In [321]:

```
import pandas as pd
pd.options.mode.chained_assignment = None
import numpy as np
import matplotlib.pyplot as plt
import time
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV, Strati
fiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC, SVC
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
ore, log_loss
from sklearn.metrics import auc, precision_recall_curve
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
SEED=42
```

1. Read and analyze data

conclusion

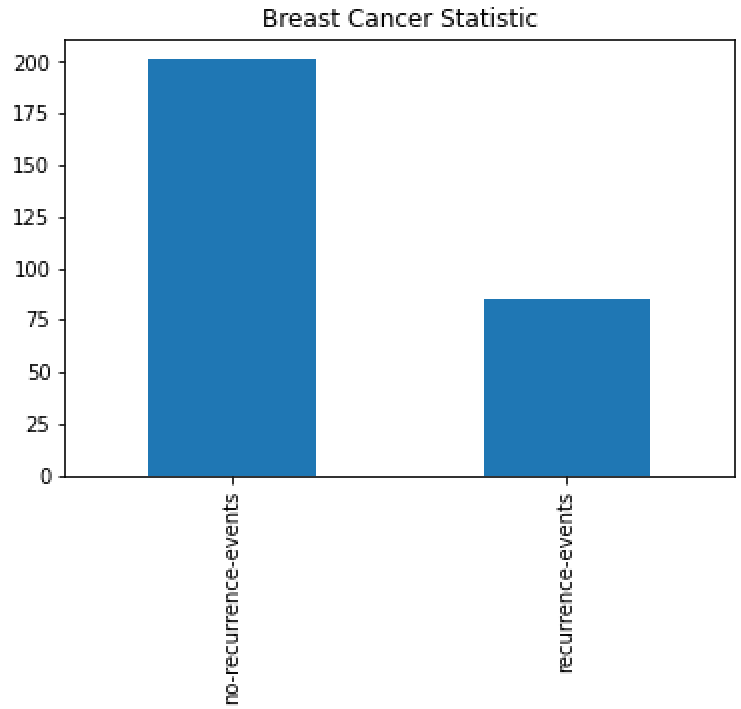
- **It is analyzed from Figure 1 that this is an unbalanced data set, which needs to be paid attention to when processing data.**
- **It can be analyzed from Figures 2, 3, and 4 that through filtering, 6 features relationships are retained --- ['age','menopause','tumor-size','inv-nodes','node-caps','deg-malig']**

In [322]:

```
data = pd.read_csv('breast-cancer.csv', header=None)
data.columns = [ 'Class', 'age','menopause','tumor-size','inv-nodes','node-caps'
,'deg-malig','breast','breast-quad','irradiat']
data['Class'].value_counts().plot(kind='bar').set_title('Breast Cancer Statisti
c')
```

Out[322]:

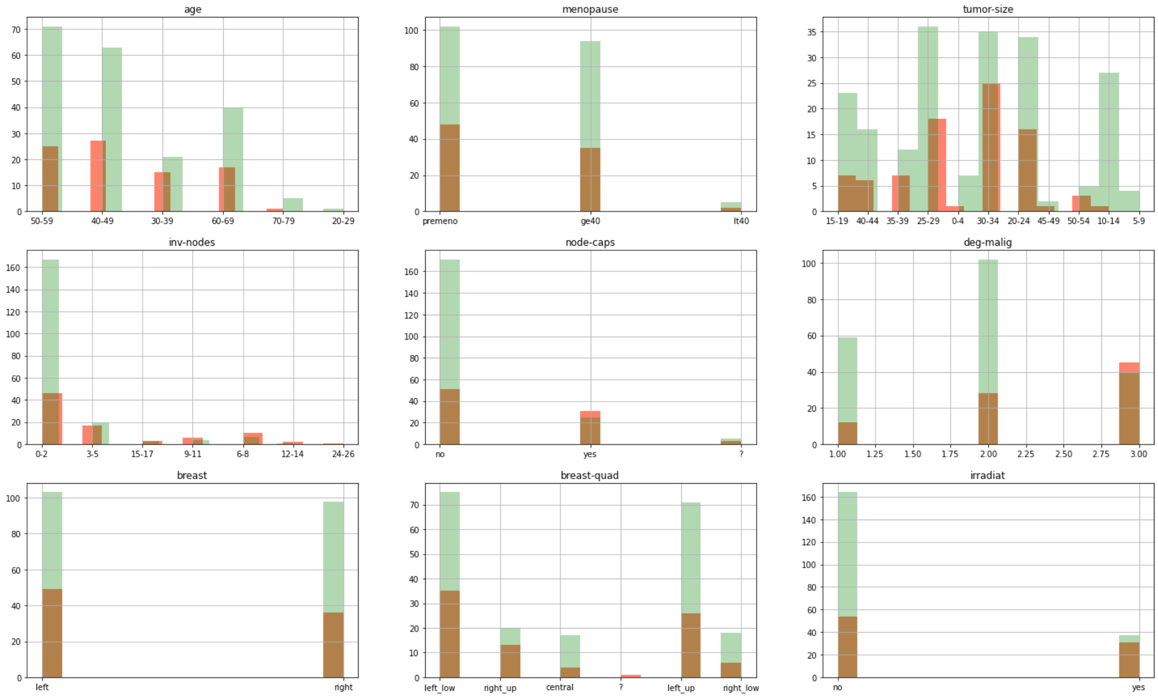
```
Text(0.5, 1.0, 'Breast Cancer Statistic')
```



In [323]:

```
features = ['age','menopause','tumor-size','inv-nodes','node-caps','deg-malig',
'breast','breast-quad','irradiat']
rows, cols = 3, 3
fig, ax = plt.subplots(rows, cols, figsize=(25,15) )
subr, subc = 0, 0

for i, feature in enumerate(features):
    if subc == cols - 1:
        subr += 1
    subc = i % cols
    print(i)

    data[data.Class=='recurrence-events'][feature].hist(bins=15, color='tomato',
alpha=0.8 ,ax=ax[subr, subc]).set_title(feature)
    data[data.Class=='no-recurrence-events'][feature].hist(bins=15, color='gree
n', alpha=0.3 ,ax=ax[subr, subc]).set_title(feature)
```

```
0
1
2
3
4
5
6
7
8
```



In [324]:

```
# process data for easy classification
data[data.isnull().any(axis = 1)]
ProcData = data.copy()

# binarize Class & node-caps & irradiat
ProcData['node-caps']= (ProcData['node-caps']=='yes').astype(int)
ProcData['irradiat']= (ProcData['irradiat']=='yes').astype(int)
ProcData['Class']= (ProcData['Class']=='recurrence-events').astype(int)

quad = {'left_up':1, 'left_low': 2, 'right_up':3, 'right_low':4, 'central':5}
ProcData = ProcData.replace({'breast-quad': quad})
ProcData['breast-quad'] = ProcData['breast-quad'].apply(pd.to_numeric, downcast=
'float', errors='coerce')
ProcData[ProcData.isnull().any(axis = 1)]
ProcData = ProcData.dropna()

breast = {'left':1, 'right':2}
ProcData = ProcData.replace({'breast': breast})


menopause = {'premeno':1, 'ge40': 2, 'lt40':3}
ProcData = ProcData.replace({'menopause': menopause})

#convert inv-nodes to the median of data.
nodes = {'0-2':1, '3-5':4,'6-8':7,'9-11':10, '12-14':13,'15-17':16,'18-20':19,'2
1-23':22,'24-26':25,'27-29':28,'30-32':31,'33-35':34,
        '36-38':37,'39':39}
ProcData = ProcData.replace({'inv-nodes': nodes})
(ProcData['inv-nodes'].describe)


#convert age to the numerical average of data.
age = {'20-29':24.5, '30-39':34.5,'40-49':44.5,'50-59':54.5, '60-69':64.5,'70-7
9':74.5,'80-89':84.5,'90-99':94.5}
ProcData = ProcData.replace({'age': age})



,
#convert tumor-size to the numerical average of data.
Tumor = {'0-4':2, '5-9':7,'10-14':12,'15-19':17, '20-24':22,'25-29':27,'30-34':3
2,'35-39':37,'40-44':42,'45-49':47,'50-54':52}
ProcData = ProcData.replace({'tumor-size': Tumor})
ProcData.head()
```

Out[324]:

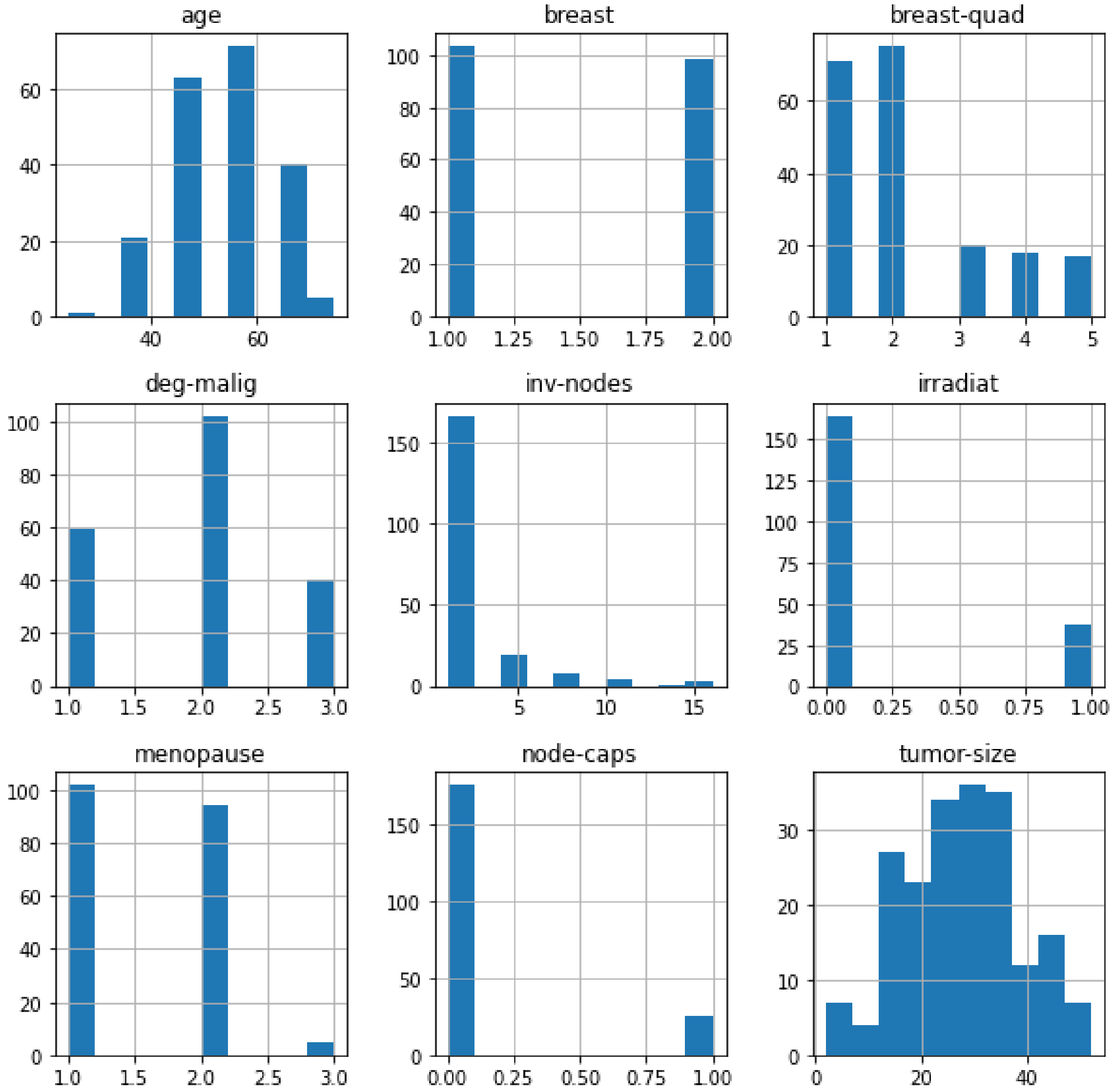
	Class	age	menopause	tumor-size	inv-nodes	node-caps	deg-malig	breast	breast-quad	irradiat
0	0	34.5	1	32	1	0	3	1	2.0	0
1	0	44.5	1	22	1	0	2	2	3.0	0
2	0	44.5	1	22	1	0	2	1	2.0	0
3	0	64.5	2	17	1	0	2	2	1.0	0
4	0	44.5	1	2	1	0	2	2	4.0	0

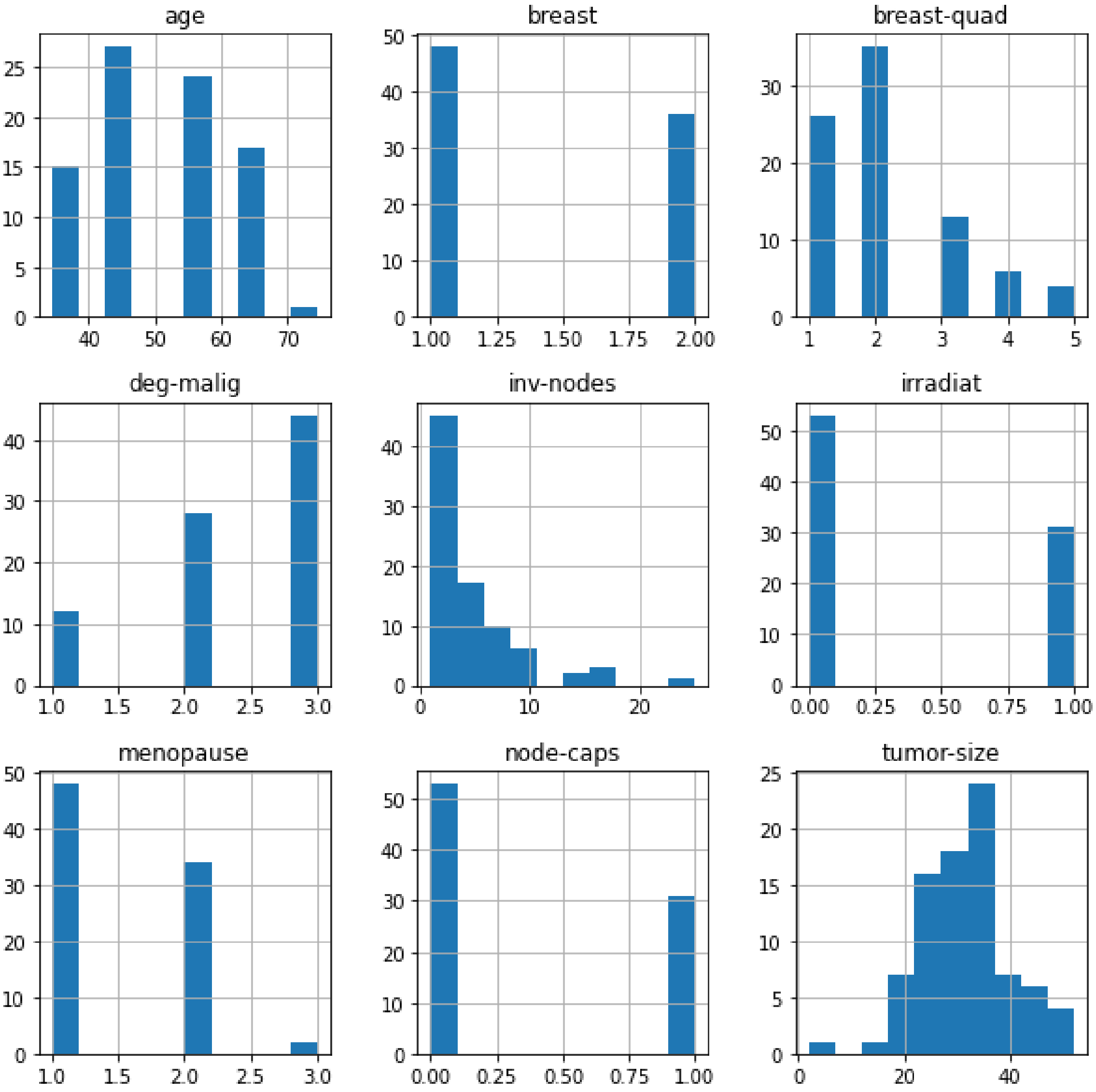
In [326]:

```
ProcData.groupby('Class').hist(figsize=(10, 10))
```

Out[326]:

```
Class
0    [[AxesSubplot(0.125,0.670278;0.215278x0.209722...
1    [[AxesSubplot(0.125,0.670278;0.215278x0.209722...
dtype: object
```



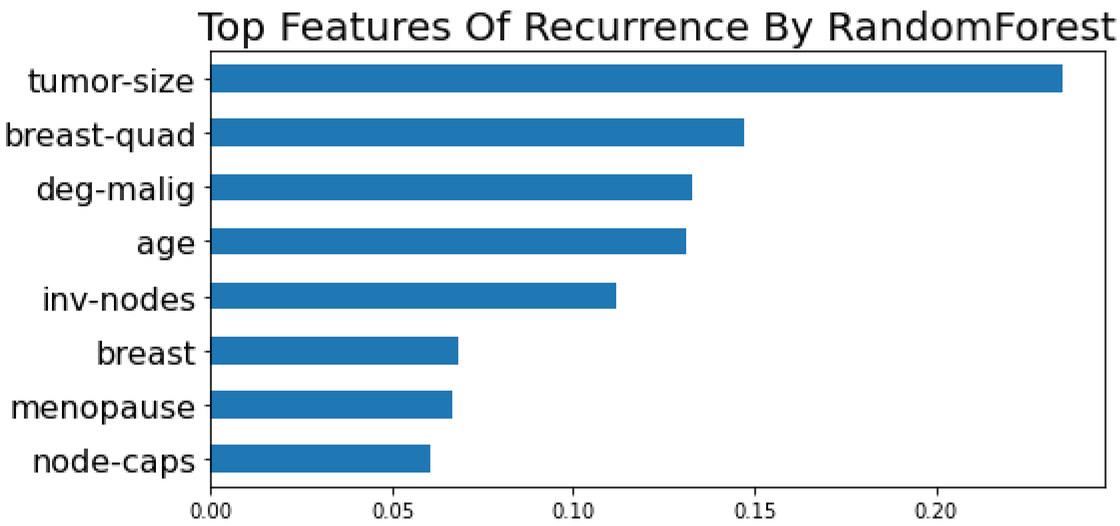

age
breast
breast-quad
deg-malig
inv-nodes
irradiat
menopause
node-caps
tumor-size

In [330]:

```
x, y = ProcData.drop('Class', axis=1).fillna(ProcData.drop('Class', axis=1).mean
()), ProcData['Class']
rfc = RandomForestClassifier(random_state=SEED, n_estimators=100)
rfc_model = rfc.fit(x, y)
(pd.Series(rfc_model.feature_importances_, index=x.columns)
    .nlargest(8)
    .plot(kind='barh', figsize=[8,4])
    .invert_yaxis())
plt.yticks(size=16)
plt.title('Top Features Of Recurrence By RandomForest', size=20)
```

Out[330]:

```
Text(0.5, 1.0, 'Top Features Of Recurrence By RandomForest')
```



2. Split data to 70:30 ratio and Fit different models

In [328]:

```
chosen_features = ['age','menopause','tumor-size','inv-nodes','node-caps','deg-m
alig']
x = ProcData[chosen_features]
y = ProcData['Class']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.3, random_s
tate=SEED, stratify=y)
print('x_train: ', x_train.shape)
print('y_train: ', y_train.shape)
print('x_test: ', x_test.shape)
print('y_test: ', y_test.shape)
```

```
x_train:  (199, 6)
y_train:  (199,)
x_test:  (86, 6)
y_test:  (86,)
```

2.1 Cross Validation

Use StratifiedKFold, especially if target class is imbalance

In [333]:

```
# for imbalance dataset, ensure the output same proportion
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=SEED)

def baseline_record(model, x_train, x_test, y_train, y_test, name):
    model.fit(x_train, y_train)
    acc = np.mean(cross_val_score(model, x_train, y_train, cv=skf, scoring='accu
racy'))
    recall = np.mean(cross_val_score(model, x_train, y_train, cv=skf, scoring='r
ecall'))
    f1 = np.mean(cross_val_score(model, x_train, y_train, cv=skf, scoring='f1'))
    rocauc = np.mean(cross_val_score(model, x_train, y_train, cv=skf, scoring='r
oc_auc'))

    y_pred = y_pred = model.predict(x_test)
    logloss = log_loss(y_test, y_pred)

    dataset_model = pd.DataFrame({'model': [name], 'acc': [acc], 'recall': [reca
ll], 'f1': [f1], 'rocauc': [rocauc], 'logloss': [logloss], 'timetaken': [0]})

    return dataset_model

lg = LogisticRegression()
dt = DecisionTreeClassifier()
gnb = GaussianNB()
knn = KNeighborsClassifier()
rfc = RandomForestClassifier()
svc = SVC()


x_train = x_train.fillna(x_train.mean())
dataset_models = pd.concat([baseline_record(lg, x_train, x_test, y_train, y_test
, 'Logistic'),
                            baseline_record(dt, x_train, x_test, y_train, y_test
, 'DecisionTree'),
                            baseline_record(gnb, x_train, x_test, y_train, y_tes
t, 'GaussianNB'),
                            baseline_record(knn, x_train, x_test, y_train, y_tes
t, 'KNN'),
                            baseline_record(rfc, x_train, x_test, y_train, y_tes
t, 'RandomForest'),
                            baseline_record(svc, x_train, x_test, y_train, y_tes
t, 'SVC')], axis=0).reset_index()


dataset_models = dataset_models.drop('index', axis=1)
dataset_models
```

Out[333]:

	model	acc	recall	f1	rocauc	logloss	timetaken
0	Logistic	0.728462	0.322727	0.413968	0.722646	11.245248	0
1	DecisionTree	0.648205	0.404545	0.398841	0.572835	11.245304	0
2	GaussianNB	0.733718	0.442424	0.498656	0.726109	10.442039	0
3	KNN	0.678205	0.216667	0.273220	0.624811	12.450108	0
4	RandomForest	0.658462	0.404545	0.411924	0.669183	10.843672	0
5	SVC	0.703590	0.000000	0.000000	0.697051	10.040342	0

3. Optimise models

In [334]:

```
def optimise_record(model, x_train, x_test, y_train, y_test, model_name):
    x_train = x_train = x_train.fillna(x_train.mean())
    model.fit(x_train, y_train)
    optimal_th = 0.2

    for i in range(0,2):
        score_list = []
        th_list = [np.linspace(optimal_th-0.4999, optimal_th+0.4999, 11),
                   np.linspace(optimal_th-0.1, optimal_th+0.1, 21),
                   np.linspace(optimal_th-0.01, optimal_th+0.01, 21)]

        for th in th_list[i]:
            if th<0:
                score_list.append(-1)
                continue
            y_pred = (model.predict_proba(x_test)[:,1] >= th)
            f1scor = f1_score(y_test, y_pred)
            score_list.append(f1scor)
        optimal_th = float(th_list[i][score_list.index(max(score_list))])

    print('optimal F1 score = {:.4f}'.format(max(score_list)))
    print('optimal threshold = {:.3f}'.format(optimal_th))

    print(model_name, 'acc score is')
    print('Training: {:.2f}%'.format(100*model.score(x_train, y_train)))
    accuracy      = np.mean(cross_val_score(model, x_train, y_train, cv=skf, sco
ring='accuracy'))
    print('Test set: {:.2f}%'.format(100*accuracy))

    y_pred = (model.predict_proba(x_test)[:,1] >= 0.25)
    print('\nAdjust to 0.25:')
    print('Precision: {:.4f},   Recall: {:.4f},   F1 Score: {:.4f}'.format(preci
sion_score(y_test, y_pred), recall_score(y_test, y_pred), f1_score(y_test, y_pre
d)))
    print(model_name, 'confusion matrix: \n', confusion_matrix(y_test, y_pred))

    y_pred = model.predict(x_test)
    print('\nDefault 0.50:')
    print('Precision: {:.4f},   Recall: {:.4f},   F1 Score: {:.4f}'.format(preci
sion_score(y_test, y_pred), recall_score(y_test, y_pred), f1_score(y_test, y_pre
d)))
    print(model_name, 'confusion matrix: \n', confusion_matrix(y_test, y_pred))

    y_pred = (model.predict_proba(x_test)[:,1] >= 0.75)
    print('\nAdjust to 0.75:')
    print('Precision: {:.4f},   Recall: {:.4f},   F1 Score: {:.4f}'.format(preci
sion_score(y_test, y_pred), recall_score(y_test, y_pred), f1_score(y_test, y_pre
d)))
    print(model_name, 'confusion matrix: \n', confusion_matrix(y_test, y_pred))

    y_pred = (model.predict_proba(x_test)[:,1] >= optimal_th)
    print('\nOptimal threshold {:.3f}'.format(optimal_th))
    precision    = precision_score(y_test, y_pred)
    recall       = recall_score(y_test, y_pred)
    f1score      = f1_score(y_test, y_pred)
    rocauc = np.mean(cross_val_score(model, x_train, y_train, cv=skf, scoring='r
oc_auc'))
    print('Precision: {:.4f},   Recall: {:.4f},   F1 Score: {:.4f}'.format(preci
sion, recall, f1score))
```

```
    print(model_name, 'confusion matrix: \n', confusion_matrix(y_test, y_pred))

    y_pred = model.predict_proba(x_test)[:,1]
    logloss      = log_loss(y_test, y_pred)
    print(model_name, 'Log-loss: {:.4f}'.format(logloss))

    dataset_model = pd.DataFrame({'model': [model_name], 'acc': [accuracy], 'rec
all': [recall], 'f1': [f1score], 'rocauc': [rocauc], 'logloss': [logloss], 'time
taken': [1000]})
    return dataset_model
```

In [335]:

```
print('\n=====LogisticRegression=====')
time1 = time.time()
kf = KFold(n_splits=5, random_state=SEED, shuffle=True)
score_list = []
c_list = 10**np.linspace(-3,3,300)
for c in c_list:
    logit = LogisticRegression(C = c)
    cvs = (cross_val_score(logit,x_train, y_train, cv=kf, scoring='f1')).mean()
    score_list.append(cvs)
print('optimal cv F1 score = {:.4f}'.format(max(score_list)))
optimal_c = float(c_list[score_list.index(max(score_list))])
print('optimal value of C = {:.3f}'.format(optimal_c))

logic = LogisticRegression(C = optimal_c)
model1 = optimise_record(logic, x_train, x_test, y_train, y_test, 'Logistic')
model1.timetaken[0] = time.time() - time1
```

```
=====LogisticRegression=====
optimal cv F1 score = 0.4574
optimal value of C = 1.289
optimal F1 score = 0.5405
optimal threshold = 0.200
Logistic acc score is
Training: 75.38%
Test set: 72.85%

Adjust to 0.25:
Precision: 0.4103,   Recall: 0.6400,   F1 Score: 0.5000
Logistic confusion matrix:
 [[38 23]
 [ 9 16]]

Default 0.50:
Precision: 0.3333,   Recall: 0.1600,   F1 Score: 0.2162
Logistic confusion matrix:
 [[53  8]
 [21  4]]

Adjust to 0.75:
Precision: 1.0000,   Recall: 0.0400,   F1 Score: 0.0769
Logistic confusion matrix:
 [[61  0]
 [24  1]]

Optimal threshold 0.200
Precision: 0.4082,   Recall: 0.8000,   F1 Score: 0.5405
Logistic confusion matrix:
 [[32 29]
 [ 5 20]]
Logistic Log-loss: 0.5717
```

In [336]:

```
print('\n=====DecisionTree=====')
time1 = time.time()
kf = KFold(n_splits=8, random_state=SEED, shuffle=True)
d_scores = []
for d in range(2, 11):
    decisiontree = DecisionTreeClassifier(max_depth=d)
    cvs = cross_val_score(decisiontree, x_train, y_train, cv=kf, scoring='f1').m
ean()
    d_scores.append(cvs)
print('optimal F1 score = {:.4f}'.format(max(d_scores)))
optimal_d = d_scores.index(max(d_scores))+2
print('optimal max_depth =', optimal_d)

decisiontree = DecisionTreeClassifier(max_depth=optimal_d)
model2 = optimise_record(decisiontree, x_train, x_test, y_train, y_test, 'Decisi
onTree')
model2.timetaken[0] = time.time() - time1
```

```
=====DecisionTree=====
optimal F1 score = 0.4189
optimal max_depth = 2
optimal F1 score = 0.5000
optimal threshold = 0.080
DecisionTree acc score is
Training: 77.89%
Test set: 76.90%

Adjust to 0.25:
Precision: 0.4800,   Recall: 0.4800,   F1 Score: 0.4800
DecisionTree confusion matrix:
 [[48 13]
 [13 12]]

Default 0.50:
Precision: 0.5714,   Recall: 0.1600,   F1 Score: 0.2500
DecisionTree confusion matrix:
 [[58  3]
 [21  4]]

Adjust to 0.75:
Precision: 0.5714,   Recall: 0.1600,   F1 Score: 0.2500
DecisionTree confusion matrix:
 [[58  3]
 [21  4]]

Optimal threshold 0.080
Precision: 0.3433,   Recall: 0.9200,   F1 Score: 0.5000
DecisionTree confusion matrix:
 [[17 44]
 [ 2 23]]
DecisionTree Log-loss: 0.5740
```

In [337]:

```
print('\n=====GaussianNB=====')
time1 = time.time()
gnb = GaussianNB()
model3 = optimise_record(gnb, x_train, x_test, y_train, y_test, 'GaussianNB')
model3.timetaken[0] = time.time() - time1
```

```
=====GaussianNB=====
optimal F1 score = 0.5714
optimal threshold = 0.100
GaussianNB acc score is
Training: 73.37%
Test set: 73.37%

Adjust to 0.25:
Precision: 0.5238,   Recall: 0.4400,   F1 Score: 0.4783
GaussianNB confusion matrix:
 [[51 10]
 [14 11]]

Default 0.50:
Precision: 0.4706,   Recall: 0.3200,   F1 Score: 0.3810
GaussianNB confusion matrix:
 [[52  9]
 [17  8]]

Adjust to 0.75:
Precision: 0.3846,   Recall: 0.2000,   F1 Score: 0.2632
GaussianNB confusion matrix:
 [[53  8]
 [20  5]]

Optimal threshold 0.100
Precision: 0.4737,   Recall: 0.7200,   F1 Score: 0.5714
GaussianNB confusion matrix:
 [[41 20]
 [ 7 18]]
GaussianNB Log-loss: 0.9399
```

In [338]:

```
print('\n=====KNN======')
time2 = time.time()
kf = KFold(n_splits=8, random_state=SEED, shuffle=True)
k_scores = []
for k in range(1, 21):
    knn = KNeighborsClassifier(n_neighbors = k)
    cvs = cross_val_score(knn, x_train, y_train, cv=kf, scoring='f1').mean()
    k_scores.append(cvs)
optimal_k = k_scores.index(max(k_scores))+1
print('optimal value of K =', optimal_k)

knn = KNeighborsClassifier(n_neighbors = optimal_k)
model4 = optimise_record(knn, x_train, x_test, y_train, y_test, 'KNN')
model4.timetaken[0] = time.time() - time2

knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
```

```
=====KNN======
optimal value of K = 7
optimal F1 score = 0.4130
optimal threshold = 0.000
KNN acc score is
Training: 73.37%
Test set: 67.86%

Adjust to 0.25:
Precision: 0.2807,   Recall: 0.6400,   F1 Score: 0.3902
KNN confusion matrix:
 [[20 41]
 [ 9 16]]

Default 0.50:
Precision: 0.2857,   Recall: 0.0800,   F1 Score: 0.1250
KNN confusion matrix:
 [[56  5]
 [23  2]]

Adjust to 0.75:
Precision: 0.0000,   Recall: 0.0000,   F1 Score: 0.0000
KNN confusion matrix:
 [[60  1]
 [25  0]]

Optimal threshold 0.000
Precision: 0.2836,   Recall: 0.7600,   F1 Score: 0.4130
KNN confusion matrix:
 [[13 48]
 [ 6 19]]
KNN Log-loss: 2.9347
```

In [339]:

```
print('\n======RandomForestClassifier======')
time3 = time.time()
kf = KFold(n_splits=6, random_state=SEED, shuffle=True)
score_list = []
n_list = []
for n in [100, 150, 200, 250, 300, 350, 400, 450, 500]:
    randomforest = RandomForestClassifier(n_estimators=n)
    cvs = (cross_val_score(randomforest, x_train, y_train, cv=kf, scoring='f1'))
.mean()
    score_list.append(cvs)
    n_list.append(n)
print('optimal F1 score = {:.4f}'.format(max(score_list)))
optimal_n = int(n_list[score_list.index(max(score_list))])
print('optimal n_estimators = {:.0f}'.format(optimal_n))

rfc = RandomForestClassifier(n_estimators=optimal_n)
model5 = optimise_record(rfc, x_train, x_test, y_train, y_test, 'RandomForest')
model5.timetaken[0] = time.time() - time3
```

```
======RandomForestClassifier======
optimal F1 score = 0.4547
optimal n_estimators = 450
optimal F1 score = 0.5085
optimal threshold = 0.300
RandomForest acc score is
Training: 91.46%
Test set: 69.86%

Adjust to 0.25:
Precision: 0.3902,   Recall: 0.6400,   F1 Score: 0.4848
RandomForest confusion matrix:
 [[36 25]
 [ 9 16]]

Default 0.50:
Precision: 0.5000,   Recall: 0.4400,   F1 Score: 0.4681
RandomForest confusion matrix:
 [[50 11]
 [14 11]]

Adjust to 0.75:
Precision: 0.6667,   Recall: 0.1600,   F1 Score: 0.2581
RandomForest confusion matrix:
 [[59  2]
 [21  4]]

Optimal threshold 0.300
Precision: 0.4412,   Recall: 0.6000,   F1 Score: 0.5085
RandomForest confusion matrix:
 [[42 19]
 [10 15]]
RandomForest Log-loss: 0.6734
```

In [346]:

```
print('\n======SVC======')
time1 = time.time()
svc = SVC(C=1.12, kernel='rbf', gamma='scale', probability=True)
model6 = optimise_record(svc, x_train, x_test, y_train, y_test, 'SVC')
model6.timetaken[0] = time.time() - time1
```

```
======SVC======
optimal F1 score = 0.5070
optimal threshold = 0.260
SVC acc score is
Training: 70.85%
Test set: 70.36%

Adjust to 0.25:
Precision: 0.2907,   Recall: 1.0000,   F1 Score: 0.4505
SVC confusion matrix:
 [[ 0 61]
 [ 0 25]]

Default 0.50:
Precision: 0.0000,   Recall: 0.0000,   F1 Score: 0.0000
SVC confusion matrix:
 [[61  0]
 [25  0]]

Adjust to 0.75:
Precision: 0.0000,   Recall: 0.0000,   F1 Score: 0.0000
SVC confusion matrix:
 [[60  1]
 [25  0]]

Optimal threshold 0.260
Precision: 0.3913,   Recall: 0.7200,   F1 Score: 0.5070
SVC confusion matrix:
 [[33 28]
 [ 7 18]]
SVC Log-loss: 0.5861
```

4. Compare with the baseline record

It can be seen that before the comparison and adjustment, the basic indicators have achieved certain improvement

In [347]:

```
optimise_models= pd.concat([model1, model2, model3, model4, model5, model6],axis
= 0).reset_index()
optimise_models.drop('index', axis=1, inplace=True)
optimise_models
```

Out[347]:

	model	acc	recall	f1	rocauc	logloss	timetaken
0	Logistic	0.728462	0.80	0.540541	0.721943	0.571700	35
1	DecisionTree	0.768974	0.92	0.500000	0.666964	0.573984	0
2	GaussianNB	0.733718	0.72	0.571429	0.726109	0.939928	0
3	KNN	0.678590	0.76	0.413043	0.636661	2.934723	1
4	RandomForest	0.698590	0.60	0.508475	0.666044	0.673368	44
5	SVC	0.703590	0.72	0.507042	0.697051	0.586078	0

In [341]:

```
dataset_models
```

Out[341]:

	model	acc	recall	f1	rocauc	logloss	timetaken
0	Logistic	0.728462	0.322727	0.413968	0.722646	11.245248	0
1	DecisionTree	0.648205	0.404545	0.398841	0.572835	11.245304	0
2	GaussianNB	0.733718	0.442424	0.498656	0.726109	10.442039	0
3	KNN	0.678205	0.216667	0.273220	0.624811	12.450108	0
4	RandomForest	0.658462	0.404545	0.411924	0.669183	10.843672	0
5	SVC	0.703590	0.000000	0.000000	0.697051	10.040342	0

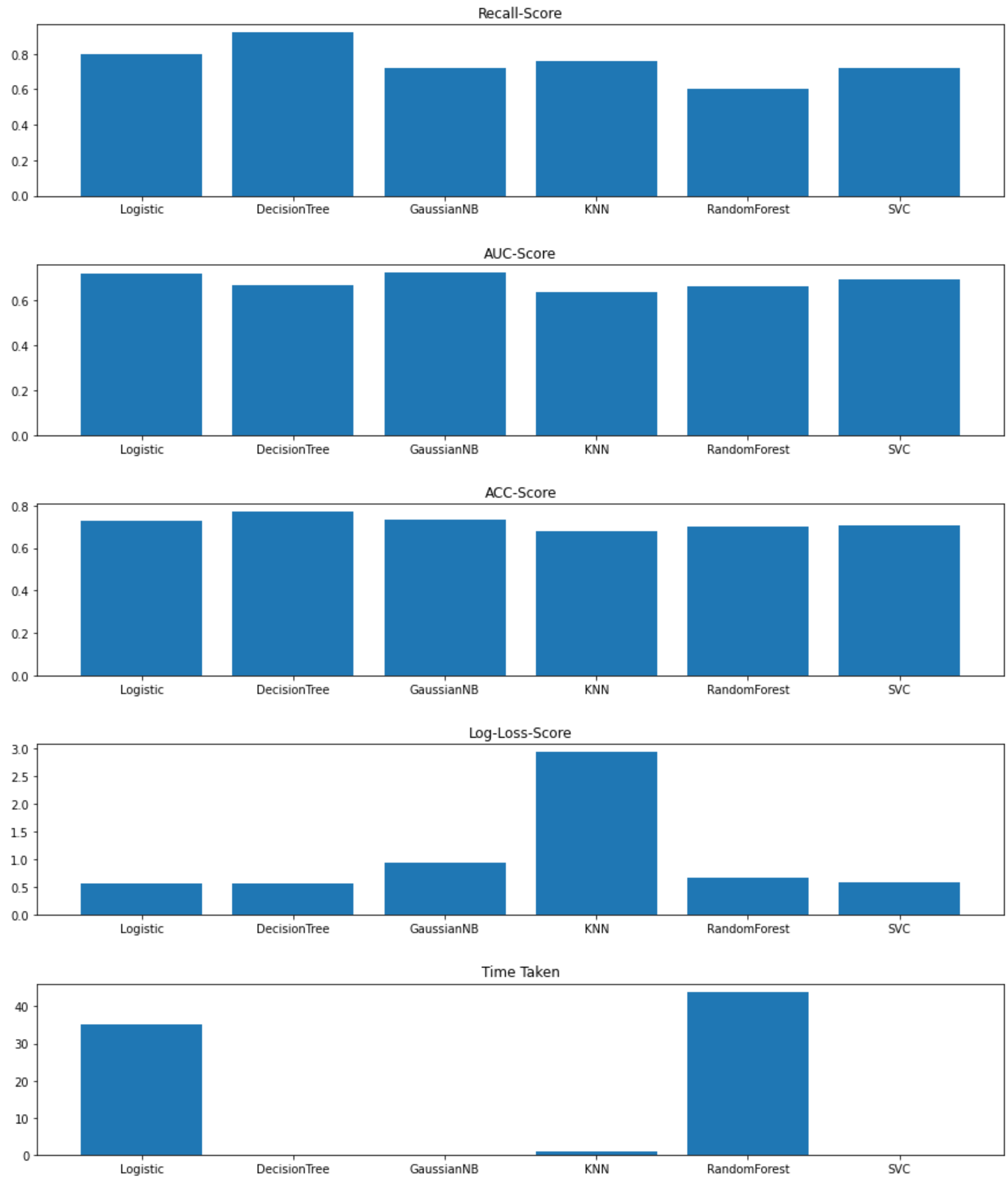
5. The Best Model

In fact, each model has achieved good results, but considering that this is an extremely imbalance data set, we should focus on the recall rate rather than the accuracy rate, so the best model is **DecisionTree**, and DT is still achieved gratifying resultson other data.

recall = 23 / (23+2) = 0.92

In [348]:

```
fig, ax = plt.subplots(5, 1, figsize=(15, 18))
ax[0].bar(optimise_models.model, optimise_models.recall)
ax[0].set_title('Recall-Score')

ax[1].bar(optimise_models.model, optimise_models.rocauc)
ax[1].set_title('AUC-Score')

ax[2].bar(optimise_models.model, optimise_models.acc)
ax[2].set_title('ACC-Score')

ax[3].bar(optimise_models.model, optimise_models.logloss)
ax[3].set_title('Log-Loss-Score')

ax[4].bar(optimise_models.model, optimise_models.timetaken)
ax[4].set_title('Time Taken')

fig.subplots_adjust(hspace=0.4, wspace=0.2)
```



In [349]:

```
bestmodel = decisiontree

def make_confusion_matrix(model, threshold=0.080):
    y_pred = (bestmodel.predict_proba(x_test)[:, 1] >= threshold)
    conf = confusion_matrix(y_test, y_pred)
    plt.figure(figsize = [5,5])
    sns.heatmap(conf, cmap=plt.cm.Blues, annot=True, square=True, fmt='d',
           xticklabels=['no-recurrence', 'recurrence'],
           yticklabels=['no-recurrence', 'recurrence']);
    plt.xlabel('prediction')
    plt.ylabel('actual')

from ipywidgets import interactive, FloatSlider
interactive(lambda threshold: make_confusion_matrix(bestmodel, threshold), thres
hold=(0.080))
```