About the Notebook

In this notebook, I tried to use different ways of machine learning in diagnosing breast cancer. First, let's find out what breast cancer is, then let's start examining the data. I hope you enjoy reading it.

Breast cancer is a disease in which cells in the breast grow out of control. There are different kinds of breast cancer. The kind of breast cancer depends on which cells in the breast turn into cancer.

Breast Cancer Diagnostic Using Different ML Models

Notebook Imports

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # data visualization library
import matplotlib.pyplot as plt

#Machine Learning Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

#import warnings library
import warnings
# ignore all warnings
warnings.filterwarnings('ignore')
```

Importing Data

In [2]:
```
data = pd.read_csv(r"C:\Users\wwwvs\Desktop\Vadhina Project\data.csv")
```

Checking for Missing Values

In [3]:
```
data.isna().sum()
```

Out[3]:
```
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
```

There is missing and unnecessary values in our dataset.

In [4]:
```
data.drop(["id","Unnamed: 32"],axis=1,inplace=True)
data.isna().sum()
```

Out[4]:
```
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
dtype: int64
```

Now we can start to EDA.

Exploratory Data Analysis

```
Data Content

ID number
Diagnosis (M = malignant, B = benign)
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 largest values) of these features were computed for each
image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.
All feature values are recoded with four significant digits.
Missing attribute values: none
Class distribution: 357 benign, 212 malignant
```

Data Content

- ID number
- Diagnosis (M = malignant, B = benign)
- 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 largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.
- All feature values are recoded with four significant digits.
- Missing attribute values: none
- Class distribution: 357 benign, 212 malignant

I got this content from DATAI Team's Feature Selection and Data Visualization notebook.

In [7]:
```
data.head()
```

Out[7]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

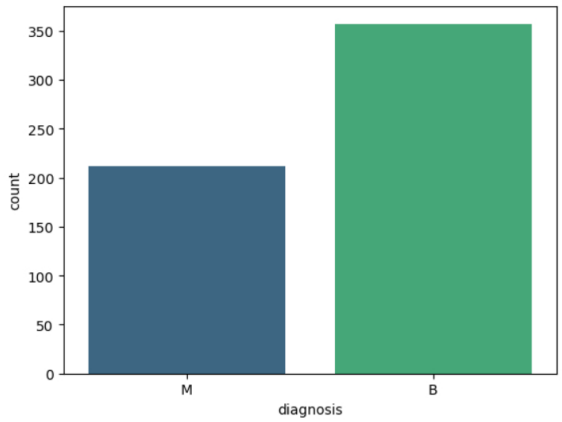
5 rows × 31 columns

In [8]:
```
col = data.columns       # .columns gives columns names in data
print(col)
```

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

In [9]:
```
n = data.diagnosis
B, M = n.value_counts()
ax = sns.countplot(n,label="Count",palette="viridis")
print('Number of Benign: ',B)
print('Number of Malignant : ',M)
```

```
Number of Benign:  357
Number of Malignant :  212
```



In [10]:
```
m = data.drop("diagnosis",axis=1)
m.describe()
```

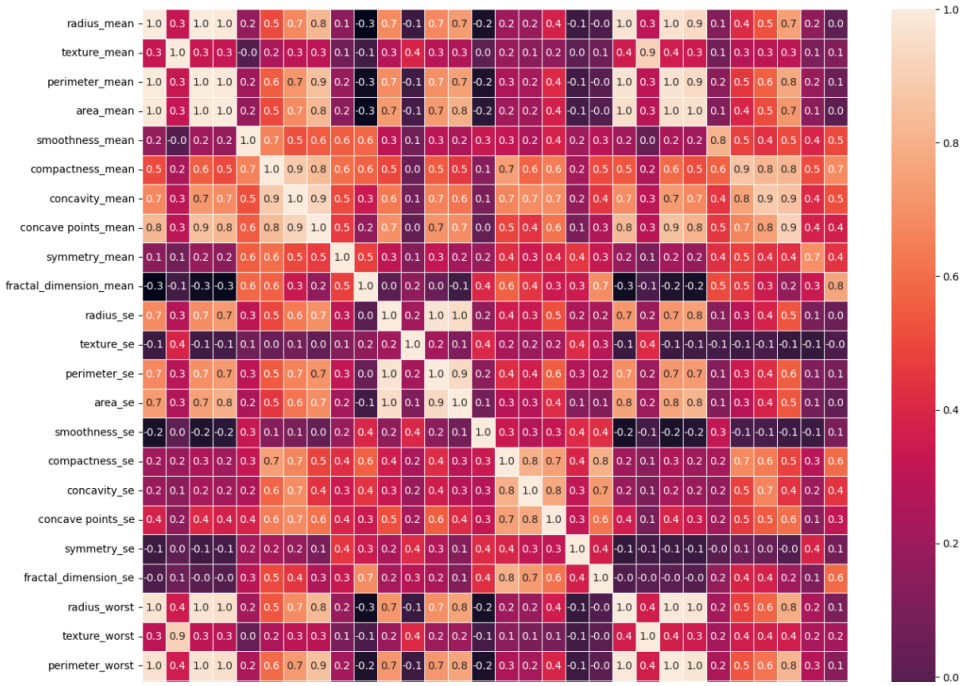
Out[10]:

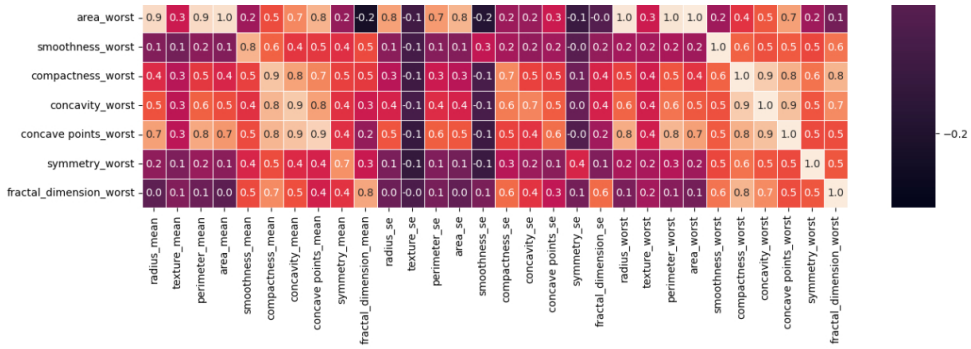
	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fracta
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	

8 rows × 30 columns

In [11]:
```
f,ax = plt.subplots(figsize=(15, 15))
sns.heatmap(m.corr(), annot=True,linewidths=.5, fmt= '.1f',ax=ax)
```

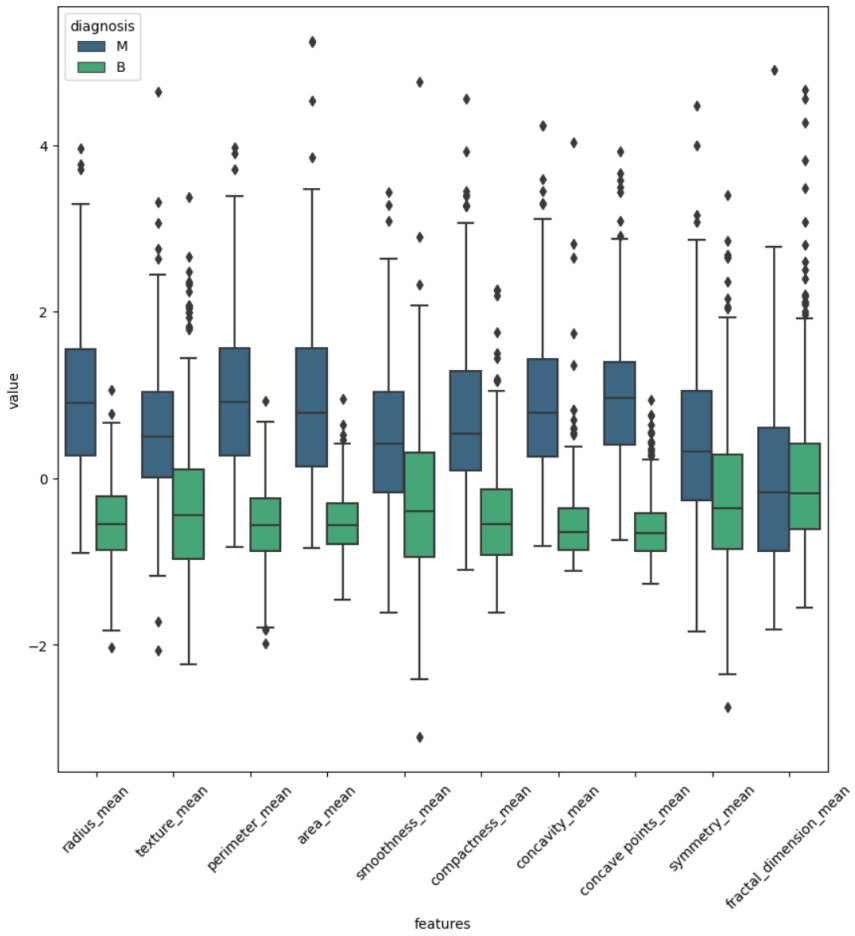
Out[11]: <AxesSubplot:>





In [12]:
```
data_dia = n
data_m = m
data_n_2 = (data_m - data_m.mean()) / (data_m.std())              # standardization
data_p = pd.concat([n,data_n_2.iloc[:,0:10]],axis=1)
data_p = pd.melt(data_p,id_vars="diagnosis",
                    var_name="features",
                    value_name='value')
plt.figure(figsize=(10,10))
sns.boxplot(x="features", y="value", hue="diagnosis", data=data_p,palette="viridis")
plt.xticks(rotation=45)
```

Out[12]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, 'radius_mean'),
  Text(1, 0, 'texture_mean'),
  Text(2, 0, 'perimeter_mean'),
  Text(3, 0, 'area_mean'),
  Text(4, 0, 'smoothness_mean'),
  Text(5, 0, 'compactness_mean'),
  Text(6, 0, 'concavity_mean'),
  Text(7, 0, 'concave points_mean'),
  Text(8, 0, 'symmetry_mean'),
  Text(9, 0, 'fractal_dimension_mean')])
```



Prepearing Data for ML

We need to convert our diagnostic values to numerical values in order to be able to process.

In [13]:
```
data.diagnosis = [1 if each == "M" else 0 for each in data.diagnosis]
data.head()
```

Out[13]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069

3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 31 columns

Now we need to normalize all values in their own way.

In [14]:
```
y = data.diagnosis.values
x_data = data.drop(["diagnosis"],axis=1)
x = (x_data - np.min(x_data))/(np.max(x_data)-np.min(x_data))
x.head()
```

Out[14]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dim
0	0.521037	0.022658	0.545989	0.363733	0.593753	0.792037	0.703140	0.731113	0.686364	
1	0.643144	0.272574	0.615783	0.501591	0.289880	0.181768	0.203608	0.348757	0.379798	
2	0.601496	0.390260	0.595743	0.449417	0.514309	0.431017	0.462512	0.635686	0.509596	
3	0.210090	0.360839	0.233501	0.102906	0.811321	0.811361	0.565604	0.522863	0.776263	
4	0.629893	0.156578	0.630986	0.489290	0.430351	0.347893	0.463918	0.518390	0.378283	

5 rows × 30 columns

Now we can create train splits and test splits.

In [15]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.3,random_state=42)
```

We made our preparations. Now let's move on to the ML part.

Logistic Regression Classifier

In [16]:
```
lr = LogisticRegression(random_state = 1) #We are building our model
lr.fit(x_train,y_train) #We are training our model
print("Print accuracy of Logistic Regression Classifier: {}".format(lr.score(x_test,y_test)))
lr_acc_score = lr.score(x_test,y_test)
```

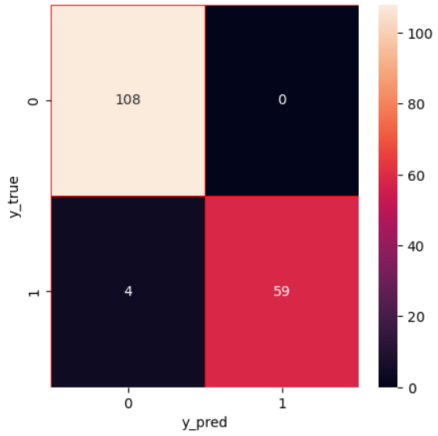
```
Print accuracy of Logistic Regression Classifier: 0.9766081871345029
```

Our model has an accuracy rate of 0.97. We can use confusion matrix to see which parts it got wrong in predicting.

In [17]:
```
y_pred = lr.predict(x_test)
y_true = y_test

cm = confusion_matrix(y_true, y_pred)

#visualize
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot = True, linewidths=0.5,linecolor="red",fmt = ".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```



K Neighbors Classifier

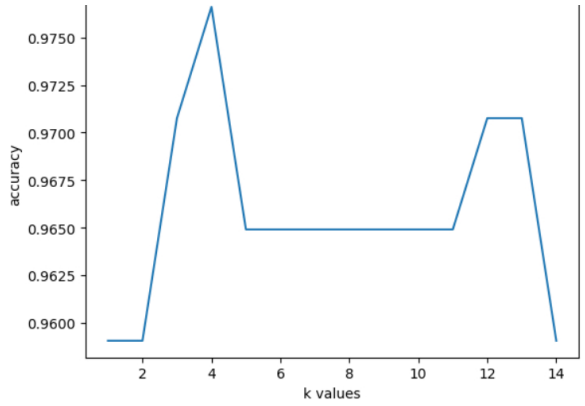
In [18]:
```
knn = KNeighborsClassifier(n_neighbors=10) #We are building our model
knn.fit(x_train,y_train) #We are training our model
print("Print accuracy of K Neighbors Classifier algo: {}".format(knn.score(x_test,y_test)))
knn_acc_score = knn.score(x_test,y_test)
```

```
Print accuracy of K Neighbors Classifier algo: 0.9649122807017544
```

We can create a for loop to find the best k value.

In [19]:
```
score_list = []
for each in range(1,15):
    knn2 = KNeighborsClassifier(n_neighbors = each)
    knn2.fit(x_train,y_train)
    score_list.append(knn2.score(x_test,y_test))

#visualize
plt.plot(range(1,15),score_list)
plt.xlabel("k values")
plt.ylabel("accuracy")
plt.savefig('plot')
plt.show()
```



Best k value is 4. Now we will try again.

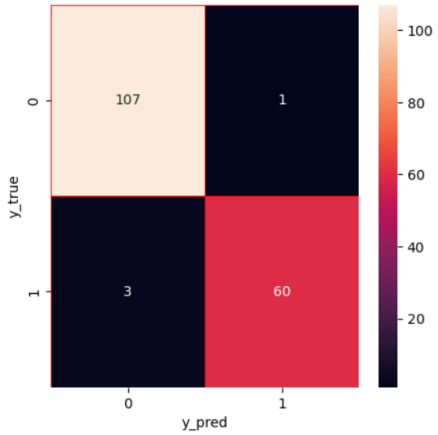
In [20]:
```
knn = KNeighborsClassifier(n_neighbors=4) #We build our model
knn.fit(x_train,y_train) #We train our model
print("Test accuracy {}".format(knn.score(x_test,y_test)))
```

```
Test accuracy 0.9766081871345029
```

In [21]:
```
y_pred = knn.predict(x_test)
y_true = y_test

cm = confusion_matrix(y_true, y_pred)

#visualize
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot = True, linewidths=0.5,linecolor="red",fmt = ".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```



Naive Bayes

In [22]:
```
nb = GaussianNB()       #We are building our model
nb.fit(x_train,y_train) #We are training our model
print("Print accuracy of naive bayes algo: {}".format(nb.score(x_test,y_test)))
nb_acc_score = nb.score(x_test,y_test)
```

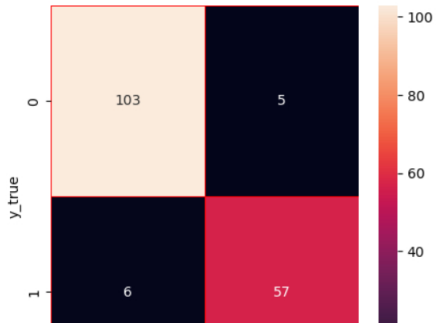
```
Print accuracy of naive bayes algo: 0.935672514619883
```

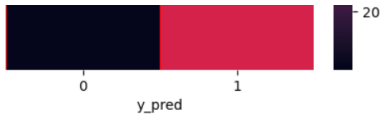
Our model has an accuracy rate of 0.93. We can use confusion matrix to see which parts it got wrong in predicting.

In [23]:
```
y_pred = nb.predict(x_test)
y_true = y_test

cm = confusion_matrix(y_true, y_pred)

#visualize
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot = True, linewidths=0.5,linecolor="red",fmt = ".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```





SVM

In [24]:
```
svm = SVC()   #We are building our model
svm.fit(x_train,y_train)      #We are training our model
print("Print accuracy of svm algo: ",svm.score(x_test,y_test))
svm_acc_score = svm.score(x_test,y_test)
```

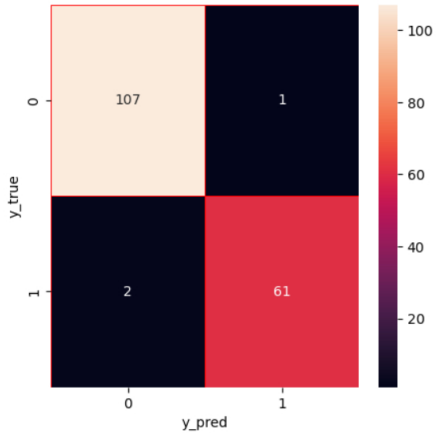
```
Print accuracy of svm algo:  0.9824561403508771
```

Our model has an accuracy rate of 0.98. We can use confusion matrix to see which parts it got wrong in predicting.

In [25]:
```
y_pred = svm.predict(x_test)
y_true = y_test

cm = confusion_matrix(y_true, y_pred)

#visualize
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot = True, linewidths=0.5,linecolor="red",fmt = ".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```



Decision Tree Classifier

In [26]:
```
dt = DecisionTreeClassifier(random_state = 1)   #We are building our model
dt.fit(x_train,y_train)       #We are training our model
print("Print accuracy of Decision Tree Classifier algo: ",dt.score(x_test,y_test))
dt_acc_score = dt.score(x_test,y_test)
```

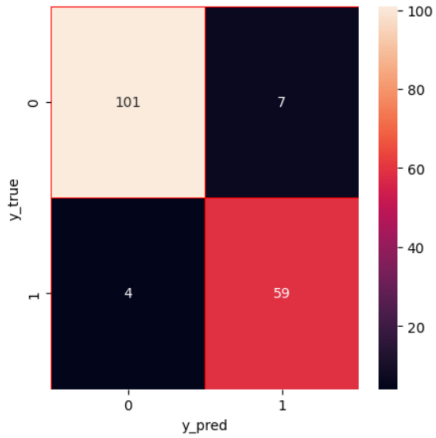
```
Print accuracy of Decision Tree Classifier algo:  0.935672514619883
```

Our model has an accuracy rate of 0.93. We can use confusion matrix to see which parts it got wrong in predicting.

In [27]:
```
y_pred = dt.predict(x_test)
y_true = y_test

cm = confusion_matrix(y_true, y_pred)

#visualize
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot = True, linewidths=0.5,linecolor="red",fmt = ".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```



Random Forest Classifier

In [28]:
```
rf = RandomForestClassifier(n_estimators=10,random_state=1)
rf.fit(x_train,y_train)
print("Print accuracy of Random Forest Classifier algo: ",rf.score(x_test,y_test))
rf_acc_score = rf.score(x_test,y_test)
```

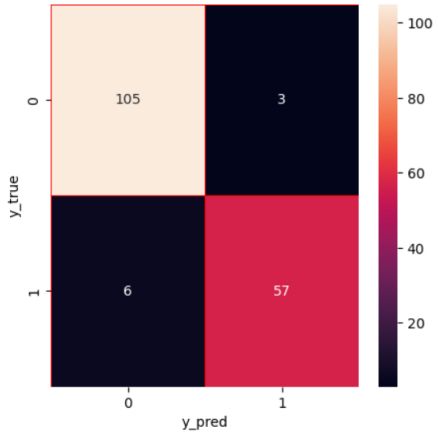
```
Print accuracy of Random Forest Classifier algo:  0.9473684210526315
```

Our model has an accuracy rate of 0.94. We can use confusion matrix to see which parts it got wrong in predicting.

In [29]:
```
y_pred = rf.predict(x_test)
y_true = y_test

cm = confusion_matrix(y_true, y_pred)

#visualize
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(cm,annot = True, linewidths=0.5,linecolor="red",fmt = ".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```



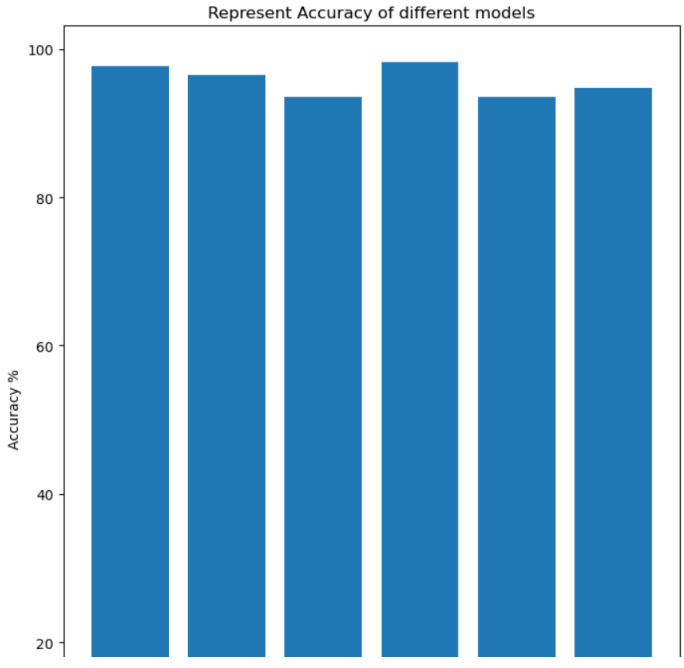
Model Evaluation

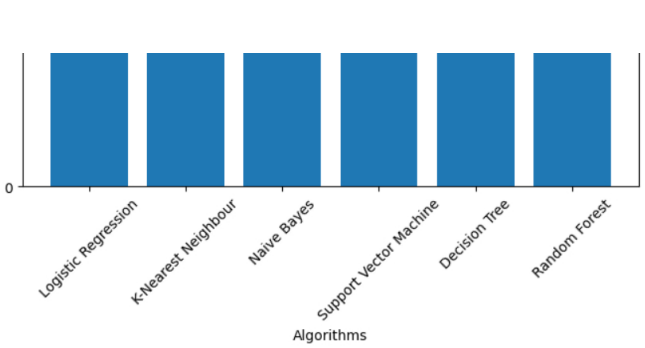
In [30]:
```
model_ev = pd.DataFrame({'Model': ['Logistic Regression','K-Nearest Neighbour','Naive Bayes','Support Vector Machine','Decision T
                         'Accuracy': [lr_acc_score*100,knn_acc_score*100,nb_acc_score*100,svm_acc_score*100,dt_acc_score*100,rf_a
model_ev
```

Out[30]:

	Model	Accuracy
0	Logistic Regression	97.660819
1	K-Nearest Neighbour	96.491228
2	Naive Bayes	93.567251
3	Support Vector Machine	98.245614
4	Decision Tree	93.567251
5	Random Forest	94.736842

In [31]:
```
#Visualize
plt.figure(figsize=(8,10))
plt.title("Represent Accuracy of different models")
plt.ylabel("Accuracy %")
plt.xlabel("Algorithms")
plt.xticks(rotation=45)
plt.bar(model_ev['Model'],model_ev['Accuracy'])
plt.show()
```





Conclusion

In short, we conclude that the diagnosis of breast cancer can best be made with the SVM algorithm, which makes accurate predictions with a rate of 98.2 percent.