Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import accuracy_score
```

Importing the dataset

```
df = pd.read_csv('cancer.csv')
df.replace('?', -99999, inplace=True)
df.drop('id', axis=1, inplace=True)
```

```
df
```

	clump_thickness	unif_cell_size	unif_cell_shape	marg_adhesion	single_epith_cell_size	bare_nuclei	bland_chrom	norm_nucleoli
0	5	1	1	1	2	1	3	1
1	5	4	4	5	7	10	3	2
2	3	1	1	1	2	2	3	1
3	6	8	8	1	3	4	3	7
4	4	1	1	3	2	1	3	1
...	...	...	...	...	...	...	...	...
694	3	1	1	1	3	2	1	1
695	2	1	1	1	2	1	1	1
696	5	10	10	3	7	3	8	10
697	4	8	6	4	3	4	10	6
698	4	8	8	5	4	5	10	4

699 rows × 10 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   clump_thickness         699 non-null    int64 
 1   unif_cell_size          699 non-null    int64 
 2   unif_cell_shape         699 non-null    int64 
 3   marg_adhesion           699 non-null    int64 
 4   single_epith_cell_size  699 non-null    int64 
 5   bare_nuclei             699 non-null    object
 6   bland_chrom             699 non-null    int64 
 7   norm_nucleoli           699 non-null    int64 
 8   mitoses                 699 non-null    int64 
 9   classes                 699 non-null    int64 
dtypes: int64(9), object(1)
memory usage: 54.7+ KB
```

```
X=np.array(df.drop(['classes'],axis=1))
y=np.array(df['classes'])
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.35, random_state = 42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

principle component analysis

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.fit_transform(X_test)
explained_variance=pca.explained_variance_ratio_
```

Fitting KNN to the Training set

```
from sklearn.neighbors import KNeighborsClassifier
knn = []
for i in range(1,21):

    classifier = KNeighborsClassifier(n_neighbors=i)
    trained_model=classifier.fit(X_train,y_train)
    trained_model.fit(X_train,y_train )
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

```
from sklearn.metrics import confusion_matrix

cm_KNN = confusion_matrix(y_test, y_pred)
print(cm_KNN)
print("Accuracy score of train KNN")
print(accuracy_score(y_train, trained_model.predict(X_train))*100)
```

```
[[160   4]
 [  3  78]]
Accuracy score of train KNN
96.0352422907489
```

```
print("Accuracy score of test KNN")
print(accuracy_score(y_test, y_pred)*100)
```

```
Accuracy score of test KNN
97.14285714285714
```

```
knn.append(accuracy_score(y_test, y_pred)*100)
```

Fitting SVM to the Training set

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
```

```
trained_model=classifier.fit(X_train,y_train)
trained_model.fit(X_train,y_train )
```

```
▾                SVC
SVC(kernel='linear', random_state=0)
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

```
from sklearn.metrics import confusion_matrix
cm_SVM = confusion_matrix(y_test, y_pred)
print(cm_SVM)
print("Accuracy score of train SVM")
print(accuracy_score(y_train, trained_model.predict(X_train))*100)
```

```
[[160   4]
 [  4  77]]
Accuracy score of train SVM
96.47577092511013
```

```
print("Accuracy score of test SVM")
print(accuracy_score(y_test, y_pred)*100)
```

```
Accuracy score of test SVM
96.73469387755102
```

```
input_data = (0,1)

# changing input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = trained_model.predict(input_data_reshaped)
print(prediction)


if (prediction[0] == 0):
  print("The Person does not have cancer Disease")

else:
  print("The Person has cancer")
```

```
[0]
The Person does not have cancer Disease
```

```
import pickle
filename = 'cancer_model.sav'
pickle.dump(trained_model, open(filename, 'wb'))
```

```
# loading the saved model
loaded_model = pickle.load(open('cancer_model.sav', 'rb'))
```

Start coding or generate with AI.