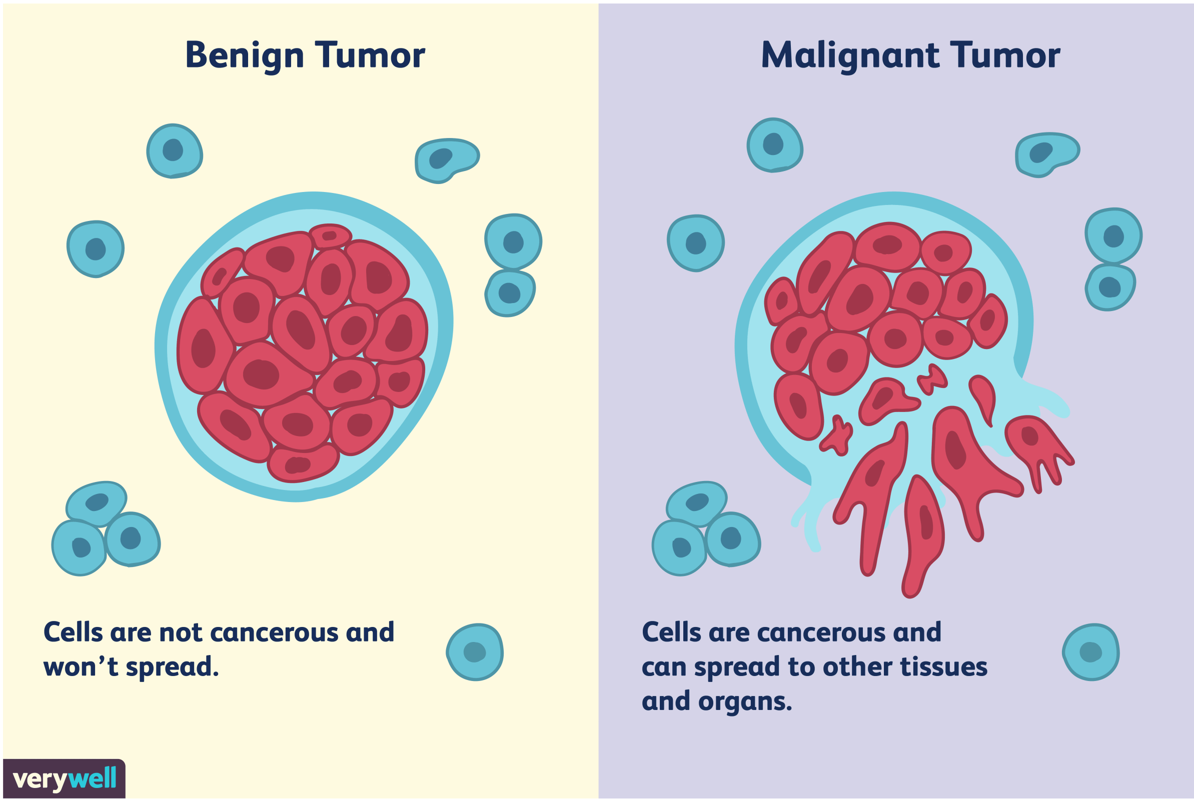
Breast Cancer Prediction

Breast cancer, the most common cancer among women worldwide accounting for 25 percent of all cancer cases and affected 2.1 million people in 2015. Early diagnosis significantly increases the chances of survival.

The key challenge in cancer detection is how to classify tumors into malignant or benign. Machine learning techniques can dramatically improve the accuracy of diagnosis. Research indicates that most experienced physicians can diagnose cancer with 79 percent accuracy while 91 percent correct diagnosis is achieved using machine learning techniques. In this case study our task is to classify tumors into malignant or benign tumors using features obtained from several cell images. Let's take a look at the cancer diagnosis and classification process. So the first step in the cancer diagnosis process is to do what we call it fine needle aspirate or FNA process which is simply extracting some of the cells out of the tumor. And at that stage we don't know if that tumor is malignant or benign. When we say malignant or benign as you guys can see these are kind of the images of the cell. This would be benign tumor and on the right side is the malignant tumor.



And when we say benign that means that the tumor is kind of not spreading across. The body of the patient is safe somehow. However if it's malignant that means it's cancerous. That means we need to intervene and actually stop the cancer growth.

Alright?

So what we do here in the machine learning aspect, so now as we extracted all these images and we want to specify if that cancer out of these images is malignant or benign, that's the whole idea. So what we do with that, we extract out of these images some features. When we say features that mean some characteristics out of the image such as radius for example of the cells such as texture, perimeter, area, smoothness and so on. And then we feed all these features into our machine learning model which is kind of a brain in a way.

OK?

The idea is we want to teach the machine how to basically classify images or classify data and tell us OK if it's malignant or benign. For example in this case without any human intervention which is going to train the model. Once the model is trained we're good to go. We can use it in practice to classify new images as we move forward.

All right.

And that's kind of the overall procedure or the cancer diagnosis procedure. And by that we conclude the problem case. And now let's shift into some of the machine learning terms.

Importing Essential Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import classification_report, confusion_matrix
```

Data Exploration

NOTE - The data is obtained from sci-kit learn's libarary. Due to this the data is already cleaned and preprocessed. So, there is no need for us to do any manipulation with the data. We can simply move forward skipping this step.

In [2]:
```
from sklearn.datasets import load_breast_cancer
```

In [3]:
```
cancer = load_breast_cancer()
```

In [4]:
```
cancer
```

Out[4]:
```
{'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]),
 'frame': None,
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) datase
t\n--------------------------------------------\n\n**Data Set Characteristics:**\n\n
:Number of Instances: 569\n\n    :Number of Attributes: 30 numeric, predictive attri
butes and the class\n\n    :Attribute Information:\n        - radius (mean of distan
ces from center to points on the perimeter)\n        - texture (standard deviation o
f gray-scale values)\n        - perimeter\n        - area\n        - smoothness (loc
al variation in radius lengths)\n        - compactness (perimeter^2 / area - 1.0)\n
- concavity (severity of concave portions of the contour)\n        - concave points
(number of concave portions of the contour)\n        - symmetry\n        - fractal d
imension ("coastline approximation" - 1)\n\n        The mean, standard error, and "w
orst" or largest (mean of the three\n        worst/largest values) of these features
were computed for each image,\n        resulting in 30 features.  For instance, fiel
d 0 is Mean Radius, field\n        10 is Radius SE, field 20 is Worst Radius.\n\n
- class:\n                - WDBC-Malignant\n                - WDBC-Benign\n\n    :Su
mmary Statistics:\n\n    ===================================== ====== ======\n
Min    Max\n    ===================================== ====== ======\n    radius (mea
n):                           6.981  28.11\n    texture (mean):
9.71   39.28\n    perimeter (mean):                     43.79  188.5\n    area (mea
n):                             143.5  2501.0\n    smoothness (mean):
0.053  0.163\n    compactness (mean):                   0.019  0.345\n    concavity
(mean):                         0.0    0.427\n    concave points (mean):
0.0    0.201\n    symmetry (mean):                      0.106  0.304\n    fractal di
mension (mean):                 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    compactnes
s (standard error):             0.002  0.135\n    concavity (standard error):
0.0    0.396\n    concave points (standard error):      0.0    0.053\n    symmetry
(standard error):                0.008  0.079\n    fractal dimension (standard error):
0.001  0.03\n    radius (worst):                        7.93   36.04\n    texture (wo
rst):                           12.02  49.54\n    perimeter (worst):                    5
0.41  251.2\n    area (worst):                          185.2  4254.0\n    smoothness
(worst):                        0.071  0.223\n    compactness (worst):
0.027  1.058\n    concavity (worst):                    0.0    1.252\n    concave po
ints (worst):                   0.0    0.291\n    symmetry (worst):
0.156  0.664\n    fractal dimension (worst):            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 Street, Olvi L. Mangasarian\n\n    :Donor: Nick Street\n\n    :Dat
e: November, 1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) d
atasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of a
fine needle\naspirate (FNA) of a breast mass.  They describe\ncharacteristics of the
cell nuclei present in the image.\n\nSeparating plane described above was obtained u
sing\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nConstruction
```

```
Via Linear Programming." Proceedings of the 4th\nMidwest Artificial Intelligence and
Cognitive Science Society,\npp. 97-101, 1992], a classification method which uses li
near\nprogramming to construct a decision tree.  Relevant features\nwere selected us
ing an exhaustive search in the space of 1-4\nfeatures and 1-3 separating planes.\n
\nThe actual linear program used to obtain the separating plane\nin the 3-dimensiona
l space is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust Linear
\nProgramming Discrimination of Two Linearly Inseparable Sets",\nOptimization Method
s and Software 1, 1992, 23-34].\n\nThis database is also available through the UW CS
ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-learn/WDBC/\n
\n.. topic:: References\n\n   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nucl
ear feature extraction \n     for breast tumor diagnosis. IS&T/SPIE 1993 Internation
al Symposium on \n     Electronic Imaging: Science and Technology, volume 1905, page
s 861-870,\n     San Jose, CA, 1993.\n   - O.L. Mangasarian, W.N. Street and W.H. Wo
lberg. Breast cancer diagnosis and \n     prognosis via linear programming. Operatio
ns Research, 43(4), pages 570-577, \n     July-August 1995.\n   - W.H. Wolberg, W.N.
Street, and O.L. Mangasarian. Machine learning techniques\n     to diagnose breast c
ancer from fine-needle aspirates. Cancer Letters 77 (1994) \n     163-171.',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'mean are
a',
        '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': 'D:\\Anaconda\\lib\\site-packages\\sklearn\\datasets\\data\\breast_canc
er.csv'}
```

In [5]:
```
cancer.keys()
```

Out[5]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'fil
ename'])
```

In [6]:
```
print(cancer['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 0 is Mean Radius, field
        10 is Radius SE, field 20 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign
```

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058
concavity (worst):	0.0	1.252
concave points (worst):	0.0	0.291
symmetry (worst):	0.156	0.664
fractal dimension (worst):	0.055	0.208

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

:Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes.

The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

```
This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction 
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on 
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and 
     prognosis via linear programming. Operations Research, 43(4), pages 570-577, 
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) 
     163-171.
```

In [7]:

```python
print(cancer['target'])
```

```
[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 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 1 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]
```

In [8]:

```python
print(cancer['target_names'])
```

```
['malignant' 'benign']
```

In [9]:

```python
print(cancer['feature_names'])
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean 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']
```

In [10]:

```python
cancer['data'].shape
```

Out[10]:

```
(569, 30)
```

In [11]:

```python
df_cancer = pd.DataFrame(np.c_[cancer['data'], cancer['target']], columns = np.appen
df_cancer.head()
```

Out[11]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	di
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	

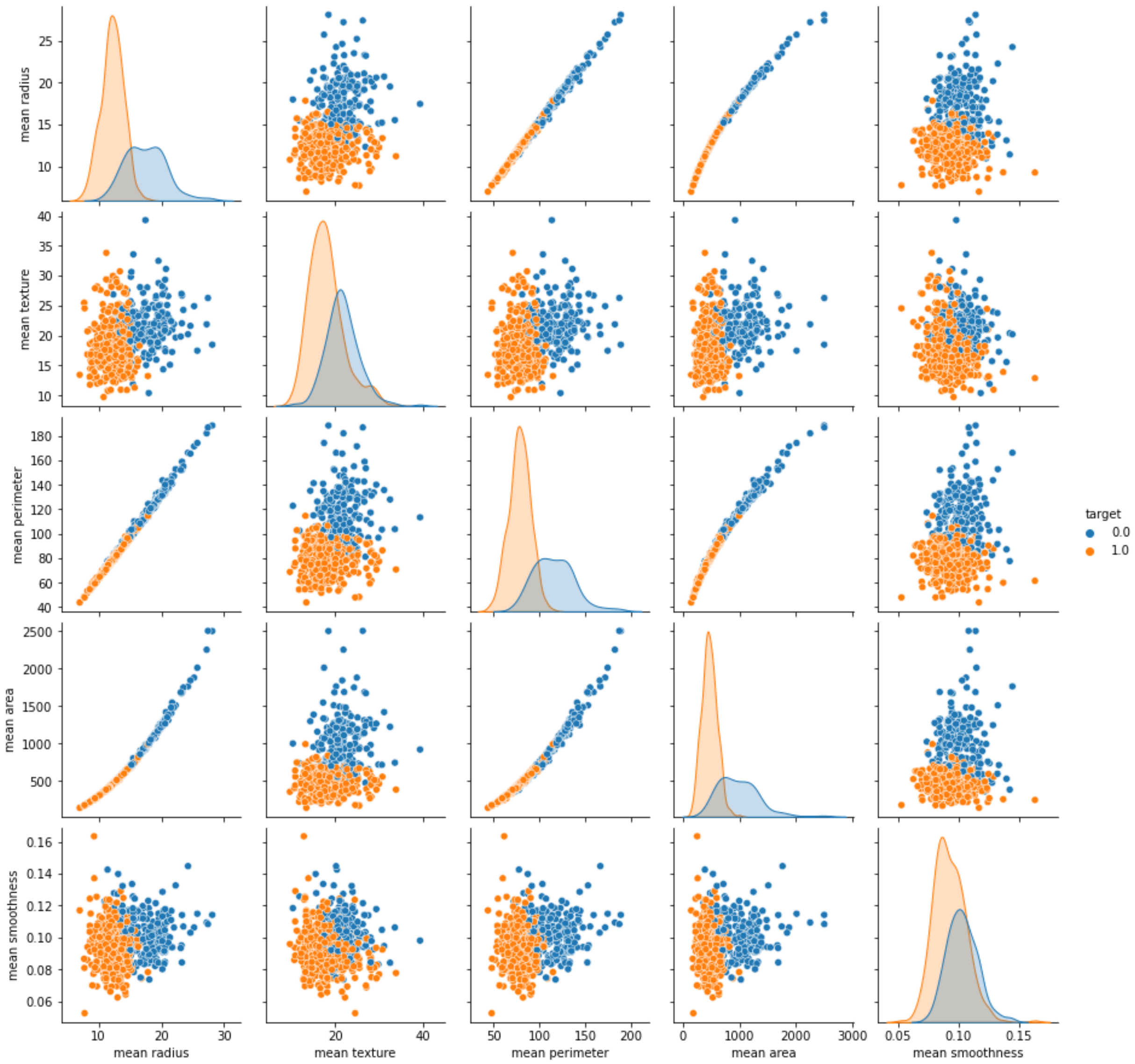
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	di
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 31 columns

Visualization

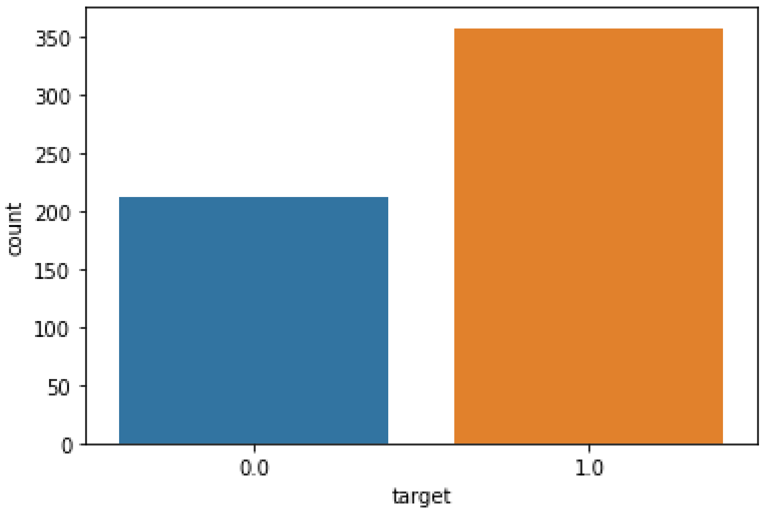
In [68]:

```
sns.pairplot(df_cancer, vars = ['mean radius', 'mean texture', 'mean perimeter', 'me
#plt.savefig('1.Pair Plot of first 5 Features.png', dpi = 300)
```



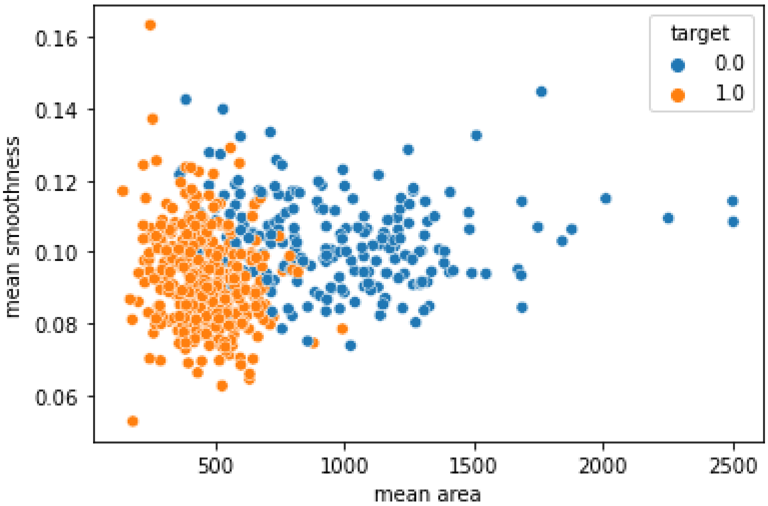
In [69]:

```
sns.countplot(df_cancer['target'])
#plt.savefig('2.Count Plot of Target.png', dpi = 300)
```



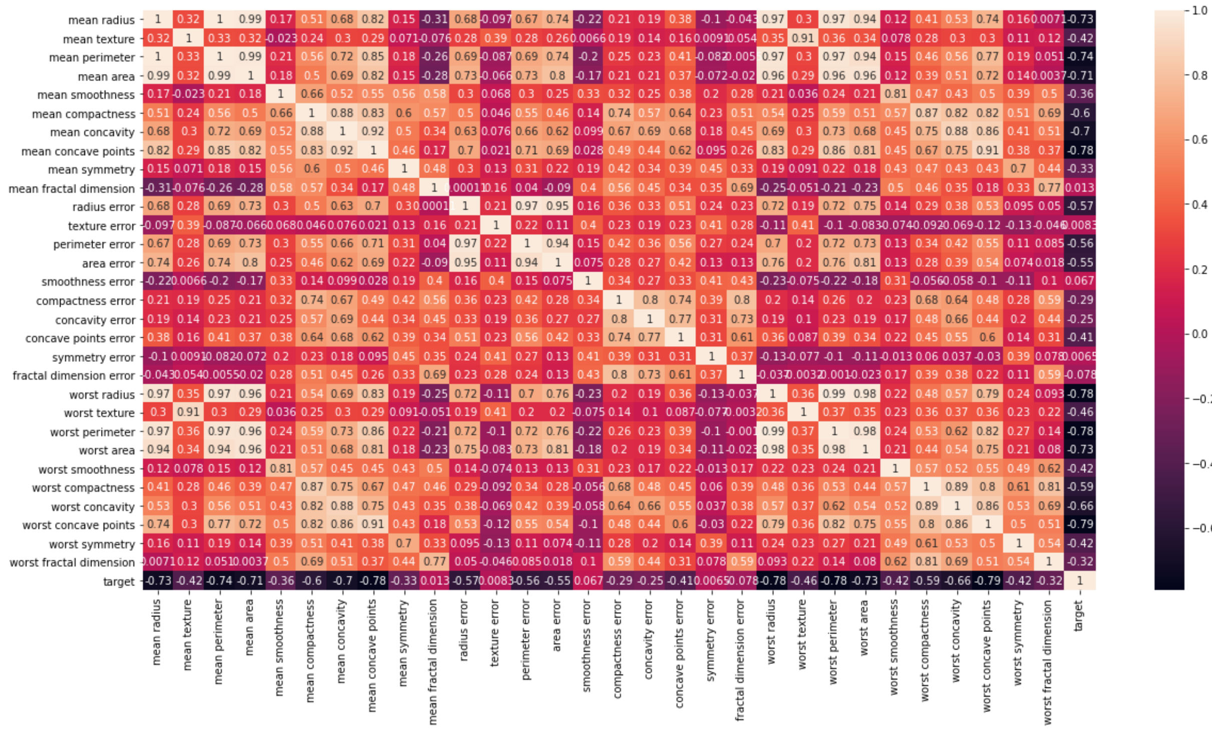
In [70]:

```python
sns.scatterplot(x = 'mean area', y = 'mean smoothness', hue = 'target', data = df_ca
#plt.savefig('3.Scatterplot of Mean(Area vs Smoothness).png', dpi = 300)
```



In [71]:

```python
plt.figure(figsize = (20, 10))
sns.heatmap(df_cancer.corr(), annot = True)
#plt.savefig('4.Heatmap - Correlation of all variables.png', dpi = 300)
```



In [16]:
```python
X = df_cancer.iloc[:, :-1]
X
```

Out[16]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809
...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587

569 rows × 30 columns

In [17]:
```python
y = df_cancer['target']
y
```

Out[17]:
```
0      0.0
1      0.0
2      0.0
3      0.0
4      0.0
      ...
564    0.0
565    0.0
566    0.0
567    0.0
568    1.0
Name: target, Length: 569, dtype: float64
```

In [18]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_st
```

In [19]:
```python
X_train
```

Out[19]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
338	10.050	17.53	64.41	310.8	0.10070	0.07326	0.02511	0.01775	0.1890
427	10.800	21.98	68.79	359.9	0.08801	0.05743	0.03614	0.01404	0.2016
406	16.140	14.86	104.30	800.0	0.09495	0.08501	0.05500	0.04528	0.1735
96	12.180	17.84	77.79	451.1	0.10450	0.07057	0.02490	0.02941	0.1900
490	12.250	22.44	78.18	466.5	0.08192	0.05200	0.01714	0.01261	0.1544

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
...	...	...	...	...	...	...	...	...	...
277	18.810	19.98	120.90	1102.0	0.08923	0.05884	0.08020	0.05843	0.1550
9	12.460	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.08543	0.2030
359	9.436	18.32	59.82	278.6	0.10090	0.05956	0.02710	0.01406	0.1506
192	9.720	18.22	60.73	288.1	0.06950	0.02344	0.00000	0.00000	0.1653
559	11.510	23.93	74.52	403.5	0.09261	0.10210	0.11120	0.04105	0.1388

455 rows × 30 columns

In [20]:
```
y_train
```

Out[20]:
```
338    1.0
427    1.0
406    1.0
96     1.0
490    1.0
      ... 
277    0.0
9      0.0
359    1.0
192    1.0
559    1.0
Name: target, Length: 455, dtype: float64
```

In [21]:
```
X_test
```

Out[21]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
512	13.40	20.52	88.64	556.7	0.11060	0.14690	0.14450	0.08172	0.2116
457	13.21	25.25	84.10	537.9	0.08791	0.05205	0.02772	0.02068	0.1619
439	14.02	15.66	89.59	606.5	0.07966	0.05581	0.02087	0.02652	0.1589
298	14.26	18.17	91.22	633.1	0.06576	0.05220	0.02475	0.01374	0.1635
37	13.03	18.42	82.61	523.8	0.08983	0.03766	0.02562	0.02923	0.1467
...	...	...	...	...	...	...	...	...	...
213	17.42	25.56	114.50	948.0	0.10060	0.11460	0.16820	0.06597	0.1308
519	12.75	16.70	82.51	493.8	0.11250	0.11170	0.03880	0.02995	0.2120
432	20.18	19.54	133.80	1250.0	0.11330	0.14890	0.21330	0.12590	0.1724
516	18.31	20.58	120.80	1052.0	0.10680	0.12480	0.15690	0.09451	0.1860
500	15.04	16.74	98.73	689.4	0.09883	0.13640	0.07721	0.06142	0.1668

114 rows × 30 columns

In [22]:
```
y_test
```

Out[22]:
```
512    0.0
457    1.0
439    1.0
298    1.0
37     1.0
      ...
213    0.0
519    1.0
432    0.0
516    0.0
500    1.0
Name: target, Length: 114, dtype: float64
```

As we saw above, our features are not scaled to each other. Hence, we will scale them using feature scaling method.

Feature Scaling

In [23]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [24]:
```
X_train
```

Out[24]:
```
array([[-1.15036482, -0.39064196, -1.12855021, ..., -0.75798367,
        -0.01614761, -0.38503402],
       [-0.93798972,  0.68051405, -0.94820146, ..., -0.60687023,
         0.09669004, -0.38615797],
       [ 0.574121  , -1.03333557,  0.51394098, ..., -0.02371948,
        -0.20050207, -0.75144254],
       ...,
       [-1.32422924, -0.20048168, -1.31754581, ..., -0.97974953,
        -0.71542314, -0.11978123],
       [-1.24380987, -0.2245526 , -1.28007609, ..., -1.75401433,
        -1.58157125, -1.00601779],
       [-0.73694129,  1.14989702, -0.71226578, ..., -0.27460457,
        -1.25895095,  0.21515662]])
```

In [25]:
```
X_test
```

Out[25]:
```
array([[-0.20175604,  0.3290786 , -0.13086754, ...,  1.3893291 ,
         1.08203284,  1.54029664],
       [-0.25555773,  1.46763319, -0.31780437, ..., -0.83369364,
        -0.73131577, -0.87732522],
       [-0.02619262, -0.8407682 , -0.09175081, ..., -0.49483785,
        -1.22080864, -0.92115937],
       ...,
       [ 1.71811488,  0.09318356,  1.7286186 , ...,  1.57630515,
         0.20317063, -0.15406178],
       [ 1.18859296,  0.34352115,  1.19333694, ...,  0.56019755,
         0.26991966, -0.27320074],
       [ 0.26263752, -0.58080224,  0.28459338, ..., -0.19383705,
        -1.15564888,  0.11231497]])
```

As we see, our features are properly scaled. Now, its time to apply an algorithm

Model Building & Evaluation

Random Forest

In [26]:
```
from sklearn.ensemble import RandomForestClassifier
random_classifier = RandomForestClassifier(n_estimators = 50, criterion = 'entropy',
random_classifier.fit(X_train, y_train)
```

Out[26]:
```
RandomForestClassifier(criterion='entropy', n_estimators=50, random_state=5)
```

In [27]:
```
y_pred_random = random_classifier.predict(X_test)
```

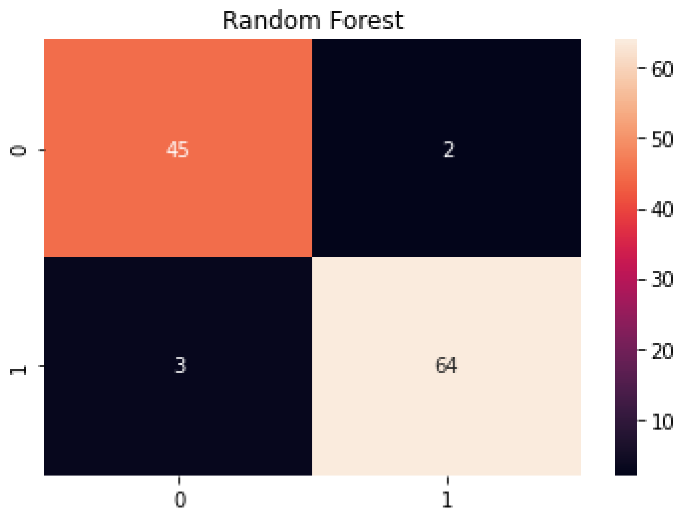
In [28]:
```
y_pred_random
```

Out[28]:
```
array([0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 0., 0., 1.,
       0., 0., 0., 0., 0., 1., 1., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 1., 0., 1., 1., 0., 1., 1., 1., 0., 0.,
       0., 0., 1., 1., 1., 1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 0.,
       0., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 0., 0., 0., 1., 0.,
       1., 1., 1., 0., 0., 1., 0., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 0., 0., 1., 0., 0., 1.])
```

Confusion Matrix & Classification Report

In [29]:
```
cm_random = confusion_matrix(y_test, y_pred_random)
```

In [72]:
```
sns.heatmap(cm_random, annot = True)
plt.title("Random Forest")
plt.savefig('5.Confusion Matrix - Random Forest.png', dpi = 300)
```



In [31]:
```
print(classification_report(y_test, y_pred_random))
```

```
              precision    recall  f1-score   support

         0.0       0.94      0.96      0.95        47
         1.0       0.97      0.96      0.96        67

    accuracy                           0.96       114
   macro avg       0.95      0.96      0.95       114
weighted avg       0.96      0.96      0.96       114
```

K-Fold Cross Validation

In [32]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = random_classifier, X = X_train, y = y_train
```

```
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 96.27 %
Standard Deviation: 3.26 %
```

Support Vector Classification

In [33]:
```
from sklearn.svm import SVC
```

In [34]:
```
svc_classifier = SVC(random_state= 1)
```

In [35]:
```
svc_classifier.fit(X_train, y_train)
```

Out[35]:
```
SVC(random_state=1)
```

In [36]:
```
y_pred_svc = svc_classifier.predict(X_test)
```

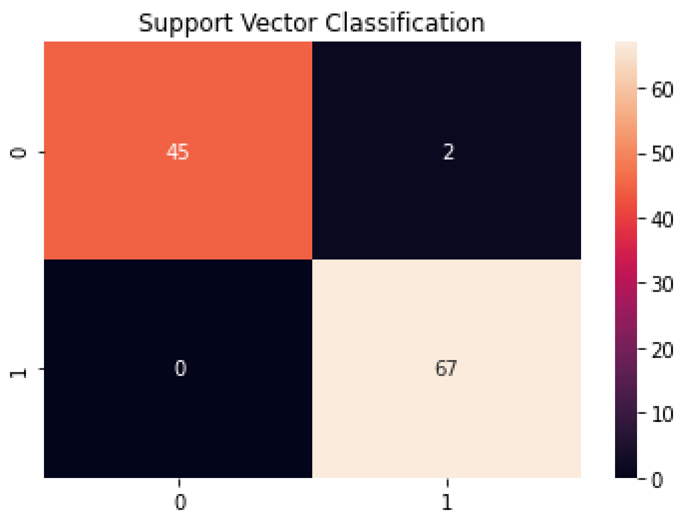
In [37]:
```
y_pred_svc
```

Out[37]:
```
array([0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1.,
       0., 0., 0., 0., 0., 1., 1., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 1., 0., 0.,
       0., 0., 1., 1., 1., 1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 0.,
       0., 1., 1., 0., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0., 1., 0.,
       1., 1., 1., 0., 0., 1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 1., 0., 1., 0., 0., 1.])
```

Confusion Matrix & Classification Report

In [38]:
```
cm_svc = confusion_matrix(y_test, y_pred_svc)
```

In [73]:
```
sns.heatmap(cm_svc, annot = True)
plt.title('Support Vector Classification')
plt.savefig('6.Confusion Matrix - Support Vector Classification.png', dpi = 300)
```



In [40]:
```
print(classification_report(y_test, y_pred_svc))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.96      0.98        47
         1.0       0.97      1.00      0.99        67

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
```

```
weighted avg       0.98      0.98      0.98       114
```

K-Fold Cross Validation

In [41]:
```python
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = svc_classifier, X = X_train, y = y_train, c
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 97.59 %
Standard Deviation: 1.53 %
```

XGBoost

In [42]:
```python
from xgboost import XGBClassifier
```

In [43]:
```python
xgb_classifier = XGBClassifier(seed=2)
```

In [44]:
```python
xgb_classifier.fit(X_train, y_train)
```

Out[44]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=2,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=2,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

In [45]:
```python
y_pred_xgb = xgb_classifier.predict(X_test)
```

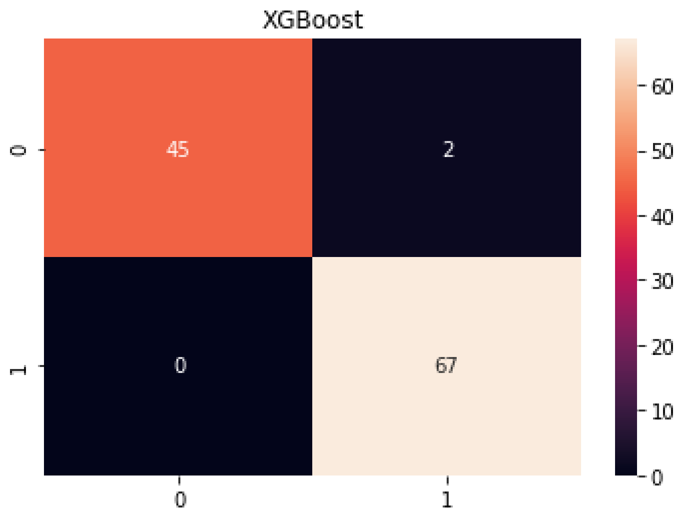
In [46]:
```python
y_pred_xgb
```

Out[46]:
```
array([0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1.,
       0., 0., 0., 0., 0., 1., 1., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 1., 0., 1., 1., 0., 1., 1., 1., 0., 0.,
       0., 0., 1., 1., 1., 1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 0.,
       0., 1., 1., 0., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0., 1., 0.,
       1., 1., 1., 0., 0., 1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 0., 0., 1., 0., 0., 1.])
```

Confusion Matrix & Classification Report

In [47]:
```python
cm_xgb = confusion_matrix(y_test, y_pred_xgb)
```

In [74]:
```python
sns.heatmap(cm_xgb, annot = True)
plt.title("XGBoost")
plt.savefig('7.Confusion Matrix - XGBoost.png', dpi = 300)
```



In [49]:
```
print(classification_report(y_test, y_pred_xgb))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.96      0.98        47
         1.0       0.97      1.00      0.99        67

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

K-Fold Cross Validation

In [50]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = xgb_classifier, X = X_train, y = y_train, c
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 96.04 %
Standard Deviation: 2.76 %
```

Parameter Tuning of SVC

In [51]:
```
from sklearn.model_selection import GridSearchCV
param_grid_svc = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel':
```

In [52]:
```
grid_svc = GridSearchCV(SVC(), param_grid_svc, refit = True, verbose = 4)
```

In [53]:
```
grid_svc.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.637, total=   0.1s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.637, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.637, total=   0.1s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.637, total=   0.0s
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV] ......... C=0.1, gamma=1, kernel=rbf, score=0.637, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ...................................
```

```
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.912, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.923, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.923, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.923, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.934, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.901, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.967, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.934, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.956, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.945, total=   0.0s
[CV] C=0.1, gamma=0.001, kernel=rbf ..................................
[CV] ...... C=0.1, gamma=0.001, kernel=rbf, score=0.736, total=   0.0s
[CV] C=0.1, gamma=0.001, kernel=rbf ..................................
[CV] ...... C=0.1, gamma=0.001, kernel=rbf, score=0.714, total=   0.0s
[CV] C=0.1, gamma=0.001, kernel=rbf ..................................
[CV] ...... C=0.1, gamma=0.001, kernel=rbf, score=0.703, total=   0.0s
[CV] C=0.1, gamma=0.001, kernel=rbf ..................................
[CV] ...... C=0.1, gamma=0.001, kernel=rbf, score=0.714, total=   0.0s
[CV] C=0.1, gamma=0.001, kernel=rbf ..................................
[CV] ...... C=0.1, gamma=0.001, kernel=rbf, score=0.714, total=   0.0s
[CV] C=1, gamma=1, kernel=rbf ........................................
[CV] ............ C=1, gamma=1, kernel=rbf, score=0.648, total=   0.0s
[CV] C=1, gamma=1, kernel=rbf ........................................
[CV] ............ C=1, gamma=1, kernel=rbf, score=0.637, total=   0.0s
[CV] C=1, gamma=1, kernel=rbf ........................................
[CV] ............ C=1, gamma=1, kernel=rbf, score=0.637, total=   0.0s
[CV] C=1, gamma=1, kernel=rbf ........................................
[CV] ............ C=1, gamma=1, kernel=rbf, score=0.648, total=   0.0s
[CV] C=1, gamma=1, kernel=rbf ........................................
[CV] ............ C=1, gamma=1, kernel=rbf, score=0.637, total=   0.0s
[CV] C=1, gamma=0.1, kernel=rbf ......................................
[CV] .......... C=1, gamma=0.1, kernel=rbf, score=0.967, total=   0.0s
[CV] C=1, gamma=0.1, kernel=rbf ......................................
[CV] .......... C=1, gamma=0.1, kernel=rbf, score=0.967, total=   0.0s
[CV] C=1, gamma=0.1, kernel=rbf ......................................
[CV] .......... C=1, gamma=0.1, kernel=rbf, score=0.923, total=   0.0s
[CV] C=1, gamma=0.1, kernel=rbf ......................................
[CV] .......... C=1, gamma=0.1, kernel=rbf, score=0.967, total=   0.0s
[CV] C=1, gamma=0.1, kernel=rbf ......................................
[CV] .......... C=1, gamma=0.1, kernel=rbf, score=0.956, total=   0.0s
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ......... C=1, gamma=0.01, kernel=rbf, score=0.978, total=   0.0s
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ......... C=1, gamma=0.01, kernel=rbf, score=0.978, total=   0.0s
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ......... C=1, gamma=0.01, kernel=rbf, score=0.934, total=   0.0s
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ......... C=1, gamma=0.01, kernel=rbf, score=0.978, total=   0.0s
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ......... C=1, gamma=0.01, kernel=rbf, score=0.989, total=   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
[CV] ........ C=1, gamma=0.001, kernel=rbf, score=0.923, total=   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
[CV] ........ C=1, gamma=0.001, kernel=rbf, score=0.967, total=   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
[CV] ........ C=1, gamma=0.001, kernel=rbf, score=0.923, total=   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
[CV] ........ C=1, gamma=0.001, kernel=rbf, score=0.967, total=   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
[CV] ........ C=1, gamma=0.001, kernel=rbf, score=0.945, total=   0.0s
```

```
[CV] C=10, gamma=1, kernel=rbf .......................................
[CV] ........... C=10, gamma=1, kernel=rbf, score=0.648, total=   0.0s
[CV] C=10, gamma=1, kernel=rbf .......................................
[CV] ........... C=10, gamma=1, kernel=rbf, score=0.637, total=   0.0s
[CV] C=10, gamma=1, kernel=rbf .......................................
[CV] ........... C=10, gamma=1, kernel=rbf, score=0.637, total=   0.0s
[CV] C=10, gamma=1, kernel=rbf .......................................
[CV] ........... C=10, gamma=1, kernel=rbf, score=0.648, total=   0.0s
[CV] C=10, gamma=1, kernel=rbf .......................................
[CV] ........... C=10, gamma=1, kernel=rbf, score=0.637, total=   0.0s
[CV] C=10, gamma=0.1, kernel=rbf .....................................
[CV] ......... C=10, gamma=0.1, kernel=rbf, score=0.956, total=   0.0s
[CV] C=10, gamma=0.1, kernel=rbf .....................................
[CV] ......... C=10, gamma=0.1, kernel=rbf, score=0.945, total=   0.0s
[CV] C=10, gamma=0.1, kernel=rbf .....................................
[CV] ......... C=10, gamma=0.1, kernel=rbf, score=0.923, total=   0.0s
[CV] C=10, gamma=0.1, kernel=rbf .....................................
[CV] ......... C=10, gamma=0.1, kernel=rbf, score=0.945, total=   0.0s
[CV] C=10, gamma=0.1, kernel=rbf .....................................
[CV] ......... C=10, gamma=0.1, kernel=rbf, score=0.945, total=   0.0s
[CV] C=10, gamma=0.01, kernel=rbf ....................................
[CV] ........ C=10, gamma=0.01, kernel=rbf, score=0.978, total=   0.0s
[CV] C=10, gamma=0.01, kernel=rbf ....................................
[CV] ........ C=10, gamma=0.01, kernel=rbf, score=0.989, total=   0.0s
[CV] C=10, gamma=0.01, kernel=rbf ....................................
[CV] ........ C=10, gamma=0.01, kernel=rbf, score=0.934, total=   0.0s
[CV] C=10, gamma=0.01, kernel=rbf ....................................
[CV] ........ C=10, gamma=0.01, kernel=rbf, score=1.000, total=   0.0s
[CV] C=10, gamma=0.01, kernel=rbf ....................................
[CV] ........ C=10, gamma=0.01, kernel=rbf, score=1.000, total=   0.0s
[CV] C=10, gamma=0.001, kernel=rbf ...................................
[CV] ....... C=10, gamma=0.001, kernel=rbf, score=0.978, total=   0.0s
[CV] C=10, gamma=0.001, kernel=rbf ...................................
[CV] ....... C=10, gamma=0.001, kernel=rbf, score=0.978, total=   0.0s
[CV] C=10, gamma=0.001, kernel=rbf ...................................
[CV] ....... C=10, gamma=0.001, kernel=rbf, score=0.934, total=   0.0s
[CV] C=10, gamma=0.001, kernel=rbf ...................................
[CV] ....... C=10, gamma=0.001, kernel=rbf, score=0.978, total=   0.0s
[CV] C=10, gamma=0.001, kernel=rbf ...................................
[CV] ....... C=10, gamma=0.001, kernel=rbf, score=0.989, total=   0.0s
[CV] C=100, gamma=1, kernel=rbf ......................................
[CV] .......... C=100, gamma=1, kernel=rbf, score=0.648, total=   0.0s
[CV] C=100, gamma=1, kernel=rbf ......................................
[CV] .......... C=100, gamma=1, kernel=rbf, score=0.637, total=   0.0s
[CV] C=100, gamma=1, kernel=rbf ......................................
[CV] .......... C=100, gamma=1, kernel=rbf, score=0.637, total=   0.0s
[CV] C=100, gamma=1, kernel=rbf ......................................
[CV] .......... C=100, gamma=1, kernel=rbf, score=0.648, total=   0.0s
[CV] C=100, gamma=1, kernel=rbf ......................................
[CV] .......... C=100, gamma=1, kernel=rbf, score=0.637, total=   0.0s
[CV] C=100, gamma=0.1, kernel=rbf ....................................
[CV] ........ C=100, gamma=0.1, kernel=rbf, score=0.945, total=   0.0s
[CV] C=100, gamma=0.1, kernel=rbf ....................................
[CV] ........ C=100, gamma=0.1, kernel=rbf, score=0.945, total=   0.0s
[CV] C=100, gamma=0.1, kernel=rbf ....................................
[CV] ........ C=100, gamma=0.1, kernel=rbf, score=0.923, total=   0.0s
[CV] C=100, gamma=0.1, kernel=rbf ....................................
[CV] ........ C=100, gamma=0.1, kernel=rbf, score=0.945, total=   0.0s
[CV] C=100, gamma=0.1, kernel=rbf ....................................
[CV] ........ C=100, gamma=0.1, kernel=rbf, score=0.934, total=   0.0s
[CV] C=100, gamma=0.01, kernel=rbf ...................................
[CV] ....... C=100, gamma=0.01, kernel=rbf, score=0.989, total=   0.0s
[CV] C=100, gamma=0.01, kernel=rbf ...................................
[CV] ....... C=100, gamma=0.01, kernel=rbf, score=0.978, total=   0.0s
[CV] C=100, gamma=0.01, kernel=rbf ...................................
[CV] ....... C=100, gamma=0.01, kernel=rbf, score=0.945, total=   0.0s
[CV] C=100, gamma=0.01, kernel=rbf ...................................
[CV] ....... C=100, gamma=0.01, kernel=rbf, score=0.989, total=   0.0s
[CV] C=100, gamma=0.01, kernel=rbf ...................................
```

```
[CV] ....... C=100, gamma=0.01, kernel=rbf, score=0.989, total=   0.0s
[CV] C=100, gamma=0.001, kernel=rbf ..................................
[CV] ...... C=100, gamma=0.001, kernel=rbf, score=0.989, total=   0.0s
[CV] C=100, gamma=0.001, kernel=rbf ..................................
[CV] ...... C=100, gamma=0.001, kernel=rbf, score=0.978, total=   0.0s
[CV] C=100, gamma=0.001, kernel=rbf ..................................
[CV] ...... C=100, gamma=0.001, kernel=rbf, score=0.945, total=   0.0s
[CV] C=100, gamma=0.001, kernel=rbf ..................................
[CV] ...... C=100, gamma=0.001, kernel=rbf, score=1.000, total=   0.0s
[CV] C=100, gamma=0.001, kernel=rbf ..................................
[CV] ...... C=100, gamma=0.001, kernel=rbf, score=0.989, total=   0.0s
[Parallel(n_jobs=1)]: Done  80 out of  80 | elapsed:    1.3s finished
```

Out[53]:
```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001],
                         'kernel': ['rbf']},
             verbose=4)
```

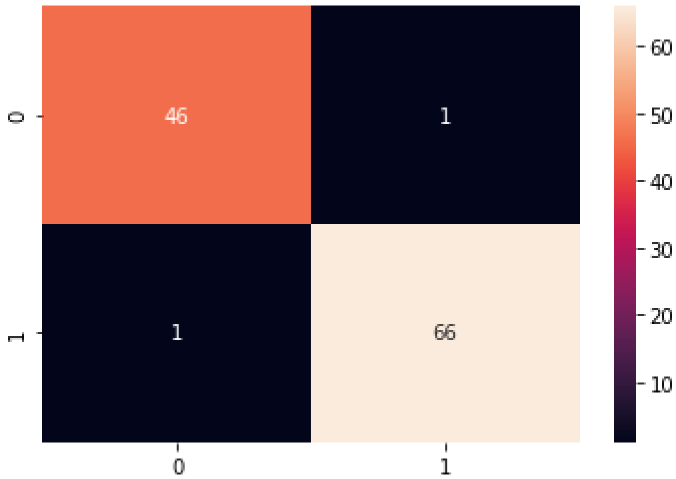
In [54]:
```
grid_svc.best_params_
```

Out[54]:
```
{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
```

In [55]:
```
grid_predictions_svc = grid_svc.predict(X_test)
```

In [56]:
```
cm = confusion_matrix(y_test, grid_predictions_svc)
```

In [75]:
```
sns.heatmap(cm, annot = True)
plt.savefig('8.Parameter Tuned Confusion Matrix - Support Vector Classification.png'
```



In [58]:
```
print(classification_report(y_test, grid_predictions_svc))
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.98      0.98        47
         1.0       0.99      0.99      0.99        67

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

Note

Results in normal SVC are better than that of parameter tuning. So will go with the results of normal SVC. However I have not removed it just for you to see it. The reason why we cannot consider the results of parameter tuning is that it gives us a 'Type 2 Error' while the one with default parameter does not give any 'Type 2 Error'.

- For a detailed explaination of my findings read my Report and Presentation.
- If you want to see only the Visualizations/Plots I have added a folder in this project's main folder named 'Visualizations' (obviously!) where I have uploaded all the images.