```
import pandas as pd
import numpy as np
from numpy import loadtxt
from numpy import sort
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.ticker import PercentFormatter
import matplotlib.ticker as mtick
from random import sample
import seaborn as sns
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils import resample
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
```

The data is obtained from: https://github.com/Agewerc/ML-Finance/blob/master/data/corporate_rating.csv

```
df_rating = pd.read_csv('https://raw.githubusercontent.com/Agewerc/ML-Finance/mas
df_rating.head()
```

	Rating	Name	Symbol	Rating Agency Name	Date	Sector	currentRatio	quickRa
0	A	Whirlpool Corporation	WHR	Egan-Jones Ratings Company	11/27/2015	Consumer Durables	0.945894	0.42
1	BBB	Whirlpool Corporation	WHR	Egan-Jones Ratings Company	2/13/2014	Consumer Durables	1.033559	0.49
2	BBB	Whirlpool Corporation	WHR	Fitch Ratings	3/6/2015	Consumer Durables	0.963703	0.45
3	BBB	Whirlpool Corporation	WHR	Fitch Ratings	6/15/2012	Consumer Durables	1.019851	0.51
4	BBB	Whirlpool Corporation	WHR	Standard & Poor's Ratings Services	10/24/2016	Consumer Durables	0.957844	0.49

5 rows × 31 columns

Exploratory Data Analysis

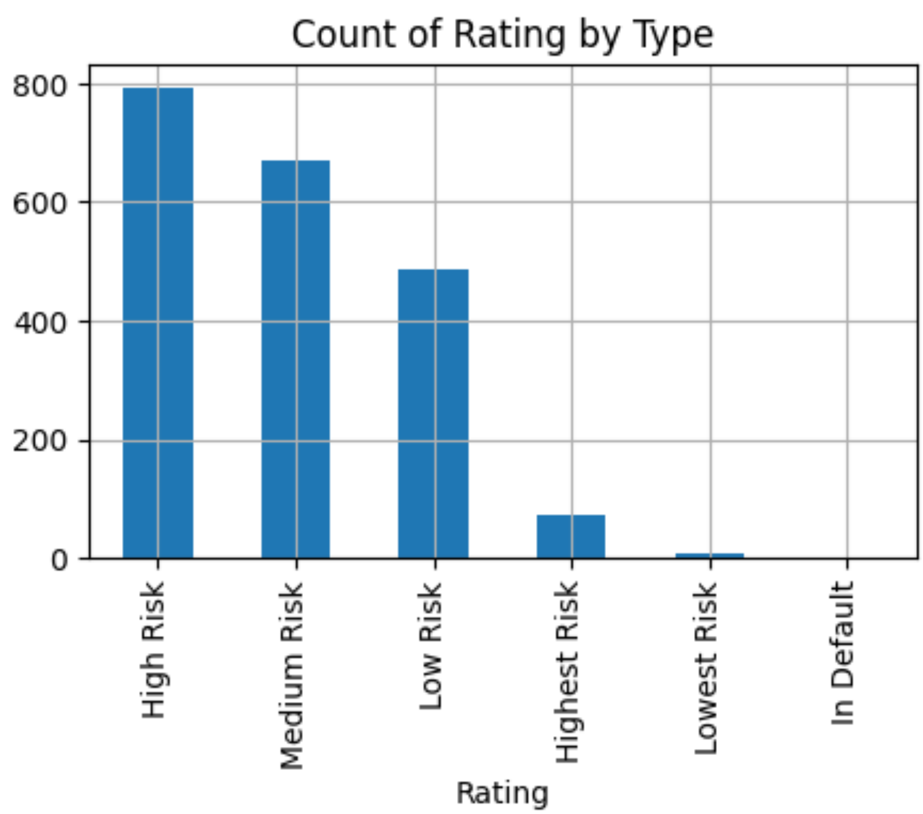
```
print("The credit rating dataset has", df_rating.shape[0], "records, each with", 
    "attributes")
df_rating.info()
```

```
The credit rating dataset has 2029 records, each with 31 attributes
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2029 entries, 0 to 2028
Data columns (total 31 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   Rating                             2029 non-null   object 
 1   Name                               2029 non-null   object 
 2   Symbol                             2029 non-null   object 
 3   Rating Agency Name                 2029 non-null   object 
 4   Date                               2029 non-null   object 
 5   Sector                             2029 non-null   object 
 6   currentRatio                       2029 non-null   float64
 7   quickRatio                         2029 non-null   float64
 8   cashRatio                          2029 non-null   float64
 9   daysOfSalesOutstanding             2029 non-null   float64
 10  netProfitMargin                    2029 non-null   float64
 11  pretaxProfitMargin                 2029 non-null   float64
 12  grossProfitMargin                  2029 non-null   float64
 13  operatingProfitMargin              2029 non-null   float64
 14  returnOnAssets                     2029 non-null   float64
 15  returnOnCapitalEmployed            2029 non-null   float64
 16  returnOnEquity                     2029 non-null   float64
 17  assetTurnover                      2029 non-null   float64
 18  fixedAssetTurnover                 2029 non-null   float64
 19  debtEquityRatio                    2029 non-null   float64
 20  debtRatio                          2029 non-null   float64
 21  effectiveTaxRate                   2029 non-null   float64
 22  freeCashFlowOperatingCashFlowRatio  2029 non-null   float64
 23  freeCashFlowPerShare               2029 non-null   float64
 24  cashPerShare                       2029 non-null   float64
 25  companyEquityMultiplier            2029 non-null   float64
 26  ebitPerRevenue                     2029 non-null   float64
 27  enterpriseValueMultiple            2029 non-null   float64
 28  operatingCashFlowPerShare          2029 non-null   float64
 29  operatingCashFlowSalesRatio        2029 non-null   float64
 30  payablesTurnover                   2029 non-null   float64
dtypes: float64(25), object(6)
memory usage: 491.5+ KB
```

We are working with **ordinal** values. A is secure, D is default-likely.

We observe that the dataset is very **unbalanced**. We have 671 triple-Bs (BBB) but only 1 D. However, we are working with Ratings from different companies such as Moody's, Standard & Poor's and more. Therefore it is preferred to simplify the labels according to investopedia.

```
df_rating.Rating.value_counts()

rating_dict = {'AAA':'Lowest Risk',  'AA':'Low Risk',  'A':'Low Risk', 'BBB':'Med
df_rating.Rating = df_rating.Rating.map(rating_dict)
ax = df_rating['Rating'].value_counts().plot(kind='bar', figsize=(5,3), title="Co
```



Unfortunately, given the lack of Credit Ratings classified as `Lowest Risk` and `In Default`, we will have to eliminate then from the table.

```
df_rating = df_rating[df_rating['Rating']!='Lowest Risk'] # filter Lowest Risk
df_rating = df_rating[df_rating['Rating']!='In Default']  # filter In Default
df_rating.reset_index(inplace = True, drop=True) # reset index
```

Descriptive Statistics

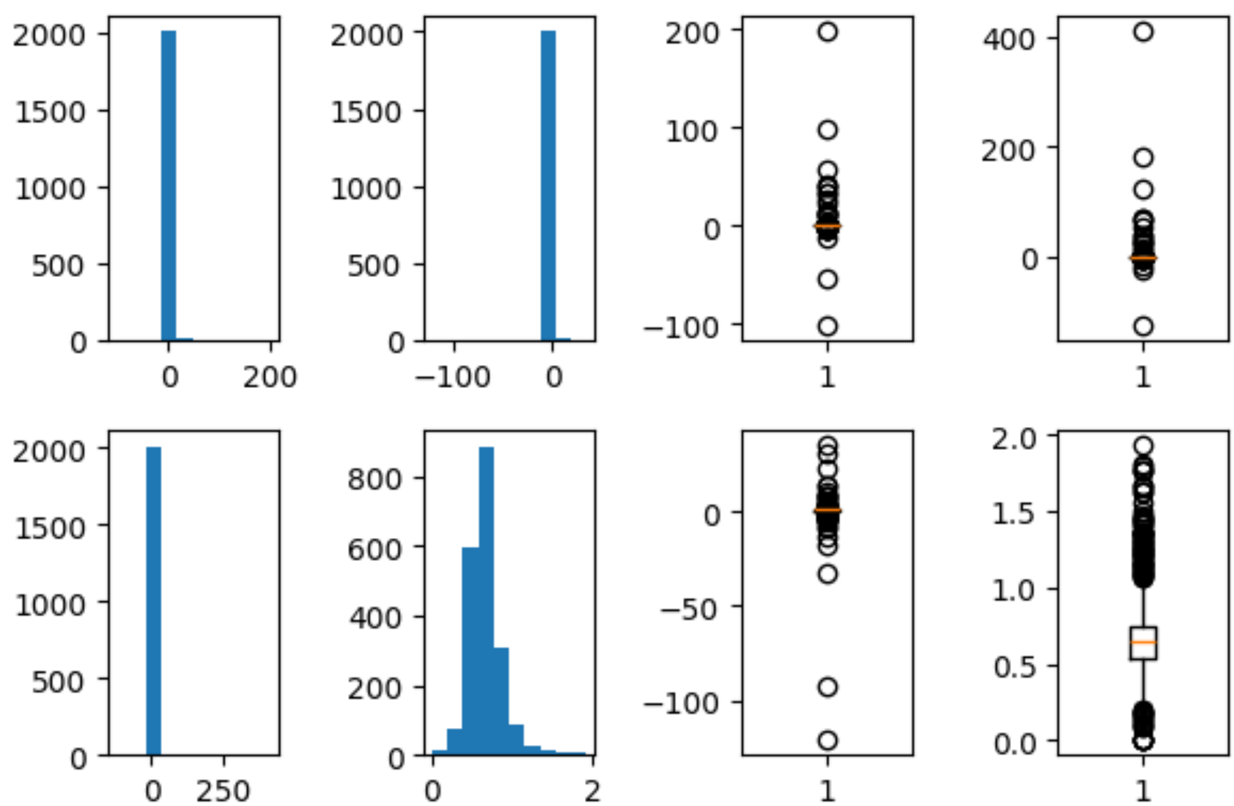
```
df_rating.describe()
```

	currentRatio	quickRatio	cashRatio	daysOfSalesOutstanding	netProfitM
count	2021.000000	2021.000000	2021.000000	2021.000000	2021.0
mean	3.535411	2.657150	0.669048	334.855415	0.2
std	44.139386	33.009920	3.590902	4456.606352	6.0
min	-0.932005	-1.893266	-0.192736	-811.845623	-101.8
25%	1.071930	0.602298	0.131433	22.806507	0.0
50%	1.492804	0.979094	0.297859	42.281804	0.0
75%	2.160710	1.450457	0.625355	59.165369	0.1
max	1725.505005	1139.541703	125.917417	115961.637400	198.5

8 rows × 25 columns

```
column_list = list(df_rating.columns[6:31])
column_list = sample(column_list,4)
print(column_list)
figure, axes = plt.subplots(nrows=2, ncols=4, figsize=(6,4))

axes[0, 0].hist(df_rating[column_list[0]])
axes[0, 1].hist(df_rating[column_list[1]])
axes[1, 0].hist(df_rating[column_list[2]])
axes[1, 1].hist(df_rating[column_list[3]])

axes[0, 2].boxplot(df_rating[column_list[0]])
axes[1, 2].boxplot(df_rating[column_list[1]])
axes[0, 3].boxplot(df_rating[column_list[2]])
axes[1, 3].boxplot(df_rating[column_list[3]])

figure.tight_layout()
```

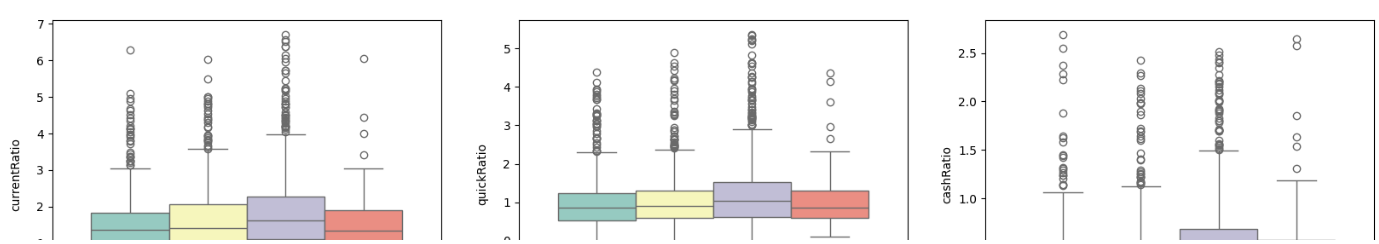
```
['netProfitMargin', 'freeCashFlowOperatingCashFlowRatio', 'operatingProfitMarg
```

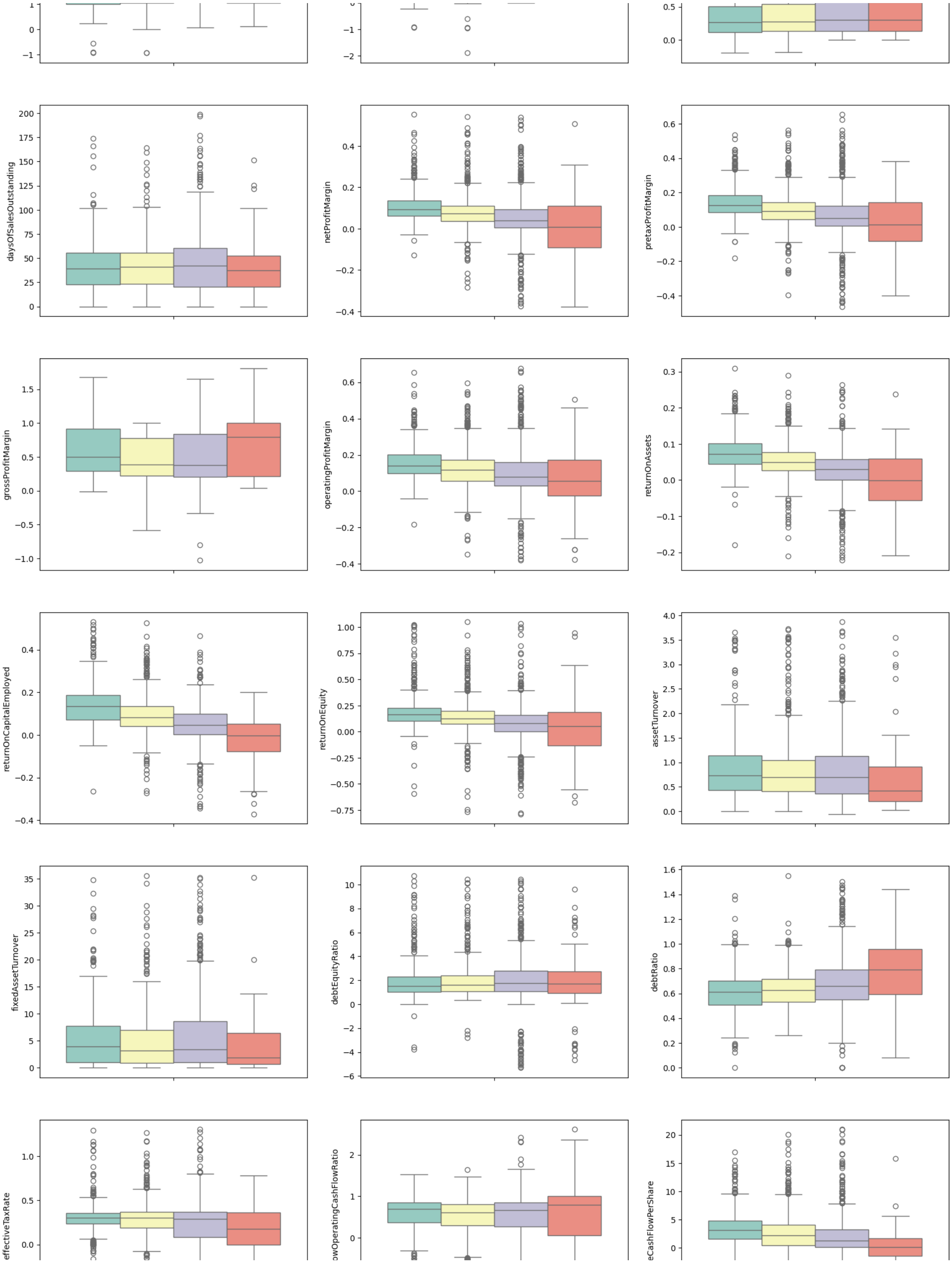


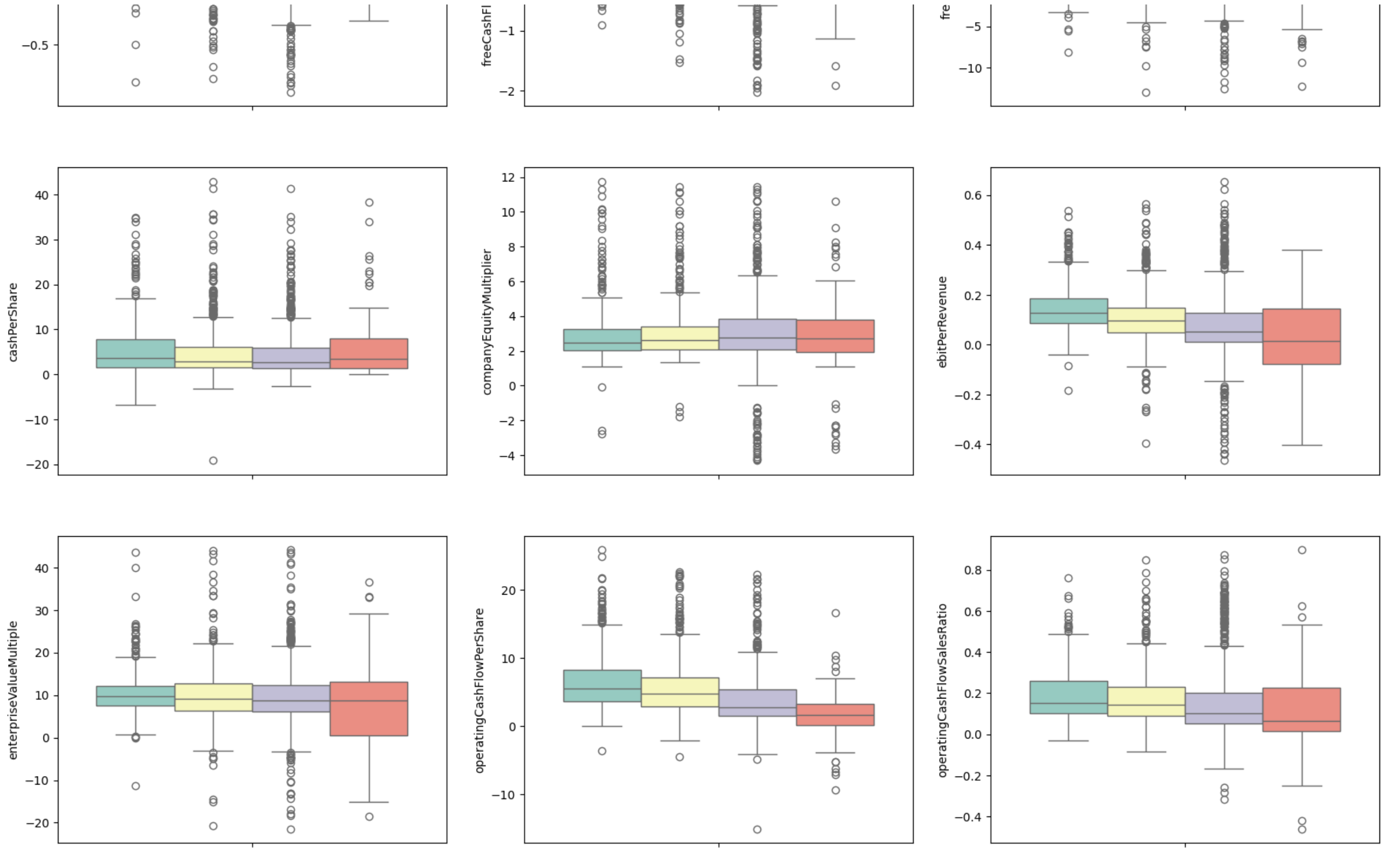
Now that we have this dataframe we can use it use it to observe the data from a different angle. We will be able to observe the distribution that was hidden by the outliers. The first step:

Plot all columns (boxplot) by each label:High Risk, Low Risk, Medium Risk, Highest Risk.

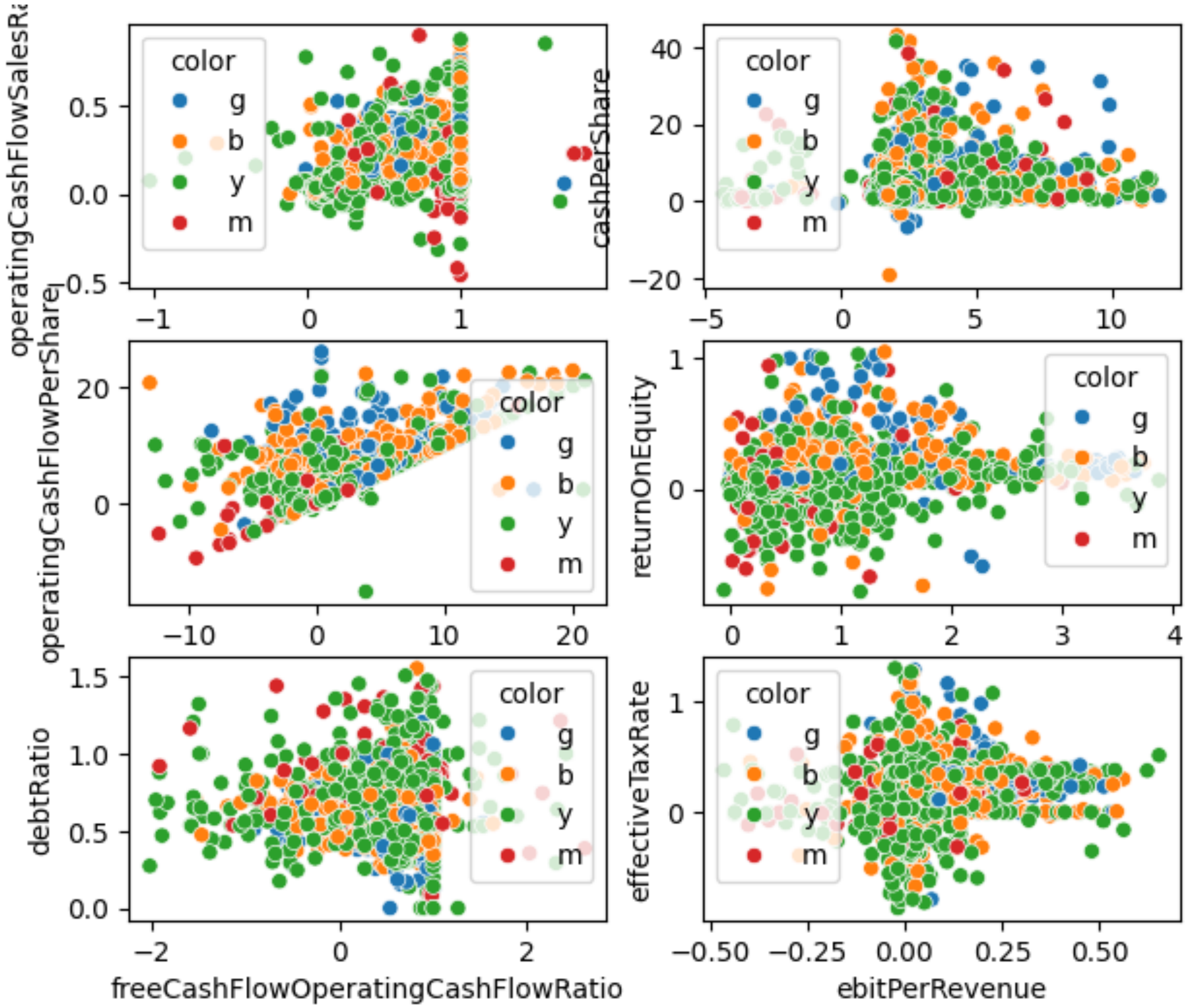
```
df_rating_no_out = df_rating.copy()

for c in df_rating_no_out.columns[6:31]:

    q05 = df_rating_no_out[c].quantile(0.10)
    q95 = df_rating_no_out[c].quantile(0.90)
    iqr = q95 - q05 #Interquartile range
    fence_low  = q05-1.5*iqr
    fence_high = q95+1.5*iqr
    df_rating_no_out.loc[df_rating_no_out[c] > fence_high,c] = df_rating_no_out[c
    df_rating_no_out.loc[df_rating_no_out[c] < fence_low,c] = df_rating_no_out[c]

figure, axes = plt.subplots(nrows=8, ncols=3, figsize=(20,44))

i = 0
j = 0

for c in df_rating_no_out.columns[6:30]:

    sns.boxplot(hue=df_rating_no_out.Rating, y=df_rating_no_out[c], palette="Set3

    if j == 2:
        j=0
        i+=1
    else:
        j+=1
```




daysOfSalesOutstanding
netProfitMargin
pretaxProfitMargin
grossProfitMargin
operatingProfitMargin
returnOnAssets
returnOnCapitalEmployed
returnOnEquity
assetTurnover
fixedAssetTurnover
debtEquityRatio
debtRatio
effectiveTaxRate
owOperatingCashFlowRatio
eCashFlowPerShare


freeCashFl
fre
cashPerShare
companyEquityMultiplier
ebitPerRevenue
enterpriseValueMultiple
operatingCashFlowPerShare
operatingCashFlowSalesRatio

```
df_rating.colors = 'a'
df_rating_no_out.loc[df_rating_no_out['Rating'] == 'Lowest Risk', 'color'] = 'r'
df_rating_no_out.loc[df_rating_no_out['Rating'] == 'Low Risk', 'color'] = 'g'
df_rating_no_out.loc[df_rating_no_out['Rating'] == 'Medium Risk', 'color'] = 'b'
df_rating_no_out.loc[df_rating_no_out['Rating'] == 'High Risk','color'] = 'y'
df_rating_no_out.loc[df_rating_no_out['Rating'] == 'Highest Risk', 'color'] = 'm'
column_list = list(df_rating.columns[6:31])
column_list = sample(column_list,12)
figure, axes = plt.subplots(nrows=3, ncols=2, figsize=(7,6))

i = 0
j = 0

for c in range(0,12, 2):

    sns.scatterplot(x = column_list[c], y=column_list[c+1], hue="color", data=df_

    if i == 1:
        i = 0
        j +=1

    else:
        i+=1
```




```
le = preprocessing.LabelEncoder()
le.fit(df_rating.Sector)
df_rating.Sector = le.transform(df_rating.Sector) # encode sector
le.fit(df_rating.Rating)
df_rating.Rating = le.transform(df_rating.Rating) # encode rating
df_train, df_test = train_test_split(df_rating, test_size=0.2, random_state = 123
X_train, y_train = df_train.iloc[:,5:31], df_train.iloc[:,0]
X_test, y_test = df_test.iloc[:,5:31], df_test.iloc[:,0]
```

Logistic Regression

```
LR_model = LogisticRegression(random_state=1234 , solver='newton-cg')
LR_model = LR_model.fit(X_train, y_train)
y_pred_LR = LR_model.predict(X_test)
Accuracy_LR = metrics.accuracy_score(y_test, y_pred_LR)
print("LR Accuracy:",Accuracy_LR)
```

```
LR Accuracy: 0.45185185185185184
/usr/local/lib/python3.11/dist-packages/sklearn/utils/optimize.py:319: Converg
  warnings.warn(
```

K-Nearest Neighbor

```
KNN_model = KNeighborsClassifier(n_neighbors = 3)
KNN_model.fit(X_train,y_train)
y_pred_KNN = KNN_model.predict(X_test)
Accuracy_KNN = metrics.accuracy_score(y_test, y_pred_KNN)
print("KNN Accuracy:",Accuracy_KNN)
```

```
KNN Accuracy: 0.562962962962963
```

Decision Tree

```
DT_model = DecisionTreeClassifier(random_state=99)
DT_model.fit(X_train,y_train)
y_pred_DT = DT_model.predict(X_test)
Accuracy_DT = metrics.accuracy_score(y_test, y_pred_DT)
print("Decision Tree Accuracy:",Accuracy_DT)
```

```
Decision Tree Accuracy: 0.5580246913580247
```

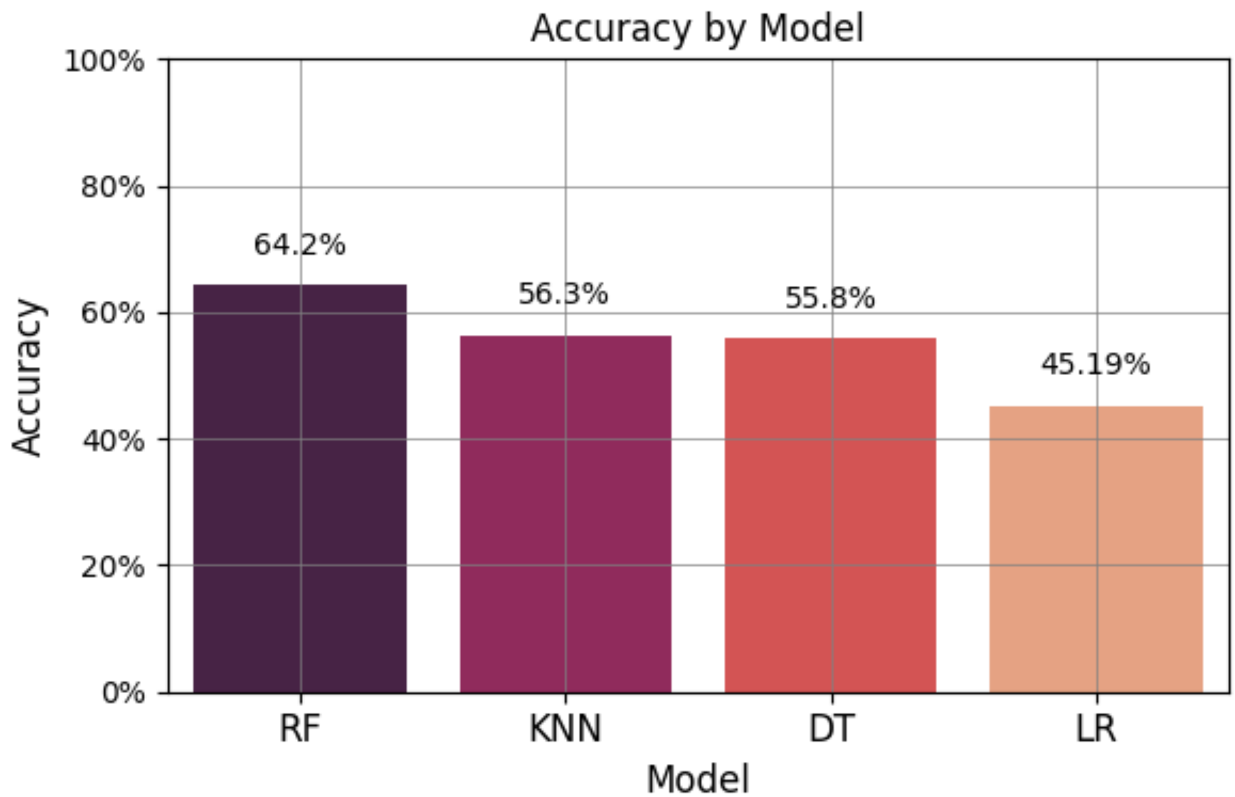
Random Forest

```
RF_model = RandomForestClassifier(random_state=1234)
RF_model.fit(X_train,y_train)
y_pred_RF = RF_model.predict(X_test)
Accuracy_RF = metrics.accuracy_score(y_test, y_pred_RF)
print("RF Accuracy:",Accuracy_RF)
```

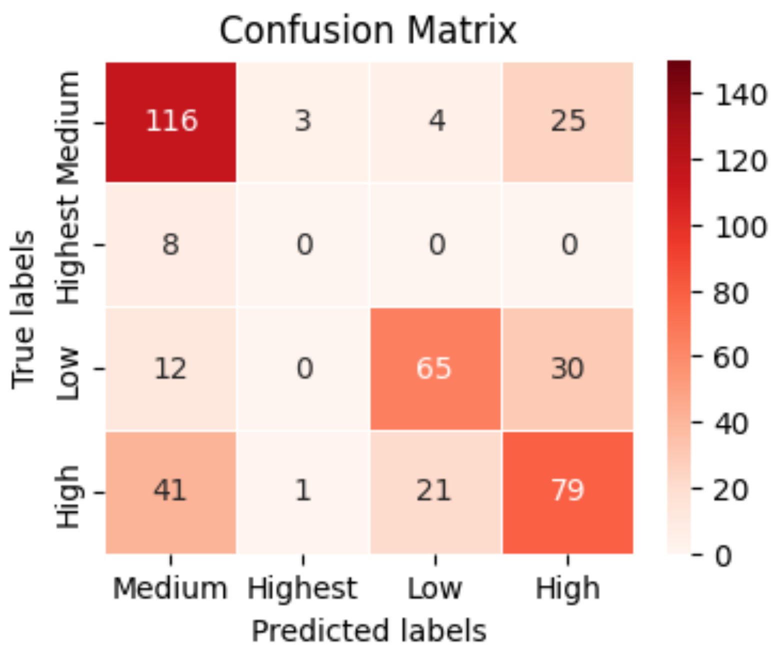
```
RF Accuracy: 0.6419753086419753
```

```
accuracy_list = [Accuracy_DT, Accuracy_RF, Accuracy_KNN, Accuracy_LR]
model_list = ['DT', 'RF', 'KNN', 'LR']

df_accuracy = pd.DataFrame({'Model': model_list, 'Accuracy': accuracy_list})
order = list(df_accuracy.sort_values('Accuracy', ascending=False).Model)
df_accuracy = df_accuracy.sort_values('Accuracy', ascending=False).reset_index().

plt.figure(figsize=(6,4))
# make barplot and sort bars
x = sns.barplot(x='Model', y="Accuracy", data=df_accuracy, order = order, palette
plt.xlabel("Model", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
plt.title("Accuracy by Model", fontsize=12)
plt.grid(linestyle='-', linewidth='0.5', color='grey')
plt.xticks(rotation=0, fontsize=12)
plt.ylim(0,1)
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter(1))

for i in range(len(model_list)):
    plt.text(x = i, y = df_accuracy.loc[i, 'Accuracy'] + 0.05, s = str(round((df_
             fontsize = 10, color='black',horizontalalignment='center')

y_value=['{:,.2f}'.format(x) + '%' for x in ax.get_yticks()]
ax.set_yticklabels(y_value)

plt.tight_layout()
```

```
<ipython-input-15-9a4a60e20f6c>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed ir

  x = sns.barplot(x='Model', y="Accuracy", data=df_accuracy, order = order, pa
<ipython-input-15-9a4a60e20f6c>:24: UserWarning: set_ticklabels() should only
  ax.set_yticklabels(y_value)
```



```
cm = confusion_matrix(y_test, y_pred_RF)
fig, ax = plt.subplots(figsize=(4,3))

sns.heatmap(cm, annot = True, ax = ax, vmin=0, vmax=150, fmt="d", linewidths=.5,
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Medium','Highest', 'Low', 'High'])
ax.yaxis.set_ticklabels(['Medium','Highest', 'Low', 'High']);

plt.show()

print(classification_report(y_test, y_pred_RF, target_names = ['Medium Risk','Hig
```



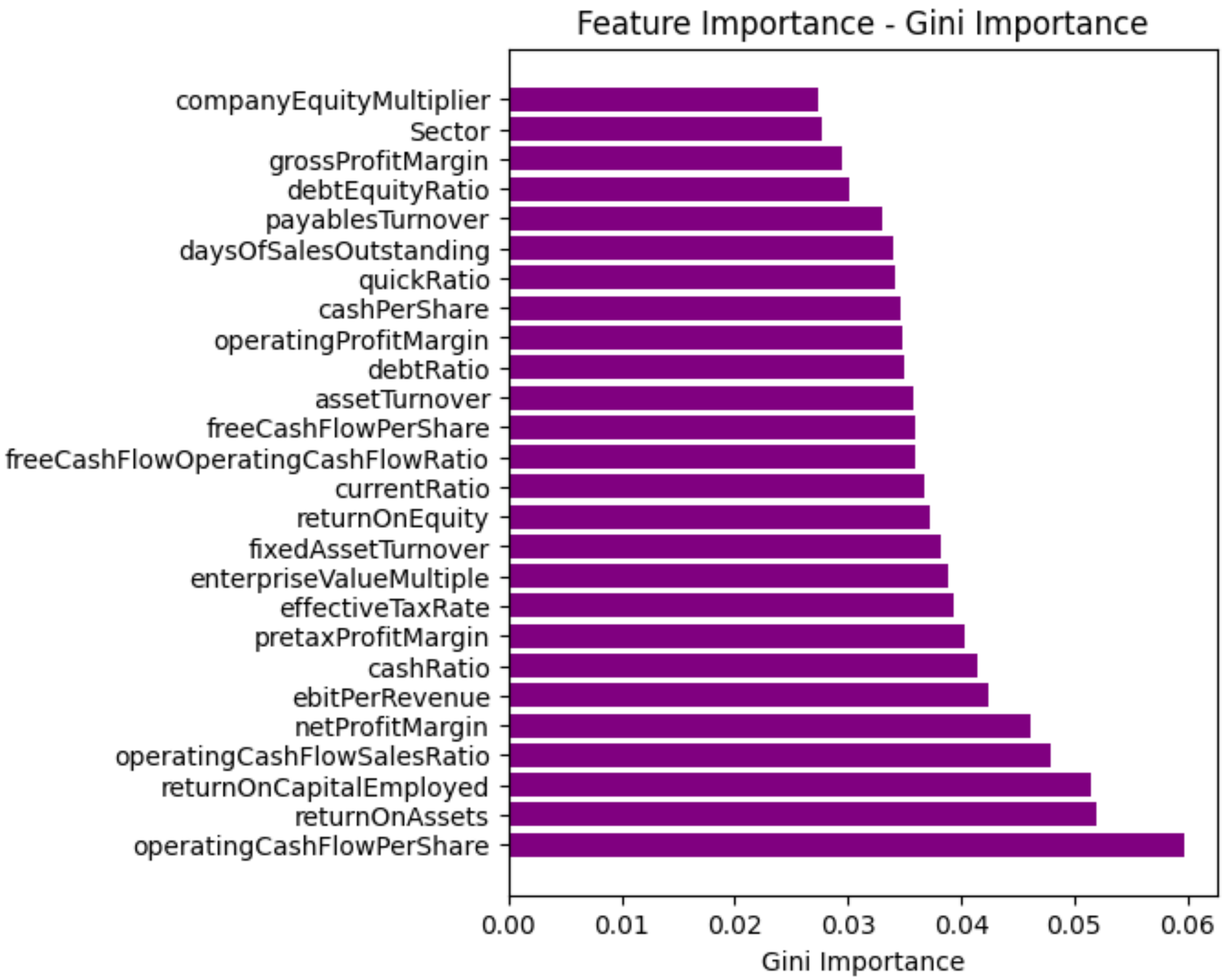
	precision	recall	f1-score	support
Medium Risk	0.66	0.78	0.71	148
Highest Risk	0.00	0.00	0.00	8
Low Risk	0.72	0.61	0.66	107
High Risk	0.59	0.56	0.57	142
accuracy			0.64	405
macro avg	0.49	0.49	0.49	405
weighted avg	0.64	0.64	0.64	405

```
# .iloc[:,5:31]
feature_names = df_rating.columns[5:31]
importances = RF_model.feature_importances_
feature_imp_df = pd.DataFrame({'Feature': feature_names, 'Gini Importance': impor
print(feature_imp_df)
```

```
                                   Feature  Gini Importance
23             operatingCashFlowPerShare         0.059711
9                          returnOnAssets         0.052009
10                returnOnCapitalEmployed         0.051512
24            operatingCashFlowSalesRatio         0.047902
5                         netProfitMargin         0.046187
21                         ebitPerRevenue         0.042412
3                               cashRatio         0.041488
6                      pretaxProfitMargin         0.040319
16                       effectiveTaxRate         0.039306
22                enterpriseValueMultiple         0.038908
13                     fixedAssetTurnover         0.038243
11                         returnOnEquity         0.037204
1                            currentRatio         0.036807
17  freeCashFlowOperatingCashFlowRatio         0.036005
18                    freeCashFlowPerShare         0.035937
12                          assetTurnover         0.035731
15                              debtRatio         0.035030
8                   operatingProfitMargin         0.034786
19                           cashPerShare         0.034606
2                              quickRatio         0.034161
4                  daysOfSalesOutstanding         0.033945
25                       payablesTurnover         0.032977
14                        debtEquityRatio         0.030076
7                       grossProfitMargin         0.029481
0                                  Sector         0.027789
20                 companyEquityMultiplier         0.027467
```

```
plt.figure(figsize=(5, 6))
plt.barh(feature_imp_df['Feature'], feature_imp_df['Gini Importance'], color = 'p
plt.xlabel('Gini Importance')
plt.title('Feature Importance - Gini Importance')
```

```
Text(0.5, 1.0, 'Feature Importance - Gini Importance')
```



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

# ... (your existing code for data loading, splitting, and model training) ...

# Get confusion matrices
# The variable y_pred_KNN was likely intended to be y_pred_knn
cm_knn = confusion_matrix(y_test, y_pred_KNN)
cm_dt = confusion_matrix(y_test, y_pred_DT)  # Similarly changed to y_pred_DT
cm_lr = confusion_matrix(y_test, y_pred_LR)  # Similarly changed to y_pred_LR


# Print confusion matrices
print("Confusion Matrix - KNN:\n", cm_knn)
print("Confusion Matrix - Decision Tree:\n", cm_dt)
print("Confusion Matrix - Logistic Regression:\n", cm_lr)
```

```
Confusion Matrix - KNN:
 [[68  1 37 50]
 [ 6  0  8  3]
 [46  3 28 23]
 [59  0 32 41]]
Confusion Matrix - Decision Tree:
 [[48  2 40 66]
 [ 7  0  6  4]
 [41  3 24 32]
 [49  3 31 49]]
Confusion Matrix - Logistic Regression:
 [[70  1 24 61]
 [ 9  0  3  5]
 [50  1 13 36]
 [74  0 17 41]]
```

Double-click (or enter) to edit

```
from sklearn.metrics import classification_report

# Assuming you have y_test (true labels) and predictions for each model:
# y_pred_knn, y_pred_dt, y_pred_lr
```

```
# Get classification reports
# Changed y_pred_knn to y_pred_KNN, y_pred_dt to y_pred_DT, and y_pred_lr to y_pr
cr_knn = classification_report(y_test, y_pred_KNN)
cr_dt = classification_report(y_test, y_pred_DT)
cr_lr = classification_report(y_test, y_pred_LR)

# Print classification reports
print("Classification Report - KNN:\n", cr_knn)
print("Classification Report - Decision Tree:\n", cr_dt)
print("Classification Report - Logistic Regression:\n", cr_lr)
```

```
Classification Report - KNN:
               precision    recall  f1-score   support

           0       0.38      0.44      0.41       156
           1       0.00      0.00      0.00        17
           2       0.27      0.28      0.27       100
           3       0.35      0.31      0.33       132

    accuracy                           0.34       405
   macro avg       0.25      0.26      0.25       405
weighted avg       0.33      0.34      0.33       405

Classification Report - Decision Tree:
               precision    recall  f1-score   support

           0       0.33      0.31      0.32       156
           1       0.00      0.00      0.00        17
           2       0.24      0.24      0.24       100
           3       0.32      0.37      0.35       132

    accuracy                           0.30       405
   macro avg       0.22      0.23      0.23       405
weighted avg       0.29      0.30      0.29       405

Classification Report - Logistic Regression:
               precision    recall  f1-score   support

           0       0.34      0.45      0.39       156
           1       0.00      0.00      0.00        17
           2       0.23      0.13      0.17       100
           3       0.29      0.31      0.30       132

    accuracy                           0.31       405
   macro avg       0.21      0.22      0.21       405
weighted avg       0.28      0.31      0.29       405
```

```
# Assuming you have your trained Decision Tree and Random Forest models:
# dt (Decision Tree)
# rf (Random Forest)

# Print feature importance for Decision Tree
print("--- Decision Tree ---")
# Changed 'dt' to 'DT_model'
print("Feature Importance:\n", DT_model.feature_importances_)

# Print feature importance for Random Forest
print("--- Random Forest ---")
# Changed 'rf' to 'RF_model'
print("Feature Importance:\n", RF_model.feature_importances_)
```

```
--- Decision Tree ---
Feature Importance:
 [0.02854921 0.0163246  0.03353558 0.06683942 0.03439571 0.05108134
 0.0134878  0.02099582 0.03162205 0.09561945 0.06684021 0.03353687
 0.01674893 0.03095554 0.00684888 0.04106688 0.05298232 0.03806299
 0.0296007  0.04490693 0.01677319 0.01904279 0.0460185  0.05996639
 0.06556633 0.03863156]
--- Random Forest ---
Feature Importance:
 [0.027789   0.03680692 0.03416117 0.04148771 0.03394548 0.04618669
 0.04031871 0.02948145 0.03478629 0.05200939 0.05151222 0.03720358
 0.03573145 0.0382434  0.03007625 0.03502998 0.03930633 0.03600466
 0.0359366  0.03460552 0.02746717 0.04241217 0.03890779 0.05971102
 0.04790215 0.0329769 ]
```