Task-5A:

Random Forest algorithm basically evaluates the estimates produced by more than one decision tree. An average value is calculated with the predicted values produced by more than one tree in parallel. Observations for trees are chosen at random. The difference from bagging is that when estimating values are requested from each tree, the trees are weighted by taking into account the error rates.

In [9]:
```
from sklearn.datasets import load_digits
digits = load_digits()
digits.keys()
```

Out[9]:
```
dict_keys(['data', 'target', 'target_names', 'images', 'DESCR'])
```

In [43]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.30, random_state=42)
```

In [44]:
```
X_train.shape
```

Out[44]:
```
(1257, 64)
```

In [45]:
```
rf_model = RandomForestClassifier()
#n_estimators (ağaç sayısı), max_features
```

In [46]:
```
rf_model
```

Out[46]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [49]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
              'max_depth': [2, 3, 5, 8, 10],
              'max_features': [20, 30, 40],
              'n_estimators': [10, 50, 100, 500],
              'min_samples_split': [3, 5, 7, 10]
             }
```

In [50]:
```
rf_grid = GridSearchCV(rf_model,
                       param_grid,
                       cv=5,
                       refit=True,
                       n_jobs=-1,
                       verbose=2)
```

In [51]:
```
rf_grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 240 candidates, totalling 1200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:    1.9s
[Parallel(n_jobs=-1)]: Done 138 tasks      | elapsed:    9.1s
[Parallel(n_jobs=-1)]: Done 341 tasks      | elapsed:   23.0s
[Parallel(n_jobs=-1)]: Done 624 tasks      | elapsed:   48.5s
[Parallel(n_jobs=-1)]: Done 989 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 1200 out of 1200 | elapsed:  2.3min finished
```

Out[51]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 5, 8, 10],
                         'max_features': [20, 30, 40],
                         'min_samples_split': [3, 5, 7, 10],
                         'n_estimators': [10, 50, 100, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=2)
```

In [32]:
```
print("Best paramters: " +str(rf_grid.best_params_))
```

```
Best paramters: {'max_depth': 10, 'max_features': 20, 'min_samples_split': 7, 'n_estimators': 50}
```

In [34]:
```
rf_tuned = RandomForestClassifier(max_depth = 10,
                                  max_features = 20,
                                  min_samples_split = 7,
                                  n_estimators = 50 )
rf_tuned.fit(X_train, y_train)
```

Out[34]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=10, max_features=20,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=7,
                       min_weight_fraction_leaf=0.0, n_estimators=50,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [37]:
```
y_pred = rf_tuned.predict(X_test)
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print("{} {:0.2f}%".format("Accuracy Score : ", score*100))
```

```
Accuracy Score :  96.85%
```

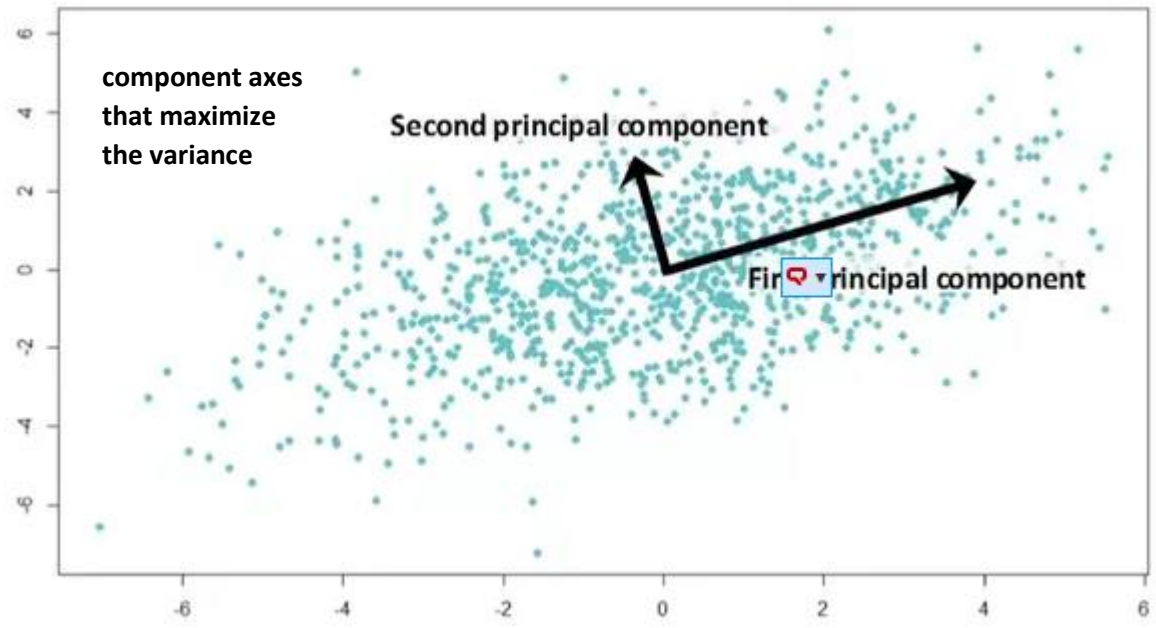
Before the model was set up, parameters were found using the GridSearchCV function to determine the best parameters. The model was fitted by defining four different parameters and CV value as 5. In total, it performed 5 different train and test operations with 240 different parameter combinations (240 * 5 = 1200 // Best parameters: {'max_depth': 10, 'max_features': 20, 'min_samples_split': 7, 'n_estimators': 50). After determining the best parameters, these values were given to the model and fit. Later, test values were given to the model and predict values were found. When the Y_test values and the estimated values were compared, the accuracy rate of the model was found to be 96.85 percent.

Task-5B:

The main purpose is a multivariate statistical analysis technique that transforms a system consisting of many interrelated variables into a new system consisting of less and unrelated new variables as linear functions of these variables. The new system should also be one that can explain as much of the total change from the previous system as possible.

PCA provides the reduction of the number of dimensions and the compression of the data by finding the general properties in the multi-dimensional data. Therefore, some features are lost, but the intention is that these lost features contain little information about the population.

In summary, it is to find the maximum variance in high dimensional data and reduce it to smaller dimensions while preserving the information.



PCA ALGORITHM

- Deciding the size to be reduced (the dimension k)

- Standardization of data

- Obtaining Eigenvalues and Eigenvectors

- Creation of the projection matrix W from the selected k eigenvalues

- Transforming the original data set using the W projection matrix and obtaining the k dimensional Y space.

The areas where PCA is used are;

- Noise Filtering : Revealing the data with Anomaly and Outlier in the dataset with PCA.
- Visualization : Visualization can be done comfortably after the size has been reduced.
- Feature Extraction : New features can be extracted from the data.
- Feature Elimination, Conversion : Removal of excessive features.

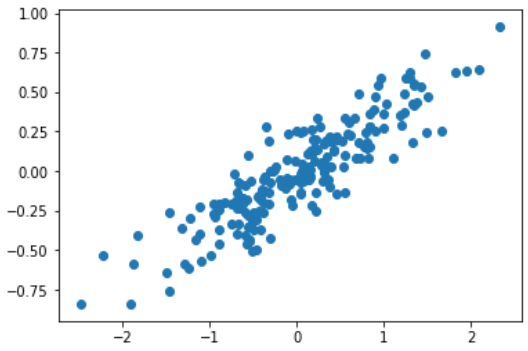
Example: We can define a random data set of 200 data in two dimensions and apply PCA.

In [59]:
```
import numpy as np
```

In [62]:
```
rng = np.random.RandomState(1)
x = np.dot(rng.rand(2,2),rng.randn(2,200)).T
print(x.shape)
```

```
(200, 2)
```

In [63]:
```
plt.scatter(x[:,0], x[:,1])
plt.show()
```



In [64]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(x)
print(pca.components_)
print(pca.explained_variance_)
```

```
[[-0.94446029 -0.32862557]
 [-0.32862557  0.94446029]]
[0.7625315 0.0184779]
```

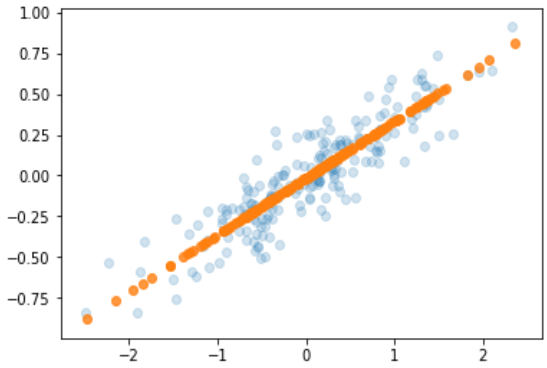
In the example above, we took the dataset directly in 2-dimensional form and applied PCA. Now we can convert this dataset to a single dimension and perform the feature extraction.

Looking at the graph, it is seen that the data set with feature extraction gives a flatter and linear ratio.

In [65]:
```
pca = PCA(n_components=1)
pca.fit(x)
x_pca = pca.transform(x)
print(x_pca.shape)
```

```
(200, 1)
```

In [66]:
```
x2 = pca.inverse_transform(x_pca)
plt.scatter(x[:,0], x[:,1], alpha=0.2)
plt.scatter(x2[:,0], x2[:,1], alpha=0.8)
plt.show()
```



Task-5C:

Standard PCA is much more difficult to use in high dimensional data. In such cases, the Randomized PCA method is used, which allows us to find the number of components faster. It can be considered as a heuristic method.

The IsoMap technique is one of the techniques used within the scope of manifold learning. PCA is successful in reflecting linear relationships between features, but weak in nonlinear relationships. IsoMap is a technique used in a dataset containing nonlinearly associated features. IsoMap stands for isometric mapping. It is a nonlinear dimensionality reduction method.

Randomized PCA

In [51]:
```
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)
```

```
['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony Blair']
(1348, 62, 47)
```

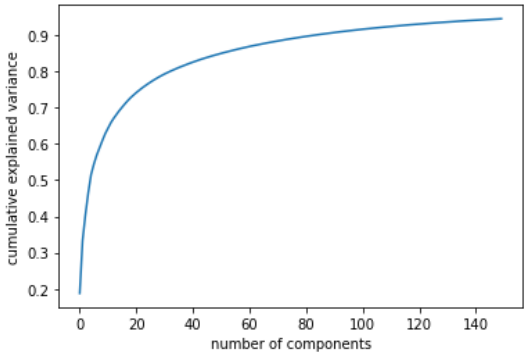
In [52]:
```
from sklearn.decomposition import PCA
pca_model = PCA(n_components=150, svd_solver='randomized', iterated_power=3)
pca_model
```

Out[52]:
```
PCA(iterated_power=3, n_components=150, svd_solver='randomized')
```

In [53]:
```
pca_model.fit(faces.data)
```

Out[53]:
```
PCA(iterated_power=3, n_components=150, svd_solver='randomized')
```

In [55]:
```
import numpy as np
plt.plot(np.cumsum(pca_model.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



The PCA model established with 150 components can explain the dataset with over 90 percent variance.

We can look at the chart below to find the number of components that we can run the model directly and extract the best dataset without giving any component number. The graph shows us that we can explain the dataset with a variance rate of more than 90 percent using 100 components.

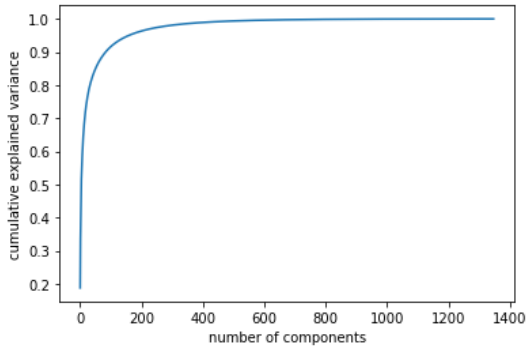
In [59]:
```
pca_model1 = PCA(svd_solver='randomized')
pca_model1
```

Out[59]:
```
PCA(svd_solver='randomized')
```

In [60]:
```
pca_model1.fit(faces.data)
```

Out[60]:
```
PCA(svd_solver='randomized')
```

In [61]:
```
plt.plot(np.cumsum(pca_model1.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```

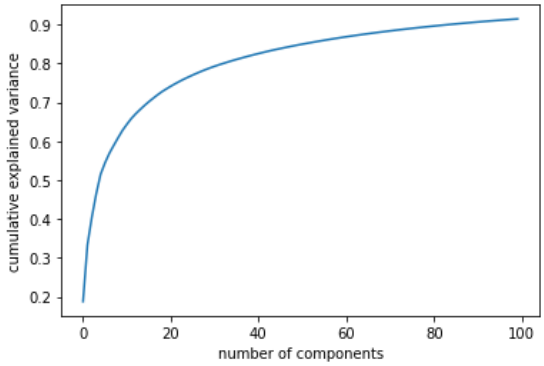


Now, when we set up a PCA model with 100 components, we see a variance rate of around 90 percent.

```
In [62]: from sklearn.decomposition import PCA
         pca_model2 = PCA(n_components=100, svd_solver='randomized', iterated_power=3)
         pca_model2
Out[62]: PCA(iterated_power=3, n_components=100, svd_solver='randomized')

In [63]: pca_model2.fit(faces.data)
Out[63]: PCA(iterated_power=3, n_components=100, svd_solver='randomized')

In [64]: plt.plot(np.cumsum(pca_model2.explained_variance_ratio_))
         plt.xlabel('number of components')
         plt.ylabel('cumulative explained variance');
```



These data show us that we need 100 components to maintain 90% of the variance. This means that data is essentially multidimensional. It cannot be defined as linearly with only a few components. We can try the IsoMap method using fewer components.

IsoMap

```
In [88]: from sklearn.datasets import fetch_lfw_people
         faces = fetch_lfw_people(min_faces_per_person=60)
         print(faces.target_names)
         print(faces.images.shape)

         ['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
          'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony Blair']
         (1348, 62, 47)

In [95]: from sklearn.manifold import Isomap
         model = Isomap(n_components=100)
         proj = model.fit_transform(faces.data)
         proj.shape
Out[95]: (1348, 100)
```

Task-5D:

In [142]:
```
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)
```

```
['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony Blair']
(1348, 62, 47)
```

In [143]:
```
from sklearn.manifold import Isomap
model = Isomap(n_components=100)
proj = model.fit_transform(faces.data)
proj.shape
```

Out[143]: (1348, 100)

In [144]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(proj, faces.target, test_size=0.20, random_state=42)
```

In [145]:
```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
```

In [146]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {
              'max_depth': [2, 3, 5, 8, 10],
              'max_features': [30, 40,50,80,100],
              'min_samples_split': [3, 5, 7, 10]
             }
grid = GridSearchCV(rf_model, param_grid, cv=3, refit=True, verbose=2)
```

In [147]:
```
grid.fit(X_train, y_train)
print(grid.best_params_)
```

```
[CV] max_depth=10, max_features=100, min_samples_split=5 .............
[CV]  max_depth=10, max_features=100, min_samples_split=5, total=   2.6s
[CV] max_depth=10, max_features=100, min_samples_split=5 .............
[CV]  max_depth=10, max_features=100, min_samples_split=5, total=   2.7s
[CV] max_depth=10, max_features=100, min_samples_split=7 .............
[CV]  max_depth=10, max_features=100, min_samples_split=7, total=   2.7s
[CV] max_depth=10, max_features=100, min_samples_split=7 .............
[CV]  max_depth=10, max_features=100, min_samples_split=7, total=   2.6s
[CV] max_depth=10, max_features=100, min_samples_split=7 .............
[CV]  max_depth=10, max_features=100, min_samples_split=7, total=   2.7s
[CV] max_depth=10, max_features=100, min_samples_split=10 ............
[CV]  max_depth=10, max_features=100, min_samples_split=10, total=   2.4s
[CV] max_depth=10, max_features=100, min_samples_split=10 ............
[CV]  max_depth=10, max_features=100, min_samples_split=10, total=   2.4s
[CV] max_depth=10, max_features=100, min_samples_split=10 ............
[CV]  max_depth=10, max_features=100, min_samples_split=10, total=   2.4s
[Parallel(n_jobs=1)]: Done 300 out of 300 | elapsed:  5.2min finished
{'max_depth': 10, 'max_features': 50, 'min_samples_split': 7}
```

In [148]:
```
rf_tuned = RandomForestClassifier(max_depth = 10,
                                  max_features = 50,
                                  min_samples_split = 7)
rf_tuned.fit(X_train, y_train)
```

Out[148]: RandomForestClassifier(max_depth=10, max_features=50, min_samples_split=7)

In [149]:
```
y_pred = rf_tuned.predict(X_test)
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print("{} {:0.2f}%".format("Accuracy Score : ", score*100))
```

```
Accuracy Score :  45.56%
```