Bankruptcy Prediction

Import packages

In [1]:
```
import os
from scipy.io import arff
import pandas as pd
import numpy as np
pd.set_option('max_columns', None)

import plotly.express as px
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler, normalize, MinMaxScaler

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoo
from sklearn.decomposition import PCA
from lightgbm import LGBMClassifier
from random import randint

from sklearn.model_selection import StratifiedKFold, StratifiedShuffleSplit
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report, confusion_matrix, f1_score,accuracy_

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 12, 7

from sklearn.naive_bayes import GaussianNB

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.utils import resample
from imblearn.over_sampling import SMOTE

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectPercentile

from sklearn.metrics import classification_report
from sklearn.metrics import average_precision_score
```

```python
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve

from sklearn.tree import export_graphviz
import graphviz
from sklearn.pipeline import make_pipeline

import xgboost as XGB
import lightgbm as lgb

from imblearn.over_sampling import SMOTE

import warnings

#Ignore warnings
warnings.filterwarnings(action='ignore')
#Import data set
```

Read the input files

In [7]:

```python
input_path = '../BankruptcyData/data/'
filename_list = []
for dirname, _, filenames in os.walk(input_path):
    for filename in filenames:
        if filename.endswith('.arff'):
            filename_list.append(os.path.join(dirname, filename))
        print(os.path.join(dirname, filename))
```

```
/content/drive/MyDrive/Assignment/data/1year.arff
/content/drive/MyDrive/Assignment/data/2year.arff
/content/drive/MyDrive/Assignment/data/3year.arff
/content/drive/MyDrive/Assignment/data/4year.arff
/content/drive/MyDrive/Assignment/data/attribute_information.txt
/content/drive/MyDrive/Assignment/data/5year.arff
```

Reading Column names

In [8]:

```python
with open(os.path.join(input_path, 'attribute_information.txt'), 'r') as f_in:
    attribute_data = f_in.readlines()
column_names = {}
for number, line in zip(range(len(attribute_data)), attribute_data):
    column_names[f'Attr{number+1}'] = line.split('\t', 1)[1].replace('\n', '')
```

In [9]:

```python
# all data files from each path stored in each dictionary
def data_reading(filename_list, categorical_column='class'):
    df = None
    for i, file_name in enumerate(filename_list):
        print("Reading file: ", file_name)

        data = arff.loadarff(file_name)
        curr_table = pd.DataFrame(data[0]).rename(columns = column_names, inplace=False
        curr_table.iloc[:, :-1] = curr_table.iloc[:, :-1].astype(np.float64)
        curr_table[categorical_column] = curr_table[categorical_column].astype(np.int64
```

```
        # fill missing value with median
        imp_mean = SimpleImputer(missing_values=np.nan, strategy='median')
        curr_table.loc[:, curr_table.columns!=categorical_column] = imp_mean.fit_transf
        curr_table['year'] = i + 1
        # save
        if df is None:
            df = curr_table.copy()
        else:
            df = df.append(curr_table, ignore_index = True)

    return df
```

In [10]:

```
df = data_reading(filename_list)
df = df[df['class']==1].drop(columns=['class'])
```

```
Reading file:  /content/drive/MyDrive/Assignment/data/1year.arff
Reading file:  /content/drive/MyDrive/Assignment/data/2year.arff
Reading file:  /content/drive/MyDrive/Assignment/data/3year.arff
Reading file:  /content/drive/MyDrive/Assignment/data/4year.arff
Reading file:  /content/drive/MyDrive/Assignment/data/5year.arff
```

Combine data

Data Exploration

In [11]:

```
analyze_df = df.copy()
analyze_df.describe()
```

Out[11]:

	net profit / total assets	total liabilities / total assets	working capital / total assets	current assets / short-term liabilities	[(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365	retained earnings / total assets	EBIT / total assets	b(
count	2091.000000	2091.000000	2091.000000	2091.000000	2.091000e+03	2091.000000	2091.000000	2(
mean	-0.319379	1.504957	-0.734570	4.138845	-7.462160e+02	-1.165242	-0.310742	
std	10.279370	17.399080	17.341001	30.144828	2.402720e+04	20.491889	10.279888	
min	-463.890000	0.000000	-479.960000	-0.403110	-1.076400e+06	-508.410000	-463.890000	
25%	-0.116210	0.455540	-0.146850	0.750725	-9.152750e+01	-0.126250	-0.116335	
50%	0.003463	0.668450	0.038857	1.084900	-3.666300e+01	0.000000	0.005142	
75%	0.050320	0.871610	0.245840	1.636950	4.928500e+00	0.000000	0.059387	
max	20.481000	480.960000	1.000000	916.500000	8.197100e+03	35.551000	20.481000	2{

In [12]:

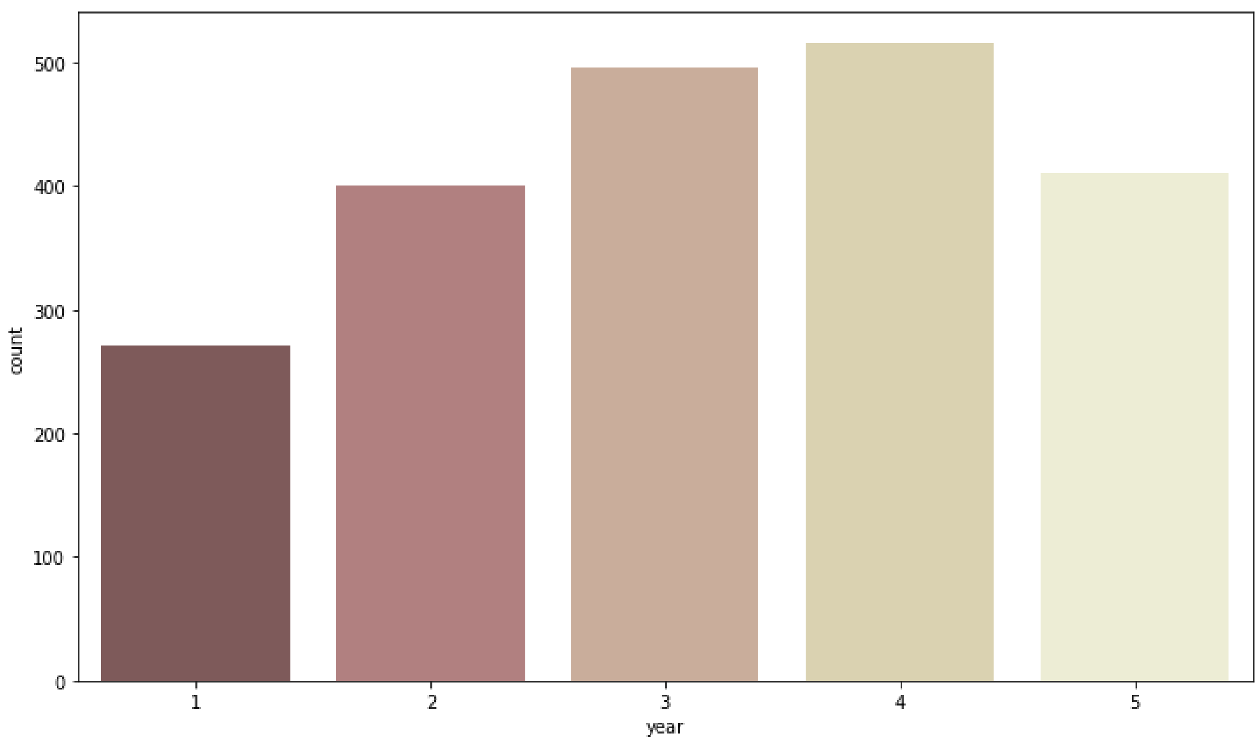
```
colors = ['Accent', 'Accent_r', 'Blues', 'Blues_r', 'BrBG', 'BrBG_r', 'BuGn', 'BuGn_r',
```

```
            'CMRmap_r', 'Dark2', 'Dark2_r', 'GnBu', 'GnBu_r', 'Greens', 'Greens_r', 'Grey
            'Oranges', 'Oranges_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastel1', 'P
            'PiYG', 'PiYG_r', 'PuBu', 'PuBuGn', 'PuBuGn_r', 'PuBu_r', 'PuOr', 'PuOr_r', '
            'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r', 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_r', '
            'Set1', 'Set1_r', 'Set2', 'Set2_r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_r
            'YlGnBu_r', 'YlGn_r', 'YlOrBr', 'YlOrBr_r', 'YlOrRd', 'YlOrRd_r', 'afmhot', '
            'binary_r', 'bone', 'bone_r', 'brg', 'brg_r', 'bwr', 'bwr_r', 'cividis', 'civ
            'coolwarm_r', 'copper', 'copper_r', 'crest', 'crest_r', 'cubehelix', 'cubehel
            'flare_r', 'gist_earth', 'gist_earth_r', 'gist_gray', 'gist_gray_r', 'gist_he
            'gist_ncar_r', 'gist_rainbow', 'gist_rainbow_r', 'gist_stern', 'gist_stern_r'
            'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'hot', 'hot_r', 'hsv
            'inferno', 'inferno_r', 'jet', 'jet_r', 'magma', 'magma_r', 'mako', 'mako_r',
            'ocean', 'ocean_r', 'pink', 'pink_r', 'plasma', 'plasma_r', 'prism', 'prism_r
            'rocket_r', 'seismic', 'seismic_r', 'spring', 'spring_r', 'summer', 'summer_r
            'tab20_r', 'tab20b', 'tab20b_r', 'tab20c', 'tab20c_r', 'terrain', 'terrain_r'
            'twilight_r', 'twilight_shifted', 'twilight_shifted_r', 'viridis', 'viridis_r
value = randint(0, len(colors)-1)
```

In [13]:

```
sns.countplot('year',data=analyze_df,palette = colors[value])
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2505ac6d90>



Are there any extreme outlier columns?

In [14]:

```
test_df = analyze_df.describe().loc['std',:]
extreme_cols = test_df[test_df>10000].index.values
extreme_cols
```

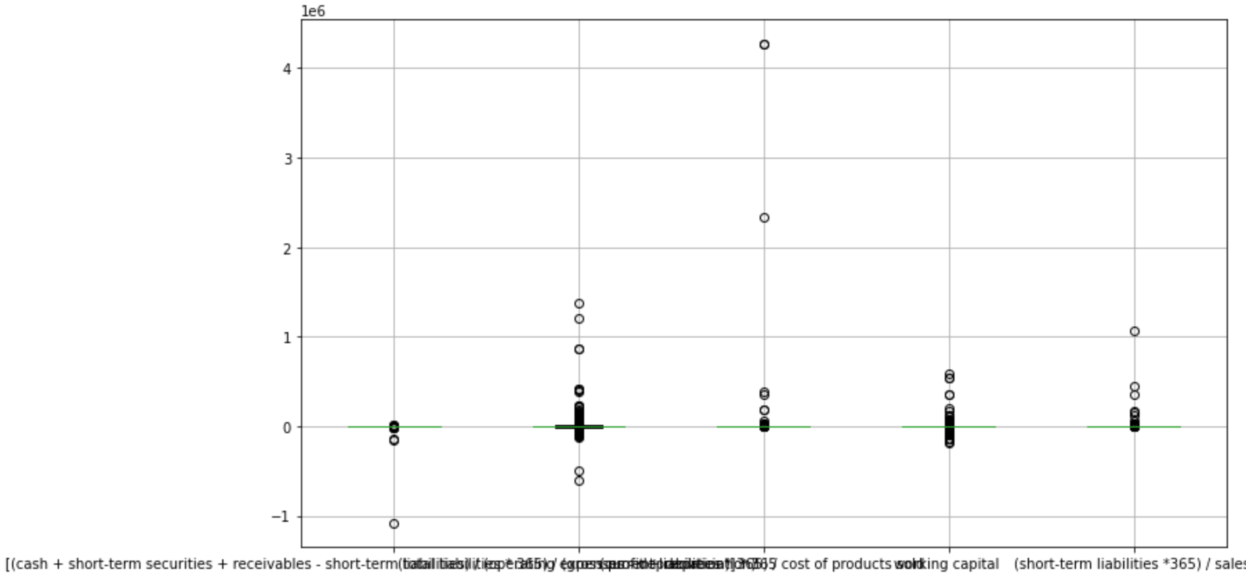
Out[14]:

```
array(['[(cash + short-term securities + receivables - short-term liabilities) / (operat
ing expenses - depreciation)] * 365 ',
       '(total liabilities * 365) / (gross profit + depreciation) ',
       '(current liabilities * 365) / cost of products sold ',
       'working capital ', '(short-term liabilities *365) / sales '],
      dtype=object)
```

In [15]:

```
analyze_df[extreme_cols].boxplot()
```

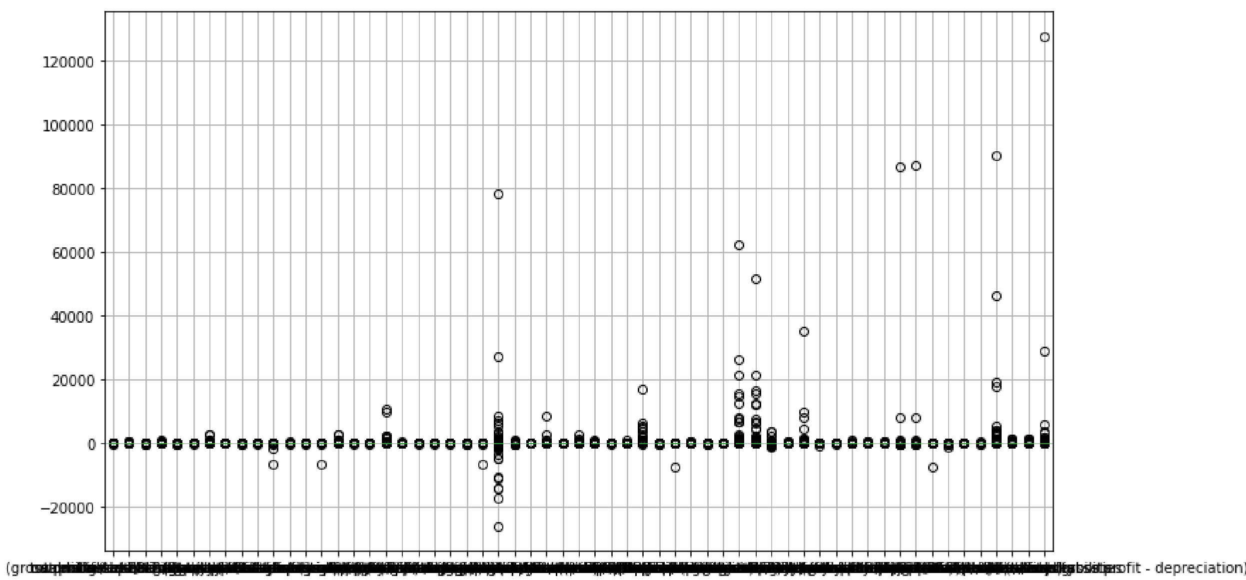
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7f25060e0d50>



In [16]:

```
regular_cols = [col for col in analyze_df.columns[:-1] if col not in extreme_cols]
analyze_df[regular_cols].boxplot()
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7f250608e150>



In [17]:

```
# remove extreme outliers
analyze_df = analyze_df.drop(columns=extreme_cols)
```

Check correlated features

In [18]:

```
correlation_matrix = analyze_df.corr()
correlation_matrix.style.background_gradient(sns.light_palette('red', as_cmap=True))
```

Out[18]:

	net profit / total assets	total liabilities / total assets	working capital / total assets	current assets / short-term liabilities	retained earnings / total assets	EBIT / total assets	book value of equity / total liabilities	sales / total assets	ec
net profit / total assets	1.000000	-0.043631	0.044392	0.000698	0.514723	0.999992	-0.000939	-0.001912	-0.2
total liabilities / total assets	-0.043631	1.000000	-0.996781	-0.007873	-0.861851	-0.043680	-0.006432	-0.011207	-0.9
working capital / total assets	0.044392	-0.996781	1.000000	0.011063	0.858797	0.044443	0.004935	0.011394	0.9
current assets / short-term liabilities	0.000698	-0.007873	0.011063	1.000000	0.005233	0.000369	0.350081	-0.015037	0.0
retained earnings / total assets	0.514723	-0.861851	0.858797	0.005233	1.000000	0.514746	0.004312	0.036750	0.6
EBIT / total assets	0.999992	-0.043680	0.044443	0.000369	0.514746	1.000000	-0.001079	-0.000945	-0.2
book value of equity / total liabilities	-0.000939	-0.006432	0.004935	0.350081	0.004312	-0.001079	1.000000	-0.024150	0.0
sales / total assets	-0.001912	-0.011207	0.011394	-0.015037	0.036750	-0.000945	-0.024150	1.000000	0.0
equity / total assets	-0.270719	-0.944904	0.941494	0.006814	0.661218	-0.270667	0.005632	0.063030	1.0
(gross profit + extraordinary items + financial expenses) / total assets	0.991181	0.029446	-0.028858	-0.000038	0.462782	0.991190	0.001687	0.004973	-0.3
gross profit / short-term liabilities	0.005669	0.001143	-0.001387	-0.129211	-0.000622	0.005713	-0.621938	0.010561	-0.0
(gross profit + depreciation) / sales	0.013449	0.029075	-0.029113	-0.005377	-0.031087	0.013450	-0.001771	-0.007818	-0.0
(gross profit + interest) / total assets	0.999992	-0.043680	0.044443	0.000369	0.514746	1.000000	-0.001079	-0.000945	-0.2

	net profit / total assets	total liabilities / total assets	working capital / total assets	current assets / short-term liabilities	retained earnings / total assets	EBIT / total assets	book value of equity / total liabilities	sales / total assets	ec
(gross profit + depreciation) / total liabilities	0.004444	0.001722	-0.000825	-0.034821	-0.001059	0.004485	-0.640452	0.011570	-0.0
total assets / total liabilities	-0.000927	-0.006482	0.004968	0.349994	0.004348	-0.001066	0.999980	-0.024352	0.0
gross profit / total assets	0.999992	-0.043680	0.044443	0.000369	0.514746	1.000000	-0.001079	-0.000945	-0.2
gross profit / sales	0.050852	0.116537	-0.117317	-0.012102	-0.125394	0.050922	-0.001819	0.008105	-0.1
(inventory * 365) / sales	0.001690	-0.008330	0.008439	0.021041	0.008128	0.001514	0.004224	-0.051297	0.0
sales (n) / sales (n-1)	0.001269	0.000019	-0.000345	-0.004515	0.000482	0.001261	-0.003711	-0.004958	-0.0
profit on operating activities / total assets	0.988608	0.008284	-0.007717	-0.000438	0.486691	0.988613	0.001474	0.003373	-0.3
net profit / sales	0.050831	0.116618	-0.117404	-0.010687	-0.125472	0.050863	-0.001365	0.008050	-0.1
gross profit (in 3 years) / total assets	0.950834	-0.181990	0.195163	0.001723	0.620099	0.950932	-0.000908	0.027285	-0.1
(equity - share capital) / total assets	-0.257971	-0.948931	0.944652	0.003664	0.673178	-0.257913	0.004533	0.040730	0.9
(net profit + depreciation) / total liabilities	0.004536	0.001676	-0.000770	-0.030276	-0.001031	0.004480	-0.637290	0.011200	-0.0
profit on operating activities / financial expenses	0.003788	-0.000573	0.000901	-0.000814	0.001363	0.003898	-0.000591	0.009316	0.0
working capital / fixed assets	0.010674	-0.015534	0.017810	0.042224	0.008803	0.010845	0.003633	-0.002200	0.0

	net profit / total assets	total liabilities / total assets	working capital / total assets	current assets / short-term liabilities	retained earnings / total assets	EBIT / total assets	book value of equity / total liabilities	sales / total assets	ec
logarithm of total assets	0.082614	-0.143391	0.130718	-0.139830	0.167954	0.082522	-0.005816	-0.230733	0.1
(total liabilities - cash) / sales	0.005692	0.275108	-0.274592	-0.004978	-0.245731	0.005658	-0.003011	-0.018623	-0.2
(gross profit + interest) / sales	0.051869	0.120034	-0.120901	-0.014105	-0.129059	0.051927	-0.002953	0.000627	-0.1
operating expenses / short-term liabilities	-0.002670	-0.005539	0.009289	0.629440	0.003846	-0.002698	0.147066	0.036306	0.0
operating expenses / total liabilities	0.003204	-0.011580	0.011417	0.543066	0.008385	0.003140	0.318413	0.062837	0.0
profit on sales / total assets	0.986957	0.000706	0.000118	0.000040	0.494300	0.986955	0.001591	0.000350	-0.3
total sales / total assets	-0.965586	0.037452	-0.038560	-0.006218	-0.516246	-0.965355	-0.006582	0.170603	0.2
(current assets - inventories) / long-term liabilities	0.002653	-0.004126	0.003390	0.013308	0.004185	0.002672	0.128711	-0.017450	0.0
constant capital / total assets	-0.270606	-0.944849	0.941578	0.008150	0.661322	-0.270556	0.005259	0.062402	0.9
profit on sales / sales	-0.000640	0.000376	-0.000511	0.002543	-0.000717	-0.000620	0.001521	0.011463	-0.0
(current assets - inventory - receivables) / short-term liabilities	-0.000453	-0.005220	0.008593	0.960960	0.003626	-0.000380	0.326829	-0.011478	0.0

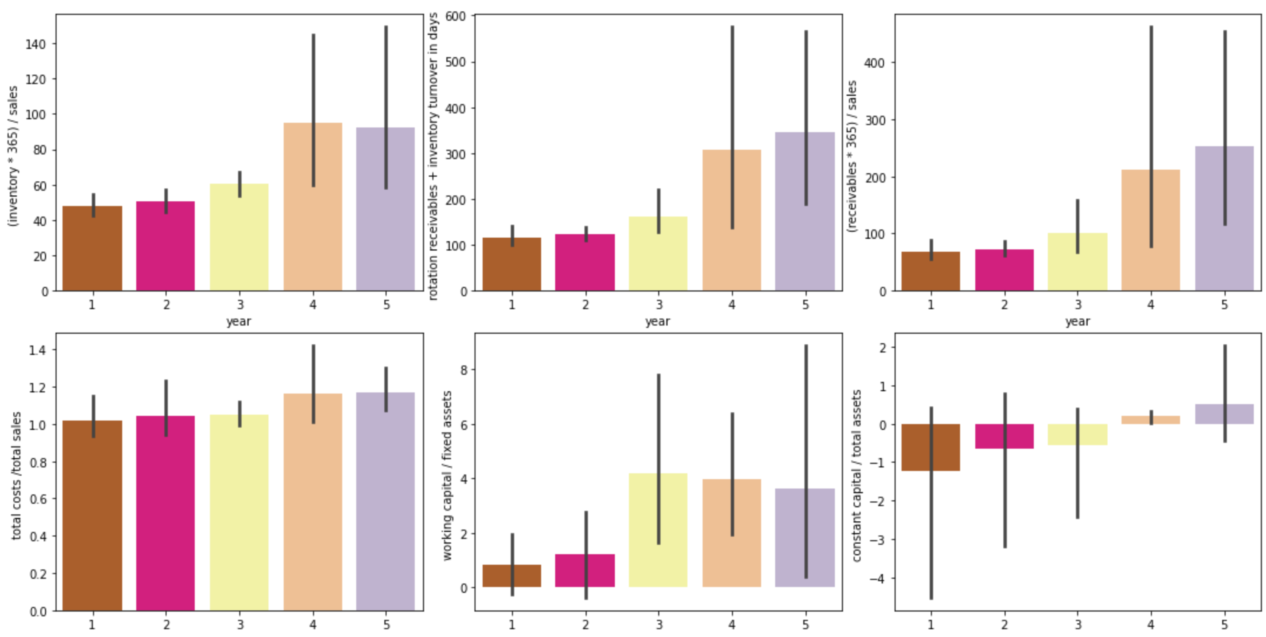
	net profit / total assets	total liabilities / total assets	working capital / total assets	current assets / short-term liabilities	retained earnings / total assets	EBIT / total assets	book value of equity / total liabilities	sales / total assets	ec
total liabilities / ((profit on operating activities + depreciation) * (12/365))	-0.021210	-0.247260	0.245682	0.000533	0.229900	-0.021189	0.000643	-0.004404	0.2
profit on operating activities / sales	0.008359	0.035174	-0.035827	-0.009876	-0.031130	0.008420	-0.001561	0.001543	-0.0
rotation receivables + inventory turnover in days	0.003037	0.019633	-0.018519	0.001443	-0.016843	0.002950	-0.002236	-0.040893	-0.0
(receivables * 365) / sales	0.003225	0.025524	-0.024218	-0.003426	-0.022128	0.003164	-0.003942	-0.036454	-0.0
net profit / inventory	0.021661	-0.069988	0.069601	0.052071	0.052846	0.021803	0.253491	-0.006171	0.0
(current assets - inventory) / short-term liabilities	-0.000246	-0.006978	0.010329	0.971863	0.004780	-0.000324	0.345324	-0.013007	0.0
(inventory * 365) / cost of products sold	0.001365	-0.004234	0.004118	0.003525	0.004290	0.001303	-0.002140	-0.023193	0.0
EBITDA (profit on operating activities - depreciation) / total assets	0.988327	0.007105	-0.006421	0.000296	0.487196	0.988311	0.001640	-0.014880	-0.3
EBITDA (profit on operating activities - depreciation) / sales	0.002143	0.013168	-0.013646	0.000184	-0.012164	0.002190	0.001527	0.016019	-0.0
current assets / total liabilities	0.002862	-0.009263	0.009732	0.883372	0.006188	0.002494	0.395765	-0.018853	0.0

	net profit / total assets	total liabilities / total assets	working capital / total assets	current assets / short-term liabilities	retained earnings / total assets	EBIT / total assets	book value of equity / total liabilities	sales / total assets	ec
short-term liabilities / total assets	-0.043623	0.996915	-0.999890	-0.009351	-0.858610	-0.043666	-0.005705	-0.009524	-0.9
(short-term liabilities * 365) / cost of products sold)	-0.063910	0.620435	-0.622614	-0.009329	-0.515414	-0.063946	-0.005443	0.000624	-0.5
equity / fixed assets	-0.100543	-0.002315	0.001264	-0.001526	-0.007938	-0.100575	-0.001310	0.616850	0.1
constant capital / fixed assets	-0.100226	-0.002312	0.001286	-0.001366	-0.007762	-0.100258	-0.001346	0.616904	0.1
(sales - cost of products sold) / sales	-0.000676	0.000364	-0.000503	0.002508	-0.000733	-0.000656	0.001509	0.011421	-0.0
(current assets - inventory - short-term liabilities) / (sales - gross profit - depreciation)	0.001964	0.001346	-0.000964	0.004793	0.000298	0.002011	0.001714	0.013315	-0.0
total costs /total sales	-0.010004	-0.006811	0.010300	0.009963	-0.012124	-0.010148	-0.002934	-0.036677	0.0
long-term liabilities / equity	0.001465	-0.002196	0.002058	-0.005822	0.003387	0.001434	-0.003883	-0.010762	0.0
sales / inventory	0.001761	-0.002945	0.002668	0.003709	0.001691	0.001785	0.000401	0.010325	0.0
sales / receivables	-0.001870	-0.005889	0.004464	0.015977	0.005669	-0.000982	-0.007240	0.395824	0.0
sales / short-term liabilities	0.000422	-0.008803	0.011347	0.625880	0.006226	0.000450	0.121387	0.053057	0.0
sales / fixed assets	-0.224381	-0.002587	0.001596	-0.002482	-0.070741	-0.224397	-0.002055	0.616716	0.1
year	-0.031943	-0.024168	0.027058	0.010528	0.004812	-0.032201	-0.004284	-0.019625	0.0

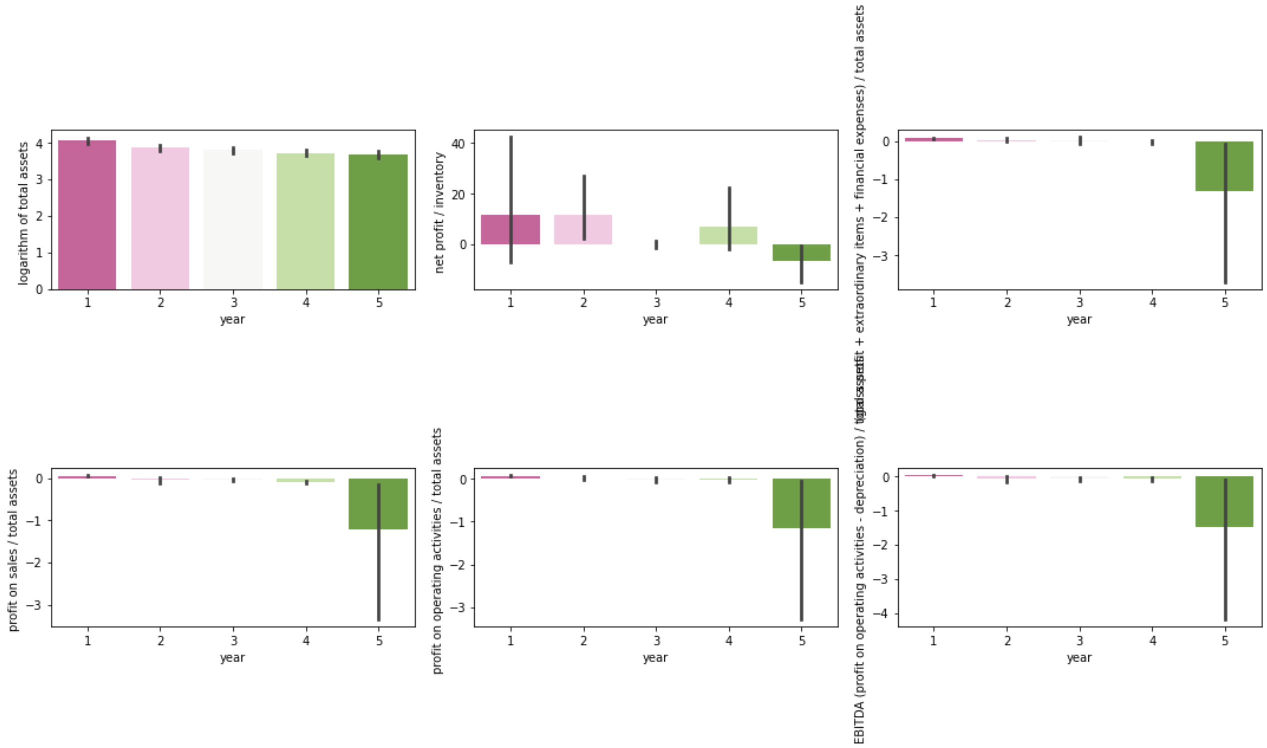
In [19]:
```
numeric_features = analyze_df.dtypes[analyze_df.dtypes != 'int64'].index
categorical_features = analyze_df.dtypes[analyze_df.dtypes == 'int64'].index

positive_corr = analyze_df[numeric_features].corrwith(analyze_df["year"]).sort_values(a
negative_corr = analyze_df[numeric_features].corrwith(analyze_df["year"]).sort_values()

positive_corr = analyze_df[positive_corr + ["year"]].copy()
negative_corr = analyze_df[negative_corr + ["year"]].copy()
```

In [20]:
```
def corrbargraph(x_value, y_value, df, filename):

    plt.figure(figsize=(15,8))
    value = randint(0, len(colors)-1)

    for i in range(1,7):
        plt.subplot(2,3,i)
        sns.barplot(x = x_value, y = y_value[i-1],data = df,palette = colors[value])

    plt.tight_layout(pad=0.5)
    plt.savefig(filename)
```

In [21]:
```
x_value = positive_corr.columns.tolist()[-1]
y_value = positive_corr.columns.tolist()[:-1]

corrbargraph(x_value, y_value, analyze_df, 'positive_correlation.png')
```

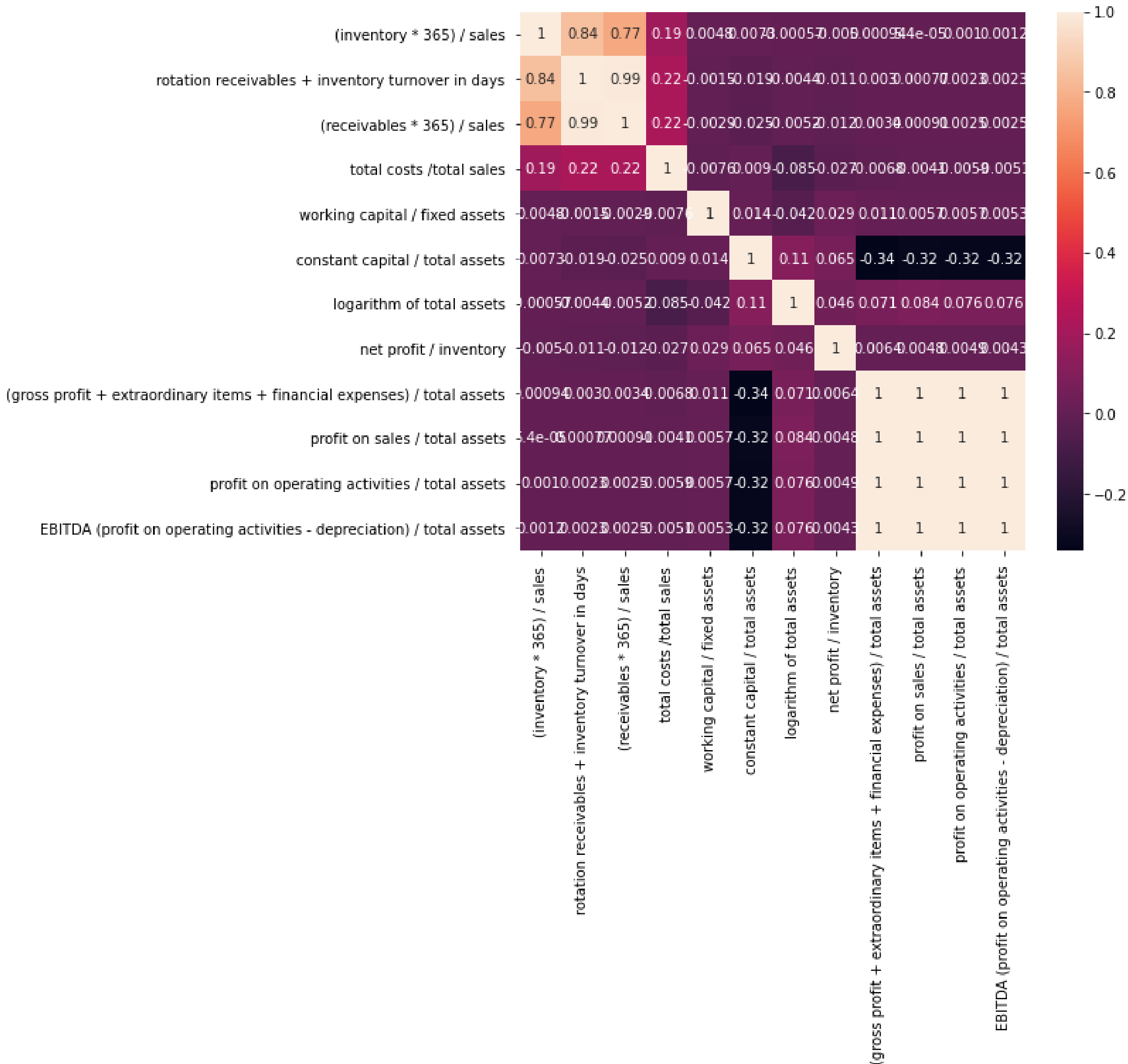


In [22]:
```
x_value = negative_corr.columns.tolist()[-1]
y_value = negative_corr.columns.tolist()[:-1]

corrbargraph(x_value, y_value, analyze_df, 'negative_correlation.png')
```



In [23]:

```
relation = positive_corr.columns.tolist()[:-1] + negative_corr.columns.tolist()[:-1]
plt.figure(figsize=(8,7))
sns.heatmap(analyze_df[relation].corr(),annot=True).figure.savefig("heat_map.png")
```



Drop correlated features

In [24]:

```
drop_correlated_fea = []
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if(correlation_matrix.iloc[i,j] >= 0.50 or correlation_matrix.iloc[i,j] <= -0.7
            if correlation_matrix.columns[j] not in drop_correlated_fea:
                drop_correlated_fea.append(correlation_matrix.columns[j])
```

In [25]:

```
print(len(drop_correlated_fea))
df_remove_corr_features = analyze_df.drop(drop_correlated_fea, axis = 1)
```

38

Issues:

1. There is too much imbalance in the data

2. Non-required features

Data Modeling

Split the dataset into training and testing sets (80% - 20%). We preserve the 20% testing set for the final evaluation.

In [26]:
```
def preprocess_inputs(df, y_column='year'):
    df = df.copy()

    # Split df into X and y
    y = df[y_column]
    X = df.drop(y_column, axis=1)

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, test_si

    # Scale X
    scaler = StandardScaler()
    scaler.fit(X_train)
    X_train = pd.DataFrame(scaler.transform(X_train), index=X_train.index, columns=X_tr
    X_test = pd.DataFrame(scaler.transform(X_test), index=X_test.index, columns=X_test.

    return X_train, X_test, y_train, y_test
```

In [27]:
```
# removing non-bankrupt companies data
final_df = analyze_df.copy()
X_train, X_test, y_train, y_test = preprocess_inputs(df_remove_corr_features)
```

Apply SMOTE only to get the train data

In [28]:
```
def preprocess_inputs_smote(df_, y_column='year'):
    from collections import Counter
    from sklearn.datasets import make_classification
    from imblearn.over_sampling import BorderlineSMOTE
    df_ = df_.copy()

    # Split df into X and y
    y_ = df_[y_column]
    X_ = df_.drop(y_column, axis=1)

    #Initializing SMOTE

    sm = BorderlineSMOTE(random_state=42)
    X_train_smote, y_train_smote = sm.fit_resample(X_train, y_train)

    sm = BorderlineSMOTE(random_state = 42)
    X_test_oversampled, y_test_oversampled = sm.fit_resample(X_train_smote, y_train_smo
    X_train_smote = pd.DataFrame(X_test_oversampled, columns=X_.columns)

    return X_train_smote, y_train_smote
```

In [29]:
```
X_train_smote, y_train_smote = preprocess_inputs_smote(df_remove_corr_features)
```

Useful function for tuning

In [30]:

```python
"""
Plot test score heatmap of parameter sets
"""
def plot_heatmap(grid, params):
    results = pd.DataFrame(grid.cv_results_)
    keys = list(params.keys())
    scores = np.array(results.mean_test_score).reshape(len(params[keys[1]]),
                                                       len(params[keys[0]]))
    plt.figure()
    mglearn.tools.heatmap(scores, xlabel=keys[0], xticklabels=params[keys[0]],
                          ylabel=keys[1], yticklabels=params[keys[1]],
                          cmap="viridis", fmt='%0.5f')
    plt.show()
```

In [39]:

```python
def MyModel(X_train, y_train, show_plots=True, score_method='recall'):

    """
    Naive Bayes
    """
    clf = GaussianNB()
    clf.fit(X_train, y_train)
    print('Finished training Naive Bayes...')

    """
    Logistic Regression
    """
    params_logit = {'polynomialfeatures__degree': [1, 2],
                 'selectpercentile__percentile': [50, 100],
                 'logisticregression__C': [0.01, 1, 100]}
    pipe_logit = make_pipeline(PolynomialFeatures(include_bias=False),
                               StandardScaler(), SelectPercentile(),
                               LogisticRegression(max_iter=1000))
    lsearch = GridSearchCV(estimator=pipe_logit,
                           scoring = 'roc_auc_ovr',
                           param_grid=params_logit,
                           cv=5,
                           n_jobs=-1)
    lsearch.fit(X_train, y_train)
    lr = lsearch.best_estimator_
    print('Finished training Logistic Regression...')


    """
    Support Vector Machine
    """
    params_svc = [{'svc__kernel' : ['rbf'],
              'svc__gamma' : [0.01, 0.1, 1, 10, 100],
              'svc__C' : [0.01, 0.1, 1, 10, 100]},
              {'svc__kernel' : ['logarithmic'],
              'svc__C' : [0.01, 0.1, 1, 10, 100]}]
    pipe_svc = make_pipeline(MinMaxScaler(), SVC(probability=True)) # no need for polyn
    ssearch = GridSearchCV(estimator=pipe_svc,
                           scoring = 'roc_auc_ovr',
                           param_grid=params_svc,
                           cv=5,
                           n_jobs=-1)
    ssearch.fit(X_train, y_train)
    svc = ssearch.best_estimator_
    print('Finished training Support Vector Machine...')
```

```
    """
    Random Forest Classifier
    """
    param_grid = {
        'n_estimators': [200, 500],
        'max_features': ['auto', 'sqrt', 'log2'],
        'max_depth' : [4,5,6,7,8],
        'criterion' :['gini', 'entropy']
    }
    rfc=RandomForestClassifier(random_state=42)
    rsearch = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
    rsearch.fit(X_train, y_train)
    rfc=rsearch.best_estimator_
    print('Finished training Random Forest Classifier...')

    # all models
    models = [clf, lr, svc, rfc]

    return models
```

In [36]:

```
mymodels = MyModel(X_train_smote, y_train_smote)
```

```
Finished training Naive Bayes...
Finished training Logistic Regression...
Finished training Support Vector Machine...
```

In [40]:

```
final_report = pd.DataFrame(index=['Naive Bayes', 'Logistic Regression',
                                   'Support Vector Machine', 'Random Forest Classifier'
                            columns=['accuracy', 'precision', 'recall', 'f1',  'f1_macr

for i in range(len(mymodels)):
    model = mymodels[i]
    name = final_report.index[i]
    report = pd.DataFrame(classification_report(y_test, model.predict(X_test), output_d

    final_report.loc[name, :] = [report.loc['accuracy', 'support'],
                             report.loc['1', 'precision'],
                             report.loc['1', 'recall'],
                             report.loc['1', 'f1-score'],
                             report.loc['macro avg', 'f1-score']]

print("Model Comparison Report:\n", final_report)
```

```
Model Comparison Report:
                           accuracy precision    recall        f1  f1_macro
Naive Bayes               0.178998  0.156863  0.872727  0.265928  0.124891
Logistic Regression        0.21957  0.204819  0.618182  0.307692  0.211585
Support Vector Machine     0.22673  0.237037  0.581818  0.336842  0.222211
Random Forest Classifier  0.794749  0.695652  0.872727  0.774194  0.793387
```

In [41]:

```
# Plotting confusion matrix for each classifier

a = 2  # number of rows
b = 2  # number of columns
c = 1  # initialize plot counter

fig = plt.figure(figsize=(40, 38))
```

```python
for i, model in enumerate(mymodels):
    name = final_report.index[i]
    y_test_pred_smote = model.predict(X_test)
    arg_test = {'y_true':y_test, 'y_pred':y_test_pred_smote}

    conf_mx0 = confusion_matrix(y_test, y_test_pred_smote)

    heat_cm0 = pd.DataFrame(conf_mx0, columns=np.unique(y_test), index = np.unique(y_te
    heat_cm0.index.name = 'Actual'
    heat_cm0.columns.name = 'Predicted'

    plt.subplot(a, b, c)
    fig.subplots_adjust(left=None, bottom=None, right= None, top=None, wspace=0.4, hspa
    sns.heatmap(heat_cm0, annot=True, fmt='.2f', square=True, annot_kws={"size": 20}, c
    c = c + 1

plt.show()
plt.savefig('confusion_matrix.png')
```


```
<Figure size 864x504 with 0 Axes>
```