K- Nearest neighbours(KNN)

In [1]:
```
# import the packages
import pandas as pd
```

In [2]:
```
# read the dataset
data = pd.read_csv("https://raw.githubusercontent.com/AP-State-Skill-Develop
```

In [3]:
```
data
```

Out[3]:

	Height	Weight	Size
0	158	58	M
1	158	59	M
2	158	63	M
3	160	59	M
4	160	60	M
5	163	60	M
6	163	61	M
7	160	64	L
8	163	64	L
9	165	61	L
10	165	61	L
11	165	62	L
12	168	62	L
13	168	63	L
14	168	66	L
15	170	63	L
16	170	64	L
17	170	68	L

In [4]:
```
data.shape
```

Out[4]:
```
(18, 3)
```

In [5]:
```
data.columns
```

Out[5]:
```
Index(['Height', 'Weight', 'Size'], dtype='object')
```

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 3 columns):
Height    18 non-null int64
Weight    18 non-null int64
Size      18 non-null object
dtypes: int64(2), object(1)
memory usage: 512.0+ bytes
```

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
Height    0
Weight    0
Size      0
dtype: int64
```

In [8]:
```
data.isnull().sum().sum()
```

Out[8]:
```
0
```

In [9]:
```
data["Size"].value_counts()
```

Out[9]:
```
L    11
M     7
Name: Size, dtype: int64
```

In [ ]:
```
# seperating features and target
```

In [11]:
```
data.columns
```

Out[11]:
```
Index(['Height', 'Weight', 'Size'], dtype='object')
```

In [12]:
```python
inpu = data[["Height","Weight"]]
inpu
```

Out[12]:

	Height	Weight
0	158	58
1	158	59
2	158	63
3	160	59
4	160	60
5	163	60
6	163	61
7	160	64
8	163	64
9	165	61
10	165	61
11	165	62
12	168	62
13	168	63
14	168	66
15	170	63
16	170	64
17	170	68

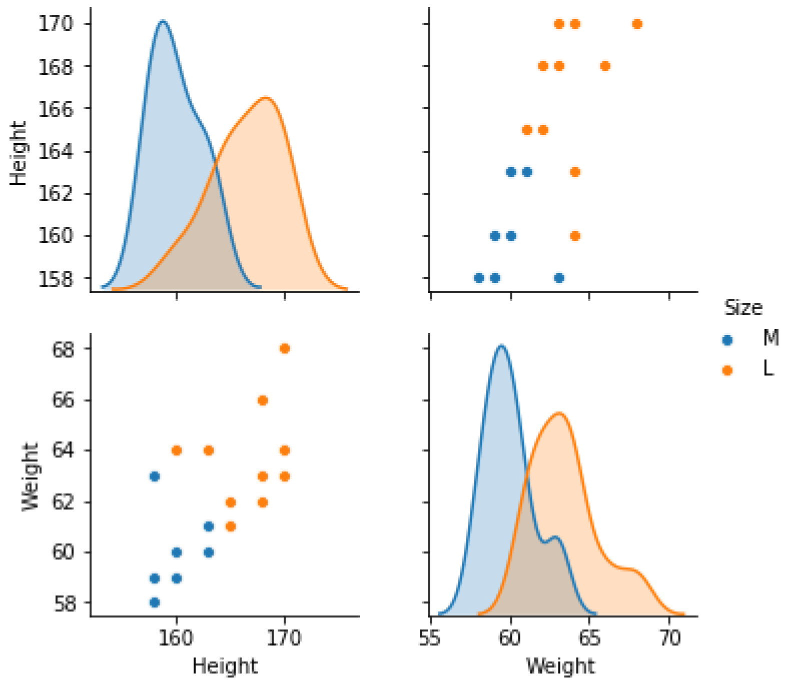
In [13]:
```
out = data["Size"]
out
```

Out[13]:
```
0     M
1     M
2     M
3     M
4     M
5     M
6     M
7     L
8     L
9     L
10    L
11    L
12    L
13    L
14    L
15    L
16    L
17    L
Name: Size, dtype: object
```

In [14]:
```
# visuvalize the data
import seaborn as sns
```

In [16]:
```
sns.pairplot(data,hue = "Size")
```

Out[16]:
```
<seaborn.axisgrid.PairGrid at 0x29a2523fb38>
```



In [ ]:

```
# get_dummies
# one hot encoder
# label encoder
```

In [17]:

```
pd.get_dummies(out)
```

Out[17]:

	L	M
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1
5	0	1
6	0	1
7	1	0
8	1	0
9	1	0
10	1	0
11	1	0
12	1	0
13	1	0
14	1	0
15	1	0
16	1	0
17	1	0

In [18]:

```
from sklearn.preprocessing import LabelEncoder
```

In [19]:

```
label = LabelEncoder()
```

In [20]:

```
d = label.fit_transform(out)
d
```

Out[20]:

```
array([1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [21]:

```
data["Size"] = d
```

In [22]:
```
data
```

Out[22]:

	Height	Weight	Size
0	158	58	1
1	158	59	1
2	158	63	1
3	160	59	1
4	160	60	1
5	163	60	1
6	163	61	1
7	160	64	0
8	163	64	0
9	165	61	0
10	165	61	0
11	165	62	0
12	168	62	0
13	168	63	0
14	168	66	0
15	170	63	0
16	170	64	0
17	170	68	0

In [23]:
```
out = data["Size"]
```

In [24]:
```
out
```

Out[24]:
```
0     1
1     1
2     1
3     1
4     1
5     1
6     1
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
15    0
16    0
17    0
Name: Size, dtype: int32
```

In [25]:
```
# import the model
```

In [26]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [27]:
```
help(KNeighborsClassifier)
```

```
Help on class KNeighborsClassifier in module sklearn.neighbors.classification:

class KNeighborsClassifier(sklearn.neighbors.base.NeighborsBase, sklearn.neighb
ors.base.KNeighborsMixin, sklearn.neighbors.base.SupervisedIntegerMixin, sklear
n.base.ClassifierMixin)
 |  KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', le
af_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)
 |  
 |  Classifier implementing the k-nearest neighbors vote.
 |  
 |  Read more in the :ref:`User Guide <classification>`.
 |  
 |  Parameters
 |  ----------
 |  n_neighbors : int, optional (default = 5)
 |      Number of neighbors to use by default for :meth:`kneighbors` queries.
 |  
 |  weights : str or callable, optional (default = 'uniform')
 |      weight function used in prediction.  Possible values:
 |  
 |      - 'uniform' : uniform weights.  All points in each neighborhood
 |        are weighted equally.
 |      - 'distance' : weight points by the inverse of their distance.
 |        in this case, closer neighbors of a query point will have a
 |        greater influence than neighbors which are further away.
 |      - [callable] : a user-defined function which accepts an
 |        array of distances, and returns an array of the same shape
 |        containing the weights.
 |  
 |  algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'}, optional
 |      Algorithm used to compute the nearest neighbors:
 |  
 |      - 'ball_tree' will use :class:`BallTree`
 |      - 'kd_tree' will use :class:`KDTree`
 |      - 'brute' will use a brute-force search.
 |      - 'auto' will attempt to decide the most appropriate algorithm
 |        based on the values passed to :meth:`fit` method.
 |  
 |      Note: fitting on sparse input will override the setting of
 |      this parameter, using brute force.
 |  
 |  leaf_size : int, optional (default = 30)
 |      Leaf size passed to BallTree or KDTree.  This can affect the
 |      speed of the construction and query, as well as the memory
 |      required to store the tree.  The optimal value depends on the
 |      nature of the problem.
 |  
 |  p : integer, optional (default = 2)
 |      Power parameter for the Minkowski metric. When p = 1, this is
 |      equivalent to using manhattan_distance (l1), and euclidean_distance
 |      (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.
 |  
 |  metric : string or callable, default 'minkowski'
 |      the distance metric to use for the tree.  The default metric is
 |      minkowski, and with p=2 is equivalent to the standard Euclidean
 |      metric. See the documentation of the DistanceMetric class for a
 |      list of available metrics.
 |  
 |  metric_params : dict, optional (default = None)
 |      Additional keyword arguments for the metric function.
 |  
 |  n_jobs : int or None, optional (default=None)
 |      The number of parallel jobs to run for neighbors search.
 |      ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
 |      ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
 |      for more details.
 |      Doesn't affect :meth:`fit` method.
 |  
 |  Examples
 |  --------
 |  >>> X = [[0], [1], [2], [3]]
 |  >>> y = [0, 0, 1, 1]
 |  >>> from sklearn.neighbors import KNeighborsClassifier
 |  >>> neigh = KNeighborsClassifier(n_neighbors=3)
 |  >>> neigh.fit(X, y) # doctest: +ELLIPSIS
 |  KNeighborsClassifier(...)
 |  >>> print(neigh.predict([[1.1]]))
 |  [0]
 |  >>> print(neigh.predict_proba([[0.9]]))
 |  [[0.66666667 0.33333333]]
 |  
 |  See also
 |  --------
 |  RadiusNeighborsClassifier
 |  KNeighborsRegressor
 |  RadiusNeighborsRegressor
 |  NearestNeighbors
 |  
 |  Notes
 |  -----
 |  See :ref:`Nearest Neighbors <neighbors>` in the online documentation
 |  for a discussion of the choice of ``algorithm`` and ``leaf_size``.
 |  
 |  .. warning::
 |  
 |     Regarding the Nearest Neighbors algorithms, if it is found that two
 |     neighbors, neighbor `k+1` and `k`, have identical distances
 |     but different labels, the results will depend on the ordering of the
 |     training data.
 |  
 |  https://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm (https://en.wiki
pedia.org/wiki/K-nearest_neighbor_algorithm)
 |  
 |  Method resolution order:
 |      KNeighborsClassifier
 |      sklearn.neighbors.base.NeighborsBase
 |      abc.NewBase
 |      sklearn.base.BaseEstimator
 |      sklearn.neighbors.base.KNeighborsMixin
 |      sklearn.neighbors.base.SupervisedIntegerMixin
 |      sklearn.base.ClassifierMixin
 |      builtins.object
 |  
 |  Methods defined here:
 |  
 |  __init__(self, n_neighbors=5, weights='uniform', algorithm='auto', leaf_siz
e=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |  
 |  predict(self, X)
 |      Predict the class labels for the provided data
 |      
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_query, n_features),                 or (n_quer
y, n_indexed) if metric == 'precomputed'
 |          Test samples.
 |      
 |      Returns
 |      -------
 |      y : array of shape [n_samples] or [n_samples, n_outputs]
 |          Class labels for each data sample.
 |  
 |  predict_proba(self, X)
 |      Return probability estimates for the test data X.
 |      
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_query, n_features),                 or (n_quer
y, n_indexed) if metric == 'precomputed'
 |          Test samples.
 |      
 |      Returns
 |      -------
 |      p : array of shape = [n_samples, n_classes], or a list of n_outputs
 |          of such arrays if n_outputs > 1.
 |          The class probabilities of the input samples. Classes are ordered
 |          by lexicographic order.
 |  
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |  
 |  __abstractmethods__ = frozenset()
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |  
 |  __getstate__(self)
 |  
 |  __repr__(self)
 |      Return repr(self).
 |  
 |  __setstate__(self, state)
 |  
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |      
 |      Parameters
 |      ----------
 |      deep : boolean, optional
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |
 |      Returns
 |      -------
 |      params : mapping of string to any
 |          Parameter names mapped to their values.
 |
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |
 |      The method works on simple estimators as well as on nested objects
 |      (such as pipelines). The latter have parameters of the form
 |      ``<component>__<parameter>`` so that it's possible to update each
 |      component of a nested object.
 |
 |      Returns
 |      -------
 |      self
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.BaseEstimator:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.neighbors.base.KNeighborsMixin:
 |
 |  kneighbors(self, X=None, n_neighbors=None, return_distance=True)
 |      Finds the K-neighbors of a point.
 |      Returns indices of and distances to the neighbors of each point.
 |
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_query, n_features),                 or (n_quer
y, n_indexed) if metric == 'precomputed'
 |          The query point or points.
 |          If not provided, neighbors of each indexed point are returned.
 |          In this case, the query point is not considered its own neighbor.
 |
 |      n_neighbors : int
 |          Number of neighbors to get (default is the value
 |          passed to the constructor).
 |
 |      return_distance : boolean, optional. Defaults to True.
 |          If False, distances will not be returned
 |
 |      Returns
 |      -------
 |      dist : array
 |          Array representing the lengths to points, only present if
 |          return_distance=True
 |      
 |      ind : array
 |          Indices of the nearest points in the population matrix.
 |      
 |      Examples
 |      --------
 |      In the following example, we construct a NeighborsClassifier
 |      class from an array representing our data set and ask who's
 |      the closest point to [1,1,1]
 |      
 |      >>> samples = [[0., 0., 0.], [0., .5, 0.], [1., 1., .5]]
 |      >>> from sklearn.neighbors import NearestNeighbors
 |      >>> neigh = NearestNeighbors(n_neighbors=1)
 |      >>> neigh.fit(samples) # doctest: +ELLIPSIS
 |      NearestNeighbors(algorithm='auto', leaf_size=30, ...)
 |      >>> print(neigh.kneighbors([[1., 1., 1.]])) # doctest: +ELLIPSIS
 |      (array([[0.5]]), array([[2]]))
 |      
 |      As you can see, it returns [[0.5]], and [[2]], which means that the
 |      element is at distance 0.5 and is the third element of samples
 |      (indexes start at 0). You can also query for multiple points:
 |      
 |      >>> X = [[0., 1., 0.], [1., 0., 1.]]
 |      >>> neigh.kneighbors(X, return_distance=False) # doctest: +ELLIPSIS
 |      array([[1],
 |             [2]]...)
 |  
 |  kneighbors_graph(self, X=None, n_neighbors=None, mode='connectivity')
 |      Computes the (weighted) graph of k-Neighbors for points in X
 |      
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_query, n_features),                 or (n_quer
y, n_indexed) if metric == 'precomputed'
 |          The query point or points.
 |          If not provided, neighbors of each indexed point are returned.
 |          In this case, the query point is not considered its own neighbor.
 |      
 |      n_neighbors : int
 |          Number of neighbors for each sample.
 |          (default is value passed to the constructor).
 |      
 |      mode : {'connectivity', 'distance'}, optional
 |          Type of returned matrix: 'connectivity' will return the
 |          connectivity matrix with ones and zeros, in 'distance' the
 |          edges are Euclidean distance between points.
 |      
 |      Returns
 |      -------
 |      A : sparse matrix in CSR format, shape = [n_samples, n_samples_fit]
 |          n_samples_fit is the number of samples in the fitted data
 |          A[i, j] is assigned the weight of edge that connects i to j.
 |      
 |      Examples
 |      --------
 |      >>> X = [[0], [3], [1]]
 |      >>> from sklearn.neighbors import NearestNeighbors
 |      >>> neigh = NearestNeighbors(n_neighbors=2)
 |      >>> neigh.fit(X) # doctest: +ELLIPSIS
 |      NearestNeighbors(algorithm='auto', leaf_size=30, ...)
 |      >>> A = neigh.kneighbors_graph(X)
 |      >>> A.toarray()
 |      array([[1., 0., 1.],
 |             [0., 1., 1.],
 |             [1., 0., 1.]])
 |
 |      See also
 |      --------
 |      NearestNeighbors.radius_neighbors_graph
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.neighbors.base.SupervisedIntegerMixin:
 |
 |  fit(self, X, y)
 |      Fit the model using X as training data and y as target values
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix, BallTree, KDTree}
 |          Training data. If array or matrix, shape [n_samples, n_features],
 |          or [n_samples, n_samples] if metric='precomputed'.
 |
 |      y : {array-like, sparse matrix}
 |          Target values of shape = [n_samples] or [n_samples, n_outputs]
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.ClassifierMixin:
 |
 |  score(self, X, y, sample_weight=None)
 |      Returns the mean accuracy on the given test data and labels.
 |
 |      In multi-label classification, this is the subset accuracy
 |      which is a harsh metric since you require for each sample that
 |      each label set be correctly predicted.
 |
 |      Parameters
 |      ----------
 |      X : array-like, shape = (n_samples, n_features)
 |          Test samples.
 |
 |      y : array-like, shape = (n_samples) or (n_samples, n_outputs)
 |          True labels for X.
 |
 |      sample_weight : array-like, shape = [n_samples], optional
 |          Sample weights.
 |
 |      Returns
 |      -------
 |      score : float
 |          Mean accuracy of self.predict(X) wrt. y.
```

In [28]:
```
knn = KNeighborsClassifier(n_neighbors=5)
```

In [29]:
```
knn.fit(inpu,out)
```

Out[29]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [30]:
```
# predict the model
pred = knn.predict(inpu)
```

In [31]:
```
pred
```

Out[31]:
```
array([1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [32]:
```
from sklearn import metrics
```

In [33]:
```
print(dir(metrics))
```

```
['SCORERS', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__
loader__', '__name__', '__package__', '__path__', '__spec__', 'accuracy_score',
'adjusted_mutual_info_score', 'adjusted_rand_score', 'auc', 'average_precision_
score', 'balanced_accuracy_score', 'base', 'brier_score_loss', 'calinski_haraba
z_score', 'check_scoring', 'classification', 'classification_report', 'cluste
r', 'cohen_kappa_score', 'completeness_score', 'confusion_matrix', 'consensus_s
core', 'coverage_error', 'davies_bouldin_score', 'euclidean_distances', 'explai
ned_variance_score', 'f1_score', 'fbeta_score', 'fowlkes_mallows_score', 'get_s
corer', 'hamming_loss', 'hinge_loss', 'homogeneity_completeness_v_measure', 'ho
mogeneity_score', 'jaccard_similarity_score', 'label_ranking_average_precision_
score', 'label_ranking_loss', 'log_loss', 'make_scorer', 'matthews_corrcoef',
'mean_absolute_error', 'mean_squared_error', 'mean_squared_log_error', 'median_
absolute_error', 'mutual_info_score', 'normalized_mutual_info_score', 'pairwis
e', 'pairwise_distances', 'pairwise_distances_argmin', 'pairwise_distances_argm
in_min', 'pairwise_distances_chunked', 'pairwise_fast', 'pairwise_kernels', 'pr
ecision_recall_curve', 'precision_recall_fscore_support', 'precision_score', 'r
2_score', 'ranking', 'recall_score', 'regression', 'roc_auc_score', 'roc_curv
e', 'scorer', 'silhouette_samples', 'silhouette_score', 'v_measure_score', 'zer
o_one_loss']
```

In [34]:
```
metrics.accuracy_score(out,pred)*100
```

Out[34]:
```
83.33333333333334
```

In [36]:
```
print(metrics.classification_report(out,pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.91      0.87        11
           1       0.83      0.71      0.77         7

   micro avg       0.83      0.83      0.83        18
   macro avg       0.83      0.81      0.82        18
weighted avg       0.83      0.83      0.83        18
```

In [37]:
```
metrics.confusion_matrix(out,pred)
```

Out[37]:
```
array([[10,  1],
       [ 2,  5]], dtype=int64)
```

Multi-class classification

In [38]:
```
data1 = pd.read_excel("winequality-red.xls")
```

In [42]:
```
data1.head()
```

Out[42]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

In [43]:
```
data1.shape
```

Out[43]:
```
(1744, 12)
```

In [44]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1744 entries, 0 to 1743
Data columns (total 12 columns):
fixed acidity           1744 non-null float64
volatile acidity        1744 non-null float64
citric acid             1744 non-null float64
residual sugar          1744 non-null float64
chlorides               1744 non-null float64
free sulfur dioxide     1744 non-null float64
total sulfur dioxide    1744 non-null float64
density                 1744 non-null float64
pH                      1744 non-null float64
sulphates               1744 non-null float64
alcohol                 1744 non-null float64
quality                 1744 non-null int64
dtypes: float64(11), int64(1)
memory usage: 163.6 KB
```

In [45]:
```
data1.isnull().sum()
```

Out[45]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

In [46]:
```
data1["quality"].value_counts()
```

Out[46]:
```
5    716
6    647
7    224
4     90
8     46
3     21
Name: quality, dtype: int64
```

In [49]:
```
# seperating features and target
features = data1.drop(["quality"],axis=1)
```

In [50]:
```
features.head()
```

Out[50]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

In [51]:
```
target = data1["quality"]
```

In [53]:
```
target.head()
```

Out[53]:
```
0    5
1    5
2    5
3    6
4    5
Name: quality, dtype: int64
```

In [57]:
```
o = label.fit_transform(target)
o
```

Out[57]:
```
array([2, 2, 2, ..., 2, 3, 2], dtype=int64)
```

In [58]:
```
data1["quality"] = o
```

In [59]:
```
data1.head()
```

Out[59]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

In [60]:
```
target = data1["quality"]
```

In [61]:
```
data1.shape
```

Out[61]:
```
(1744, 12)
```

In [62]:
```
# splitting the data for training and testing
```

In [63]:
```
from sklearn.model_selection import train_test_split
```

In [103]:
```
x_train,x_test,y_train,y_test = train_test_split(features,target,test_size =
```

In [104]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [105]:
```
knn1 = KNeighborsClassifier(n_neighbors=3)
```

In [106]:
```
knn1.fit(x_train,y_train)
```

Out[106]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [107]:
```
pred1 = knn1.predict(x_train)
```

In [108]:
```
pred1
```

Out[108]:
```
array([2, 4, 4, ..., 4, 2, 2], dtype=int64)
```

In [109]:
```
knn1.predict(x_test)
```

Out[109]:
```
array([3, 3, 1, 3, 1, 2, 3, 3, 3, 3, 4, 3, 3, 4, 2, 1, 2, 2, 2, 2, 2, 4,
       2, 5, 2, 3, 5, 4, 2, 4, 2, 3, 2, 2, 3, 4, 3, 3, 2, 5, 4, 2, 2, 2,
       2, 3, 3, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 4, 2, 3, 3, 2, 3, 4, 3, 3,
       3, 2, 2, 3, 2, 3, 2, 4, 3, 3, 3, 3, 4, 1, 2, 2, 4, 2, 3, 3, 2, 2,
       3, 3, 4, 2, 5, 4, 0, 2, 3, 2, 2, 3, 3, 2, 2, 2, 2, 2, 2, 3, 2, 4,
       4, 4, 2, 2, 2, 1, 2, 3, 1, 1, 2, 2, 2, 3, 1, 2, 2, 2, 4, 4, 1, 4,
       2, 2, 3, 2, 1, 2, 1, 2, 2, 3, 2, 5, 4, 2, 3, 3, 1, 2, 2, 3, 2, 1,
       3, 2, 3, 3, 2, 3, 3, 2, 2, 4, 1, 2, 2, 3, 2, 2, 3, 3, 3, 2, 3, 2,
       2, 2, 2, 5, 2, 3, 3, 3, 2, 3, 2, 3, 4, 3, 2, 3, 5, 2, 3, 3, 2, 3,
       2, 2, 2, 3, 3, 1, 4, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 2, 2, 3,
       3, 2, 2, 3, 2, 3, 2, 3, 2, 3, 3, 2, 3, 3, 2, 2, 2, 2, 5, 2, 2, 1,
       2, 3, 1, 3, 2, 2, 2, 5, 2, 2, 1, 2, 3, 4, 1, 3, 4, 3, 3, 2, 2, 2,
       3, 2, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 3, 4, 2, 2, 1, 3, 4, 2,
       5, 4, 3, 1, 1, 1, 3, 2, 1, 2, 2, 2, 3, 2, 3, 3, 0, 3, 3, 2, 2, 3,
       3, 4, 3, 2, 3, 3, 2, 3, 2, 2, 3, 4, 2, 2, 2, 4, 3, 4, 2, 3, 4, 3,
       2, 2, 4, 2, 2, 2, 1, 3, 3, 3, 1, 2, 2, 3, 3, 2, 2, 2, 1, 2, 2, 3,
       4, 2, 2, 3, 3, 2, 3, 2, 4, 2, 3, 2, 3, 2, 1, 2, 2, 0, 3, 3, 2, 4,
       2, 3, 3, 2, 2, 3, 3, 2, 2, 3, 3, 3, 3, 3, 3, 2, 2, 1, 2, 3, 2, 0,
       2, 3, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 3, 2, 2, 3, 3, 2, 2, 2, 4,
       3, 2, 3, 3, 3, 2, 4, 4, 2, 3, 1, 2, 2, 1, 2, 4, 3, 3, 3, 1, 2, 2,
       2, 3, 1, 3, 2, 4, 3, 4, 2, 1, 2, 3, 5, 1, 4, 2, 2, 2, 2, 2, 3, 2,
       3, 3, 0, 3, 4, 2, 3, 2, 4, 2, 3, 2, 1, 3, 4, 2, 3, 3, 3, 3, 2, 5,
       2, 2, 4, 3, 3, 4, 2, 2, 1, 3, 2, 2, 2, 2, 3, 4, 2, 3, 2, 3, 2, 2,
       2, 2, 3, 1, 2, 4, 3, 3, 2, 3, 3, 2, 3, 3, 2, 3, 4, 3, 3, 2, 2, 3,
       2, 4, 2, 1, 2, 2, 3, 4, 2, 2, 3, 2, 2, 3, 2, 4, 3, 2, 2, 2, 3, 2,
       2, 2, 4, 2, 2, 2, 2, 3, 2, 3, 3, 3, 2, 3, 2, 2, 2, 4, 4, 2, 3, 3,
       1, 2, 2, 2], dtype=int64)
```

In [110]:
```
metrics.accuracy_score(y_train,pred1)
```

Out[110]:
```
0.7320205479452054
```

In [111]:
```
metrics.confusion_matrix(y_train,pred1)
```

Out[111]:
```
array([[ 14,   0,   0,   0,   0,   0],
       [  1,  34,  12,  10,   2,   0],
       [  1,  12, 391,  65,   5,   0],
       [  3,  14, 100, 307,  21,   1],
       [  2,   1,  27,  23,  90,   1],
       [  0,   1,   5,   6,   0,  19]], dtype=int64)
```

Logistic Regression

In [112]:
```
import pandas as pd
```

In [114]:
```
from sklearn import datasets
```

In [116]:
```
dir(datasets)
```

Out[116]:
```
['__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '_svmlight_format',
 'base',
 'california_housing',
 'clear_data_home',
 'covtype',
 'dump_svmlight_file',
 'fetch_20newsgroups',
 'fetch_20newsgroups_vectorized',
 'fetch_california_housing',
 'fetch_covtype',
 'fetch_kddcup99',
 'fetch_lfw_pairs',
 'fetch_lfw_people',
 'fetch_mldata',
 'fetch_olivetti_faces',
 'fetch_openml',
 'fetch_rcv1',
 'fetch_species_distributions',
 'get_data_home',
 'kddcup99',
 'lfw',
 'load_boston',
 'load_breast_cancer',
 'load_diabetes',
 'load_digits',
 'load_files',
 'load_iris',
 'load_linnerud',
 'load_mlcomp',
 'load_sample_image',
 'load_sample_images',
 'load_svmlight_file',
 'load_svmlight_files',
 'load_wine',
 'make_biclusters',
 'make_blobs',
 'make_checkerboard',
 'make_circles',
 'make_classification',
 'make_friedman1',
 'make_friedman2',
 'make_friedman3',
 'make_gaussian_quantiles',
 'make_hastie_10_2',
 'make_low_rank_matrix',
 'make_moons',
 'make_multilabel_classification',
 'make_regression',
 'make_s_curve',
 'make_sparse_coded_signal',
 'make_sparse_spd_matrix',
 'make_sparse_uncorrelated',
 'make_spd_matrix',
 'make_swiss_roll',
 'mlcomp',
 'mldata',
 'mldata_filename',
 'olivetti_faces',
 'openml',
 'rcv1',
 'samples_generator',
 'species_distributions',
 'svmlight_format',
 'twenty_newsgroups']
```

In [117]:

```
cancer = datasets.load_breast_cancer()
```

```
In [118]:  cancer
```

```
Out[118]: {'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
                   1.189e-01],
                  [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
                   8.902e-02],
                  [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
                   8.758e-02],
                  ...,
                  [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
                   7.820e-02],
                  [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
                   1.240e-01],
                  [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
                   7.039e-02]]),
           'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
          1, 1,
                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                  0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
                  1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
                  1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
                  1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
                  0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                  1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
                  1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
                  0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
                  1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
                  1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                  0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
                  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
                  0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
                  0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
                  1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
                  1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
                  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
                  1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                  1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
                  1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
                  1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
           'target_names': array(['malignant', 'benign'], dtype='<U9'),
           'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic)
          dataset\n--------------------------------------------\n\n**Data Set Character
          istics:**\n\n    :Number of Instances: 569\n\n    :Number of Attributes: 30 n
          umeric, predictive attributes and the class\n\n    :Attribute Information:\n
          - radius (mean of distances from center to points on the perimeter)\n
          - texture (standard deviation of gray-scale values)\n        - perimeter\n
          - area\n        - smoothness (local variation in radius lengths)\n        - c
          ompactness (perimeter^2 / area - 1.0)\n        - concavity (severity of conca
          ve portions of the contour)\n        - concave points (number of concave port
          ions of the contour)\n        - symmetry \n        - fractal dimension ("coas
          tline approximation" - 1)\n\n        The mean, standard error, and "worst" or
          largest (mean of the three\n        largest values) of these features were co
          mputed for each image,\n        resulting in 30 features.  For instance, fiel
```

```
d 3 is Mean Radius, field\n        13 is Radius SE, field 23 is Worst Radiu
s.\n\n        - class:\n                - WDBC-Malignant\n                - W
DBC-Benign\n\n    :Summary Statistics:\n\n    ===============================
====== ====== ======\n                                          Min    Max\n
===================================== ====== ======\n    radius (mean):
6.981  28.11\n    texture (mean):                      9.71   39.28\n    per
imeter (mean):                   43.79  188.5\n    area (mean):
143.5  2501.0\n    smoothness (mean):                   0.053  0.163\n    co
mpactness (mean):                  0.019  0.345\n    concavity (mean):
0.0    0.427\n    concave points (mean):               0.0    0.201\n    sym
metry (mean):                    0.106  0.304\n    fractal dimension (mea
n):            0.05   0.097\n    radius (standard error):            0.112
2.873\n    texture (standard error):            0.36   4.885\n    perimeter
(standard error):           0.757  21.98\n    area (standard error):
6.802  542.2\n    smoothness (standard error):         0.002  0.031\n    com
pactness (standard error):         0.002  0.135\n    concavity (standard erro
r):           0.0    0.396\n    concave points (standard error):      0.0
0.053\n    symmetry (standard error):           0.008  0.079\n    fractal di
mension (standard error):   0.001  0.03\n    radius (worst):
7.93   36.04\n    texture (worst):                     12.02  49.54\n    per
imeter (worst):                  50.41  251.2\n    area (worst):
185.2  4254.0\n    smoothness (worst):                  0.071  0.223\n    co
mpactness (worst):                 0.027  1.058\n    concavity (worst):
0.0    1.252\n    concave points (worst):              0.0    0.291\n    sym
metry (worst):                   0.156  0.664\n    fractal dimension (wors
t):           0.055  0.208\n    ===================================== ======
======\n\n    :Missing Attribute Values: None\n\n    :Class Distribution: 212
- Malignant, 357 - Benign\n\n    :Creator:  Dr. William H. Wolberg, W. Nick S
treet, Olvi L. Mangasarian\n\n    :Donor: Nick Street\n\n    :Date: November,
1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) dataset
s.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of
a fine needle\naspirate (FNA) of a breast mass.  They describe\ncharacteristi
cs of the cell nuclei present in the image.\n\nSeparating plane described abo
ve was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Deci
sion Tree\nConstruction Via Linear Programming." Proceedings of the 4th\nMidw
est Artificial Intelligence and Cognitive Science Society,\npp. 97-101, 199
2], a classification method which uses linear\nprogramming to construct a dec
ision tree.  Relevant features\nwere selected using an exhaustive search in t
he space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual linear pro
gram used to obtain the separating plane\nin the 3-dimensional space is that
described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust Linear\nProgramm
ing Discrimination of Two Linearly Inseparable Sets",\nOptimization Methods a
nd Software 1, 1992, 23-34].\n\nThis database is also available through the U
W CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-lea
rn/WDBC/\n\n.. topic:: References\n\n   - W.N. Street, W.H. Wolberg and O.L.
Mangasarian. Nuclear feature extraction \n     for breast tumor diagnosis. IS
&T/SPIE 1993 International Symposium on \n     Electronic Imaging: Science an
d Technology, volume 1905, pages 861-870,\n     San Jose, CA, 1993.\n   - O.
L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and \n
prognosis via linear programming. Operations Research, 43(4), pages 570-577,
\n     July-August 1995.\n   - W.H. Wolberg, W.N. Street, and O.L. Mangasaria
n. Machine learning techniques\n     to diagnose breast cancer from fine-need
le aspirates. Cancer Letters 77 (1994) \n     163-171.',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'me
an area',
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension'], dtype='<U23'),
 'filename': 'C:\\Users\\Alekhya\\Anaconda3\\lib\\site-packages\\sklearn\\dat
asets\\data\\breast_cancer.csv'}
```

In [121]:

```
# selecting features and target

input_data = pd.DataFrame(cancer["data"],columns = ['mean radius', 'mean tex
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension'])
```

In [122]:

```
input_data.head()
```

Out[122]:

mean ymmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	cc
0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	
0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	
0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	
0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	
0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	

In [123]:

```
input_data.shape
```

Out[123]:

```
(569, 30)
```

In [125]:

```
output_data = pd.DataFrame(cancer["target"],columns=["target"])
```

In [126]:
```
output_data.head()
```

Out[126]:

	target
0	0
1	0
2	0
3	0
4	0

In [128]:
```
output_data["target"].value_counts()
```

Out[128]:
```
1    357
0    212
Name: target, dtype: int64
```

In [129]:
```
output_data.isnull().sum()
```

Out[129]:
```
target    0
dtype: int64
```

In [130]:
```
input_data.isnull().sum()
```

Out[130]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
dtype: int64
```

In [131]:
```
# splitting the data for training and testing
```

In [132]:
```
from sklearn.model_selection import train_test_split
```

In [133]:
```
x_train,x_test,y_train,y_test = train_test_split(input_data,output_data,
                                                 test_size=0.3,random_state=
```

In [134]:
```
# select the model
```

In [135]:
```
from sklearn.linear_model import LogisticRegression
```

In [136]:
```
log = LogisticRegression()
```

In [137]:
```
log.fit(x_train,y_train)
```

```
C:\Users\Alekhya\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
33: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\Alekhya\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: D
ataConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[137]:
```
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 [138]:
```
pred2 = log.predict(x_test)
pred2
```

Out[138]:
```
array([1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1])
```

In [141]:
```
metrics.accuracy_score(y_test,pred2)
```

Out[141]:
```
0.9473684210526315
```

In [142]:
```
metrics.confusion_matrix(y_test,pred2)
```

Out[142]:
```
array([[ 57,   5],
       [  4, 105]], dtype=int64)
```

In [143]:
```
print(metrics.classification_report(y_test,pred2))
```

```
              precision    recall  f1-score   support

           0       0.93      0.92      0.93        62
           1       0.95      0.96      0.96       109

   micro avg       0.95      0.95      0.95       171
   macro avg       0.94      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171
```

In [ ]:
```

```