In [15]:
```
import pandas as pd
import numpy as np
import seaborn as sb
from matplotlib import pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression,LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
```

Loading data

This will inolve pandas library where, csv file is loaded

In [16]:
```
#loading data
data=pd.read_csv('data.csv')
#data.head()
```

In [17]:
```
#check the shape of the data
data.shape
```

Out[17]:
```
(6819, 96)
```

The loaded data has 6819 rows and 96 columns

Check for na values

In [20]:
```
#are there any columns with null values
for i in range(data.shape[1]):
    if data.isna().any()[i]==True:
        print(data.columns[i],"-------------",data.isna().any()[i])
    else:
        pass
        #print(data.columns[i],"--------------",data.isna().any()[i])
        #this would display all column if first case is false
#since no values printed out as true then there are no na values
```

Since there are many columns, for loop is utilised to check all the columns. If the column has any na values the column is indicated by a bool variable true while otherwise indicated false. For this case there are no na values since all the columns are indicated false.

In [21]:
```
#selecting variables to work with
sel_data=data.iloc[:,[0,1,2,3,4,13,5,6]]
sel_data.head()
```

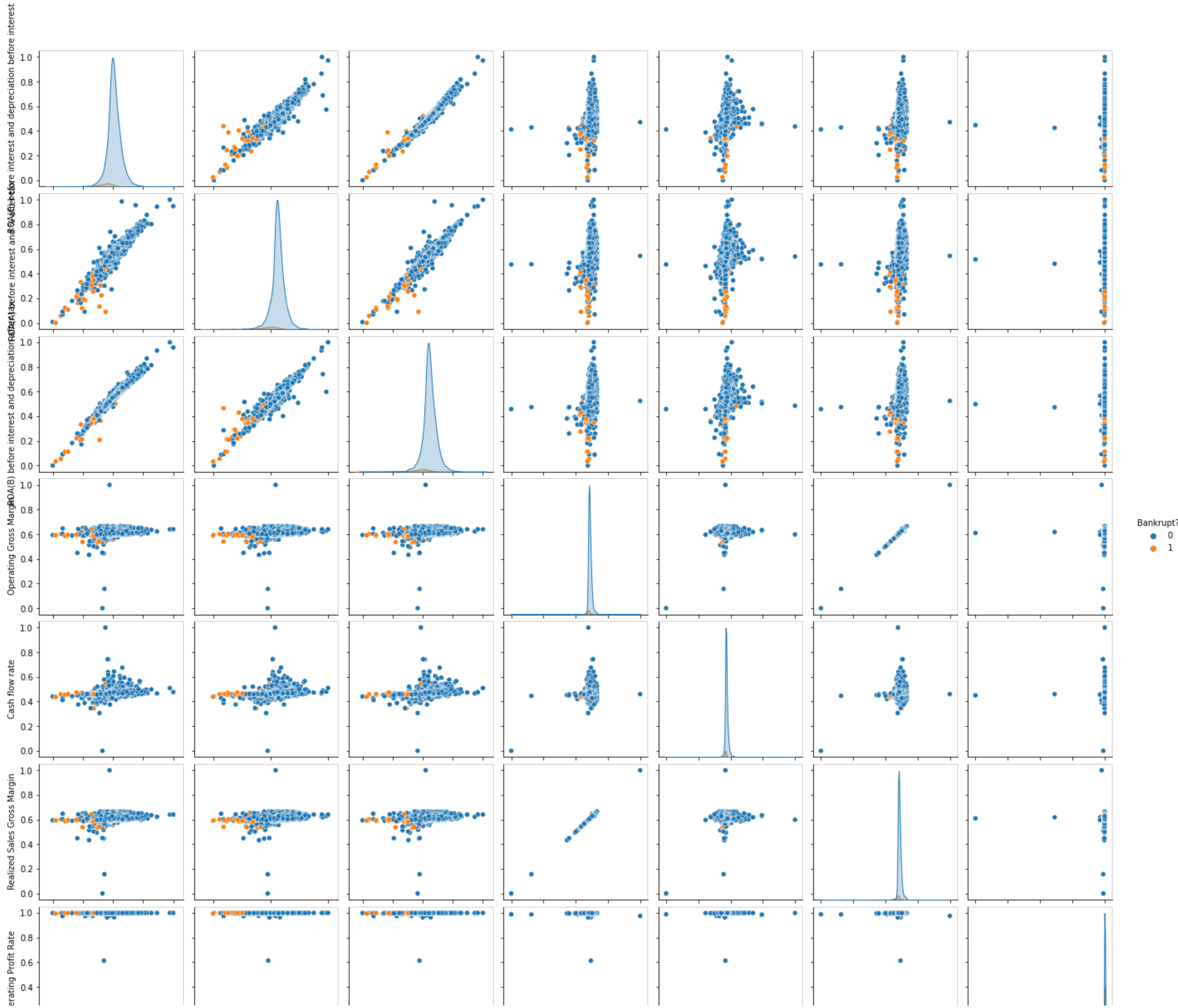
Out[21]:

	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Cash flow rate	Realized Sales Gross Margin	Operating Profit Rate
0	1	0.370594	0.424389	0.405750	0.601457	0.458143	0.601457	0.998969
1	1	0.464291	0.538214	0.516730	0.610235	0.461867	0.610235	0.998946
2	1	0.426071	0.499019	0.472295	0.601450	0.458521	0.601364	0.998857
3	1	0.399844	0.451265	0.457733	0.583541	0.465705	0.583541	0.998700
4	1	0.465022	0.538432	0.522298	0.598783	0.462746	0.598783	0.998973

Considering that there are many columns, few columns can be selected for analysis. In this case 7 columns are selected where, Bankrupt? variable is intended to be used in classification model as the target with the rest of the columns used as features. In the case of linear regression model Realized sales gross margin is used as the response variable where the rest of the columns are used as independent variables.

Exploring data

In [22]:
```
#check distribution and relationship of features variables (columns) in relation to bankrupt
#and relationship between variables using scatterplots
#combined using pairplot
pair=sb.pairplot(sel_data,hue="Bankrupt?",diag_kind="kde")
pair.fig.set_figwidth(20)
pair.fig.set_figwidth(20)
```


Operating Gross Margin
Cash flow rate
Realized Sales Gross Margin
Bankrupt?
0
1

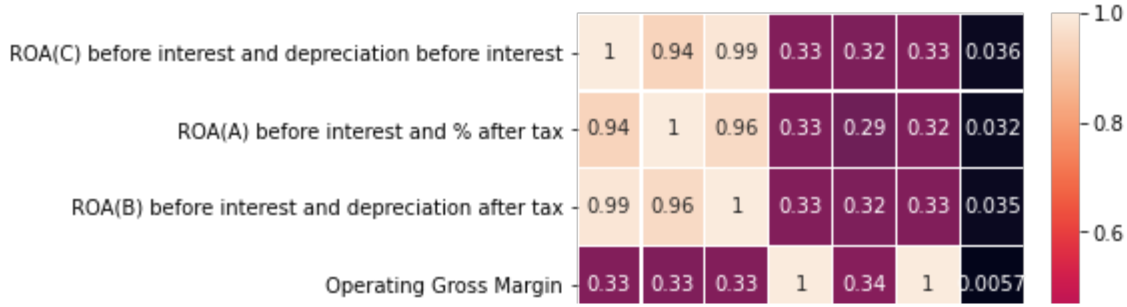
From the density kernel plots for the combined cases i.e 0,1 the four variables selected i.e ROA(C),ROA(A), ROA(B),operating gross margin and realized sales gross margin tend to be normally distributed. Relationship indicated by the scatterplots tend to be linear for some cases for instance ROA(A) and ROA(B) have a strong positive correlation, this implies that increase in ROA(A) increases ROA(B). Where operating gross margin and realized gross margin have a perfect correlation. This can be elaborated by creating a correlation matrix.

In [23]:

```
#correlation matrix between the features/independent variables
nums=sel_data.iloc[:,[1,2,3,4,5,6,7]]
plt.figure(figsize=(5,5))
sb.heatmap(nums.corr(),linewidths=0.2,annot=True)
```

Out[23]:

```
<AxesSubplot:>
```



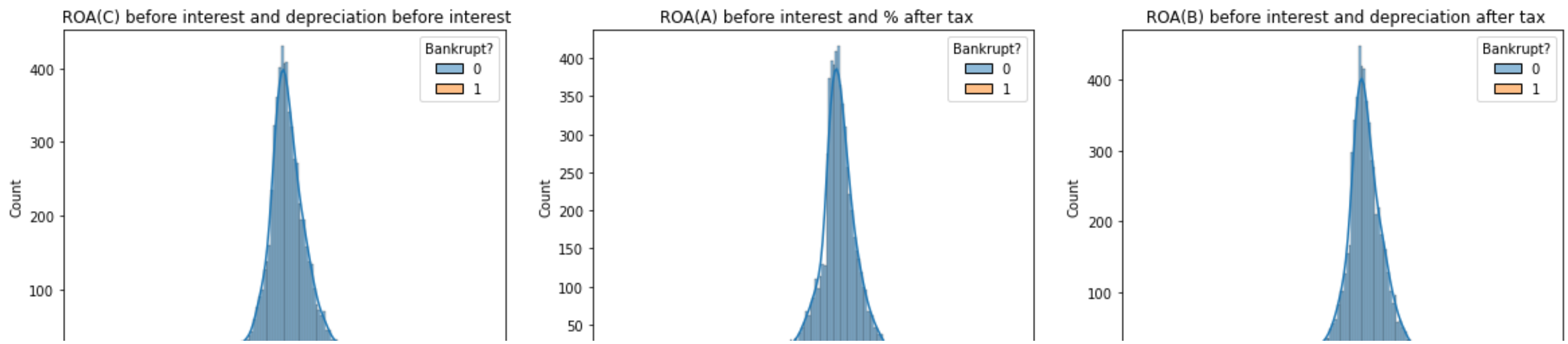
Considering the Pearson's correlation coefficient in this case, coefficients greater than 0.75 indicate a strong positive correlation while those below 0.5 indicate poor positive correlation.

In [24]:

```
#elaborating on distribution using histogram and kernel density
#The code merges the plots into subplots
fig,axes=plt.subplots(2,3,figsize=(20,10))
sb.histplot(ax=axes[0,0],data=sel_data,x=sel_data.columns[1],hue=sel_data.columns[0],kde=True)
axes[0,0].set_title(sel_data.columns[1])
sb.histplot(ax=axes[0,1],data=sel_data,x=sel_data.columns[2],hue=sel_data.columns[0],kde=True)
axes[0,1].set_title(sel_data.columns[2])
sb.histplot(ax=axes[0,2],data=sel_data,x=sel_data.columns[3],hue=sel_data.columns[0],kde=True)
axes[0,2].set_title(sel_data.columns[3])
sb.histplot(ax=axes[1,0],data=sel_data,x=sel_data.columns[4],hue=sel_data.columns[0],kde=True)
sb.histplot(ax=axes[1,1],data=sel_data,x=sel_data.columns[5],hue=sel_data.columns[0],kde=True)
sb.histplot(ax=axes[1,2],data=sel_data,x=sel_data.columns[6],hue=sel_data.columns[0],kde=True)
```

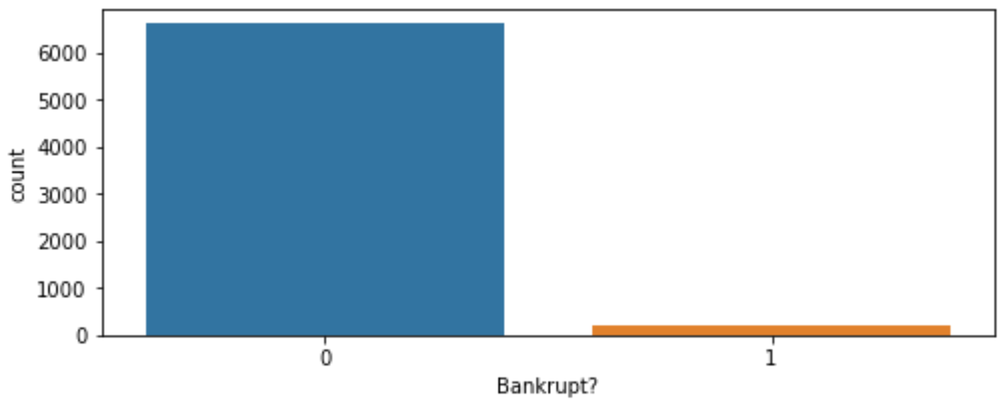
Out[24]:

```
<AxesSubplot:xlabel=' Realized Sales Gross Margin', ylabel='Count'>
```



This plots indicate distributions for each numerical variable, from the view the variables tend to be normally ditributed both indicated by histogram and kernel density plot.

In [25]:
```
#frequency plot for the class labels
plt.figure(figsize=(8,3))
sb.countplot(data=data,x="Bankrupt?")
plt.show()
```



indicates there few cases of bankruptcy compared to none. This cases are below 1000.

Investigating effects of various variables on Bankrupt?

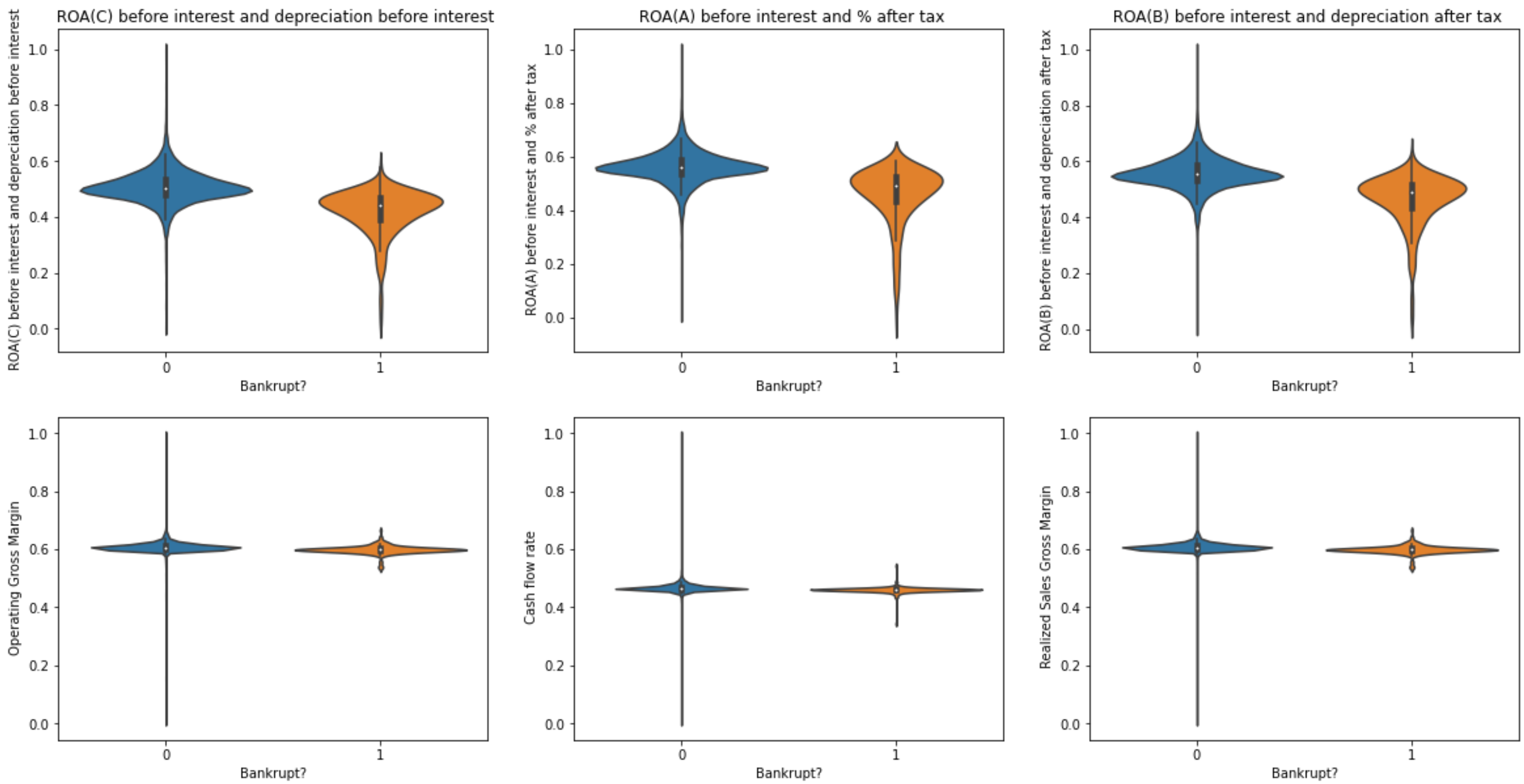
In this case, violinplots are appropriate in comparing effects of selected numerical variables on the two classe in variable Bankrupt?. The intention is to determine if the variables had similar influence on the two states i.e 0 and 1 (not bankrupt and bankrupt).

In [26]:

```
fig,axes=plt.subplots(2,3,figsize=(20,10))
sb.violinplot(ax=axes[0,0],data=sel_data,y=sel_data.columns[1],x=sel_data.columns[0],kde=True)
axes[0,0].set_title(sel_data.columns[1])
sb.violinplot(ax=axes[0,1],data=sel_data,y=sel_data.columns[2],x=sel_data.columns[0],kde=True)
axes[0,1].set_title(sel_data.columns[2])
sb.violinplot(ax=axes[0,2],data=sel_data,y=sel_data.columns[3],x=sel_data.columns[0],kde=True)
axes[0,2].set_title(sel_data.columns[3])
sb.violinplot(ax=axes[1,0],data=sel_data,y=sel_data.columns[4],x=sel_data.columns[0],kde=True)
sb.violinplot(ax=axes[1,1],data=sel_data,y=sel_data.columns[5],x=sel_data.columns[0],kde=True)
sb.violinplot(ax=axes[1,2],data=sel_data,y=sel_data.columns[6],x=sel_data.columns[0],kde=True)
```

Out[26]:

```
<AxesSubplot:xlabel='Bankrupt?', ylabel=' Realized Sales Gross Margin'>
```



The plots indicate the distribution of numerical variables from the selected data for each category based on the kernel density estimation. Where also they indicate the median point (white marker). The rectangle indicates the first and third quartiles. The plots indicates that operating gross margin and realised sales gross margin had similar effect on variable Bankrupt, where the other variavbles had different effect.

In [27]:
```
#grouping the selected data based on the two classes
#obtain the mean and standard deviation of the variables
sel_data.groupby(["Bankrupt?"]).agg({sel_data.columns[1]:['mean','std'],
                                    sel_data.columns[2]:['mean','std'],
                                    sel_data.columns[3]:['mean','std'],
                                    sel_data.columns[4]:['mean','std'],
                                    sel_data.columns[5]:['mean','std'],
                                    sel_data.columns[6]:['mean','std']})
```

Out[27]:

	ROA(C) before interest and depreciation before interest		ROA(A) before interest and % after tax		ROA(B) before interest and depreciation after tax		Operating Gross Margin		Cash flow rate		Realized Sales Gross Margin	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
Bankrupt?												
0	0.508069	0.057694	0.562015	0.060898	0.556659	0.057864	0.608257	0.016920	0.467656	0.017149	0.608237	0.016903
1	0.418503	0.081068	0.456947	0.107674	0.461483	0.091825	0.598670	0.014595	0.460681	0.011266	0.598717	0.014583

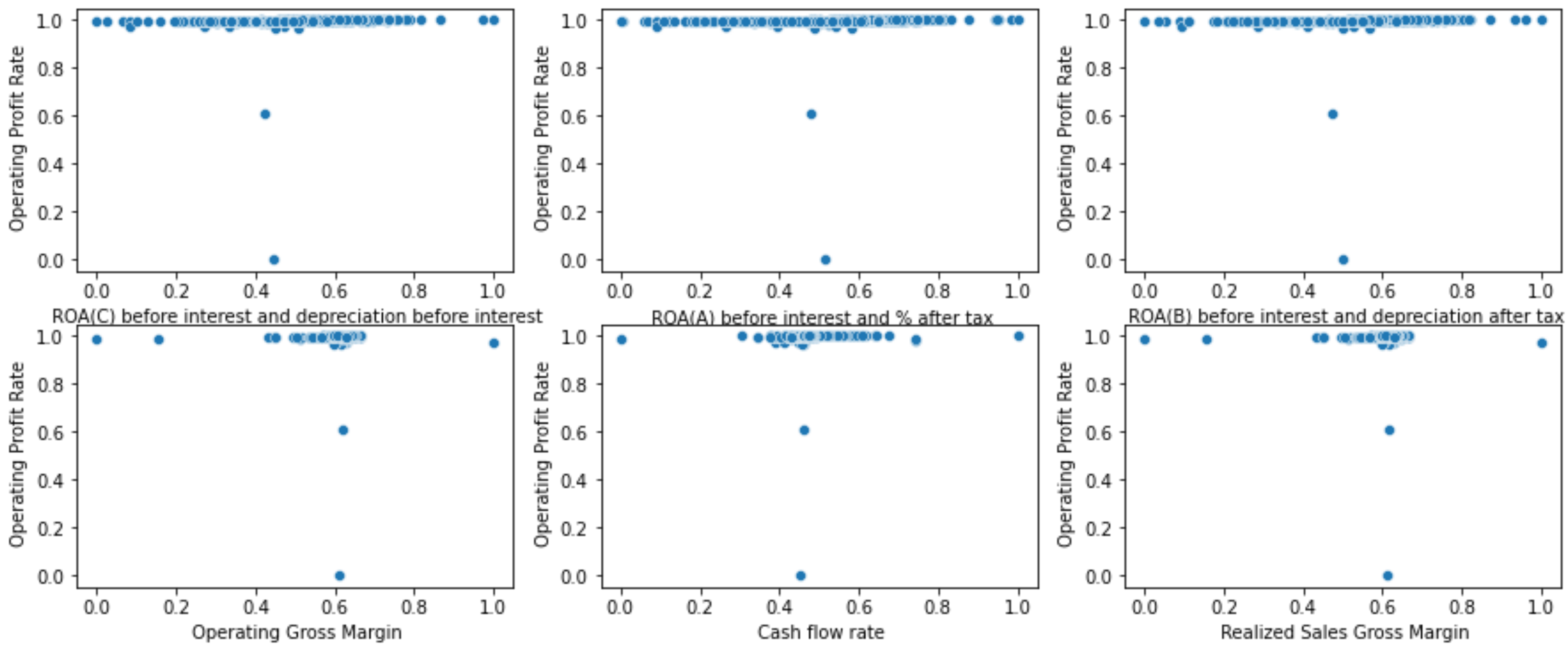
The selected data is grouped based on the variable Bankrupt?, which cab be considered binary variable. summary statistic, where for this case mean and standard deviation is computed. From this, average and degree of spread is defined for each class. For instance the average cashflow rate for cases considered not bankrupt (0) is 0.467 while the standard deviation is 0.017 while for cases considered bankrupt the average cashflow is 0.460 and standard deviation is 0.01. This indicate that there is no much difference in how the cash flow rate is spread since the values for two classes are close. For the two classes cash flow rate is clustered around 0.46.

Linear regression

Consider using operating profit rate as the dependent variable while the rest variables as independent variables. Investigate the linear relationship between independent variables and dependent variable

In [28]:
```
#using scatterplots
fig,axes=plt.subplots(2,3,figsize=(15,6))
sb.scatterplot(ax=axes[0,0],data=sel_data,x=sel_data[sel_data.columns[1]],y=sel_data[sel_data.columns[7]])
sb.scatterplot(ax=axes[0,1],data=sel_data,x=sel_data[sel_data.columns[2]],y=sel_data[sel_data.columns[7]])
sb.scatterplot(ax=axes[0,2],data=sel_data,x=sel_data[sel_data.columns[3]],y=sel_data[sel_data.columns[7]])
sb.scatterplot(ax=axes[1,0],data=sel_data,x=sel_data[sel_data.columns[4]],y=sel_data[sel_data.columns[7]])
sb.scatterplot(ax=axes[1,1],data=sel_data,x=sel_data[sel_data.columns[5]],y=sel_data[sel_data.columns[7]])
sb.scatterplot(ax=axes[1,2],data=sel_data,x=sel_data[sel_data.columns[6]],y=sel_data[sel_data.columns[7]])
```

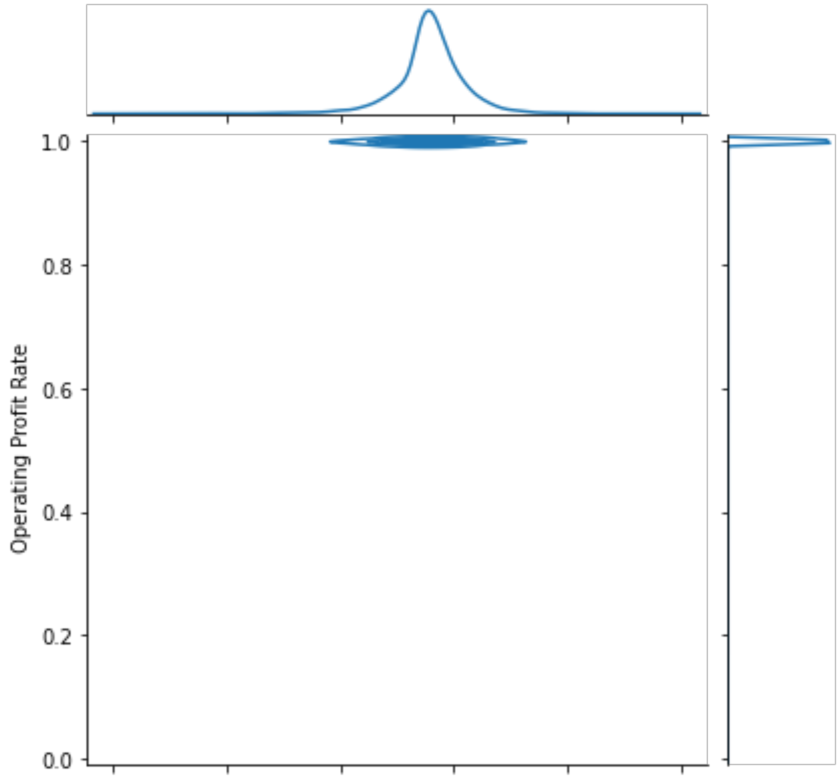
Out[28]:
```
<AxesSubplot:xlabel=' Realized Sales Gross Margin', ylabel=' Operating Profit Rate'>
```



In [ ]:
```
#jointplot illustrating distribution and relationship between dependent and independent variable.
sb.jointplot(data=sel_data,x=sel_data[sel_data.columns[1]],y=sel_data[sel_data.columns[7]])
```

In [14]:
```
#this indicates distribution based on kernel density function
sb.jointplot(data=sel_data,x=sel_data[sel_data.columns[2]],y=sel_data[sel_data.columns[7]],kind="kde")
```

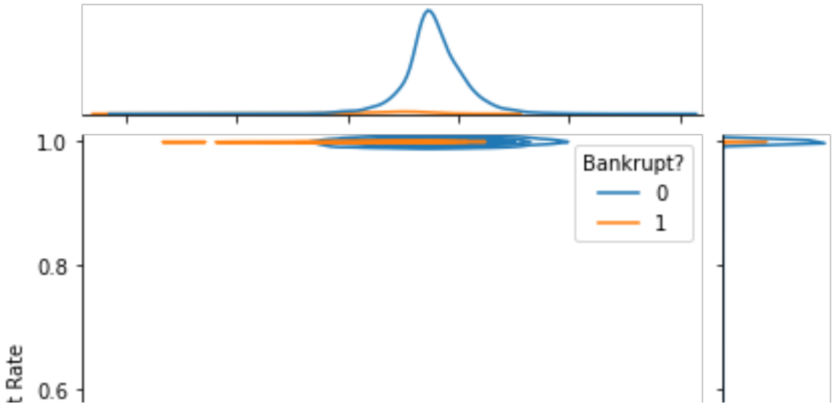
Out[14]:
```
<seaborn.axisgrid.JointGrid at 0x7f7bfa0bee50>
```



This indicates that both the dependent and independent variables are normally distributed.

In [15]:
```
sb.jointplot(data=sel_data,x=sel_data[sel_data.columns[3]],y=sel_data[sel_data.columns[7]],hue=sel_data[sel_
```

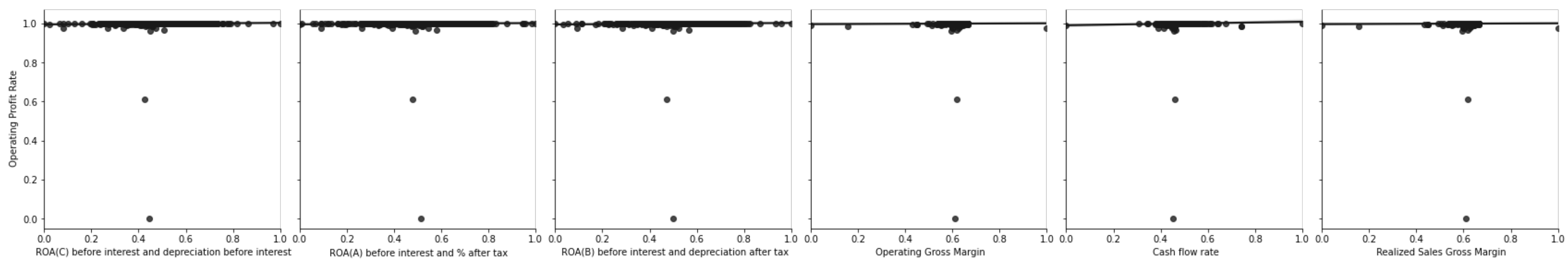
Out[15]:
```
<seaborn.axisgrid.JointGrid at 0x7f7bca221c40>
```



Based on the two classes, distribution for each variable in each category can be viewed.

In [29]:
```
#simple linear relationship where realized sales gross margin is dependent.
cols=[sel_data.columns[1],sel_data.columns[2],sel_data.columns[3],sel_data.columns[4],sel_data.columns[5],se
pair = sb.PairGrid(sel_data, y_vars=[sel_data.columns[7]], x_vars=cols, height=4)
pair.map(sb.regplot, color=".1")
```

Out[29]: <seaborn.axisgrid.PairGrid at 0x7fdb31ea58b0>



The first four variables fits well, which indicates a strong linear relationship. Therefore this variables may be considered strong determiners of realized sales gross margin.

Logistic regression

The model intends to classify the features that define the variable bankrupt? as either 0 or 1 which indicate not bankrupt and bankrupt respectively

SPLITTING DATA

As previous data is splitted into train and test parts at a ratio of 0.75 to 0.25

In [30]:
```
class_=sel_data[sel_data.columns[0]]
feat=sel_data.iloc[:,[1,2,3,4,5,6]]
x_tr,x_te,y_tr,y_te=train_test_split(feat,class_,test_size=0.25,random_state=1)
```

FITTING THE MODEL

In [31]:
```
log_model=LogisticRegression(random_state=1)
log_model.fit(x_tr,y_tr)
```

Out[31]:
```
LogisticRegression(random_state=1)
```

In [32]:
```
coef=log_model.coef_
coef
coef_df=pd.DataFrame(coef[0],feat.columns,columns=["coefficients"])
coef_df
```

Out[32]:

	coefficients
ROA(C) before interest and depreciation before interest	-3.885556
ROA(A) before interest and % after tax	-4.904621
ROA(B) before interest and depreciation after tax	-4.086118
Operating Gross Margin	-0.807530
Cash flow rate	-0.150970
Realized Sales Gross Margin	-0.807424

odds ratio

In [33]:
```
odds_df=pd.DataFrame(np.exp(coef[0]),feat.columns,columns=["odds ratio"])
odds_df
```

Out[33]:

	odds ratio
ROA(C) before interest and depreciation before interest	0.020536
ROA(A) before interest and % after tax	0.007412
ROA(B) before interest and depreciation after tax	0.016804

	odds ratio
Operating Gross Margin	0.445958
Cash flow rate	0.859873

EVALUATION

In this section, utilise the classification report and confusion metric to determine how well the model performs.

In [34]:
```
preds=log_model.predict(x_te)
print(classification_report(y_te,preds))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      1644
           1       0.14      0.02      0.03        61

    accuracy                           0.96      1705
   macro avg       0.55      0.51      0.50      1705
weighted avg       0.94      0.96      0.95      1705
```

The model has an acuracy value of 0.96 which can be translated to 96% accuracy which can be considered excellent. Precision indicate how well each class was predicted, from the values class 0 were predicted with 0.96 accuracy while class 1 was predicted at an accuracy of 0.14

In [35]:
```
sb.heatmap(confusion_matrix(y_te,preds),annot=True,linewidths=.2)
```

Out[35]:
```
<AxesSubplot:>
```

The confusion matrix indicates how each class was predicted. For instance 1600 class 0 were predicted 0 whre it was actually true while 6 class 0 were predicted 0 where it was actually class 1.

KNN model

Same as logistic regression this model is used in classification, therefore utilise in classification of the two classes in this case.

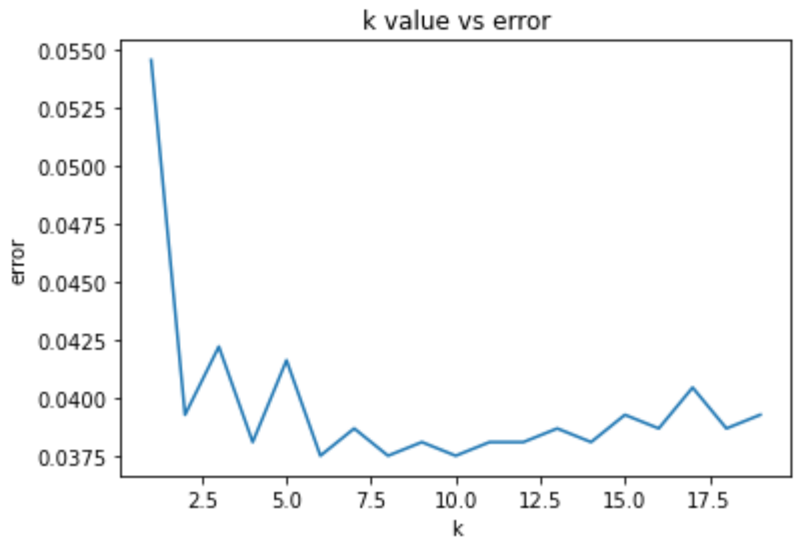
OPTIMISING K-VALUE

In this case the objective is to find the value of k that minimises the error in prediction of new classes. Using the for loop, assign k to values between 0-20 until the value that results to minimal error is obtained

In [36]:

```
#choosing the k-value
error=[]
ks=[]
for i in range(1,20):
    mod1=KNeighborsClassifier(n_neighbors=i)
    mod1.fit(x_tr,y_tr)
    pred=mod1.predict(x_te)
    er=np.mean(pred!= y_te)
    error.append(er)
    ks.append(i)

plt.plot(ks,error)
plt.xlabel("k")
plt.ylabel("error")
plt.title("k value vs error")
plt.show()
```



The plot indicates that the error is minimal when the value of k=10, error reduces as the value of k increases. Error is minimal at k=10

FITTING THE MODEL

fitting the model with k=10

In [37]:
```
mod=KNeighborsClassifier(n_neighbors=10)
mod.fit(x_tr,y_tr)
```

Out[37]:
```
KNeighborsClassifier(n_neighbors=10)
```

EVALUATING THE MODEL

In [38]:
```
preds=mod.predict(x_te)
print(classification_report(y_te,preds))
```

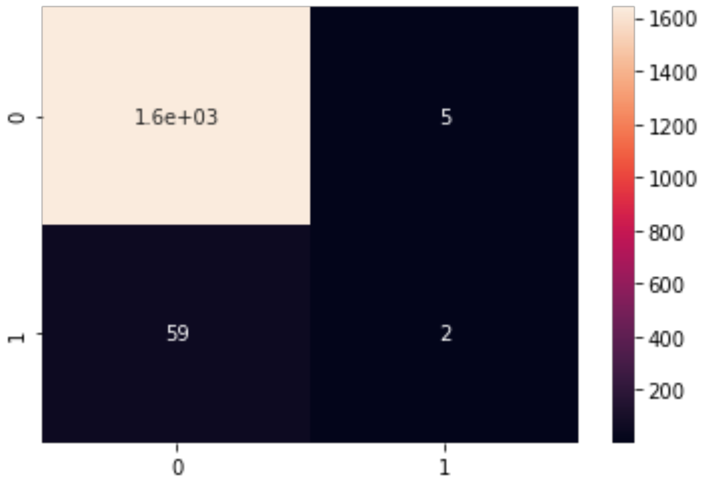
```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98      1644
           1       0.29      0.03      0.06        61

    accuracy                           0.96      1705
   macro avg       0.63      0.51      0.52      1705
weighted avg       0.94      0.96      0.95      1705
```

In [39]:
```
sb.heatmap(confusion_matrix(y_te,preds),annot=True)
```

Out[39]: <AxesSubplot:>



The model performs with an accuracy score of 0.97 which can be considered excellent. 1600 features are predicted as class 0 while it is actually true while 5 are predicted as class 0 while it is actually class 1.

Conclusion

From the three models, the data suits well the models, where minimal cleaning of data was performed. This suggests that selection of model for your data is of importance, considering that there are wide varieties of machine learning model that can be utilised. This models helps in deriving insights from the data and understanding aspects related to the data. This plays a key role in decision making in all sectors of human life. It can be concluded that the 6 features used in the classification models are key determiners of bankruptcy where they inluences heavly the results as seen from the model performance. In linear regression model the 5 independent variables hugely affect the outcome of realized sales gross margin. Considering that the data is obtained from economic/ finance sector, this results can be utilised in making decision which may influence the form and performace of this sector and other related sectors.

Decision Tree Classifier

Training the model using previously splitted data.

In [40]:
```
#training a decision tree classifier
dtc=DecisionTreeClassifier(random_state=1)
dtc.fit(x_tr,y_tr)
t_pred=dtc.predict(x_te)
t_pred
```

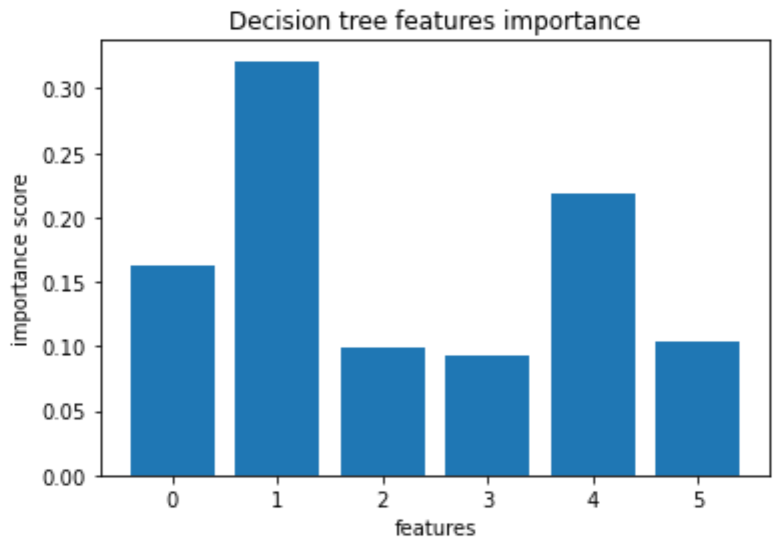
Out[40]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

Determining feature importance

In [41]:
```
feat_imp=dtc.feature_importances_
for feature,importance in enumerate(feat_imp):
    print("feature: ",feature," score: ",importance,"feature name: ",x_te.columns[feature])
```

```
feature:  0  score:  0.16314873022483772 feature name:   ROA(C) before interest and depreciation before inte
rest
feature:  1  score:  0.3213941748637301 feature name:   ROA(A) before interest and % after tax
feature:  2  score:  0.0997431656775361 feature name:   ROA(B) before interest and depreciation after tax
feature:  3  score:  0.09343063552242636 feature name:   Operating Gross Margin
feature:  4  score:  0.21904318905541256 feature name:   Cash flow rate
feature:  5  score:  0.10324010465605714 feature name:   Realized Sales Gross Margin
```

In [42]:
```
#displaying the feature importance for each attribute using a bar plot
plt.bar([i for i in range(len(feat_imp))],feat_imp)
plt.xlabel("features")
plt.ylabel("importance score")
plt.title("Decision tree features importance")
plt.show()
```



the bar plot indicates that, ROA(A) has the highest importance.

Evaluating the tree

In [43]:
```
accuracy_score(y_te,t_pred)
```

Out[43]: 0.9483870967741935

In [44]:
```
print(classification_report(y_te,t_pred))
```

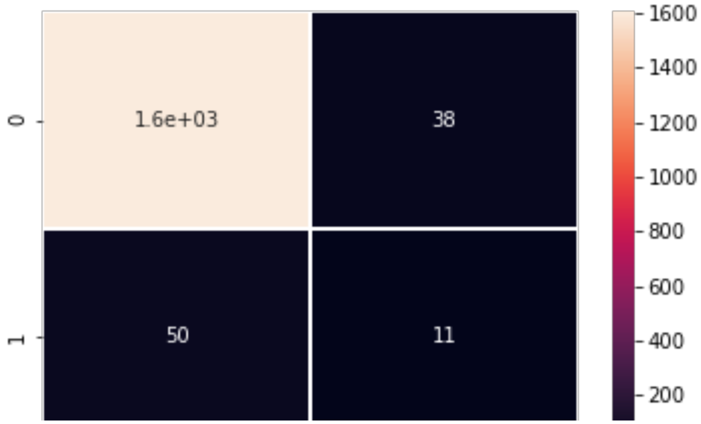
```
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      1644
           1       0.22      0.18      0.20        61

    accuracy                           0.95      1705
   macro avg       0.60      0.58      0.59      1705
weighted avg       0.94      0.95      0.95      1705
```

In [45]:
```
sb.heatmap(confusion_matrix(y_te,t_pred),annot=True,linewidth=0.5)
```

Out[45]: <AxesSubplot:>



TUNING HYPERPARAMETERS

In [46]:
```
tuned_dtc=DecisionTreeClassifier(max_depth=1,min_samples_split=3,min_samples_leaf=2)
tuned_dtc.fit(x_tr,y_tr)
tuned_pred=tuned_dtc.predict(x_te)
tuned_pred
```

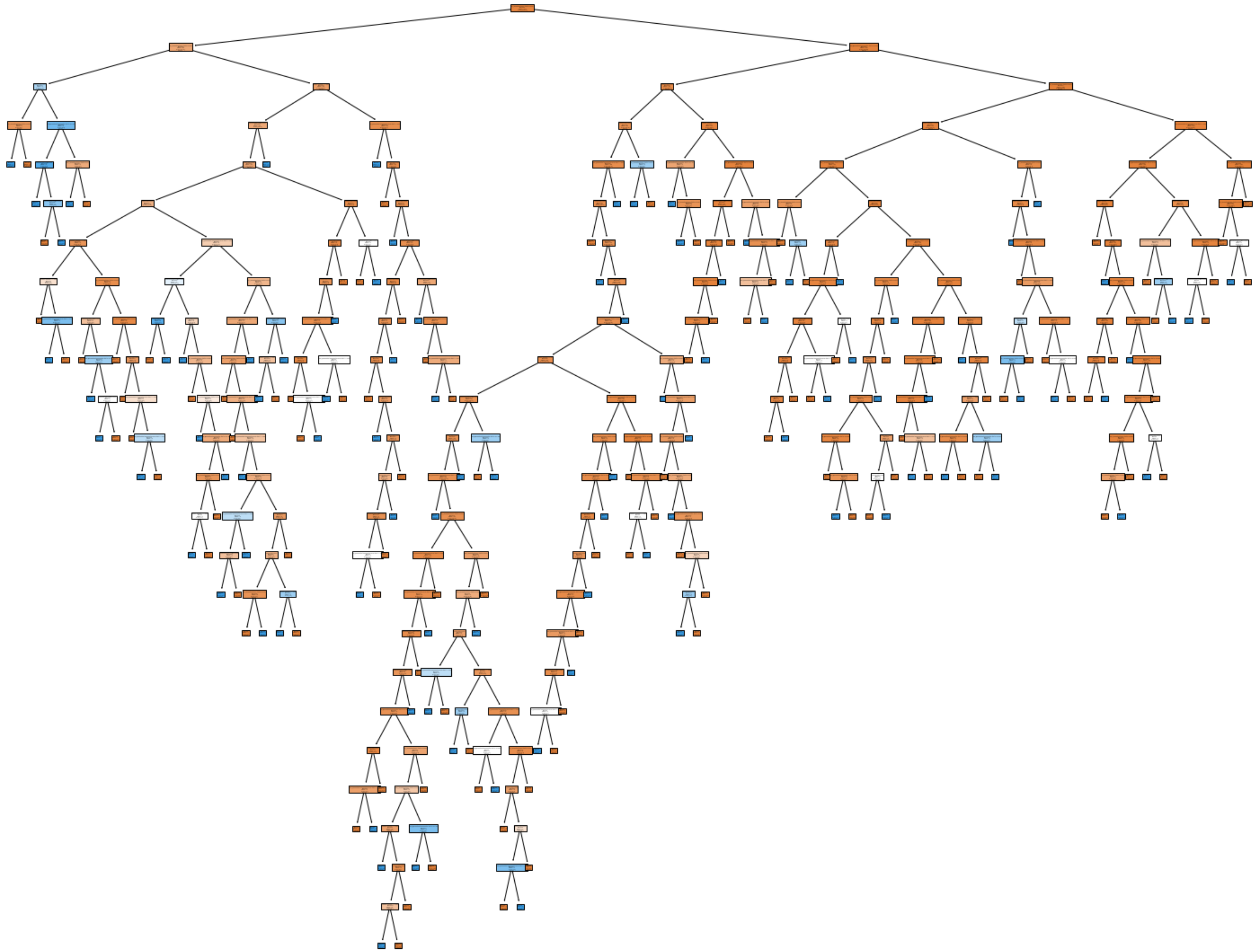
Out[46]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [34]:
```
accuracy_score(y_te,tuned_pred)
```

Out[34]:
```
0.9642228739002933
```

Visualizing Tree

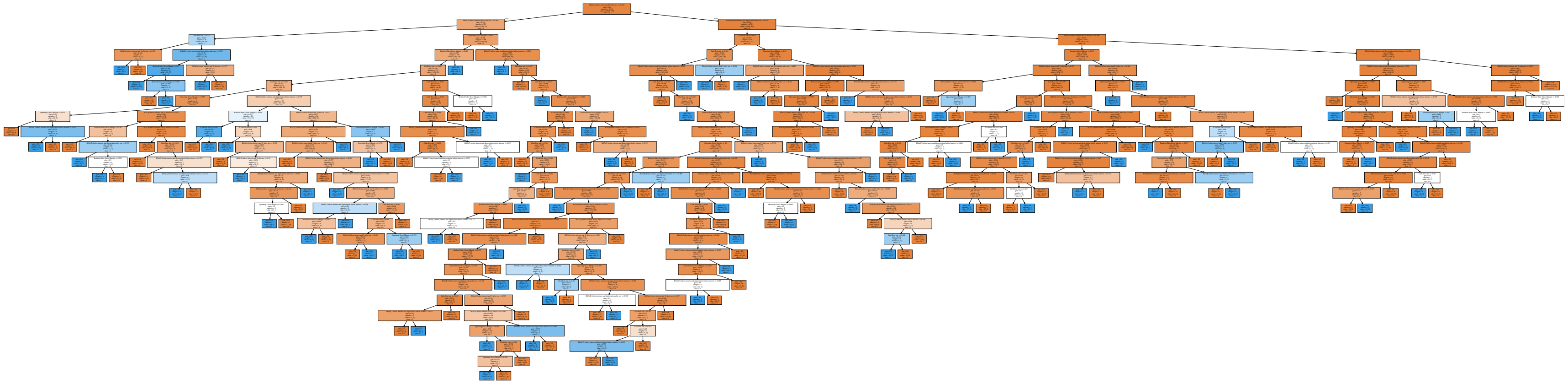
In [47]:
```
from sklearn import tree
feature_names=[i for i in x_te.columns]
fig = plt.figure(figsize=(25,20))
tr_ = tree.plot_tree(dtc,
                   feature_names=feature_names,
                   class_names=["0","1"],
                   filled=True)
```



In [49]:
```
import graphviz
dot_data = tree.export_graphviz(dtc, out_file=None,
                                feature_names=feature_names,
                                class_names=["0","1"],
                                filled=True)

# Draw graph
tree_ = graphviz.Source(dot_data, format="png")
tree_
```

Out[49]:



In [50]:
```
#saving the tree in png file
tree_.render("bankruptcy")
```

Out[50]: 'bankruptcy.png'

Random Forests Classifier

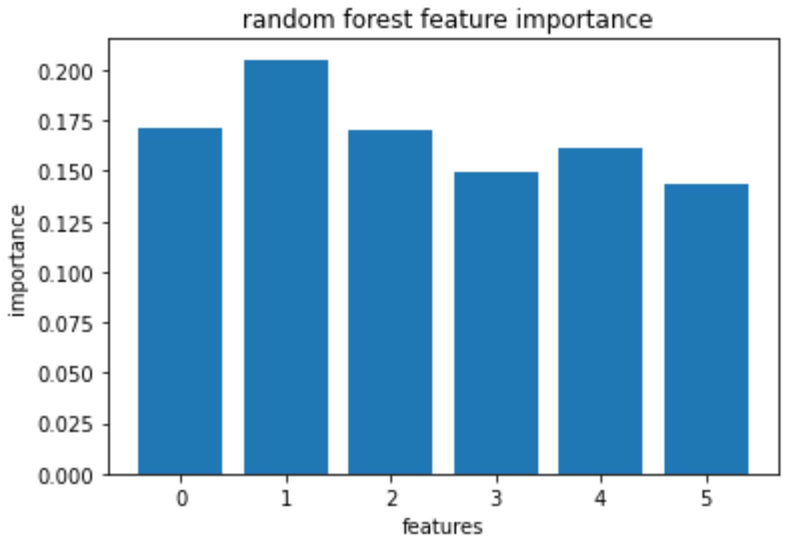
In [51]:
```
rfc=RandomForestClassifier(random_state=1)
rfc.fit(x_tr,y_tr)
rfc_pred=rfc.predict(x_te)
rfc_pred
```

Out[51]: array([0, 0, 0, ..., 0, 0, 0])

In [52]:
```
accuracy_score(y_te,rfc_pred)
```

Out[52]: 0.9601173020527859

In [53]:
```
#feature importance bar plot
plt.bar([i for i in range(len(rfc.feature_importances_))],rfc.feature_importances_)
plt.xlabel("features")
plt.ylabel("importance")
plt.title(" random forest feature importance")
plt.show()
```



Similar to decision tree classifier ROA(A) can be considered the most important feature.

HYPERPARAMETERS

Consider setting some random forest classifier parameters to some random value, where effect of the parameters can be determined.

In [54]:
```
rfc_tuned=RandomForestClassifier(random_state=1,n_estimators=100,max_depth=1,min_samples_split=3,min_samples
rfc_tuned.fit(x_tr,y_tr)
rfc_t_pred=rfc_tuned.predict(x_te)
rfc_t_pred
```

Out[54]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [55]:
```
accuracy_score(y_te,rfc_t_pred)
```

Out[55]:
```
0.9642228739002933
```

In [56]:
```
import warnings
warnings.filterwarnings('ignore')
print(classification_report(y_te,rfc_t_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      1644
           1       0.00      0.00      0.00        61

    accuracy                           0.96      1705
   macro avg       0.48      0.50      0.49      1705
weighted avg       0.93      0.96      0.95      1705
```

In [ ]: