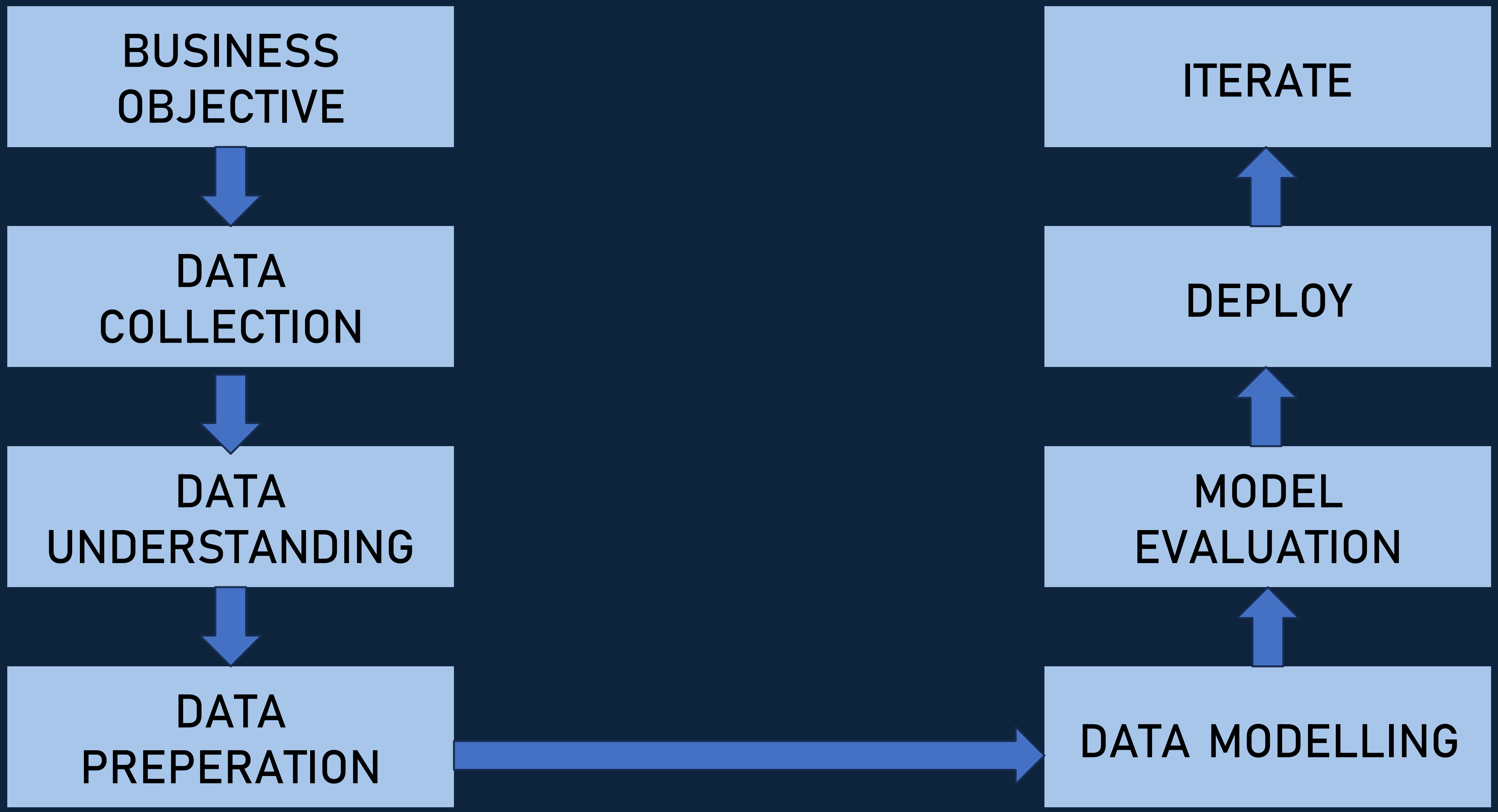
BANKRUPTCY PREVENTION

CLASSIFICATION
PROBLEM

BUSINESS OBJECTIVE

This is a classification project, since the variable to predict is binary (bankruptcy or non-bankruptcy). The goal here is to model the probability that a business goes bankrupt from different features.

PROJECT FLOW



EXPLORATORY DATA ANALYSIS

01 . Importing the necessary libraries for further analysis

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings('ignore')
```

02. Importing the DataSet - Bank

```
In [2]: bank = pd.read_csv("bank.csv", sep = ';', header = 0)
```

```
In [3]: bank
```

Out[3]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_risk	class
0	0.5	1.0	0.0	0.0	0.0	0.5	bankruptcy
1	0.0	1.0	0.0	0.0	0.0	1.0	bankruptcy
2	1.0	0.0	0.0	0.0	0.0	1.0	bankruptcy
3	0.5	0.0	0.0	0.5	0.0	1.0	bankruptcy
4	1.0	1.0	0.0	0.0	0.0	1.0	bankruptcy
...	...	...	...	...	...	...	...
245	0.0	1.0	1.0	1.0	1.0	1.0	non-bankruptcy
246	1.0	1.0	0.5	1.0	1.0	0.0	non-bankruptcy
247	0.0	1.0	1.0	0.5	0.5	0.0	non-bankruptcy
248	1.0	0.0	0.5	1.0	0.5	0.0	non-bankruptcy
249	1.0	0.0	0.5	0.5	1.0	1.0	non-bankruptcy

250 rows × 7 columns

03. This data has 250 entries with 7 columns. The columns represent different aspects like industrial risk and credibility. These factors influence a category called "class," helping us understand how different factors affect business results.

```
In [5]: bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 7 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   industrial_risk         250 non-null    float64
 1    management_risk        250 non-null    float64
 2    financial_flexibility  250 non-null    float64
 3    credibility            250 non-null    float64
 4    competitiveness        250 non-null    float64
 5    operating_risk         250 non-null    float64
 6    class                  250 non-null    object
dtypes: float64(6), object(1)
memory usage: 13.8+ KB
```

```
In [7]: bank.shape
Out[7]: (250, 7)
```

04. The data is complete, with no missing values. Every aspect, like industrial risk and credibility, has proper information. This ensures our analysis is accurate and dependable for making conclusions.

```
In [11]: bank.isnull().sum()

Out[11]: industrial_risk          0
          management_risk         0
          financial_flexibility   0
          credibility             0
          competitiveness         0
          operating_risk          0
          class                   0
         dtype: int64
```

04. The columns show things like how risky a business is and how trustworthy it seems, rated from 0 to 1. These ratings help predict if a company could go bankrupt, giving us an idea of how stable the business is.

In [12]:
```
bank_new = bank.iloc[:,:]
bank_new
industrial_risk
```

Out[12]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_risk	class
0	0.5	1.0	0.0	0.0	0.0	0.5	bankruptcy
1	0.0	1.0	0.0	0.0	0.0	1.0	bankruptcy
2	1.0	0.0	0.0	0.0	0.0	1.0	bankruptcy
3	0.5	0.0	0.0	0.5	0.0	1.0	bankruptcy
4	1.0	1.0	0.0	0.0	0.0	1.0	bankruptcy
...	...	...	...	...	...	...	...
245	0.0	1.0	1.0	1.0	1.0	1.0	non-bankruptcy
246	1.0	1.0	0.5	1.0	1.0	0.0	non-bankruptcy
247	0.0	1.0	1.0	0.5	0.5	0.0	non-bankruptcy
248	1.0	0.0	0.5	1.0	0.5	0.0	non-bankruptcy
249	1.0	0.0	0.5	0.5	1.0	1.0	non-bankruptcy

250 rows × 7 columns

04. A new column called "class_yn" is created with all entries set to 1. This doesn't alter the main data but adds a new way to categorize, making it easier to analyze and classify things in certain ways.

In [13]:
```
bank_new["class_yn"] = 1
bank_new
```

Out[13]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_risk	class	class_yn
0	0.5	1.0	0.0	0.0	0.0	0.5	bankruptcy	1
1	0.0	1.0	0.0	0.0	0.0	1.0	bankruptcy	1
2	1.0	0.0	0.0	0.0	0.0	1.0	bankruptcy	1
3	0.5	0.0	0.0	0.5	0.0	1.0	bankruptcy	1
4	1.0	1.0	0.0	0.0	0.0	1.0	bankruptcy	1
...	...	...	...	...	...	...	...	...
245	0.0	1.0	1.0	1.0	1.0	1.0	non-bankruptcy	1
246	1.0	1.0	0.5	1.0	1.0	0.0	non-bankruptcy	1
247	0.0	1.0	1.0	0.5	0.5	0.0	non-bankruptcy	1
248	1.0	0.0	0.5	1.0	0.5	0.0	non-bankruptcy	1
249	1.0	0.0	0.5	0.5	1.0	1.0	non-bankruptcy	1

250 rows × 8 columns

04. The way we represent business outcomes has changed. Now, 0 means bankruptcy, and 1 means non-bankruptcy. So, companies labeled as "bankruptcy" before are now marked as 0, making it easier to understand and analyze the data for better decision-making.

In [13]:
```
bank_new["class_yn"] = 1
bank_new
```

Out[13]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_risk	class	class_yn
0	0.5	1.0	0.0	0.0	0.0	0.5	bankruptcy	1
1	0.0	1.0	0.0	0.0	0.0	1.0	bankruptcy	1
2	1.0	0.0	0.0	0.0	0.0	1.0	bankruptcy	1
3	0.5	0.0	0.0	0.5	0.0	1.0	bankruptcy	1
4	1.0	1.0	0.0	0.0	0.0	1.0	bankruptcy	1
...	...	...	...	...	...	...	...	...
245	0.0	1.0	1.0	1.0	1.0	1.0	non-bankruptcy	1
246	1.0	1.0	0.5	1.0	1.0	0.0	non-bankruptcy	1
247	0.0	1.0	1.0	0.5	0.5	0.0	non-bankruptcy	1
248	1.0	0.0	0.5	1.0	0.5	0.0	non-bankruptcy	1
249	1.0	0.0	0.5	0.5	1.0	1.0	non-bankruptcy	1

250 rows × 8 columns

04. We removed the 'class' column, keeping only numbers and the 'class_yn' column. This simplifies our data, helping us focus on important factors for predicting bankruptcy. Now, it's ready for detailed analysis and predictions.

In [17]:
```
bank_new.drop(' class', inplace = True, axis =1)
bank_new.head()
```

Out[17]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_risk	class_yn
0	0.5	1.0	0.0	0.0	0.0	0.5	1
1	0.0	1.0	0.0	0.0	0.0	1.0	1
2	1.0	0.0	0.0	0.0	0.0	1.0	1
3	0.5	0.0	0.0	0.5	0.0	1.0	1
4	1.0	1.0	0.0	0.0	0.0	1.0	1

Value Counts Check

```
df["industrial_risk"].value_counts()

1.0    89
0.5    81
0.0    80
Name: industrial_risk, dtype: int64
```

```
df[" management_risk"].value_counts()

1.0    119
0.5     69
0.0     62
Name:  management_risk, dtype: int64
```

```
df[" financial_flexibility"].value_counts()

0.0    119
0.5     74
1.0     57
Name:  financial_flexibility, dtype: int64
```

```
In [16]: df[" credibility"].value_counts()

Out[16]: 0.0    94
         1.0    79
         0.5    77
         Name:  credibility, dtype: int64
```

```
In [17]: df[" competitiveness"].value_counts()

Out[17]: 0.0    103
         1.0     91
         0.5     56
         Name:  competitiveness, dtype: int64
```

```
In [18]: df[" operating_risk"].value_counts()

Out[18]: 1.0    114
         0.0     79
         0.5     57
         Name:  operating_risk, dtype: int64
```

Rename the columns

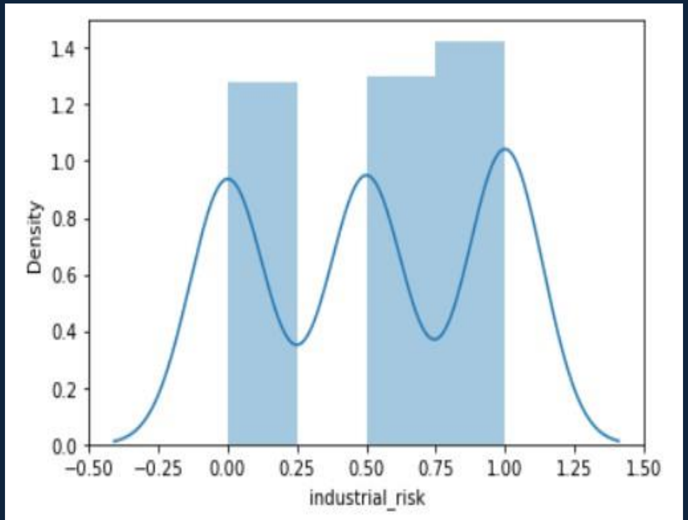
```
In [19]: df1 = df.rename({' industrial_risk': ' Industrial',
                           ' management_risk': 'Management',
                           ' financial_flexibility': 'Financial',
                           ' credibility': 'Credibility',
                           ' competitiveness': 'Competitive',
                           ' operating_risk': 'Operational',
                          ' class' : 'class'},
                          axis=1)
```

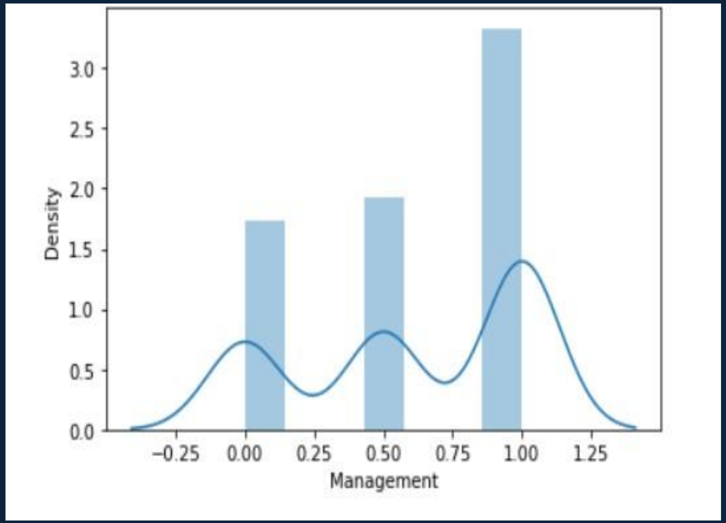
```
In [20]: df1.head()
```

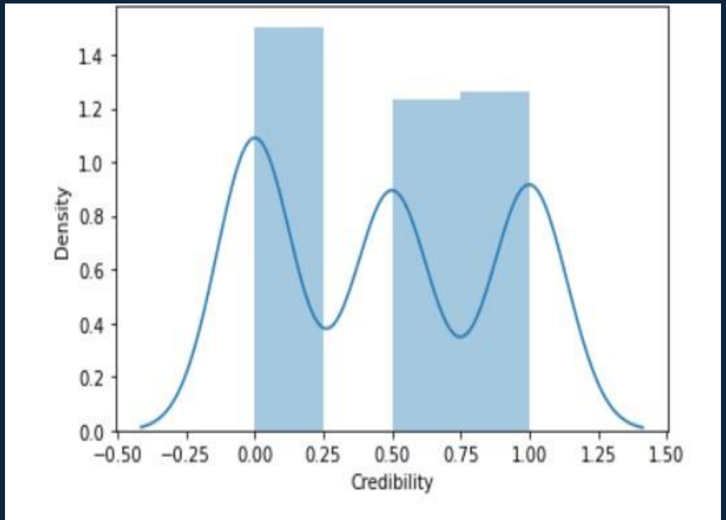
Out[20]:

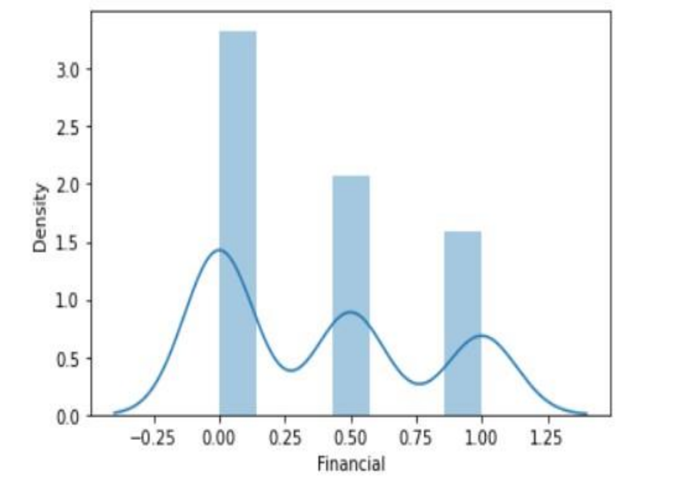
	industrial_risk	Management	Financial	Credibility	Competitive	Operational	class
0	0.5	1.0	0.0	0.0	0.0	0.5	bankruptcy
1	0.0	1.0	0.0	0.0	0.0	1.0	bankruptcy
2	1.0	0.0	0.0	0.0	0.0	1.0	bankruptcy
3	0.5	0.0	0.0	0.5	0.0	1.0	bankruptcy
4	1.0	1.0	0.0	0.0	0.0	1.0	bankruptcy

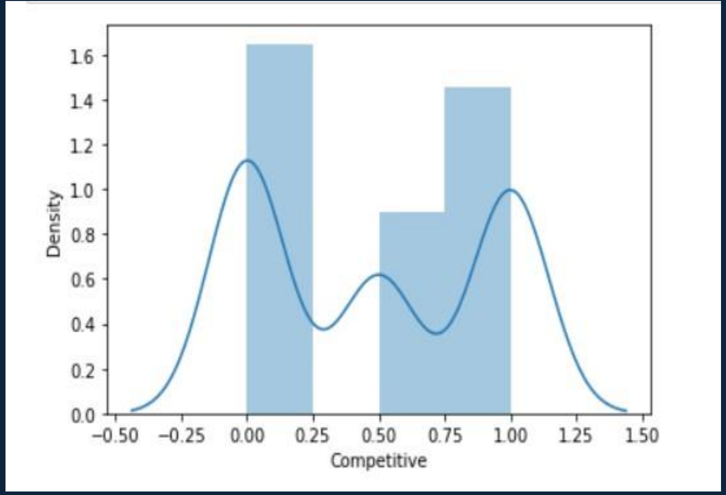
Distance Plot

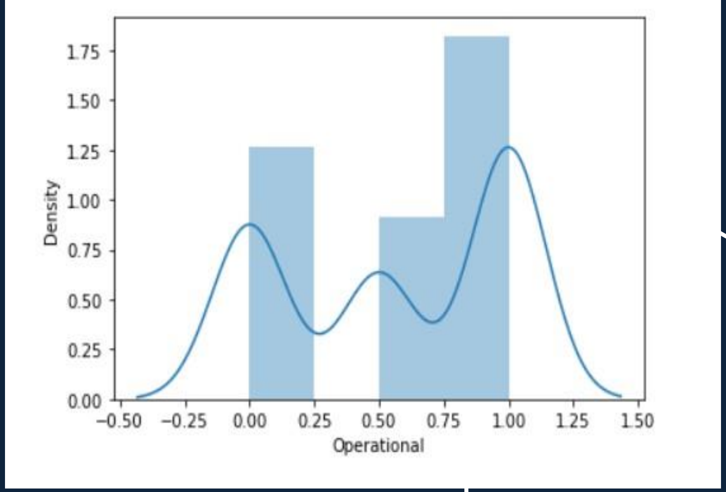




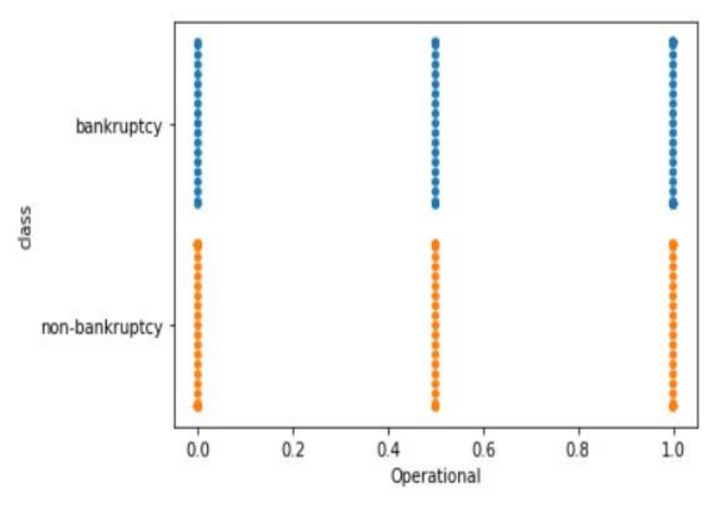




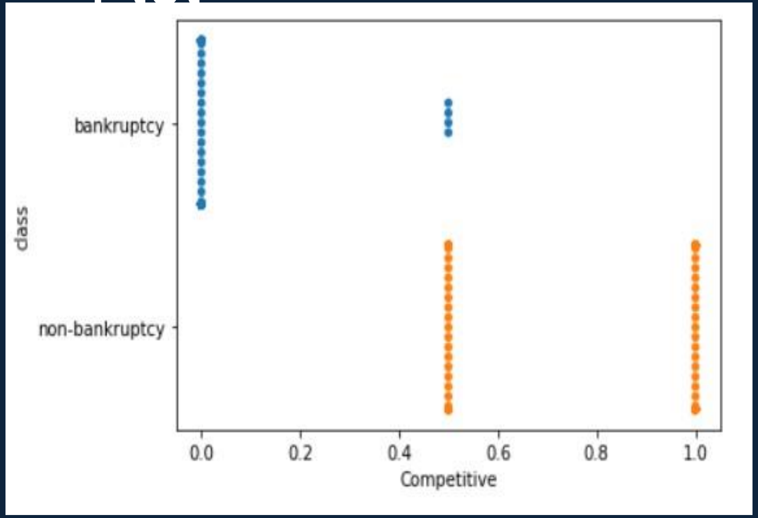




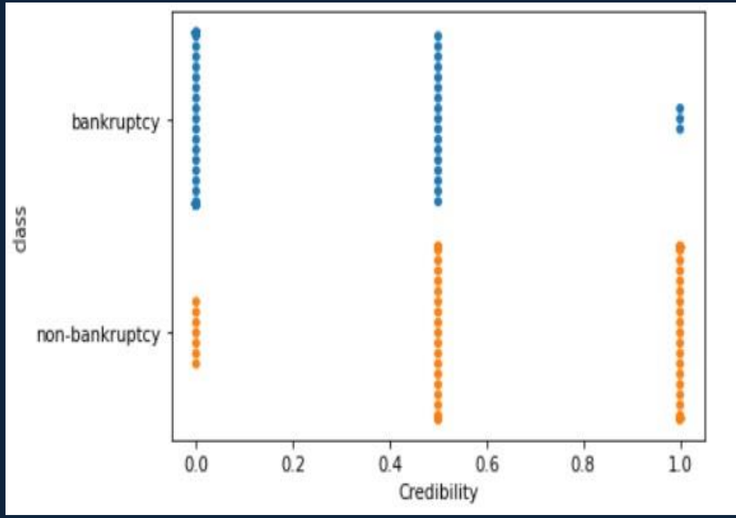
Swarm Plot



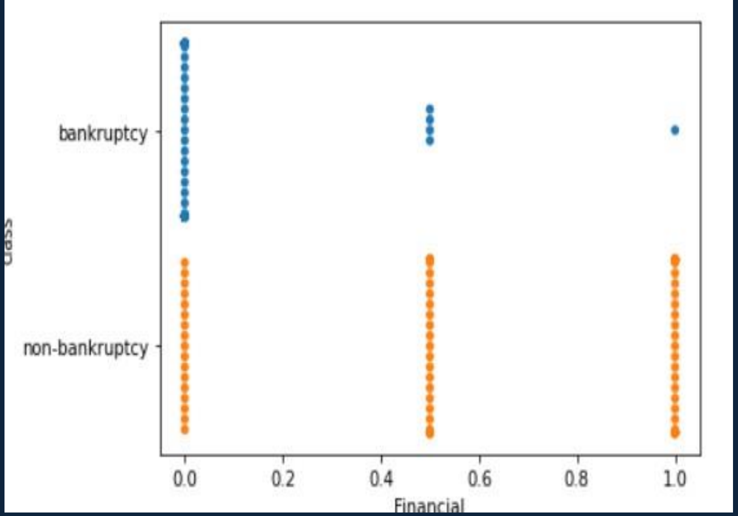
class
bankruptcy
non-bankruptcy
0.0
0.2
0.4
0.6
0.8
1.0
Operational



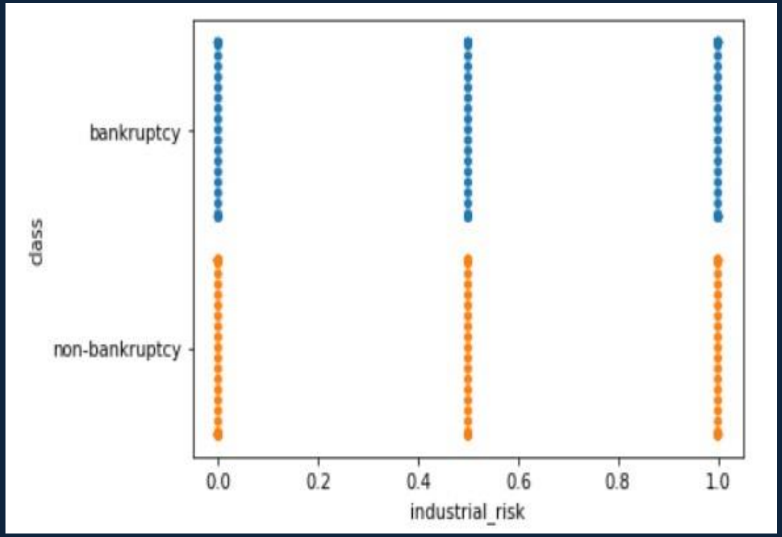
class
bankruptcy
non-bankruptcy
0.0
0.2
0.4
0.6
0.8
1.0
Competitive



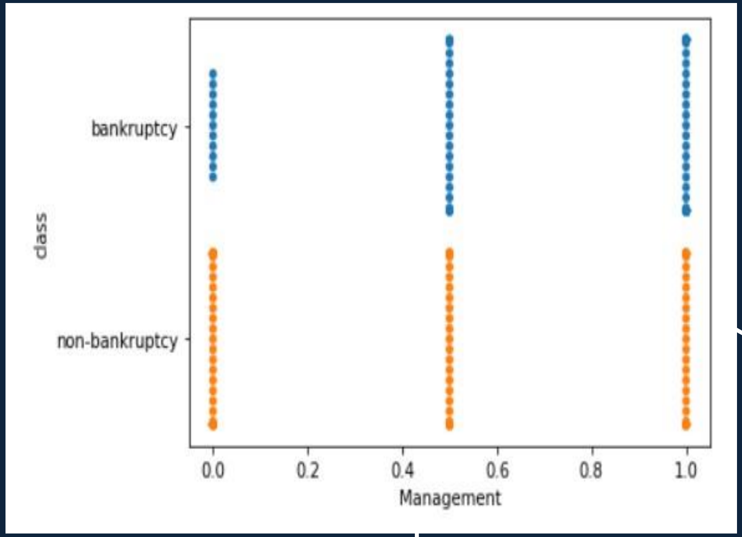
class
bankruptcy
non-bankruptcy
0.0
0.2
0.4
0.6
0.8
1.0
Credibility



bankruptcy
non-bankruptcy
0.0
0.2
0.4
0.6
0.8
1.0
Financial

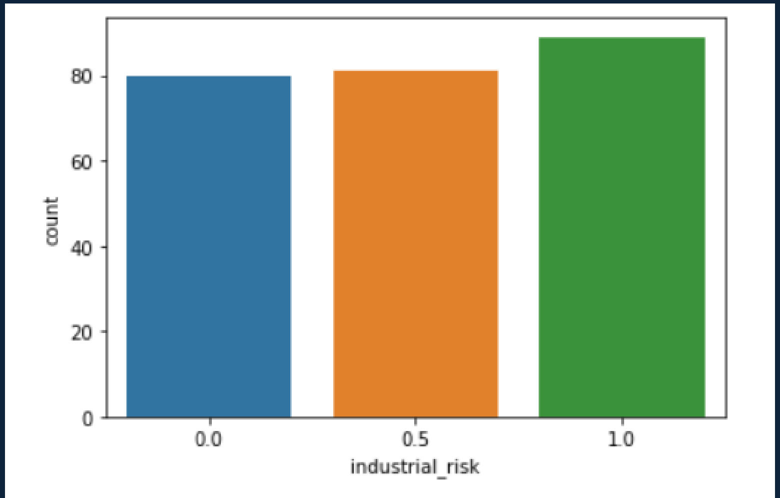


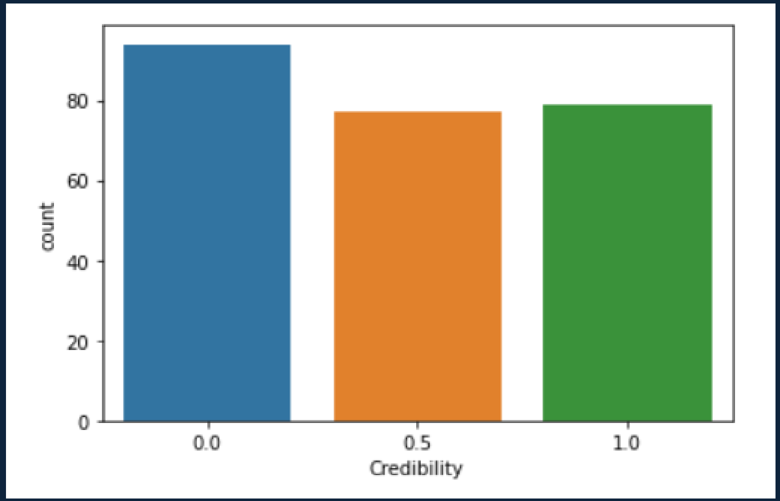
class
bankruptcy
non-bankruptcy
0.0
0.2
0.4
0.6
0.8
1.0
industrial_risk

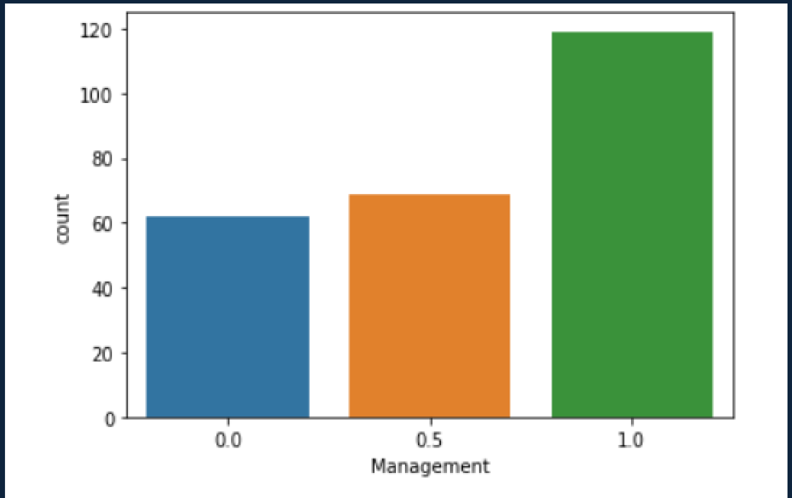


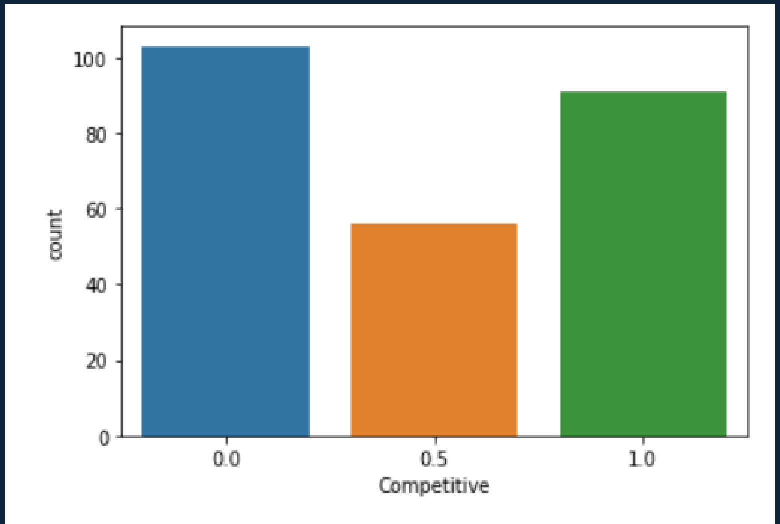
class
bankruptcy
non-bankruptcy
0.0
0.2
0.4
0.6
0.8
1.0
Management

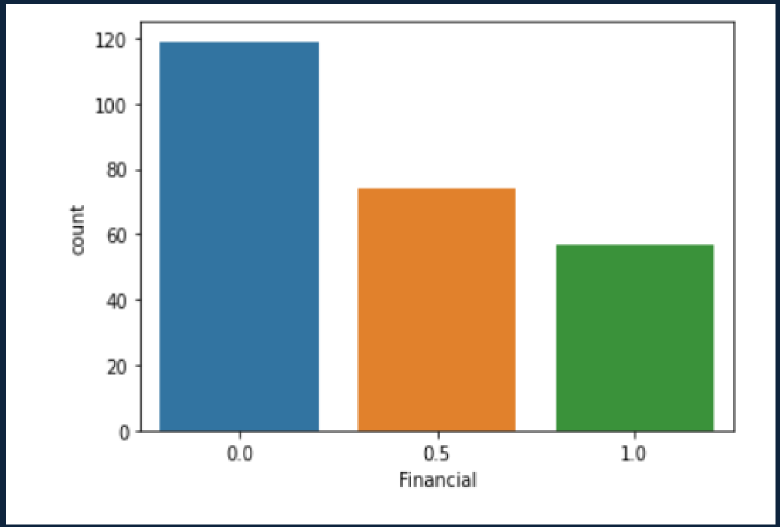
Count Plot

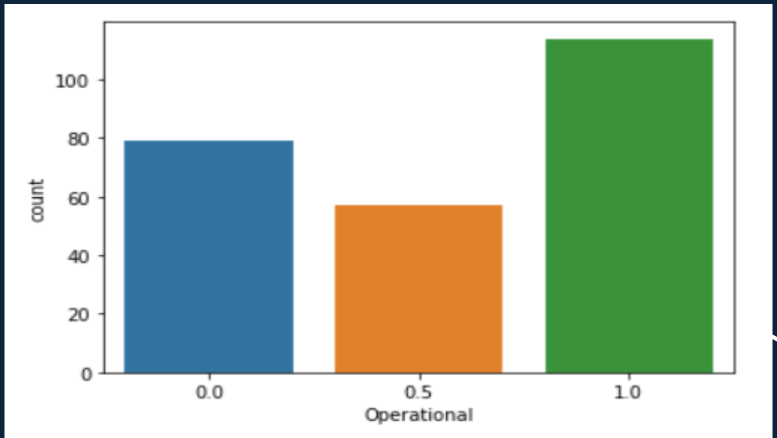

count
0
20
40
60
80
0.0
0.5
1.0
industrial_risk


count
0
20
40
60
80
0.0
0.5
1.0
Credibility

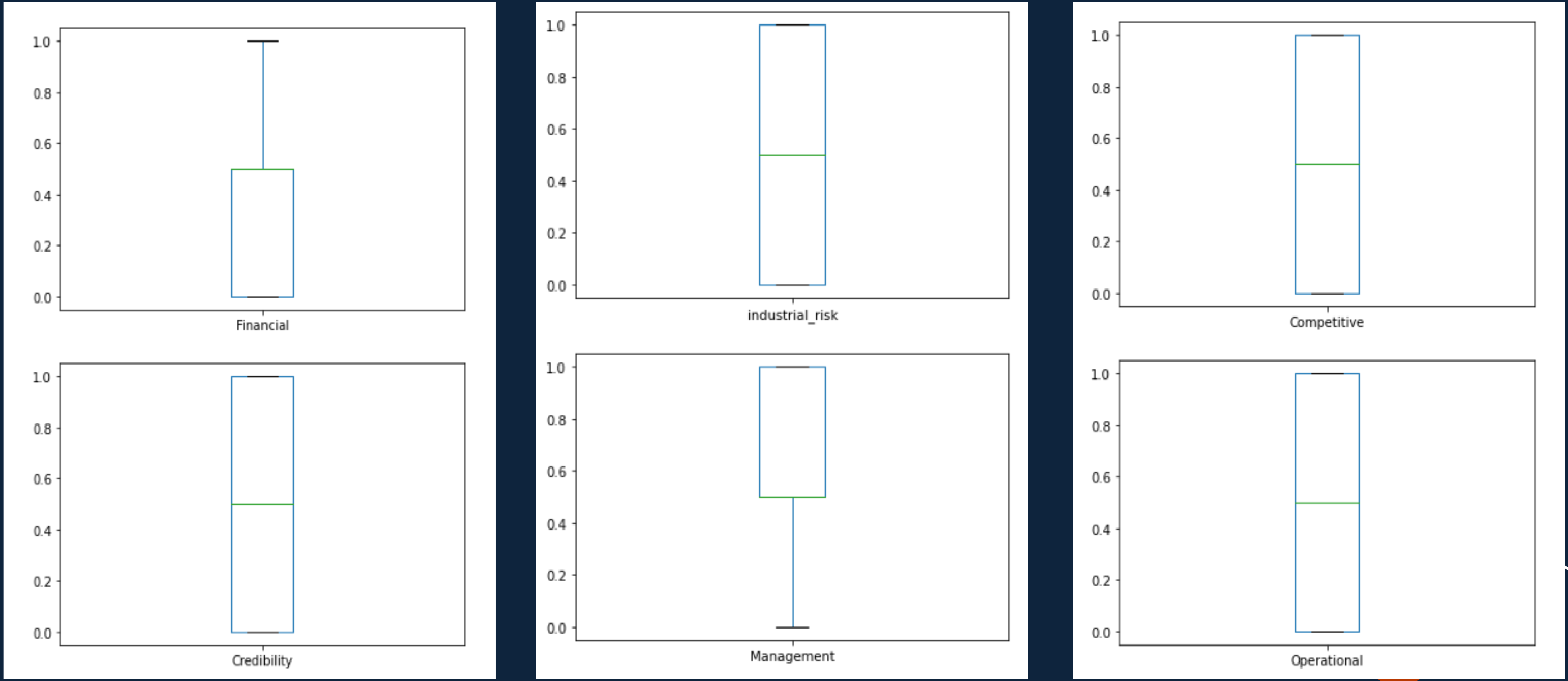

count
0
20
40
60
80
100
120
0.0
0.5
1.0
Management


count
0
20
40
60
80
100
0.0
0.5
1.0
Competitive


count
0
20
40
60
80
100
120
0.0
0.5
1.0
Financial


count
0
20
40
60
80
100
0.0
0.5
1.0
Operational

BOX Plot



Financial
Credibility
industrial_risk
Management
Competitive
Operational
0.0
0.2
0.4
0.6
0.8
1.0

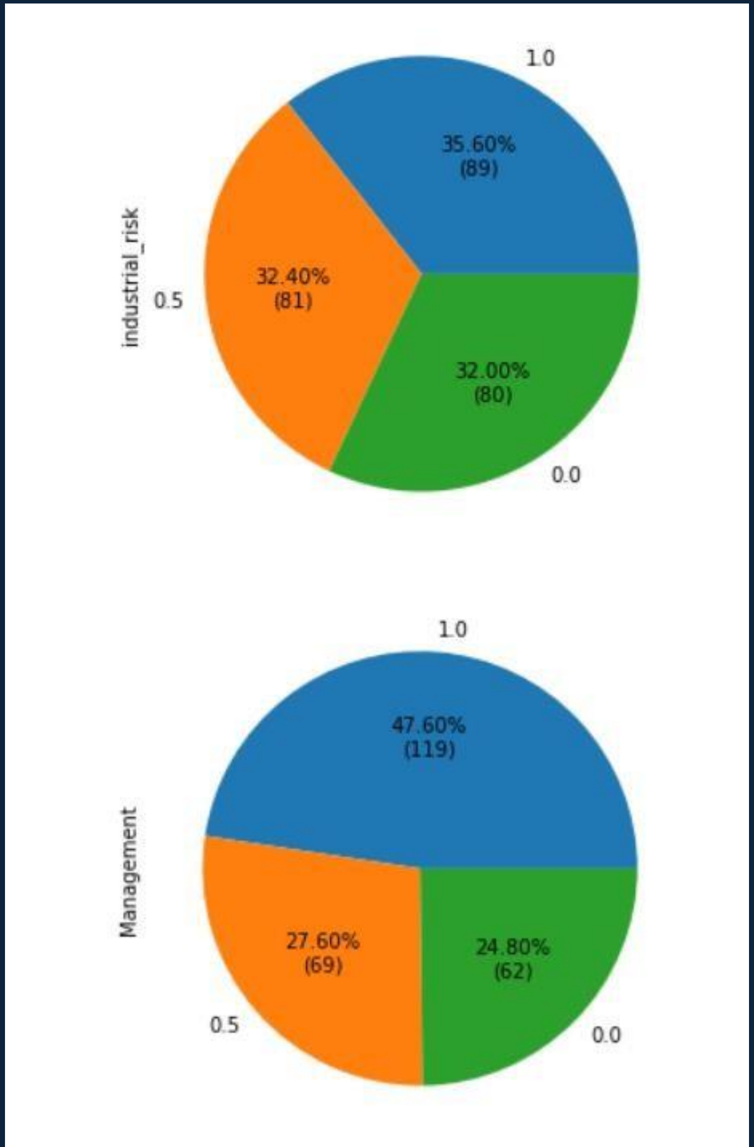
The provided code identifies and removes outliers from specific features in a DataFrame, resulting in a cleaned DataFrame without outliers.

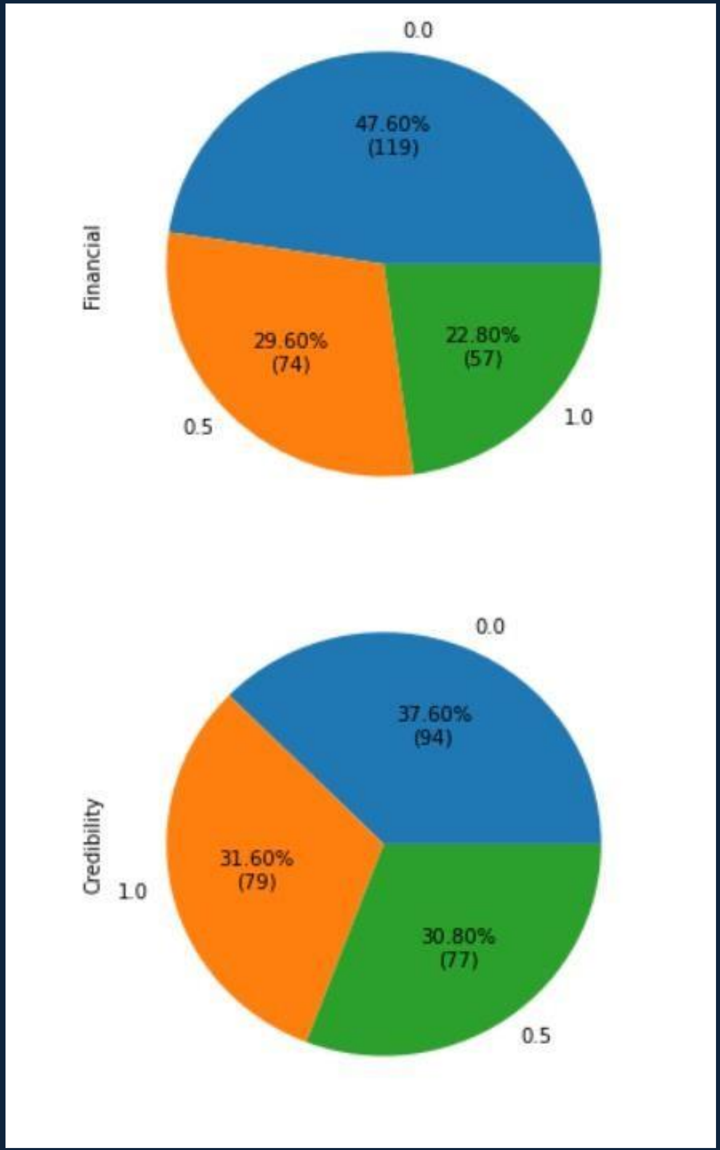
```
In [41]: def outliers(df,ft):
             Q1 = df[ft].quantile(0.25)
             Q3 = df[ft].quantile(0.75)
             IQR = Q3-Q1

             lower_bound = Q1-1.5* IQR
             upper_bound = Q3 + 1.5 * IQR
             ls = df.index[(df[ft] < lower_bound) | (df[ft] > upper_bound)]
             return ls


         index_list = []
         for feature in ["industrial_risk","Management","Financial","Credibility","Competitive","Operational"]:
             index_list.extend(outliers(df1,feature))


         index_list

         def remove(df,ls):
             ls = sorted(set(ls))
             df = df.drop(ls)
             return df

         df_cleaned = remove(df1,index_list)
         df_cleaned.shape


         df_cleaned.shape
```

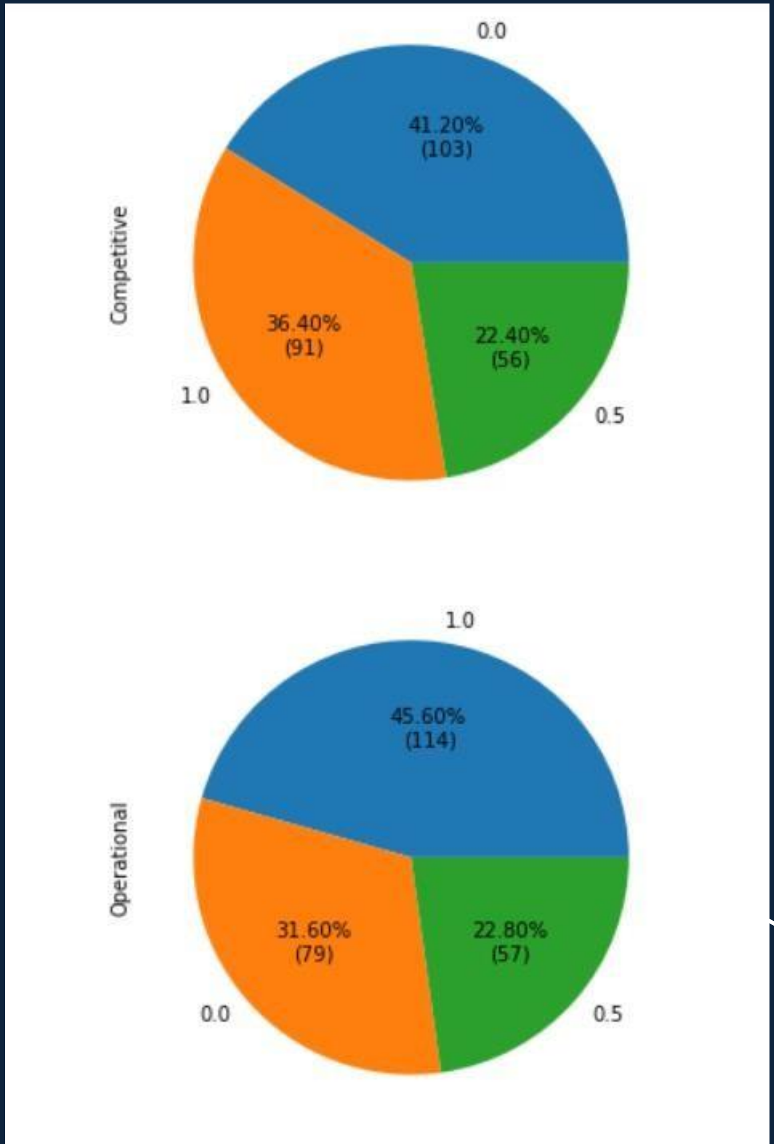
```
df1.head()
```

	industrial_risk	Management	Financial	Credibility	Competitive	Operational	class
0	0.5	1.0	0.0	0.0	0.0	0.5	bankruptcy
1	0.0	1.0	0.0	0.0	0.0	1.0	bankruptcy
2	1.0	0.0	0.0	0.0	0.0	1.0	bankruptcy
3	0.5	0.0	0.0	0.5	0.0	1.0	bankruptcy
4	1.0	1.0	0.0	0.0	0.0	1.0	bankruptcy

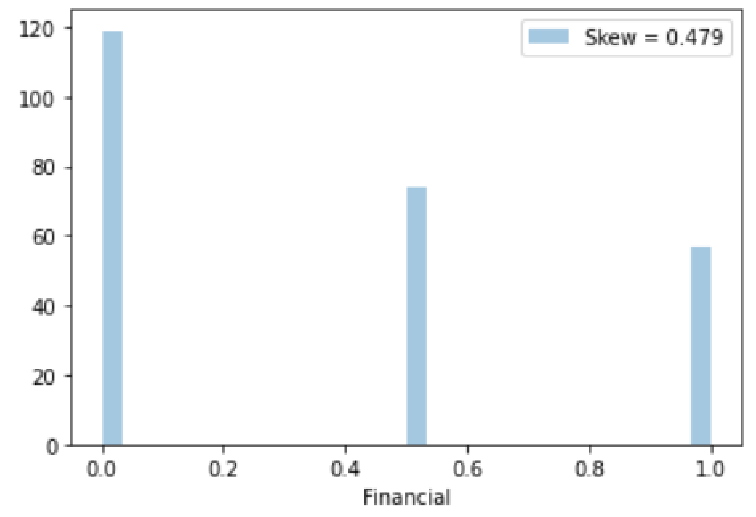
HISTOGRAM

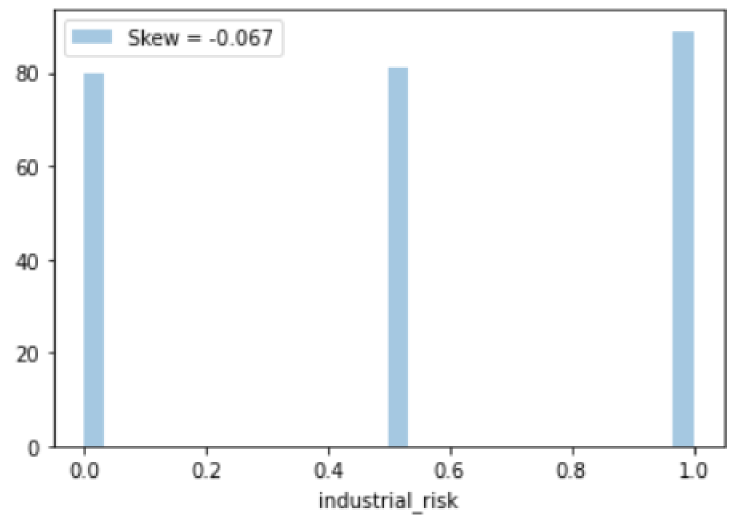

industrial_risk
1.0
35.60%
(89)
0.5
32.40%
(81)
0.0
32.00%
(80)
Management
1.0
47.60%
(119)
0.5
27.60%
(69)
0.0
24.80%
(62)

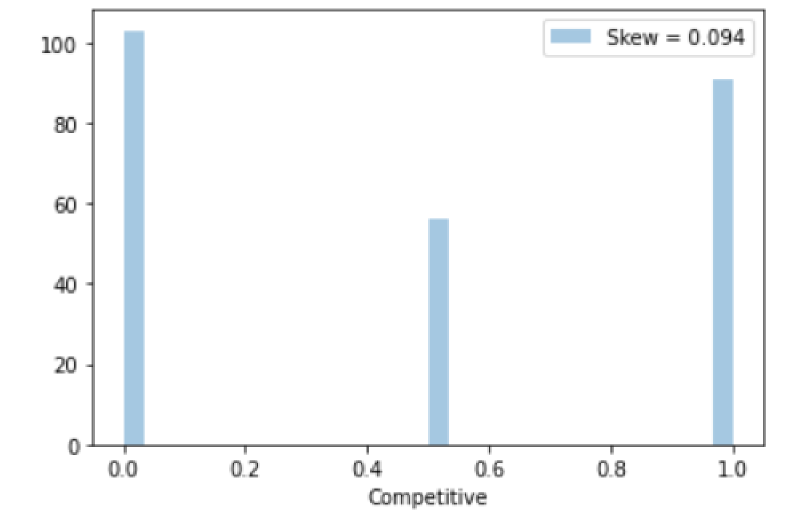

Financial
0.0
47.60%
(119)
0.5
29.60%
(74)
1.0
22.80%
(57)
Credibility
0.0
37.60%
(94)
1.0
31.60%
(79)
0.5
30.80%
(77)

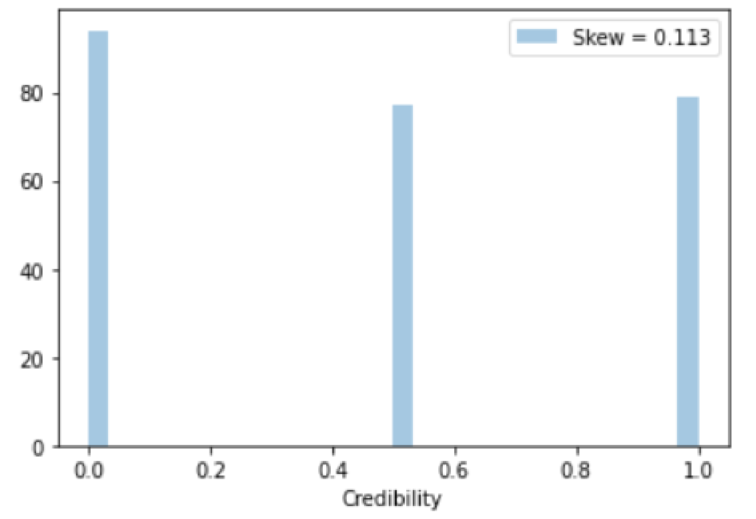

Competitive
0.0
41.20%
(103)
1.0
36.40%
(91)
0.5
22.40%
(56)
Operational
1.0
45.60%
(114)
0.0
31.60%
(79)
0.5
22.80%
(57)

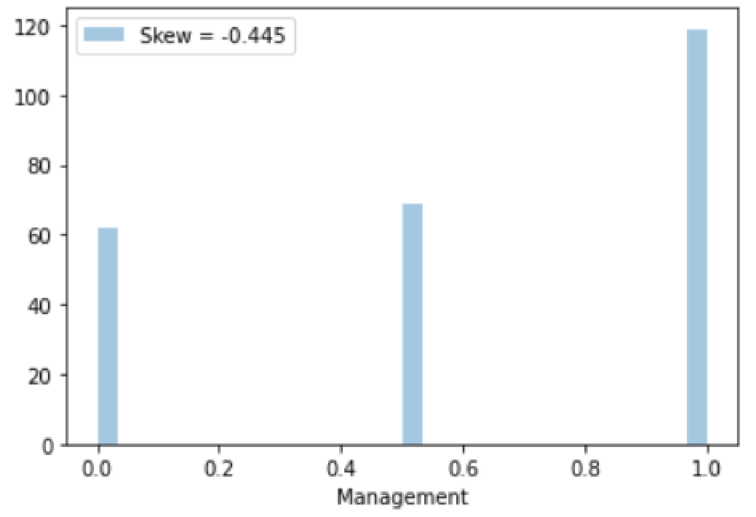
Finding the Skewness

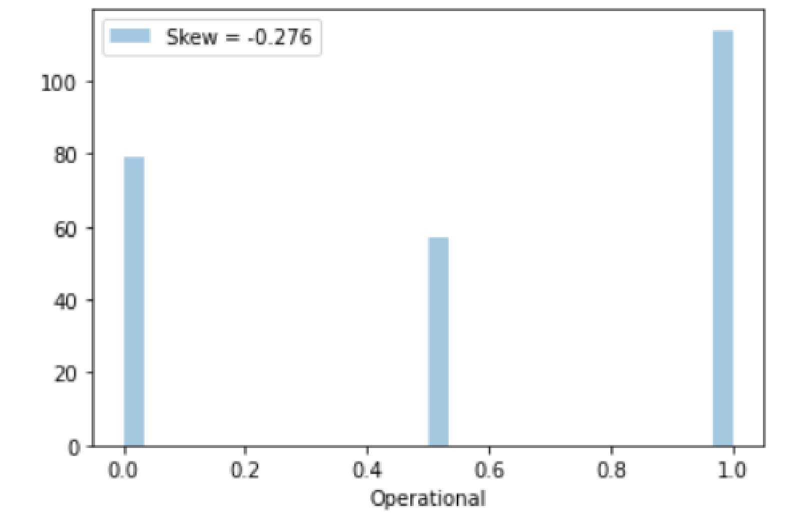




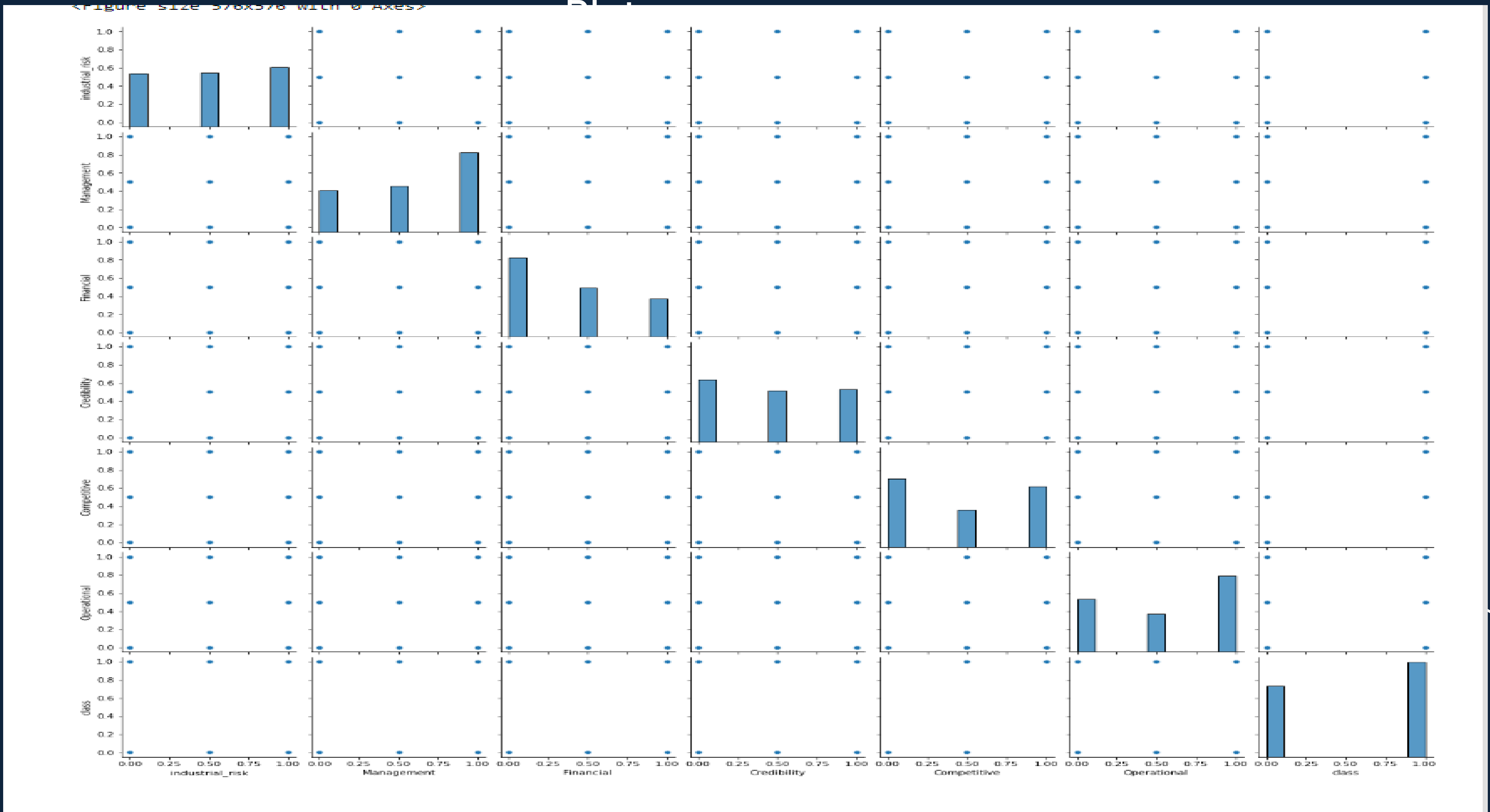




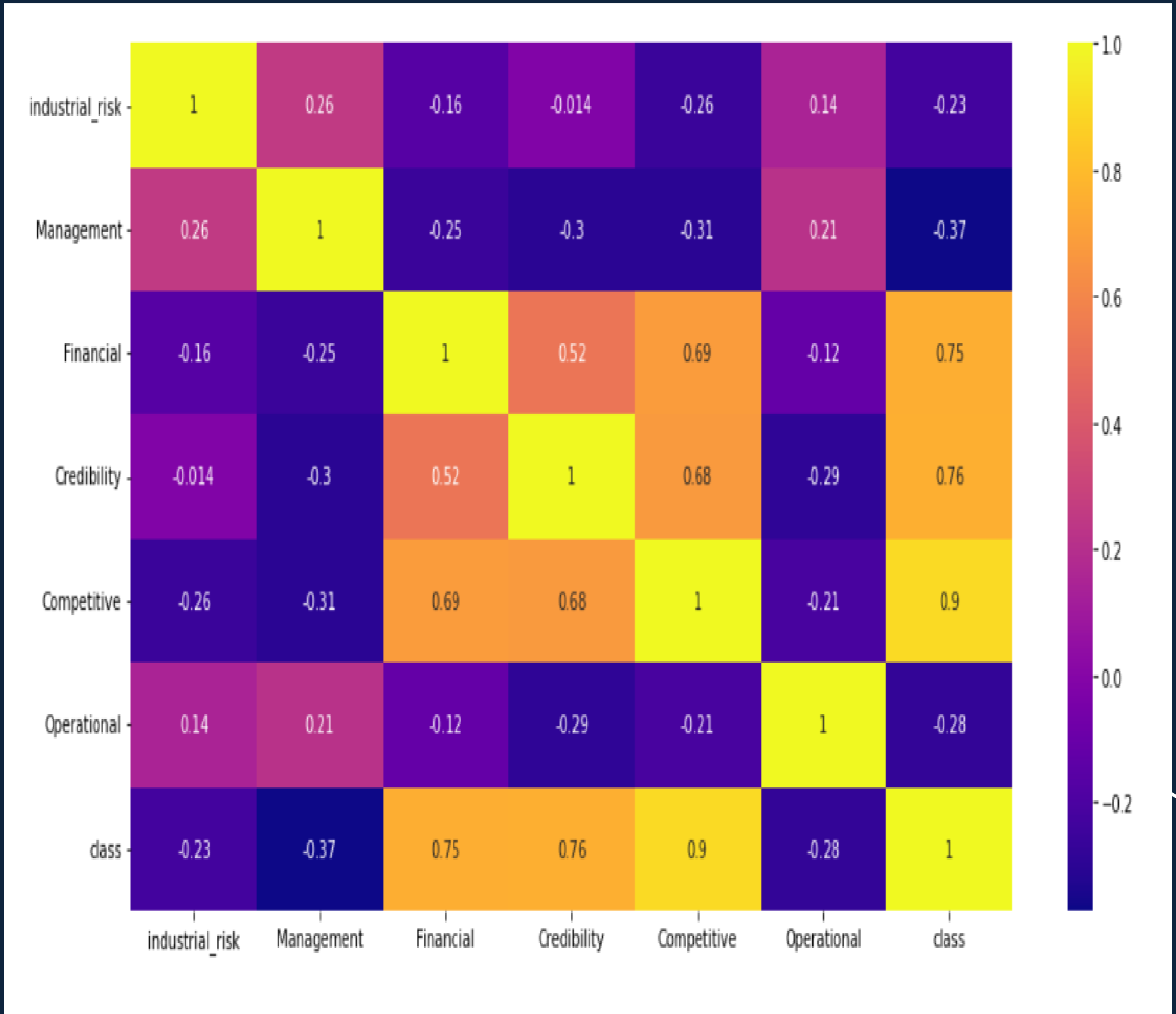




PAIR Plot


<Figure size 576x576 with 0 Axes>
industrial_risk
Management
Financial
Credibility
Competitive
Operational
class

- Upon examining feature-to-feature and feature-to-target relationships using pair plots, no evident linear connections were found. However, positive correlations were noticed between "Financial," "Credibility," and "Competitive" features with the target variable. Notably, "Financial" and "Competitive" exhibited a strong positive correlation of 0.69.



MODEL BUILDING

- Split the data set in to train and test

```
from sklearn.model_selection import train_test_split

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = .25,random_state = 42)
```

```
X_train.shape
```

```
(187, 6)
```

```
X_test.shape
```

```
(63, 6)
```

```
Y_train.shape
```

```
(187,)
```

```
Y_test.shape
```

```
(63,)
```

- A logistic regression model was trained and evaluated on the training and test datasets. The model demonstrated high accuracy on both sets, indicating effective classification. Sensitivity and precision scores were balanced, with F1 scores reflecting a good balance between precision and recall on both training and test data, indicating a robust model.

```
train_cm1

array([[ 84,   1],
       [  0, 102]], dtype=int64)

test_cm1

array([[22,  0],
       [ 0, 41]], dtype=int64)
```

```
Accuracy_score_train_1 = accuracy_score(Y_train,Y_pred_train).round(3) #
Accuracy_score_train_1

0.995

Accuracy_score_test_1 = accuracy_score(Y_test,Y_pred_test).round(3)
Accuracy_score_test_1

1.0
```

```
Precision_score_train_1 = precision_score(Y_train,Y_pred_train).round(3
Precision_score_train_1

0.99

Precision_score_test_1 = precision_score(Y_test,Y_pred_test).round(3)

F1_score_train_1=f1_score(Y_train,Y_pred_train).round(3) # (2 * PRECISI
F1_score_train_1

0.995

F1_score_test_1 = f1_score(Y_test,Y_pred_test).round(3)
F1_score_test_1

1.0
```

```
Sensitivity_score_train_1 = recall_score(Y_train,Y_pred_train).round(3)
Sensitivity_score_train_1

1.0

Sensitivity_score_test_1 = recall_score(Y_test,Y_pred_test).round(3)
Sensitivity_score_test_1

1.0
```

- The K-Nearest Neighbors (KNN) classifier with k=5 and Euclidean distance metric exhibited exceptional performance. Both training and test sets achieved high accuracy (98.4%), showcasing the model's strong predictive ability. Sensitivity on the training set was perfect (100%), indicating its ability to capture all positive instances. Precision was high (99%) on the training set and perfect (100%) on the test set, demonstrating the model's precision in identifying positive cases. F1 scores, reflecting a balance between precision and recall, were also impressively high, underscoring the model's overall effectiveness and reliability in classification tasks.

```
train_cm2
```

```
array([[ 84,   1],
       [  0, 102]], dtype=int64)
```

```
test_cm2
```

```
Accuracy_score_train_2 = accuracy_score(Y_train,Y_pred_train).round(3)
Accuracy_score_train_2 = Accuracy_score_test_2 = accuracy_score(Y_test,Y_pred_test).round(3)
Accuracy_score_test_2
```

```
0.984
```

```
Accuracy_score_test_2 = accuracy_score(Y_test,Y_pred_test).round(3)
Accuracy_score_test_2
```

```
0.984
```

```
Sensitivity_score_train_2 = recall_score(Y_train,Y_pred_train).round(3)
Sensitivity_score_train_2
```

```
1.0
```

```
Sensitivity_score_test_2 = recall_score(Y_test,Y_pred_test).round(3)
Sensitivity_score_test_2
```

```
0.976
```

```
Precision_score_train_2 = precision_score(Y_train,Y_pred_train).round(3)
Precision_score_train_2
```

```
0.99
```

```
Precision_score_test_2 = precision_score(Y_test,Y_pred_test).round(3)
Precision_score_test_2
```

```
1.0
```

```
F1_score_train_2 = f1_score(Y_train,Y_pred_train).round(3)
F1_score_train_2
```

```
0.995
```

```
F1_score_test_2 = f1_score(Y_test,Y_pred_test).round(3)
F1_score_test_2
```

```
0.988
```

- The Multinomial Naive Bayes classifier performed remarkably well. Both training and test sets demonstrated high accuracy (97.3% and 100%, respectively). Sensitivity and precision scores were excellent, with perfect scores indicating accurate detection of positive cases. F1 scores were also high, highlighting the model's balanced precision and recall, showcasing its robustness in classifying instances.

```
train_cm3
```

```
array([[ 80,   5],
       [  0, 102]], dtype=int64)
```

```
test_cm3
```

```
array([[22,  0],
       [ 0, 41]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score
```

```
Accuracy_score_train_3 = accuracy_score(Y_train,Y_pred_train).round(3)
Accuracy_score_train_3
```

```
0.973
```

```
Accuracy_score_test_3 = accuracy_score(Y_test,Y_pred_test).round(3)
Accuracy_score_test_3
```

```
1.0
```

```
Sensitivity_score_train_3 = recall_score(Y_train,Y_pred_train).round(3)
Sensitivity_score_train_3
```

```
1.0
```

```
Sensitivity_score_test_3 = recall_score(Y_test,Y_pred_test).round(3)
Sensitivity_score_test_3
```

```
1.0
```

```
Precision_score_train_3 = precision_score(Y_train,Y_pred_train).round(3)
Precision_score_train_3
```

```
0.953
```

```
Precision_score_test_3 = precision_score(Y_test,Y_pred_test).round(3)
Precision_score_test_3
```

```
1.0
```

```
F1_score_train_3 = f1_score(Y_train,Y_pred_train).round(3)
F1_score_train_3
```

```
0.976
```

```
F1_score_test_3 = f1_score(Y_test,Y_pred_test).round(3)
F1_score_test_3
```

```
1.0
```

- The Support Vector Machine with a polynomial function displayed outstanding performance. In the training set, it achieved perfect accuracy and sensitivity, indicating flawless classification of positive instances. On the test set, it demonstrated high accuracy and precision, with a slightly lower sensitivity, suggesting effective detection while maintaining high precision. The F1 score on the test set remained exceptionally high, underlining the model's robustness in balancing precision and recall.

```
train_cm4
```
```
array([[ 85,   0],
       [  0, 102]], dtype=int64)
```
```
test_cm4
```
```
array([[22,  0],
       [ 1, 40]], dtype=int64)
```
```
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score
```
```
Accuracy_score_train_4 = accuracy_score(Y_train,Y_pred_train).round(3)
Accuracy_score_train_4
```
```
1.0
```
```
Accuracy_score_test_4 = accuracy_score(Y_test,Y_pred_test).round(3)
Accuracy_score_test_4
```
```
0.984
```
```
Sensitivity_score_train_4 = recall_score(Y_train,Y_pred_train).round(3)
Sensitivity_score_train_4
```
```
1.0
```
```
Sensitivity_score_test_4 = recall_score(Y_test,Y_pred_test).round(3)
Sensitivity_score_test_4
```
```
0.976
```
```
Precision_score_train_4 = precision_score(Y_train,Y_pred_train).round(3)
Precision_score_train_4
```
```
1.0
```
```
Precision_score_test_4 = precision_score(Y_test,Y_pred_test).round(3)
Precision_score_test_4
```
```
1.0
```
```
F1_score_train_4 = f1_score(Y_train,Y_pred_train).round(3)
F1_score_train_4
```
```
1.0
```
```
F1_score_test_4 = f1_score(Y_test,Y_pred_test).round(3)
F1_score_test_4
```
```
0.988
```

- All the presented classifier models, including Logistic Regression, K-Nearest Neighbors, Multinomial Naive Bayes, and Support Vector Machine with a polynomial function, exhibited exceptional performance. They achieved high accuracy, sensitivity, and precision, ensuring accurate identification of positive cases while maintaining a strong balance between precision and recall. These models demonstrate their reliability and effectiveness in various classification tasks, making them valuable choices for different applications.
- We've chosen the logistic regression model. We'll use cross-validation to ensure it works well on different data parts and prevent it from becoming too tailored to our current dataset, preventing potential issues with overfitting.

MODEL EVALUATION

- The evaluation results indicate a perfect classification performance, with an accuracy, precision, recall, and F1-score of 1.0. The confusion matrix shows that all 50 samples were correctly classified into "bankruptcy" and "non-bankruptcy" categories. The classification report further confirms this flawless performance, suggesting that the model is highly accurate and reliable in distinguishing between the two classes, showcasing excellent predictive capabilities.

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report

# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)

# Calculate precision, recall, and F1-score
precision = precision_score(y_test, predictions, average='weighted')
recall = recall_score(y_test, predictions, average='weighted')
f1 = f1_score(y_test, predictions, average='weighted')

print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)

# Print confusion matrix
conf_matrix = confusion_matrix(y_test, predictions)
print("Confusion Matrix:")
print(conf_matrix)

# Print classification report
class_report = classification_report(y_test, predictions)
print("Classification Report:")
print(class_report)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
Confusion Matrix:
[[21  0]
 [ 0 29]]
Classification Report:
                precision    recall  f1-score   support

    bankruptcy       1.00      1.00      1.00        21
non-bankruptcy       1.00      1.00      1.00        29

      accuracy                           1.00        50
     macro avg       1.00      1.00      1.00        50
  weighted avg       1.00      1.00      1.00        50
```

- REGULARISATION
- 01. LASSO
- 02. RIDGE
- Both logistic regression models achieved perfect accuracy (1.0), indicating flawless classification on the test data. The first model utilized L2 regularization, while the second employed L1 regularization (Lasso). L2 regularization maintains all features, while L1 can lead to feature selection. Despite this difference, both models performed exceptionally well, demonstrating their robustness in correctly predicting the target variable, showcasing their high predictive power.
- The Logistic Regression model with L2 regularization (Ridge) achieved a perfect accuracy score of 1.0, indicating flawless classification on the test data. This underscores the model's robustness and ability to precisely predict the target variable, showcasing its high performance and reliability.

Regularization to logistic regression

```
C = 1.0  # Regularization parameter
model = LogisticRegression(penalty='l2', C=C)

# Train the model using the training data
model.fit(X_train, y_train)

# Make predictions on the test data
predictions = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Assuming X contains your features and ' class' (with a space) is your correct target variable
X = df.drop(columns=[' class'])  # Features
y = df[' class']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Logistic Regression model with L1 regularization (Lasso)
# You can adjust the value of C to control the regularization strength
lasso_C = 1.0  # Regularization parameter for Lasso
lasso_model = LogisticRegression(penalty='l1', C=lasso_C, solver='liblinear')

# Train the Lasso model using the training data
lasso_model.fit(X_train, y_train)

# Make predictions using the Lasso model
lasso_predictions = lasso_model.predict(X_test)

# Evaluate the Lasso model
lasso_accuracy = accuracy_score(y_test, lasso_predictions)
print("Lasso Accuracy:", lasso_accuracy)
```

```
Lasso Accuracy: 1.0
```

```
# Create a Logistic Regression model with L2 regularization (Ridge)
# You can adjust the value of C to control the regularization strength
ridge_C = 1.0  # Regularization parameter for Ridge
ridge_model = LogisticRegression(penalty='l2', C=ridge_C)

# Train the Ridge model using the training data
ridge_model.fit(X_train, y_train)

# Make predictions using the Ridge model
ridge_predictions = ridge_model.predict(X_test)

# Evaluate the Ridge model
ridge_accuracy = accuracy_score(y_test, ridge_predictions)
print("Ridge Accuracy:", ridge_accuracy)
```

```
Ridge Accuracy: 1.0
```

CROSS VALIDATION

Both Lasso (L1 regularization) and Ridge (L2 regularization) Logistic Regression models demonstrated exceptional performance during 10-fold cross-validation. They consistently achieved high accuracy, with average scores of approximately 99.6%. These results suggest the models' robustness and generalizability, indicating their ability to maintain accurate predictions across different subsets of the data.

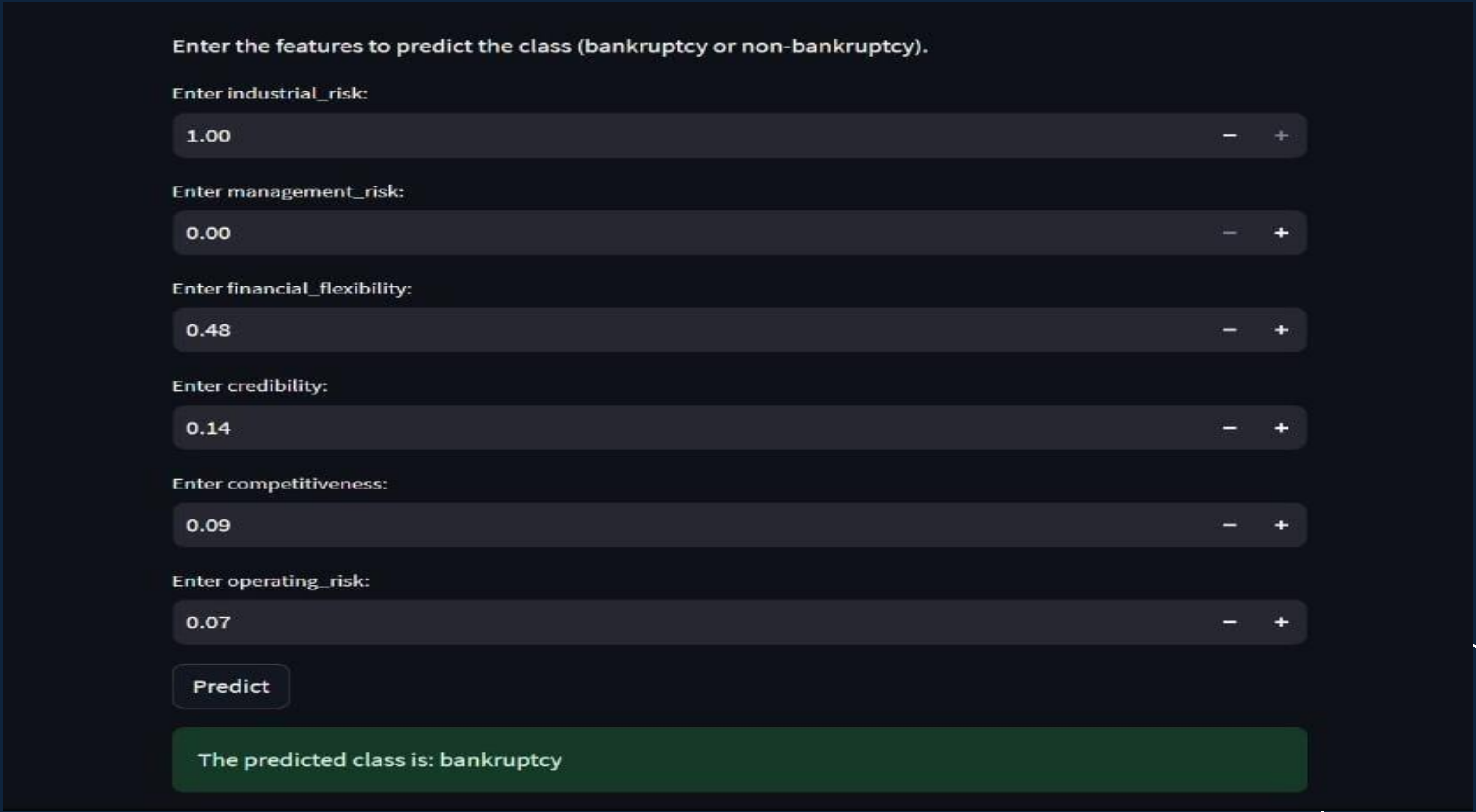
```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

# Assuming X contains your features and ' class' (with a space) is your correct target variable
X = df.drop(columns=[' class'])  # Features
y = df[' class']  # Target variable

# Create Lasso (L1 regularization) Logistic Regression model
lasso_model = LogisticRegression(penalty='l1', solver='liblinear')

# Perform 10-fold cross-validation for Lasso model
lasso_scores = cross_val_score(lasso_model, X, y, cv=10, scoring='accuracy')
print("Lasso Cross-Validation Scores:", lasso_scores)
print("Lasso Average Accuracy:", lasso_scores.mean())

# Create Ridge (L2 regularization) Logistic Regression model
ridge_model = LogisticRegression(penalty='l2')

# Perform 10-fold cross-validation for Ridge model
ridge_scores = cross_val_score(ridge_model, X, y, cv=10, scoring='accuracy')
print("Ridge Cross-Validation Scores:", ridge_scores)
print("Ridge Average Accuracy:", ridge_scores.mean())
```

```
Lasso Cross-Validation Scores: [1.   1.   1.   1.   1.   1.   0.96 1.   1.   1.  ]
Lasso Average Accuracy: 0.9960000000000001
Ridge Cross-Validation Scores: [1.   1.   1.   1.   1.   1.   0.96 1.   1.   1.  ]
Ridge Average Accuracy: 0.9960000000000001
```

DEPLOYMENT

For deployment, we opted for Logistic Regression due to its consistent high performance, as demonstrated through cross-validation. We fine-tuned the model by exploring various parameters, ensuring its accuracy in predicting bankruptcy occurrences. Leveraging Streamlit, we created an intuitive interface for users to input data, allowing real-time predictions on the likelihood of bankruptcy. This reliable, well-tuned model forms the backbone of our deployment, offering users a trustworthy tool for making informed decisions regarding financial stability.

```python
import streamlit as st
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Load the data
def load_data():
    df = pd.read_csv('path/to/your/bank.csv', delimiter=';')  # Adjust the delimiter if necessary
    return df

df = load_data()

# Preprocess the data
X = df.drop(columns=[' class'])  # Features
y = df[' class']  # Target variable

# Standardize features (if necessary)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Create and train the model
model = LogisticRegression(penalty='l1', solver='liblinear')
model.fit(X_scaled, y)

# Streamlit UI
st.title("Bankruptcy Prediction App")
st.write("Enter the features to predict the class (bankruptcy or non-bankruptcy).")

# Get user input for features
input_features = []
for feature_name in X.columns:
    input_feature = st.number_input(f"Enter {feature_name}:", min_value=0.0, max_value=1.0, step=0.01)
    input_features.append(input_feature)

# Predict the class based on user input
if st.button("Predict"):
    user_input = pd.DataFrame([input_features], columns=X.columns)
    user_input_scaled = scaler.transform(user_input)
    prediction = model.predict(user_input_scaled)
    st.success(f"The predicted class is: {prediction[0]}")
```


Enter the features to predict the class (bankruptcy or non-bankruptcy).
Enter industrial_risk:
1.00
Enter management_risk:
0.00
Enter financial_flexibility:
0.48
Enter credibility:
0.14
Enter competitiveness:
0.09
Enter operating_risk:
0.07
Predict
The predicted class is: bankruptcy

THANK YOU