Description

- This analysis is used to detect breast cancer, based off of data.
- The dataset consists of features related to characteristics of cancer tumor i.e. radius
- the dataset is labeled by types of cancer, Malignant and Benign.

Initial plan for Data exploration

- After importing dataset in form of dataframe, the features which include NaN are checked and cleaned
- The types of cancer are preliminary checked and visualized by value_counts() and bar plot, respectively.
- The type of each features should be preprocessed to analyze the insight.

1. Data preparation

In [2]:

```
#Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [9]:
```
#Load the data
df = pd.read_csv('data.csv')
df.head(7)
#column: M = Malignant, B = Benign
```

Out[9]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	poin
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	
5	843786	M	12.45	15.70	82.57	477.1	0.12780	0.17000	0.1578	
6	844359	M	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.1127	

7 rows × 33 columns

In [10]:
```
#Count the number of rows and columns in the data
df.shape
```

Out[10]: (569, 33)

In [11]:
```
#Count the number of empty (NaN, NAN, na) values in each column
df.isna().sum()
```

Out[11]:
```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

In [14]:
```
#Drop the column with all missing values
df = df.dropna(axis=1)
```

In [15]:
```
#Get the new count of the number of rows and column
df.shape
```

Out[15]:
```
(569, 32)
```

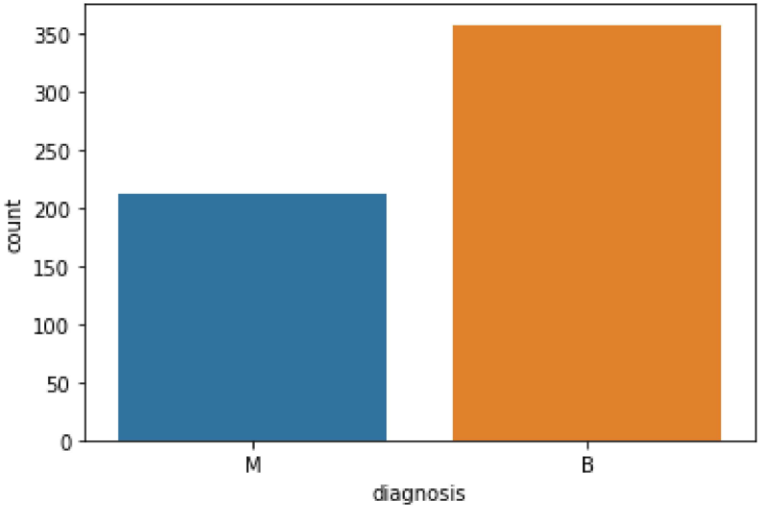
In [16]:
```
#Get a count of the number of Malignant (M) or Benign (B) cells
df['diagnosis'].value_counts()
```

Out[16]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [18]:

```
#Visualiza the count
sns.countplot(df['diagnosis'], label = 'count')
```

Out[18]: <AxesSubplot:xlabel='diagnosis', ylabel='count'>



In [19]:
```python
#Look at the data type to see which column need to be encoded
df.dtypes
#We found that the 'diagnosis' = string of B or M
```

Out[19]:
```
id                           int64
diagnosis                   object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
dtype: object
```

In order to analyze insight, the labels are encoded into numnerical parameters

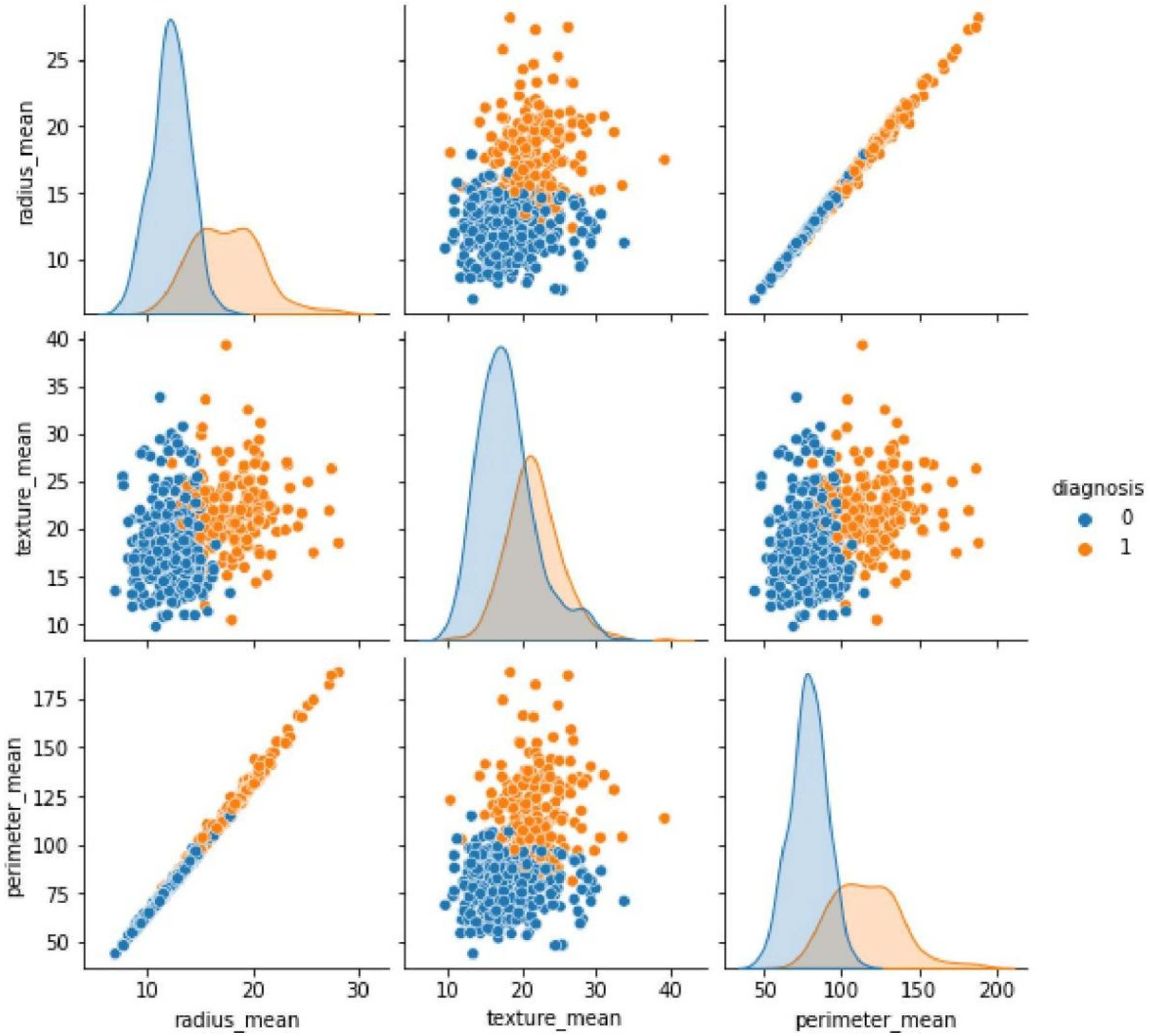
In [23]:
```
#Encode the categorical data values
from sklearn.preprocessing import LabelEncoder
labelencoder_Y = LabelEncoder()
df.iloc[:,1] = labelencoder_Y.fit_transform(df.iloc[:,1].values)
#df.iloc[:,1].values #for 'diagnosis' column transfromed from B,M to 0,1
```

Out[23]:
```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64
```

Pair plot are used to find the relationship between types of cancer and main features

In [25]:
```
#Create a pair plot
sns.pairplot(df.iloc[:,1:5], hue = 'diagnosis')
```

Out[25]: <seaborn.axisgrid.PairGrid at 0x151bbfe9ca0>



In [27]:

```
#Print the first 5 rows of the new data
df.head(5)
```

Out[27]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	poin
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	

5 rows × 32 columns

The correlation can be used to figure out the impact between each features

In [31]:
```python
#Get the correlation of the columns
df.iloc[:,1:12].corr()
#we can see how one column can influence the other
#redius_mean has a influence on the diagnosis
```

Out[31]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_
diagnosis	1.000000	0.730029	0.415185	0.742636	0.708984	0.358560	0.596534	0.69
radius_mean	0.730029	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.67
texture_mean	0.415185	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.30
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.7
area_mean	0.708984	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.68
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.52
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.88
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.92
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.50
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.33

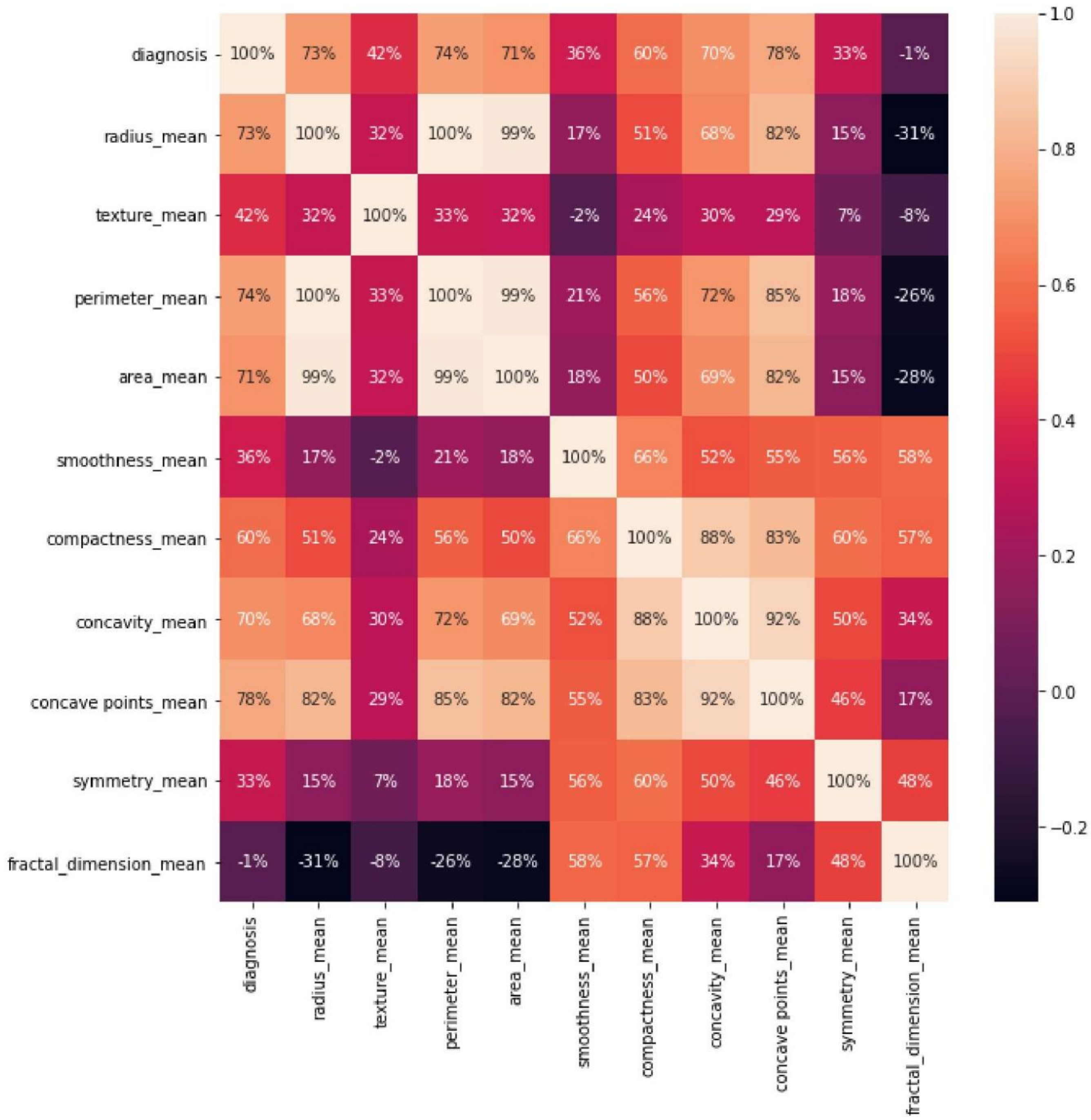
The correlation can be easily analyzed by heatmap plot.The higher percentage means the parameters have impact to each other.

In [35]:

```python
#Visualize the correlation
plt.figure(figsize=(10,10))
sns.heatmap(df.iloc[:,1:12].corr(), annot = True, fmt = '.0%')
```

Out[35]:

```
<AxesSubplot:>
```



diagnosis
radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

In order to create some prediction model, the dataset must be defined as features and labels. The dataset is also splitted into training and test set to train and validate the model, respectively.

In [38]:
```
#Split the data set into independent (X) and dependent (Y) data sets
X = df.iloc[:,2:31].values #conditions for cancer
Y = df.iloc[:,1].values #diagnosis
```

In [39]:
```
#Split the data set into 75% training and 25% testing
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 0)
```

The features are scaled to prevent the influence of some features.

In [42]:
```
#Scale the data (Feature Scaling)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

The model prediction in this work consists of three model including Logistic Regression, Decision Tree and Random Forest Classifier. The model accuracy are measured by the score function

In [45]:

```python
#Create a function for the models
def models(X_train, Y_train):

    #Logistic Regression
    from sklearn.linear_model import LogisticRegression
    log = LogisticRegression(random_state = 0)
    log.fit(X_train, Y_train)

    #Decision Tree
    from sklearn.tree import DecisionTreeClassifier
    tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
    tree.fit(X_train, Y_train)

    #Random Forest Classifier
    from sklearn.ensemble import RandomForestClassifier
    forest = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
    forest.fit(X_train, Y_train)

    #Print the models accuracy on the training data
    print('[0]Logistic Regression Traing Accuracy:', log.score(X_train, Y_train))
    print('[1]Decision Tree Classifier Traing Accuracy:', tree.score(X_train, Y_train))
    print('[2]Random Forest Classifier Traing Accuracy:', forest.score(X_train, Y_train))

    return log, tree, forest
```

In [46]:

```python
#Getting all of the models
model = models(X_train, Y_train)
```

```
[0]Logistic Regression Traing Accuracy: 0.9906103286384976
[1]Decision Tree Classifier Traing Accuracy: 1.0
[2]Random Forest Classifier Traing Accuracy: 0.9953051643192489
```

Cofusion Matrix which is described by predicted and actual information are provided to check the null and alternative hypothesis

In [52]:

```python
#Test model accuracy on test data on confusion matrix
from sklearn.metrics import confusion_matrix

for i in range(len(model)):
    print('Model ', i)
    cm = confusion_matrix(Y_test, model[i].predict(X_test))
    TP = cm[0][0]
    TN = cm[1][1]
    FN = cm[1][0]
    FP = cm[0][1]

    print(cm)
    print('Testing accuracy = ', (TP + TN)/(TP + TN + FN + FP))
    print()
```

```
Model  0
[[86  4]
 [ 3 50]]
Testing accuracy =  0.951048951048951

Model  1
[[83  7]
 [ 2 51]]
Testing accuracy =  0.9370629370629371

Model  2
[[87  3]
 [ 2 51]]
Testing accuracy =  0.965034965034965
```

The another way to calculate the precision, recall F1-score and support can be performed easier by classification_report from sklearn.metrics

In [56]:

```
#Show another way to get matrics of the models
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

for i in range(len(model)):
    print('Model ', i)
    print(classification_report(Y_test, model[i].predict(X_test)))
    print(accuracy_score(Y_test, model[i].predict(X_test)))
    print()
```

```
Model  0
              precision    recall  f1-score   support

           0       0.97      0.96      0.96        90
           1       0.93      0.94      0.93        53

    accuracy                           0.95       143
   macro avg       0.95      0.95      0.95       143
weighted avg       0.95      0.95      0.95       143

0.951048951048951

Model  1
              precision    recall  f1-score   support

           0       0.98      0.92      0.95        90
           1       0.88      0.96      0.92        53

    accuracy                           0.94       143
   macro avg       0.93      0.94      0.93       143
weighted avg       0.94      0.94      0.94       143

0.9370629370629371

Model  2
              precision    recall  f1-score   support

           0       0.98      0.97      0.97        90
           1       0.94      0.96      0.95        53
```

```
    accuracy                           0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.97      0.97      0.97       143

0.965034965034965
```

The example of prediction using Random Forest Classifier is the results from prediction.

In [57]:
```python
#Print the prediction of Random Forest Classifier model
pred = model[2].predict(X_test)
print(pred)
print()
print(Y_test)
```

```
[1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 0
 1 0 1 0 0 1 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0 0
 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0
 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1]

[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 0
 1 0 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0 1
 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0
 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1]
```

Suggestions for next steps

This dataset can be used to create the model for prediction the cancer in new patients. The accuracy for using in real situation is very important. More features or details of survey from real patients can be used to deveolop the model. Moreover, this kind of dataset can be used to perform the unsupervised model to group the types of patient.

In [ ]:
```python

```