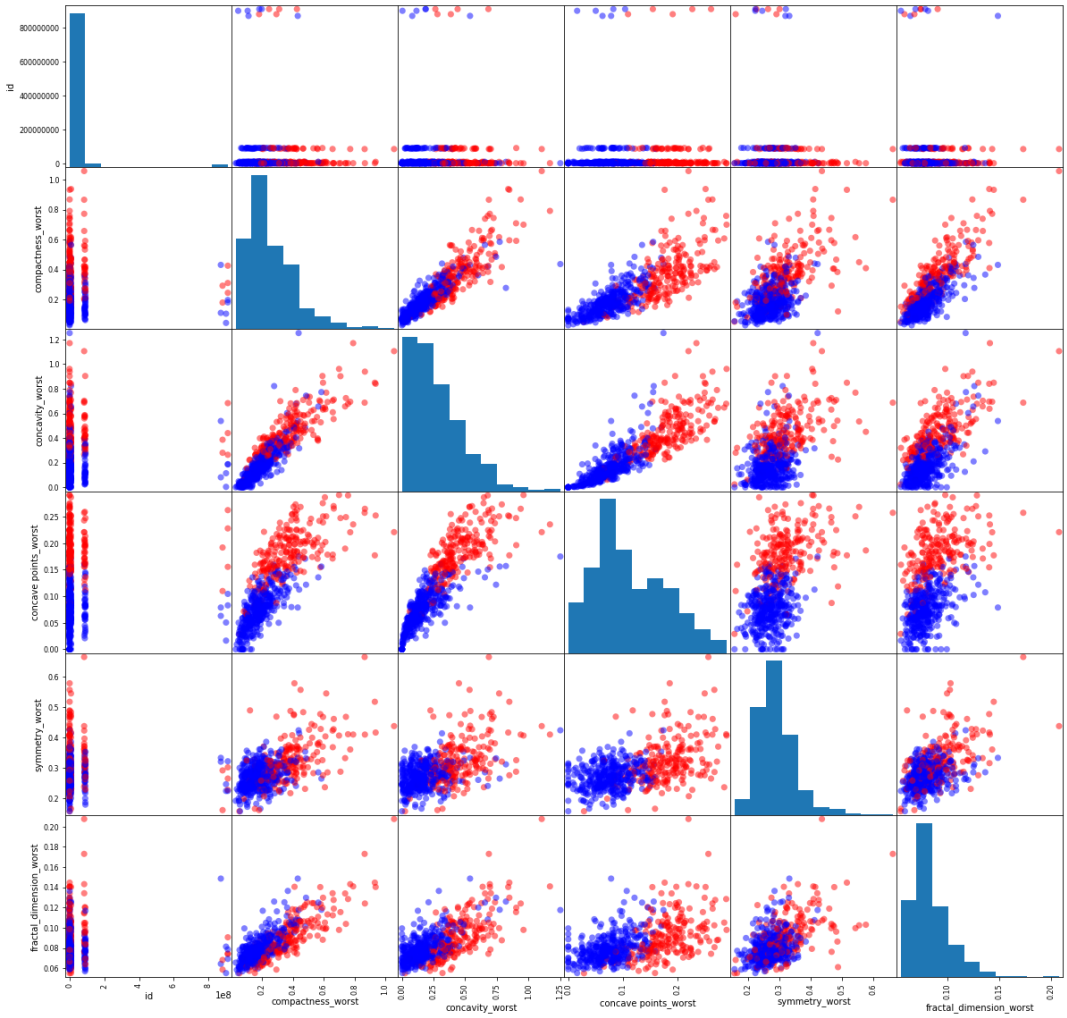
```
df = pd.read_csv("KNNAlgorithmDataset.csv")
```

data visualisation

- "scatter_matrix" function takes a data frame and plots some scatter plot
- each scatter plot show relation between two features and flags

```
plottingDf = df.drop(columns=["Unnamed: 32"], inplace=False)
colors = ["red" if cur == "M" else "blue" for cur in df.diagnosis]
for i in range(1, 27):
    plottingDf.drop(columns=[df.columns[i]], inplace=True)

pd.plotting.scatter_matrix(plottingDf, figsize=(20, 20), color=colors, s=200);
```



prepare input and output

- we have to "drop" some columns(features) in order to shape input
- diagnosis are the labels(m or b)
- we change m to 1 and b to 0

```
df.diagnosis = [1 if cur == "M" else 0 if cur == "B" else -1 for cur in df.diagnosis]
y = df.diagnosis

x = df.drop(columns=["id", "diagnosis", "Unnamed: 32"], inplace=False)
x.describe()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_me
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800

8 rows × 30 columns

normalize data

- as you can see in the upper table some feature's ranges are very small meanwhile some are very bin
- in order to normalize data we have to decrease this difference(notice range of data in table shown after normalize data)
- axis=0 means we want to normalize columns

```
from sklearn.preprocessing import normalize

norm_data = normalize(x.values, norm="l2", axis=0)

x = pd.DataFrame(norm_data, columns=x.columns)
x.describe()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mea
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	0.040678	0.040919	0.040534	0.036935	0.041483	0.037410	0.031208
std	0.010147	0.009124	0.010709	0.019848	0.006055	0.018936	0.028017
min	0.020101	0.020598	0.019300	0.008093	0.022657	0.006949	0.000000
25%	0.033689	0.034301	0.033130	0.023705	0.037183	0.023276	0.010389
50%	0.038497	0.039965	0.038009	0.031082	0.041272	0.033212	0.021628
75%	0.045437	0.046244	0.045880	0.044144	0.045332	0.046754	0.045933
max	0.080939	0.083325	0.083078	0.141055	0.070344	0.123840	0.149994

8 rows × 30 columns

split test and train data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=.30, shuffle=True)
```

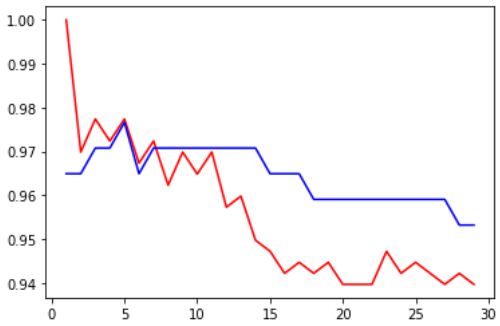
```
from sklearn.neighbors import KNeighborsClassifier
```

avoid over fitting and under fitting

- KNeighborsClassifier arguments:
  - metric: to specify the algorithm used to calculate distance
  - n_neighbors: number of neighbors
- we increase number of neighbors from 1 to 30 in order to determine witch one gives best result.
- as plot shows:
  - when number of neighbors is too small we have over fitting
  - when number of neighbors is too big we have under fitting

```
train_score_list = []
test_score_list = []
n_neighbors = []
for i in range(1, 30):
    knn = KNeighborsClassifier(n_neighbors=i, metric="minkowski", p=2)
    n_neighbors.append(i)
    knn.fit(X_train, y_train)
    test_score_list.append(knn.score(X_test, y_test))
    train_score_list.append(knn.score(X_train, y_train))

plt.plot(n_neighbors, train_score_list, c='r')
plt.plot(n_neighbors, test_score_list, c='b')
plt.show()
print(np.mean(test_score_list))
```



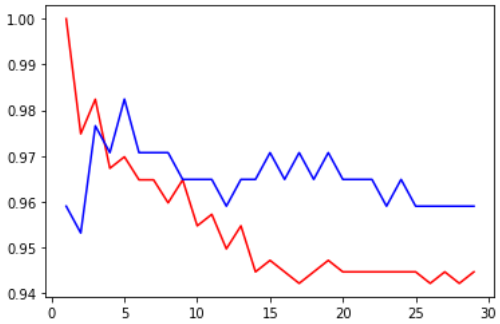

```
0.9645089735833835
```

distance metrics comparison(Euclidean(p=2) vs manhattan(p=1))

- by setting metric to "minkowski" and p=1 metric will be:manhattan
- by setting metric to "minkowski" and p=2 metric will be:Euclidean
- there is no big performance different between these two metrics(according to mean of all scores)
- Euclidean seems to be more stable

```
train_score_list = []
test_score_list = []
n_neighbors = []
for i in range(1, 30):
    knn = KNeighborsClassifier(n_neighbors=i, metric="minkowski", p=1)
    n_neighbors.append(i)
    knn.fit(X_train, y_train)
    test_score_list.append(knn.score(X_test, y_test))
    train_score_list.append(knn.score(X_train, y_train))

plt.plot(n_neighbors, train_score_list, c='r')
plt.plot(n_neighbors, test_score_list, c='b')
plt.show()
print(np.mean(test_score_list))
```




```
0.9653155878201247
```

```
knn = KNeighborsClassifier(n_neighbors=10, metric="minkowski", p=2)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

```
0.9707602339181286
```