Python For Data Analysis

Polish companies bankruptcy Dataset

- Classification
- Number of instances: 10503
- Number of attributes: 64
- Missing values: Yes

The dataset is about bankruptcy prediction of Polish companies.The bankrupt companies were analyzed in the period 2000-2012, while the still operating companies were evaluated from 2007 to 2013.

Data Cleaning

- Loading files
- Setting columns types
- Removing duplicates

Entrée [2]:
```
# Reading the first data file into dataframe
data = arff.loadarff('datasets/1year.arff')
df = pd.DataFrame(data[0])

# Reading the second data file into dataframe
data2 = arff.loadarff('datasets/2year.arff')
df2 = pd.DataFrame(data2[0])

# Reading the third data file into dataframe
data3 = arff.loadarff('datasets/3year.arff')
df3 = pd.DataFrame(data3[0])

# Reading the fourth data file into dataframe
data4 = arff.loadarff('datasets/4year.arff')
df4 = pd.DataFrame(data4[0])

# Reading the fifth data file into dataframe
data5 = arff.loadarff('datasets/5year.arff')
df5 = pd.DataFrame(data5[0])
```

Entrée [3]:
```
# Making one dataframe by appending the 4 dataframe to the first one
df = df.append([df2, df3, df4, df5], ignore_index=True)
len(df)
```

Out[3]: 43405

Entrée [4]:
```
df.head()
```

Out[4]:

	Attr1	Attr2	Attr3	Attr4	Attr5	Attr6	Attr7	Attr8	Attr9	Attr10	...	Attr56	Attr57	Attr58	Attr59	Attr60	Attr61	Attr62
0	0.200550	0.37951	0.39641	2.0472	32.3510	0.38825	0.249760	1.33050	1.1389	0.50494	...	0.121960	0.39718	0.87804	0.001924	8.4160	5.1372	82.658
1	0.209120	0.49988	0.47225	1.9447	14.7860	0.00000	0.258340	0.99601	1.6996	0.49788	...	0.121300	0.42002	0.85300	0.000000	4.1486	3.2732	107.350
2	0.248660	0.69592	0.26713	1.5548	-1.1523	0.00000	0.309060	0.43695	1.3090	0.30408	...	0.241140	0.81774	0.76599	0.694840	4.9909	3.9510	134.270
3	0.081483	0.30734	0.45879	2.4928	51.9520	0.14988	0.092704	1.86610	1.0571	0.57353	...	0.054015	0.14207	0.94598	0.000000	4.5746	3.6147	86.435
4	0.187320	0.61323	0.22960	1.4063	-7.3128	0.18732	0.187320	0.63070	1.1559	0.38677	...	0.134850	0.48431	0.86515	0.124440	6.3985	4.3158	127.210

5 rows × 65 columns

Entrée [5]:
```
df['class'] = df['class'].astype('int')
```

Entrée [6]:
```
# Drop duplicates
df.drop_duplicates(keep=False, inplace=True)
```

Handle missing values

Entrée [9]:
```
# How many total missing values do we have?
total_cells = np.product(df.shape)
total_missing = missing_values_count.sum()

# percent of data that is missing
print(f"The percent of the missing data is: {total_missing / total_cells * 100:.2f}%")
```

```
The percent of the missing data is: 1.47%
```

Entrée [10]:
```
# Getting the mean values of the dataframe columns
mean_values = df.mean()
```

Entrée [11]:
```
# Filling the missing data by mean values
df = df.fillna(mean_values)
```

In order to fill the nan values of the dataframe first I get the mean value for each column by (df.mean()) method and after that filling the nan value of each column

The second approach is to drop the rows which contains nan values, but I found that it will drop around 23000 records more than the half of the

Entrée [12]:
```
# Getting the number of missing data points per column after filling the nan values
missing_values_count_after_filling_NAN_values = df.isnull().sum()
missing_values_count_after_filling_NAN_values
```

Out[12]:
```
Attr1     0
Attr2     0
Attr3     0
Attr4     0
Attr5     0
          ..
Attr61    0
Attr62    0
Attr63    0
Attr64    0
class     0
Length: 65, dtype: int64
```

Analyze Data

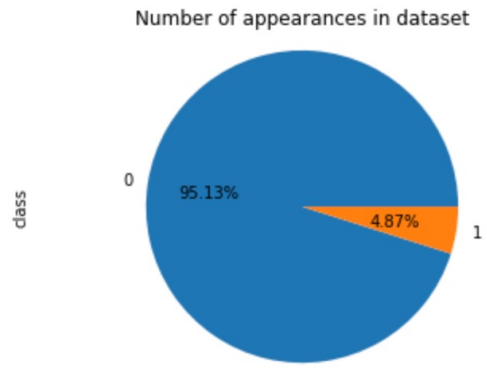
Percentage appearance bankrupt class

```
Entrée [13]: df['class'].value_counts()
```

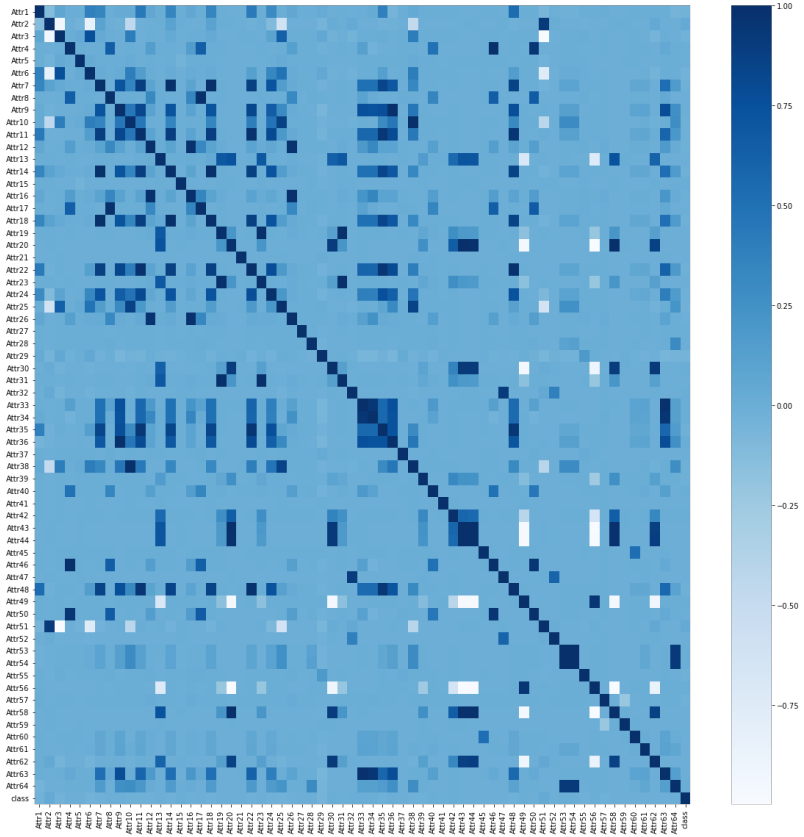
```
Out[13]: 0    40528
         1     2075
         Name: class, dtype: int64
```

```
Entrée [14]: # Make a pie chart
             df['class'].value_counts().plot(kind='pie', autopct='%1.2f%%')
             plt.axis('equal')
             plt.title('Number of appearances in dataset')
```

```
Out[14]: Text(0.5, 1.0, 'Number of appearances in dataset')
```



Correlation Matrix



Entrée [16]:

```
# Return dataframe with pairs of attributes and corresponding Kendall's tau value
# excluding pairs of 2 same attributes:

def corrank(matrix: pd.DataFrame) -> pd.DataFrame:
    return pd.DataFrame([[(i, j), matrix.loc[i, j]] for i, j in list(itertools.combinations(matrix, 2))],
                        columns=['pairs', 'corr'])
```

Entrée [17]:

```
# Below is the sorted list of attributes pairs and their correlation based on Kendall's tau value.

corr_matrix = df[df.columns.difference(['class'])].corr(method="kendall")
corr_matrix = corrank(corr_matrix)
print(corr_matrix.sort_values(by='corr', ascending=False))
```

```
                pairs      corr
308   (Attr14, Attr18)  0.999863
360    (Attr14, Attr7)  0.999839
582    (Attr18, Attr7)  0.999702
1296  (Attr32, Attr52)  0.993930
529    (Attr17, Attr8)  0.970671
...                ...       ...
1275  (Attr32, Attr33) -0.988587
1333  (Attr33, Attr52) -0.994401
2001  (Attr62, Attr63) -0.995808
1709  (Attr44, Attr61) -0.997669
478    (Attr17, Attr2) -0.998801

[2016 rows x 2 columns]
```

Entrée [18]:

```
print(corr_matrix[corr_matrix['corr'].between(0.8, 1, inclusive="both")].sort_values(by='corr', ascending=False)[0:10
```

```
                pairs      corr
308   (Attr14, Attr18)  0.999863
360    (Attr14, Attr7)  0.999839
582    (Attr18, Attr7)  0.999702
1296  (Attr32, Attr52)  0.993930
529    (Attr17, Attr8)  0.970671
123    (Attr10, Attr8)  0.962666
430   (Attr16, Attr26)  0.957736
60     (Attr1, Attr7)   0.940513
4      (Attr1, Attr14)  0.940374
8      (Attr1, Attr18)  0.940238
```

Kendall Tau Correlation

- We keep features which have correlation between [-0.8; 0.8]

Features to keep should be in range [-0.8, 0.8] based on Kendall coefficient values:

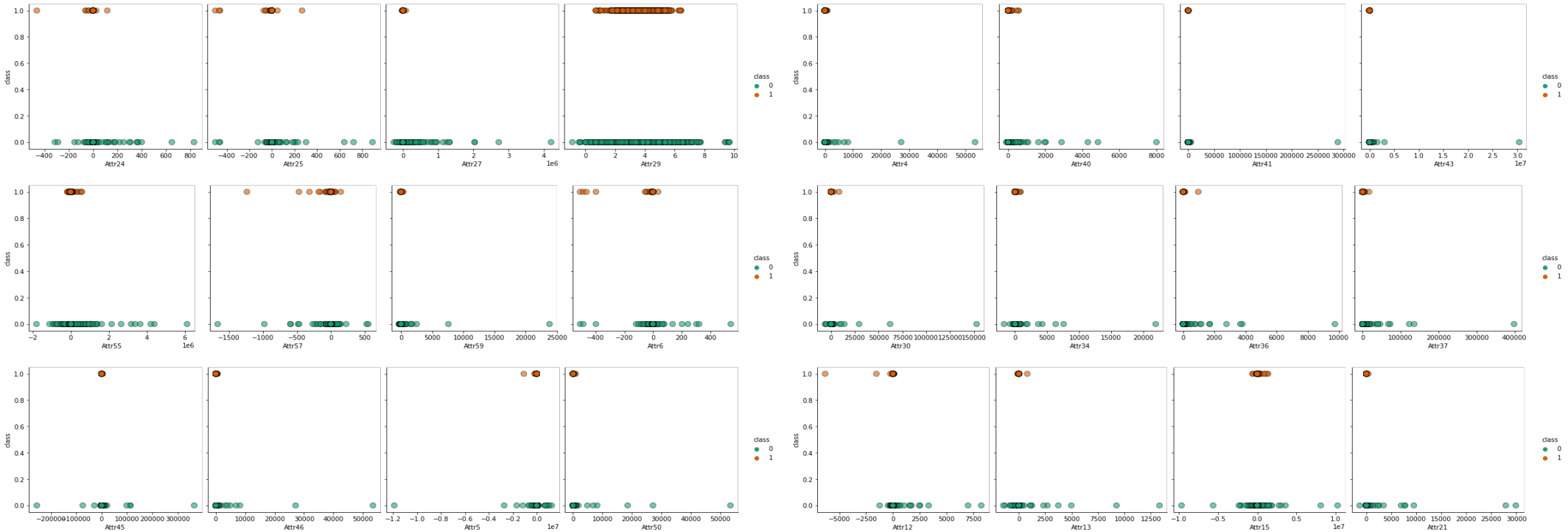
Entrée [20]:

```
columns_name = set(df.columns)

columns_name.difference_update(
    [corr[0] for corr in corr_matrix[corr_matrix['corr'].between(-1, -0.8, inclusive="neither")]['pairs']])
columns_name.difference_update(
    [corr[1] for corr in corr_matrix[corr_matrix['corr'].between(-1, -0.8, inclusive="neither")]['pairs']])
columns_name.difference_update(
    [corr[0] for corr in corr_matrix[corr_matrix['corr'].between(0.8, 1, inclusive="neither")]['pairs']])
columns_name.difference_update(
    [corr[1] for corr in corr_matrix[corr_matrix['corr'].between(0.8, 1, inclusive="neither")]['pairs']])
columns_name = list(columns_name)
columns_name
```

Out[20]:

```
['Attr9',
 'Attr46',
 'Attr36',
 'Attr59',
 'Attr43',
 'Attr57',
 'Attr27',
 'Attr45',
 'Attr40',
 'Attr21',
 'Attr34',
 'Attr6',
 'Attr12',
 'Attr24',
 'Attr37',
 'Attr55',
 'Attr29',
 'Attr41',
 'Attr50',
 'Attr4',
 'Attr13',
 'class',
 'Attr64',
 'Attr15',
 'Attr5',
 'Attr25',
 'Attr30']
```

Pairplot


class
0
1
Attr24
Attr25
Attr27
Attr29
Attr4
Attr40
Attr41
Attr43
Attr55
Attr57
Attr59
Attr6
Attr30
Attr34
Attr36
Attr37
Attr45
Attr46
Attr5
Attr50
Attr12
Attr13
Attr15
Attr21

Dealing with imbalanced data (SMOTE)

Dealing with imbalanced data

Entrée [28]:
```
X = df[df.columns.difference(['class'])]
Y = df['class']
```

Entrée [29]:
```
sm = SMOTE(random_state=111)
X_smote, y_smote = sm.fit_resample(X, Y)
```

Entrée [30]:
```
# Splitting the data after balancing
X_train_smote, X_test_smote, y_train_smote, y_test_smote = train_test_split(X_smote, y_smote, test_size=0.3)
```

Entrée [31]:
```
# Normalizing the features
min_max_scaler = preprocessing.MinMaxScaler()
X_train_smote_scaled = min_max_scaler.fit_transform(X_train_smote)
y_test_smote_scaled = min_max_scaler.fit_transform(X_test_smote)
```

Modeling

Train / Validate model

Entrée [32]:

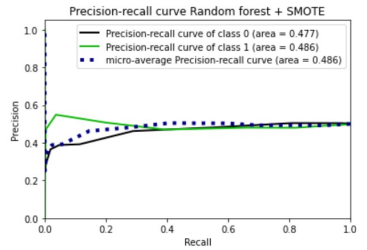
```
def save_model(model: any, features: np.ndarray, label: pd.Series, out_dir: str) -> None:
    model.fit(features, label)
    pickle.dump(model, open(out_dir, 'wb'))


def validate(in_dir: str, X_test: np.ndarray, y_test: pd.Series, title: str) -> None:
    model = pickle.load(open(in_dir, 'rb'))
    y_probas = model.predict_proba(X_test)
    plot_precision_recall(
        y_test,
        y_probas,
        title=f"Precision-recall curve {title}"
    )
    plt.show()
```

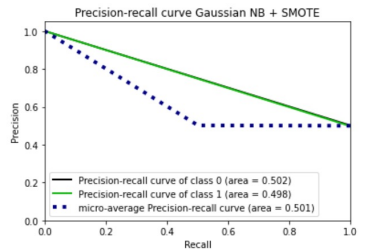
Different Models

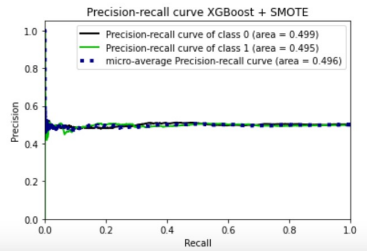
- Random forest classifier with default parameters run on oversampled and imputed data (SMOTE) leads to micro-average AUC around 0.502, the algorithm performs poorly.
- Naive Bayes classifier performs poorly on oversampled and imputed data (SMOTE), AUC = 0.5
- XGBoost classifier also performs poorly to the data oversampled with SMOTE and imputed data leads to the mirco-average precision-recall rate only 0.474, and there is a high risk that this model is underfitted

Entrée [34]:
```
train_models = [
    (RandomForestClassifier(), X_train_smote_scaled, y_train_smote, 'models/random_forest_SMOTE.pkl'),
    (GaussianNB(), X_train_smote_scaled, y_train_smote, 'models/gaussian_nb_SMOTE.pkl'),
    (xgboost.XGBClassifier(), X_train_smote_scaled, y_train_smote, 'models/xgboost_SMOTE.pkl')
]

for clr, features, label, out_dir in train_models:
    save_model(clr, features, label, out_dir)
```

Entrée [35]:
```
test_models = [
    ('models/random_forest_SMOTE.pkl', y_test_smote_scaled, y_test_smote, 'Random forest + SMOTE'),
    ('models/gaussian_nb_SMOTE.pkl', y_test_smote_scaled, y_test_smote, 'Gaussian NB + SMOTE'),
    ('models/xgboost_SMOTE.pkl', y_test_smote_scaled, y_test_smote, 'XGBoost + SMOTE')
]

for in_dir, X_test, y_test, title in test_models:
    validate(in_dir, X_test, y_test, title)
```







Grid Search

- As it is seen from the two plots, AdaBoost classifier is likely to be overfitted on oversampled (SMOTE) data.
- The best classifier is random forest that was fitted on oversampled (SMOTE) data with preliminarly dropped missing values, area under the precision-recall curve is 0.965

Entrée [36]:
```
# Run grid search on random forest
param_grid = {
    'max_depth': [3, 5, 7, 8]
}
```

Entrée [37]:
```
CV_rfc = GridSearchCV(estimator=RandomForestClassifier(), param_grid=param_grid, cv=5)
CV_rfc.fit(X_smote, y_smote)

CV_rfc.best_params_
```

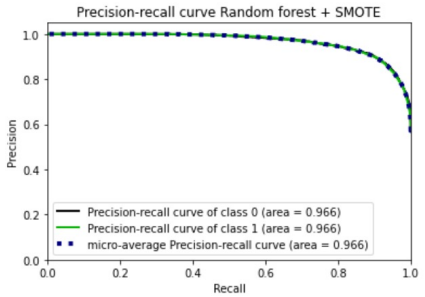
Out[37]: {'max_depth': 8}

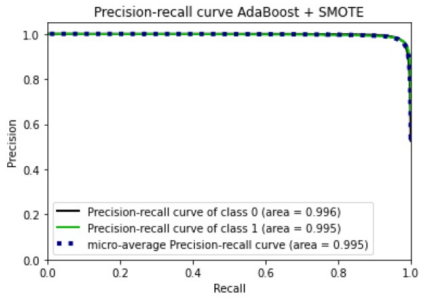
Entrée [38]:
```
# Run grid search on AdaBoost

param_grid = {'base_estimator__max_depth': [3, 5, 7, 10],
              'base_estimator': [DecisionTreeClassifier()]}

CV_rfc = GridSearchCV(estimator=AdaBoostClassifier(), param_grid=param_grid, cv=5)
CV_rfc.fit(X_smote, y_smote)

CV_rfc.best_params_
```

Out[38]:
```
{'base_estimator': DecisionTreeClassifier(max_depth=10),
 'base_estimator__max_depth': 10}
```

Entrée [39]:
```
train_models = [
    (RandomForestClassifier(max_depth=8), X_train_smote, y_train_smote, 'models/random_forest2_SMOTE.pkl'),
    (AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=10)), X_train_smote, y_train_smote,
     'models/adaboost_SMOTE.pkl')
]

for clr, features, label, out_dir in train_models:
    save_model(clr, features, label, out_dir)
```

Entrée [40]:
```
test_models = [
    ('models/random_forest2_SMOTE.pkl', X_test_smote, y_test_smote, 'Random forest + SMOTE'),
    ('models/adaboost_SMOTE.pkl', X_test_smote, y_test_smote, 'AdaBoost + SMOTE')
]

for in_dir, X_test, y_test, title in test_models:
    validate(in_dir, X_test, y_test, title)
```





API

Empty Fields

Bankruptcy Prediction

Gross Profit / Short-Term Liabilities
Gross Profit / Short-Term Liabilities

(Gross Profit + Depreciation) / Sales
(Gross Profit + Depreciation) / Sales

(Total Liabilities * 365) / (Gross Profit + Depreciation)
(Total Liabilities * 365) / (Gross Profit + Depreciation)

sales (n) / sales (n-1)
sales (n) / sales (n-1)

Gross Profit (In 3 Years) / Total Assets
Gross Profit (In 3 Years) / Total Assets

(Equity - Share Capital) / Total Assets
(Equity - Share Capital) / Total Assets

Profit On Operating Activities / Financial Expenses
Profit On Operating Activities / Financial Expenses

Logarithm Of Total Assets
Logarithm Of Total Assets

(Total Liabilities - Cash) / Sales
(Total Liabilities - Cash) / Sales

Operating Expenses / Total Liabilities
Operating Expenses / Total Liabilities

Total Sales / Total Assets
Total Sales / Total Assets

(Current Assets - Inventories) / Long-Term Liabilities
(Current Assets - Inventories) / Long-Term Liabilities

Current Assets / Short-Term Liabilities
Current Assets / Short-Term Liabilities

(Current Assets - Inventory - Receivables) / Short-Term Liabilities
(Current Assets - Inventory - Receivables) / Short-Term Liabilities

Total Liabilities / ((Profit On Operating Activities + Depreciation) * (12/365))
Total Liabilities / ((Profit On Operating Activities + Depreciation) * (12/365))

Rotation Receivables + Inventory Turnover In Days
Rotation Receivables + Inventory Turnover In Days

Net Profit / Inventory
Net Profit / Inventory

(Current Assets - Inventory) / Short-Term Liabilities
(Current Assets - Inventory) / Short-Term Liabilities

[(Cash + Short-Term Securities + Receivables - Short-Term Liabilities) / (Operating Expenses - Depreciation)] * 365
[(Cash + Short-Term Securities + Receivables - Short-Term Liabilities) / (Operating

Current Assets / Total Liabilities
Current Assets / Total Liabilities

Working Capital
Working Capital

(Current Assets - Inventory - Short-Term Liabilities) / (Sales - Gross Profit - Depreciation)
(Current Assets - Inventory - Short-Term Liabilities) / (Sales - Gross Profit - Depreci

Long-Term Liabilities / Equity
Long-Term Liabilities / Equity

Retained Earnings / Total Assets
Retained Earnings / Total Assets

Sales / Fixed Assets
Sales / Fixed Assets

Sales / Total Assets
Sales / Total Assets

Fill all fields

Clear Values

Missing values will be replaced with random values between the minimum and the maximum value of this column in the complete dataset

Fields Filled

Bankruptcy Prediction

Gross Profit / Short-Term Liabilities
6004,0

(Gross Profit + Depreciation) / Sales
991,12

(Total Liabilities * 365) / (Gross Profit + Depreciation)
9551226,13

sales (n) / sales (n-1)
24515,08

Gross Profit (In 3 Years) / Total Assets
-439,54

(Equity - Share Capital) / Total Assets
-202,99

Profit On Operating Activities / Financial Expenses
2764885,06

Logarithm Of Total Assets
0,32

(Total Liabilities - Cash) / Sales
83652,07

Operating Expenses / Total Liabilities
6443,09

Total Sales / Total Assets
3756,98

(Current Assets - Inventories) / Long-Term Liabilities
289028,01

Current Assets / Short-Term Liabilities
22194,83

(Current Assets - Inventory - Receivables) / Short-Term Liabilities
7367,02

Total Liabilities / ((Profit On Operating Activities + Depreciation) * (12/365))
230798,56

Rotation Receivables + Inventory Turnover In Days
18753400,36

Net Profit / Inventory
127105,44

(Current Assets - Inventory) / Short-Term Liabilities
35131,3

[(Cash + Short-Term Securities + Receivables - Short-Term Liabilities) / (Operating Expenses - Depreciation)] * 365
-6951055,08

Current Assets / Total Liabilities
34817,39

Working Capital
5419386,52

(Current Assets - Inventory - Short-Term Liabilities) / (Sales - Gross Profit - Depreciation)
-1172,83

Long-Term Liabilities / Equity
14493,32

Retained Earnings / Total Assets
230,84

Sales / Fixed Assets
7361,93

Sales / Total Assets
145,65

Predict

Clear Values

Missing values will be replaced with random values between the minimum and the maximum value of this column in the complete dataset

Bankrupt

Bankruptcy Prediction

The Company will bankrupt

Net Profit / Total Assets

7896.33

Equity / Total Assets

7603.28

Gross Profit / Short-Term Liabilities

3796158.29

(Gross Profit + Depreciation) / Sales

11561.02

(Total Liabilities * 365) / (Gross Profit + Depreciation)

-371.33

(Gross Profit + Depreciation) / Total Liabilities

294.16

Gross Profit / Sales

1702936.25

(Inventory * 365) / Sales

2.36

Sales(n) / Sales(n-1)

66895.59

Profit On Operating Activities / Total Assets

20622.55

No Bankrupt

Bankruptcy Prediction

The Company will not bankrupt

Net Profit / Total Assets

-607.93

Equity / Total Assets

892.55

Gross Profit / Short-Term Liabilities

3249336.91

(Gross Profit + Depreciation) / Sales

21982.9

(Total Liabilities * 365) / (Gross Profit + Depreciation)

316.73

(Gross Profit + Depreciation) / Total Liabilities

-441.7

Gross Profit / Sales

2008180.31

(Inventory * 365) / Sales

3.41