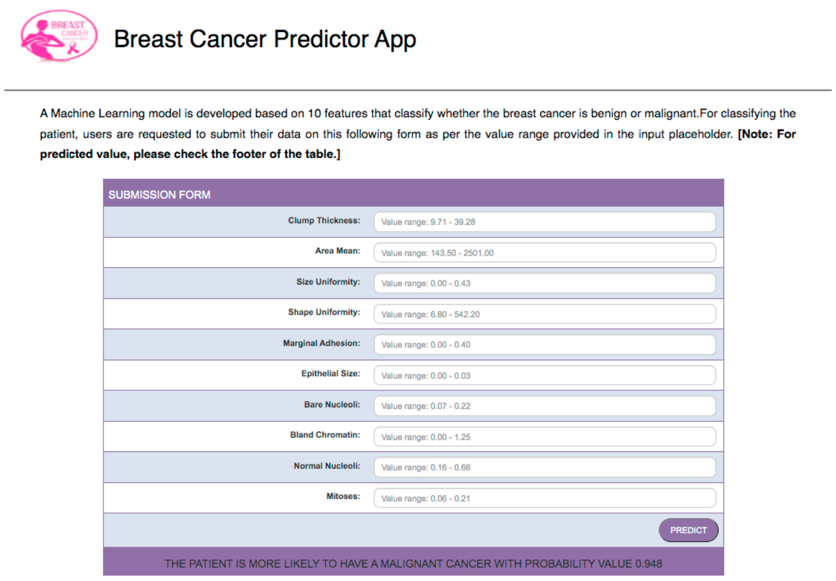

Breast Cancer Predictor App

A Machine Learning model is developed based on 10 features that classify whether the breast cancer is benign or malignant.For classifying the patient, users are requested to submit their data on this following form as per the value range provided in the input placeholder. **[Note: For predicted value, please check the footer of the table.]**

SUBMISSION FORM

Clump Thickness:	Value range: 9.71 - 39.28
Area Mean:	Value range: 143.50 - 2501.00
Size Uniformity:	Value range: 0.00 - 0.43
Shape Uniformity:	Value range: 6.80 - 542.20
Marginal Adhesion:	Value range: 0.00 - 0.40
Epithelial Size:	Value range: 0.00 - 0.03
Bare Nucleoli:	Value range: 0.07 - 0.22
Bland Chromatin:	Value range: 0.00 - 1.25
Normal Nucleoli:	Value range: 0.16 - 0.66
Mitoses:	Value range: 0.06 - 0.21

THE PATIENT IS MORE LIKELY TO HAVE A MALIGNANT CANCER WITH PROBABILITY VALUE 0.948

Breast cancer is one of the most common cancers among women worldwide, representing the majority of new cancer cases and cancer-related deaths according to global statistics, making it a significant public health problem in today's society. The early diagnosis of Breast cancer can improve the prognosis and chance of survival significantly, as it can promote timely clinical treatment to patients. Further accurate classification of benign tumors can prevent patients undergoing unnecessary treatments.

Benign: Not likely to get cancer (2)

Malignant: Likely to get cancer (4)

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

In [ ]:
```
data=pd.read_csv("/content/breastCancer.csv")
data.head()
```

Out[ ]:

	id	clump_thickness	size_uniformity	shape_uniformity	marginal_adh
0	1000025	5	1	1	
1	1002945	5	4	4	
2	1015425	3	1	1	
3	1016277	6	8	8	
4	1017023	4	1	1	

```
In [ ]: data['class'].unique()
```

```
Out[ ]: array([2, 4])
```

```
In [ ]: data['class'].value_counts()
```

```
Out[ ]: 2    458
        4    241
        Name: class, dtype: int64
```

We can change the 2 and 4 ---> 0 and 1 respectively

```
In [ ]: data['class'] = data['class'].replace(2,0)
```

```
In [ ]: data['class'] = data['class'].replace(4,1)
```

```
In [ ]: data['class'].unique()
```

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

```
In [ ]: data['clump_thickness'].min()
```

```
Out[ ]: 1
```

```
In [ ]: data['clump_thickness'].max()
```

```
Out[ ]: 10
```

Attribute Information:

1. Sample code number: id number
2. Clump Thickness: 1 - 10
3. Uniformity of Cell Size: 1 - 10
4. Uniformity of Cell Shape: 1 - 10
5. Marginal Adhesion: 1 - 10
6. Single Epithelial Cell Size: 1 - 10
7. Bare Nuclei: 1 - 10
8. Bland Chromatin: 1 - 10
9. Normal Nucleoli: 1 - 10
10. Mitoses: 1 - 10
11. Class: (2 for benign, 4 for malignant)

```
In [ ]:
```

Exploratory Data Analysis

```
In [ ]: data['class'].value_counts()
```

```
Out[ ]: 2    458
        4    241
        Name: class, dtype: int64
```

```
In [ ]: data.dtypes #checking the data types of each column
```

```
Out[ ]: id                      int64
        clump_thickness         int64
        size_uniformity         int64
        shape_uniformity        int64
        marginal_adhesion       int64
        epithelial_size         int64
        bare_nucleoli          object
        bland_chromatin         int64
        normal_nucleoli         int64
        mitoses                 int64
        class                   int64
        dtype: object
```

```
In [ ]: data['bare_nucleoli'] #let's inspect the 'bare_nucleoli' column
```

```
Out[ ]: 0       1
        1      10
        2       2
        3       4
        4       1
               ..
        694     2
        695     1
        696     3
        697     4
        698     5
        Name: bare_nucleoli, Length: 699, dtype: object
```

```
In [ ]: data['bare_nucleoli'].isna().sum()
```

```
Out[ ]: 0
```

```
In [ ]: data[data['bare_nucleoli']=='?'] #checking the presence of '?' in the 'bare_
```

Out[ ]:

	id	clump_thickness	size_uniformity	shape_uniformity	marginal_a
23	1057013	8	4	5	
40	1096800	6	6	6	
139	1183246	1	1	1	
145	1184840	1	1	3	
158	1193683	1	1	2	
164	1197510	5	1	1	
235	1241232	3	1	4	
249	169356	3	1	1	
275	432809	3	1	3	
292	563649	8	8	8	
294	606140	1	1	1	
297	61634	5	4	3	
315	704168	4	6	5	
321	733639	3	1	1	
411	1238464	1	1	1	
617	1057067	1	1	1	

In [ ]:

```
```

In [ ]:

```
data[data['bare_nucleoli']=='?'].sum() # alternatively
```

Out[ ]:

```
id                           13721250
clump_thickness                    54
size_uniformity                    39
shape_uniformity                   46
marginal_adhesion                  29
epithelial_size                    39
bare_nucleoli        ????????????????
bland_chromatin                    50
normal_nucleoli                    44
mitoses                            16
class                              36
dtype: object
```

Alternatively

Using the isdigit() function

In [ ]:

```
digits_in_bare_nucleoli= pd.DataFrame(data.bare_nucleoli.str.isdigit())
digits_in_bare_nucleoli
```

Out[ ]:

	bare_nucleoli
0	True
1	True
2	True
3	True
4	True
...	...
694	True
695	True
696	True
697	True
698	True

699 rows × 1 columns

In [ ]:
```
#df[digits_in_hp['horsepower'] == False]
data[digits_in_bare_nucleoli['bare_nucleoli']== False]
```

Out[ ]:

	id	clump_thickness	size_uniformity	shape_uniformity	marginal_a
23	1057013	8	4	5	
40	1096800	6	6	6	
139	1183246	1	1	1	
145	1184840	1	1	3	
158	1193683	1	1	2	
164	1197510	5	1	1	
235	1241232	3	1	4	
249	169356	3	1	1	
275	432809	3	1	3	
292	563649	8	8	8	
294	606140	1	1	1	
297	61634	5	4	3	
315	704168	4	6	5	
321	733639	3	1	1	
411	1238464	1	1	1	
617	1057067	1	1	1	

Let us replace these missing values with NaN

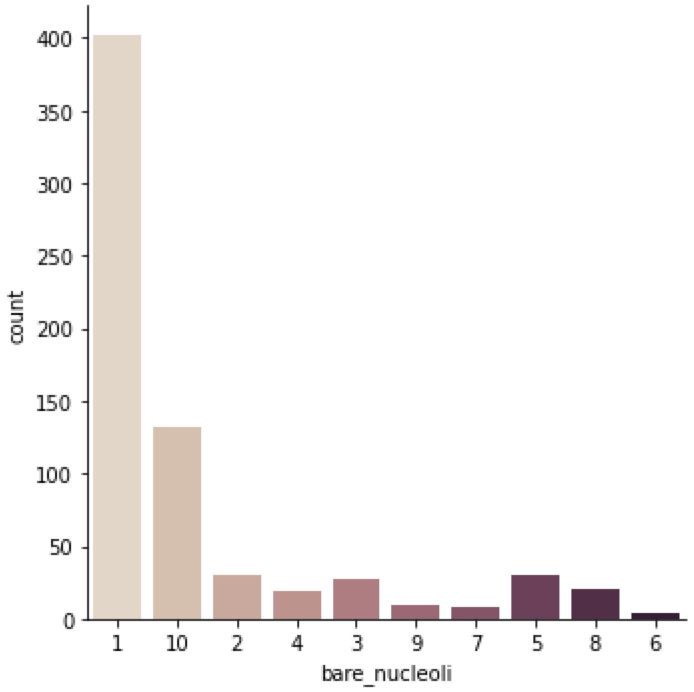
```
In [ ]: df= data.replace('?', np.nan)
```

```
In [ ]: df.isna().sum()
```

```
Out[ ]: id                      0
        clump_thickness         0
        size_uniformity         0
        shape_uniformity        0
        marginal_adhesion       0
        epithelial_size         0
        bare_nucleoli          16
        bland_chromatin         0
        normal_nucleoli         0
        mitoses                 0
        class                   0
        dtype: int64
```

```
In [ ]: sns.catplot(x="bare_nucleoli", kind="count", palette="ch:.25", data=df)
```

```
Out[ ]: <seaborn.axisgrid.FacetGrid at 0x7f194a239950>
```



we will not drop the **NaN**s in our dataset, we will rather replace them with the **Median** of the column. We can also use the **Mode**(i.e. the most frequent number in the column). The mean will not be a good idea in this case since the data is not normally distributed.

```
In [ ]: df.median()
```

```
Out[ ]: id                     1171710.0
        clump_thickness              4.0
        size_uniformity              1.0
        shape_uniformity             1.0
        marginal_adhesion            1.0
        epithelial_size              2.0
        bare_nucleoli                1.0
        bland_chromatin              3.0
        normal_nucleoli              1.0
        mitoses                      1.0
        class                        2.0
        dtype: float64
```

```
In [ ]: df = df.fillna(df.median())
```

```
In [ ]: df.dtypes
```

```
Out[ ]: id                      int64
        clump_thickness         int64
        size_uniformity         int64
        shape_uniformity        int64
        marginal_adhesion       int64
        epithelial_size         int64
        bare_nucleoli          object
        bland_chromatin         int64
        normal_nucleoli         int64
        mitoses                 int64
        class                   int64
        dtype: object
```

we notice that the **bare_nucleoli** feature is showing **object** although it is supposed to be integer. We will then manually convert it to integer.

```
In [ ]: df['bare_nucleoli'] = df['bare_nucleoli'].astype('int64')
```

```
In [ ]: df.dtypes
```

```
Out[ ]: id                     int64
        clump_thickness        int64
        size_uniformity        int64
        shape_uniformity       int64
        marginal_adhesion      int64
        epithelial_size        int64
        bare_nucleoli          int64
        bland_chromatin        int64
        normal_nucleoli        int64
        mitoses                int64
        class                  int64
        dtype: object
```

```
In [ ]: #dropping the index of the dataset

        df.drop('id', axis=1, inplace=True)
```

In [ ]:
```
df.head()
```

Out[ ]:

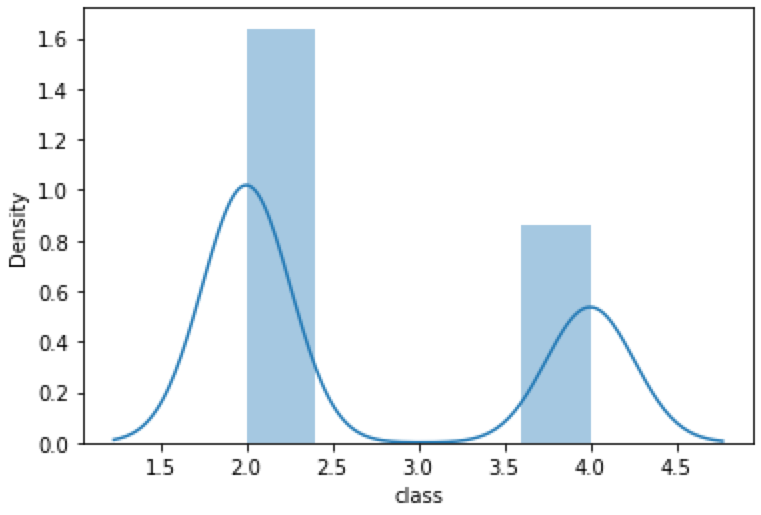
	clump_thickness	size_uniformity	shape_uniformity	marginal_adhesion	epi
0	5	1	1	1	
1	5	4	4	5	
2	3	1	1	1	
3	6	8	8	1	
4	4	1	1	3	

Univariate Data Analysis of the **Class** column

In [ ]:
```
sns.distplot(df['class'])
```

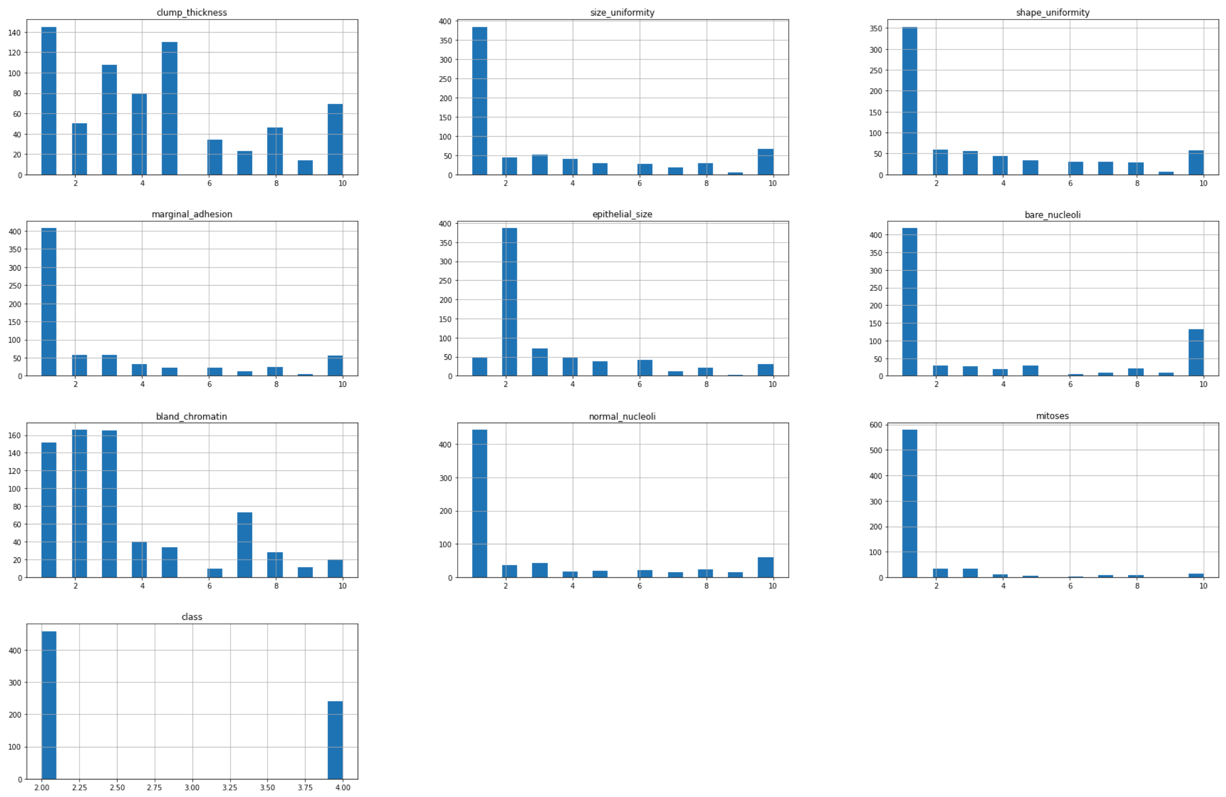
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level func
tion with similar flexibility) or `histplot` (an axes-level function for his
tograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1947404450>
```



Multivariate Data Analysis

In [ ]:
```
df.hist(bins=20, figsize=(30,30), layout=(6,3));
```



Most of the features are right skewed

Let's See the Correlation among these attributes

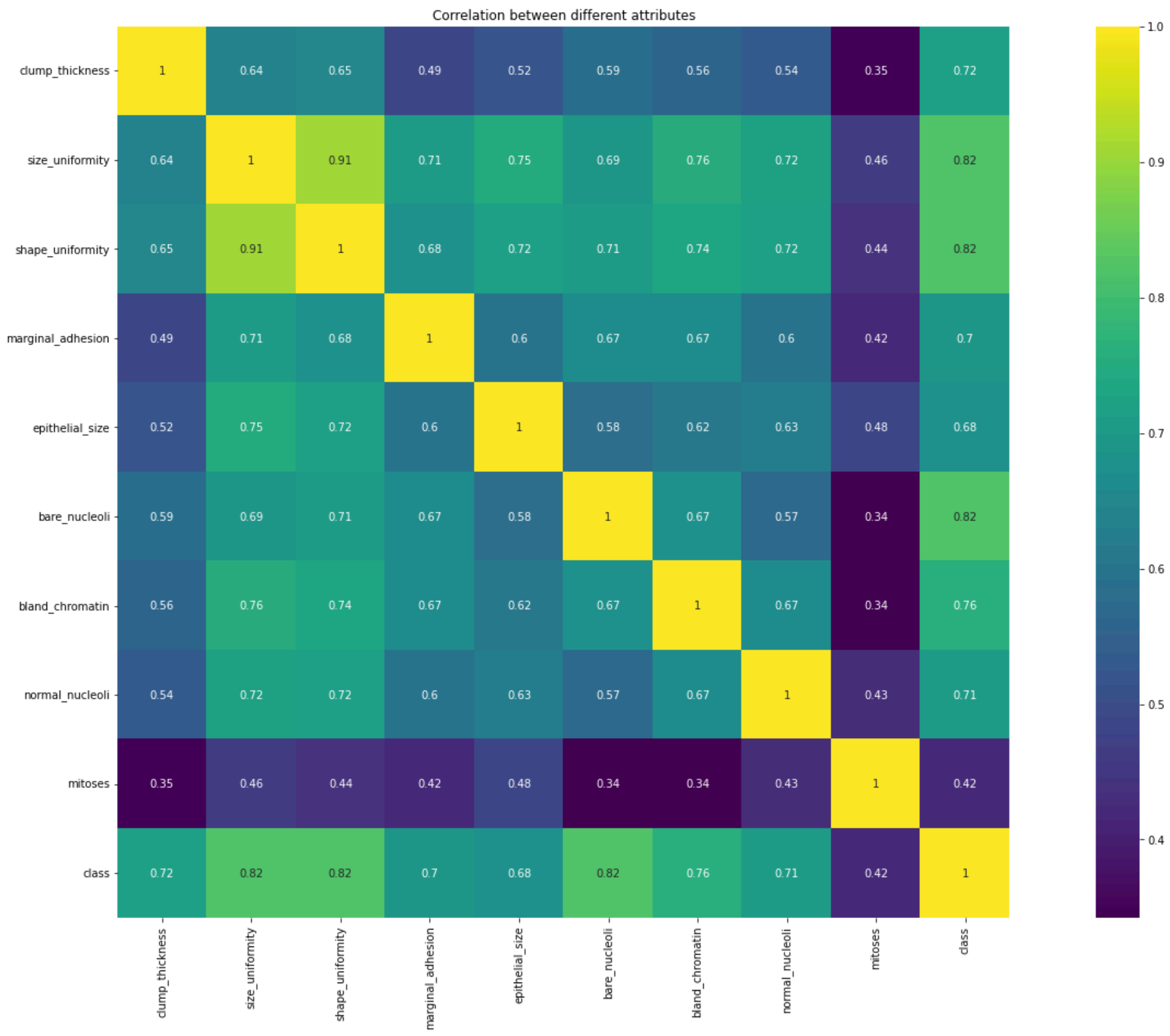
```
In [ ]: df.corr()
```

Out[ ]:

	clump_thickness	size_uniformity	shape_uniformity	margi
clump_thickness	1.000000	0.644913	0.654589	
size_uniformity	0.644913	1.000000	0.906882	
shape_uniformity	0.654589	0.906882	1.000000	
marginal_adhesion	0.486356	0.705582	0.683079	
epithelial_size	0.521816	0.751799	0.719668	
bare_nucleoli	0.590008	0.686673	0.707474	
bland_chromatin	0.558428	0.755721	0.735948	
normal_nucleoli	0.535835	0.722865	0.719446	
mitoses	0.350034	0.458693	0.438911	
class	0.716001	0.817904	0.818934	

```
In [ ]: #Heatmap of the correlation between the indepent attributes

        plt.figure(figsize=(28,15))
        sns.heatmap(df.corr(), vmax=1, square=True,annot=True,cmap='viridis')
```

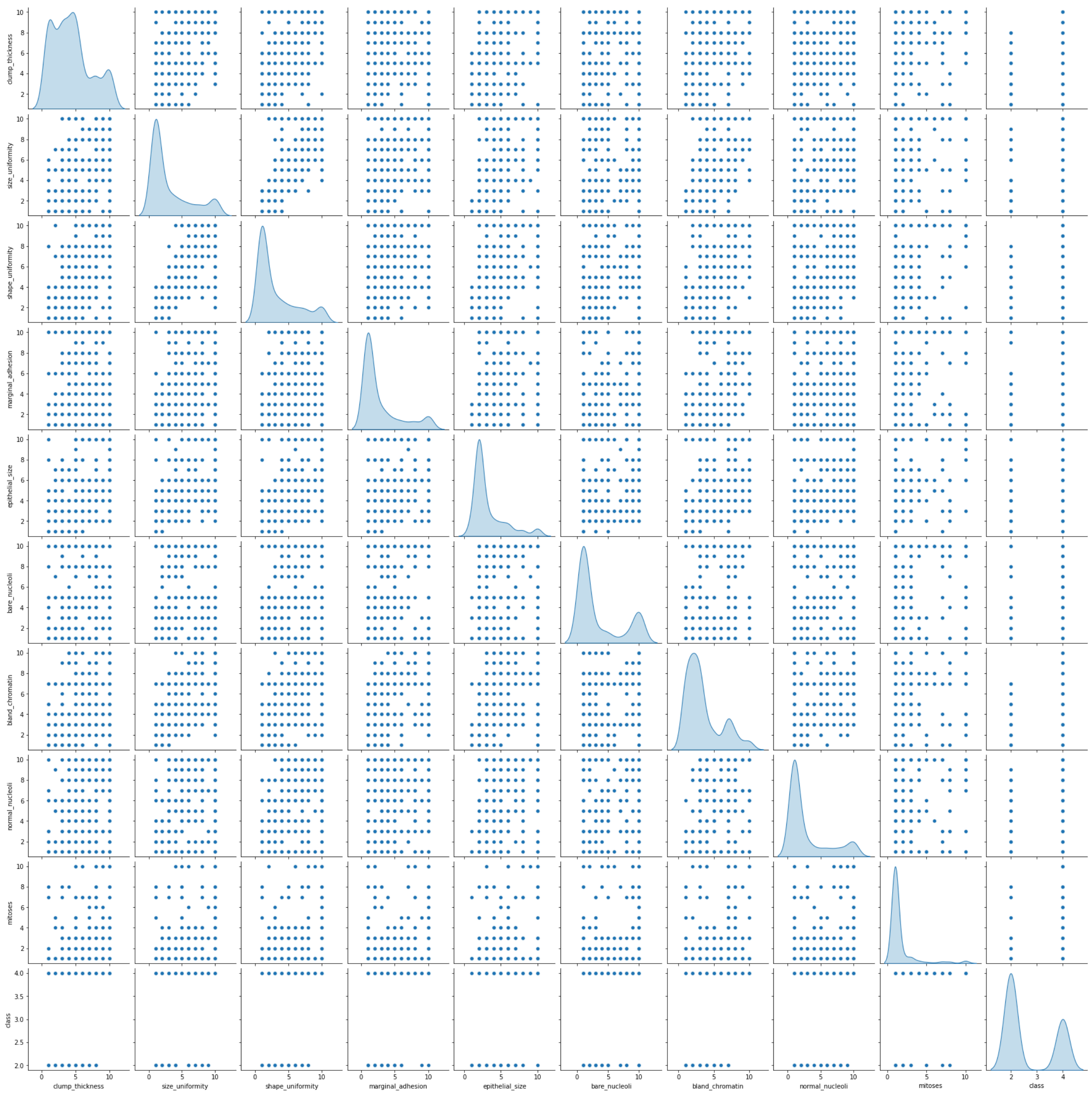
```
plt.title('Correlation between different attributes')
plt.show()
```



	clump_thickness	size_uniformity	shape_uniformity	marginal_adhesion	epithelial_size	bare_nucleoli	bland_chromatin	normal_nucleoli	mitoses	class
clump_thickness	1	0.64	0.65	0.49	0.52	0.59	0.56	0.54	0.35	0.72
size_uniformity	0.64	1	0.91	0.71	0.75	0.69	0.76	0.72	0.46	0.82
shape_uniformity	0.65	0.91	1	0.68	0.72	0.71	0.74	0.72	0.44	0.82
marginal_adhesion	0.49	0.71	0.68	1	0.6	0.67	0.67	0.6	0.42	0.7
epithelial_size	0.52	0.75	0.72	0.6	1	0.58	0.62	0.63	0.48	0.68
bare_nucleoli	0.59	0.69	0.71	0.67	0.58	1	0.67	0.57	0.34	0.82
bland_chromatin	0.56	0.76	0.74	0.67	0.62	0.67	1	0.67	0.34	0.76
normal_nucleoli	0.54	0.72	0.72	0.6	0.63	0.57	0.67	1	0.43	0.71
mitoses	0.35	0.46	0.44	0.42	0.48	0.34	0.34	0.43	1	0.42
class	0.72	0.82	0.82	0.7	0.68	0.82	0.76	0.71	0.42	1

```
In [ ]: #Pairplot of the correlation/distribution between various independent attrib
        sns.pairplot(df, diag_kind="kde")
```

```
Out[ ]: <seaborn.axisgrid.PairGrid at 0x7f19452e8d10>
```



Building Our Model

In [ ]:
```
# Dividing our dataset into training and testing set

X = df.drop('class', axis=1)  #selecting all the attributes except the class
y = df['class'] #selecting class attribute.
```

In [ ]:
```
#Splitting our data into 70:30
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, ra
```

KNeighborsClassifier

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

        KNN = KNeighborsClassifier(n_neighbors= 5 , weights = 'distance' )
```

```
In [ ]: # Call Nearest Neighbour algorithm

        KNN.fit(X_train, y_train)

Out[ ]: KNeighborsClassifier(weights='distance')
```

```
In [ ]: X_test
```

Out[ ]:

	clump_thickness	size_uniformity	shape_uniformity	marginal_adhesion
584	5	1	1	6
417	1	1	1	1
606	4	1	1	2
349	4	2	3	5
134	3	1	1	1
...	...	...	...	...
440	10	4	3	10
299	9	1	2	6
577	1	1	1	1
103	8	2	3	1
659	1	1	1	1

210 rows × 9 columns

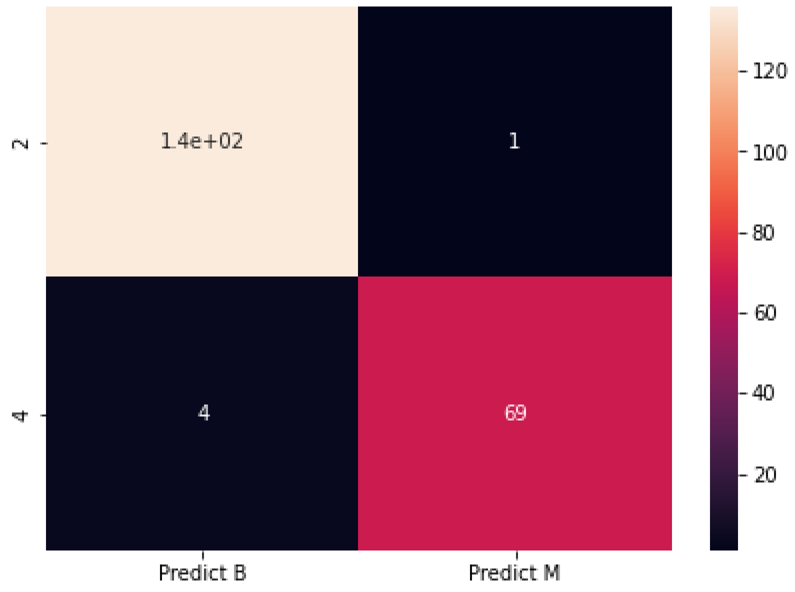
```
In [ ]: predicted_1 = KNN.predict(X_test)
        predicted_1

Out[ ]: array([2, 2, 2, 4, 2, 2, 4, 2, 2, 2, 4, 4, 2, 2, 4, 4, 2, 2, 2, 2, 2, 4,
               4, 2, 4, 2, 4, 4, 2, 2, 2, 4, 4, 4, 4, 4, 2, 4, 2, 2, 2, 2, 2, 2,
               2, 4, 2, 2, 2, 2, 2, 2, 4, 2, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2,
               2, 4, 2, 2, 2, 2, 2, 2, 4, 4, 2, 2, 2, 2, 4, 4, 2, 4, 2, 2, 2, 4,
               4, 2, 4, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 4, 2, 4, 2,
               4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 2, 4, 2, 4, 2, 2, 2, 4,
               4, 2, 4, 2, 2, 2, 4, 4, 2, 4, 2, 2, 2, 4, 2, 4, 2, 2, 2, 2, 2, 4,
               2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4,
               4, 4, 4, 4, 4, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 4, 2, 4, 4, 4,
               4, 2, 2, 2, 2, 2, 4, 4, 4, 2, 2, 2])
```

```
In [ ]: from scipy.stats import zscore

        print('KNeighborsClassifier Agorithm is predicting at {0:.2g}%'.format(KNN.s

KNeighborsClassifier Agorithm is predicting at 98%
```

```
In [ ]: from sklearn import metrics

        print("Confusion Matrix For KNeighborsClassifier")
        cm=metrics.confusion_matrix(y_test, predicted_1, labels=[2, 4])

        df_cm = pd.DataFrame(cm, index = [i for i in [2,4]],
                          columns = [i for i in ["Predict B","Predict M"]])
        plt.figure(figsize = (7,5))
        sns.heatmap(df_cm, annot=True)
```

```
Confusion Matrix For KNeighborsClassifier
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1940bb2210>
```



Support Vector Machine

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


        svc= SVC(gamma=0.025, C=3, kernel='linear')
        svc.fit(X_train, y_train)
```

```
Out[ ]: SVC(C=3, gamma=0.025, kernel='linear')
```

```
In [ ]: predicted_2 = svc.predict(X_test)
        predicted_2
```

```
Out[ ]:  array([2, 2, 2, 4, 2, 2, 4, 2, 2, 2, 4, 4, 2, 4, 4, 4, 2, 2, 2, 2, 2, 4,
                4, 2, 4, 2, 4, 4, 2, 2, 2, 4, 4, 4, 4, 4, 2, 4, 2, 2, 2, 2, 2, 2,
                2, 4, 2, 2, 2, 2, 2, 2, 4, 2, 4, 2, 4, 2, 2, 4, 2, 2, 2, 2, 2, 2,
                2, 4, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 4, 4, 2, 4, 2, 2, 2, 4,
                4, 2, 4, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 4, 2, 4, 2,
                4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 2, 2, 2, 4, 2, 2, 2, 4,
                4, 2, 4, 2, 2, 2, 4, 4, 2, 4, 2, 2, 2, 4, 2, 4, 2, 2, 2, 2, 2, 4,
                2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4,
                4, 4, 4, 4, 4, 2, 2, 2, 2, 4, 2, 2, 4, 2, 2, 2, 2, 4, 2, 4, 4, 4,
                4, 2, 2, 2, 2, 2, 4, 4, 4, 2, 4, 2])
```

```
In [ ]: print('SupportVectorClassifier Agorithm is predicting at {0:.2g}%'.format(sv
```

```
SupportVectorClassifier Agorithm is predicting at 97%
```

```
In [ ]: knnPredictions=pd.DataFrame(predicted_1)
        svcPredictions=pd.DataFrame(predicted_2)
```

```
In [ ]: #@title
        df1=pd.concat([knnPredictions,svcPredictions],axis=1)
```

```
In [ ]: df1.columns=[['knnPredictions','svcPredictions']]
```

```
In [ ]: df1
```

Out[ ]:

	knnPredictions	svcPredictions
0	2	2
1	2	2
2	2	2
3	4	4
4	2	2
...	...	...
205	4	4
206	4	4
207	2	2
208	2	4
209	2	2

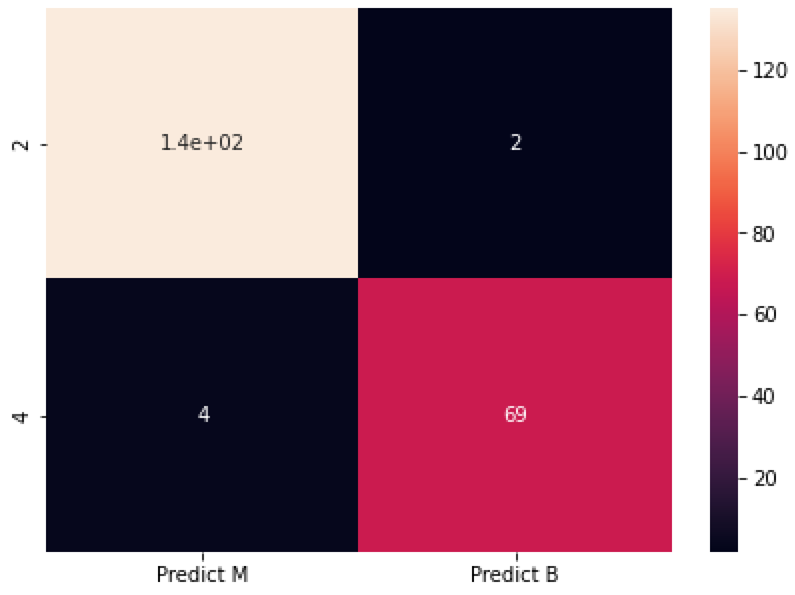
210 rows × 2 columns

```
In [ ]: print("Confusion Matrix For SupportVectorMachine")
        cm=metrics.confusion_matrix(y_test, predicted_2, labels=[2, 4])

        df_cm = pd.DataFrame(cm, index = [i for i in [2,4]],
                          columns = [i for i in ["Predict M","Predict B"]])
```

```
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True)
```

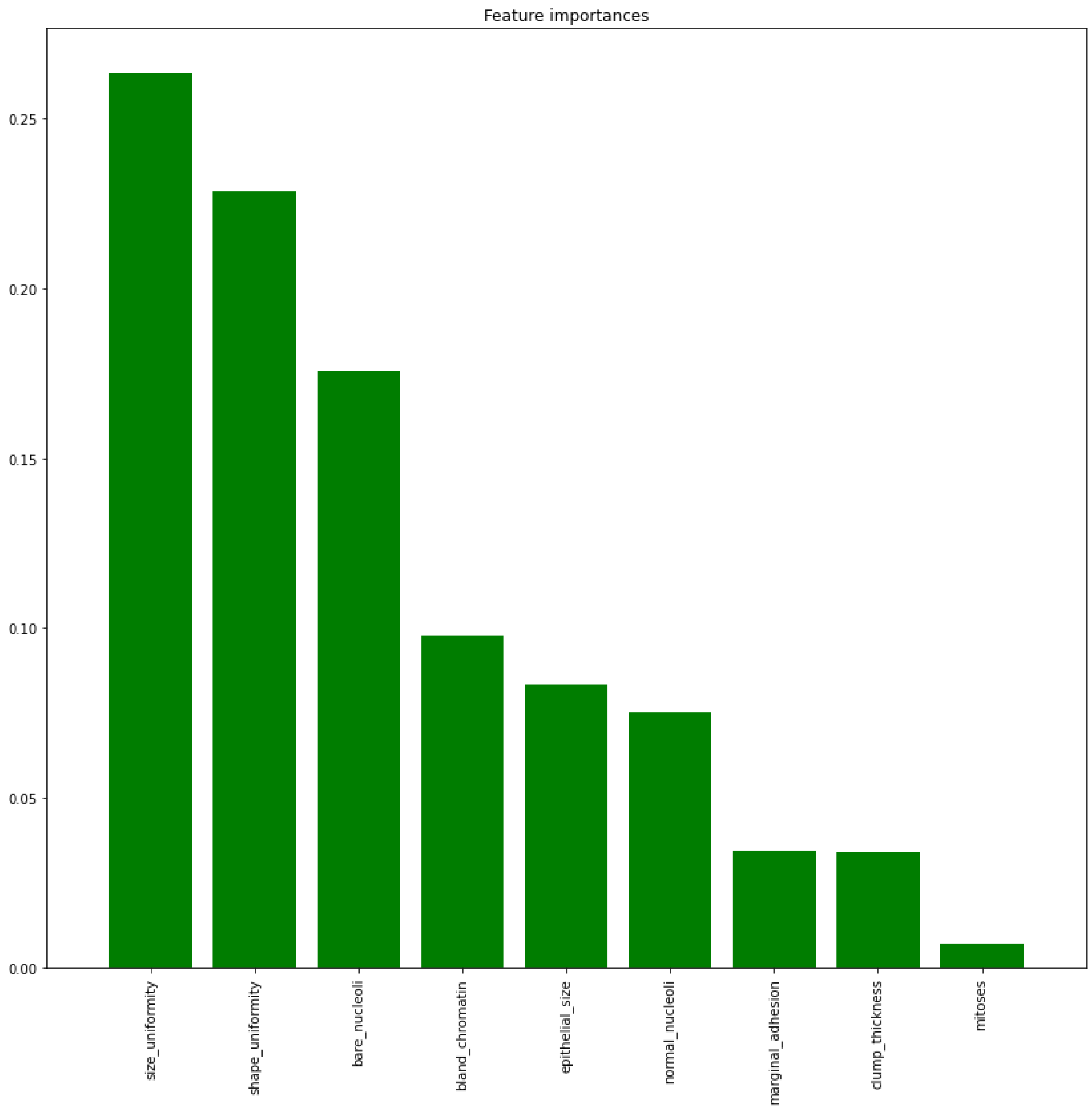
```
Confusion Matrix For SupportVectorMachine
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f19430e0310>
```



Feature Importance

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf = rf.fit(X_train, y_train)
importances = rf.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
# print("Feature ranking:")

# for f in range(X_train.shape[1]):
#     print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[

# Plot the feature importances of the forest

plt.figure(1, figsize=(14, 13))
plt.title("Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices],
       color="g",  align="center")
plt.xticks(range(X_train.shape[1]), X_train.columns[indices],rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```



```
In [ ]: df.head()
```

Out[ ]:

	clump_thickness	size_uniformity	shape_uniformity	marginal_adhesion	epi
0	5	1	1	1	
1	5	4	4	5	
2	3	1	1	1	
3	6	8	8	1	
4	4	1	1	3	

```
In [ ]: df.shape

Out[ ]: (699, 10)
```

```
In [ ]: from sklearn.preprocessing import StandardScaler

        scaler = StandardScaler()

        x_train = scaler.fit_transform(X_train)
        x_test = scaler.transform(X_test)

        import pickle

        # Creating a pickle file for the classifier
        pickle.dump(scaler, open('scaler.pkl', 'wb'))
```

```
In [ ]: # SelectedFeatures = df[['size_uniformity','bare_nucleoli','shape_uniformity
```

```
In [ ]: # x_train, x_test, y_train, y_test = train_test_split(SelectedFeatures, y, t

        # rf = RandomForestClassifier()
        # rf.fit(X_train, y_train)
```

```
In [ ]: import pickle

        # Creating a pickle file for the classifier
        pickle.dump(svc, open('model.pkl', 'wb'))
```

```
In [ ]:
```