Assignment- ML Role

PART A. Train a Classifier to classify stress companies vs non-stress companies. Use Yahoo Finance to collect Financial data for Indian companies (min 100 companies). Convert financial variables into features based on the following ratios. Current ratio Debt to equity ratio ROI (Return on investment) ROA (Return on assets) Inventory turnover ratio. Net profit margin Perform EDA(exploratory data analysis) on features. Label the data using the logic given below: A company is called in stress when its revenue to expenses ratio is less than 1. Train a machine learning classifier(Logistic regression, support vector machine, naive bays, decision tree) to predict the classes and get the correlation of coefficients (weights) with respect to the degree of stress. Use the ensemble model to predict the final degree of stress. Evaluate the model on test data with evaluation matrices.

Use Yahoo Finance to collect Financial data for Indian companies (min 100 companies).

In [456]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from numpy import NaN
from glob import glob
import re
import time
from datetime import datetime
import lxml
from lxml import html
import requests


pd.set_option('max_columns', 200)
pd.set_option('max_rows', 300)
pd.set_option('display.expand_frame_repr', True)
```

In [217]:

```
def get_page(url):
    # Set up the request headers that we're going to use, to simulate
    # a request by the Chrome browser. Simulating a request from a browser
    # is generally good practice when building a scraper
    headers = {
        'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,ima
ge/apng,*/*;q=0.8,application/signed-exchange;v=b3',
        'Accept-Encoding': 'gzip, deflate, br',
        'Accept-Language': 'en-US,en;q=0.9',
        'Cache-Control': 'max-age=0',
        'Pragma': 'no-cache',
        'Referrer': 'https://google.com',
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH
TML, like Gecko) Chrome/77.0.3865.120 Safari/537.36'
    }

    return requests.get(url, headers=headers)

def parse_rows(table_rows):
    parsed_rows = []

    for table_row in table_rows:
        parsed_row = []
        el = table_row.xpath("./div")

        none_count = 0

        for rs in el:
            try:
                (text,) = rs.xpath('.//span/text()[1]')
                parsed_row.append(text)
            except ValueError:
                parsed_row.append(np.NaN)
                none_count += 1

        if (none_count < 2):
            parsed_rows.append(parsed_row)

    return pd.DataFrame(parsed_rows)

def clean_data(df):
    df = df.set_index(0) # Set the index to the first column: 'Period Ending'.
    df = df.transpose() # Transpose the DataFrame, so that our header contains the acco
unt names

    # Rename the "Breakdown" column to "Date"
    cols = list(df.columns)
    cols[0] = 'Date'
    df = df.set_axis(cols, axis='columns', inplace=False)

    numeric_columns = list(df.columns)[1::] # Take all columns, except the first (which
is the 'Date' column)

    for column_index in range(1, len(df.columns)): # Take all columns, except the first
(which is the 'Date' column)
        df.iloc[:,column_index] = df.iloc[:,column_index].str.replace(',', '') # Remove
the thousands separator
        df.iloc[:,column_index] = df.iloc[:,column_index].astype(np.float64) # Convert
 the column to float64
```

```
    return df
```

In [4]:

```
#getting data from tables
def scrape_table(url):

    # Fetch the page that we're going to parse
    page = get_page(url);

    # Parse the page with LXML, so that we can start doing some XPATH queries
    # to extract the data that we want
    tree = html.fromstring(page.content)
    time.sleep(5)
    # Fetch all div elements which have class 'D(tbr)'
    table_rows = tree.xpath("//div[contains(@class, 'D(tbr)')]")
    time.sleep(5)
    # Ensure that some table rows are found; if none are found, then it's possible
    # that Yahoo Finance has changed their page layout, or have detected
    # that you're scraping the page.
    assert  len(table_rows) > 0
    time.sleep(5)
    df = parse_rows(table_rows)
    df = clean_data(df)
    time.sleep(5)

    return df
```

In [5]:

```
#geting data of multiple compines
def scrape(symbol):
    print('Attempting to scrape data for ' + symbol)

    df_balance_sheet = scrape_table('https://finance.yahoo.com/quote/' + symbol + '/bal
ance-sheet?p=' + symbol)
    df_balance_sheet = df_balance_sheet.set_index('Date')

    df_income_statement = scrape_table('https://finance.yahoo.com/quote/' + symbol + '/
financials?p=' + symbol)
    df_income_statement = df_income_statement.set_index('Date')

    df_cash_flow = scrape_table('https://finance.yahoo.com/quote/' + symbol + '/cash-fl
ow?p=' + symbol)
    df_cash_flow = df_cash_flow.set_index('Date')

    df_joined = df_balance_sheet \
        .join(df_income_statement, on='Date', how='outer', rsuffix=' - Income Statemen
t') \
        .join(df_cash_flow, on='Date', how='outer', rsuffix=' - Cash Flow') \
        .dropna(axis=1, how='all') \
        .reset_index()

    df_joined.insert(1, 'Symbol', symbol)

    return df_joined
```

In [6]:

```
def scrape_multi(symbols):
    return pd.concat([scrape(symbol) for symbol in symbols], sort=False)
```

In [24]:

```
#geeting only syombols of the compines

from bs4 import BeautifulSoup
import requests
list=[]
url='https://in.finance.yahoo.com/most-active?count=100&offset=100'
header = {
        'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.
3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9'
}
response=requests.get(url,headers=header)
soup=BeautifulSoup(response.content, 'lxml')
for item in soup.select('.simpTblRow'):

        #print(item.select('[aria-label=Symbol]')[0].get_text())
    list1.append((item.select('[aria-label=Symbol]')[0].get_text()))
    #print('____________________________')
```

In [ ]:

```
df_combine = scrape_multi(list)
```

In [118]:

```
#saving scarpped data from yahoo finance
date = datetime.today().strftime('%Y-%m-%d')
writer = pd.ExcelWriter('df8.xlsx')
df_combine.to_excel(writer)
writer.save()
```

In [121]:

```
#importing required libraries
import os
import pandas as pd
cwd = os.path.abspath('')
files = os.listdir(cwd)
```

In [122]:

```
cd C:\Users\harsha.teja\Desktop\solvedo\data
```

```
C:\Users\harsha.teja\Desktop\solvedo\data
```

In [163]:

```
#Loading the data
df_total = pd.DataFrame()
for file in files:                          # loop through Excel files
    if file.endswith('.xlsx'):
        excel_file = pd.ExcelFile(file)
        sheets = excel_file.sheet_names
        for sheet in sheets:                # loop through sheets inside an Excel file
            df = excel_file.parse(sheet_name = sheet)
            df_total = df_total.append(df)
df_total.to_excel('combined_file.xlsx')
```

In [164]:

```
data = pd.read_excel('combined_file.xlsx')
```

In [165]:

```
data.shape
```

Out[165]:

```
(549, 65)
```

In [166]:

```
data
```

Out[166]:

	Unnamed: 0	Unnamed: 0.1	index	Symbol	Date	Total Assets	Total Liabilities Net Minority Interest
0	0	0	3/31/2020	SANWARIA.NS	3/31/2020	3719967.0	9382262.0
1	1	1	3/31/2019	SANWARIA.NS	3/31/2019	17949803.0	11438281.0
2	2	2	3/31/2018	SANWARIA.NS	3/31/2018	17316959.0	11375655.0
3	3	3	NaN	SANWARIA.NS	ttm	NaN	NaN
4	4	0	3/31/2020	EDELWEISS.NS	3/31/2020	542803210.0	470732440.0
...	...	...	...	...	...	...	...
544	107	3	NaN	MRPL.NS	ttm	NaN	NaN
545	108	0	3/31/2020	GODREJCP.NS	3/31/2020	149570100.0	70586500.0
546	109	1	3/31/2019	GODREJCP.NS	3/31/2019	141700800.0	69031600.0
547	110	2	3/31/2018	GODREJCP.NS	3/31/2018	139627100.0	77044000.0
548	111	3	NaN	GODREJCP.NS	ttm	NaN	NaN

549 rows × 65 columns

In [167]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 549 entries, 0 to 548
Data columns (total 65 columns):
 #   Column                                        Non-Null
Count  Dtype
---  ------                                        ---------
-----  -----
 0   Unnamed: 0                                    549 non-n
ull    int64
 1   Unnamed: 0.1                                  549 non-n
ull    int64
 2   index                                         409 non-n
ull    object
 3   Symbol                                        549 non-n
ull    object
 4   Date                                          549 non-n
ull    object
 5   Total Assets                                  409 non-n
ull    float64
 6   Total Liabilities Net Minority Interest       409 non-n
ull    float64
 7   Total Equity Gross Minority Interest          409 non-n
ull    float64
 8   Total Capitalization                          409 non-n
ull    float64
 9   Common Stock Equity                           409 non-n
ull    float64
 10  Net Tangible Assets                           409 non-n
ull    float64
 11  Working Capital                               359 non-n
ull    float64
 12  Invested Capital                              409 non-n
ull    float64
 13  Tangible Book Value                           409 non-n
ull    float64
 14  Total Debt                                    384 non-n
ull    float64
 15  Net Debt                                      296 non-n
ull    float64
 16  Share Issued                                  408 non-n
ull    float64
 17  Ordinary Shares Number                        408 non-n
ull    float64
 18  Total Revenue                                 538 non-n
ull    float64
 19  Cost of Revenue                               458 non-n
ull    float64
 20  Gross Profit                                  458 non-n
ull    float64
 21  Operating Expense                             474 non-n
ull    float64
 22  Operating Income                              474 non-n
ull    float64
 23  Net Non Operating Interest Income Expense     471 non-n
ull    float64
 24  Pretax Income                                 539 non-n
ull    float64
 25  Tax Provision                                 529 non-n
ull    float64
 26  Net Income Common Stockholders                539 non-n
ull    float64
```

```
 27  Diluted NI Available to Com Stockholders                 539 non-n
ull    float64
 28  Basic Average Shares                                     343 non-n
ull    float64
 29  Diluted Average Shares                                   343 non-n
ull    float64
 30  Rent Expense Supplemental                                284 non-n
ull    float64
 31  Total Expenses                                           474 non-n
ull    float64
 32  Net Income from Continuing & Discontinued Operation      539 non-n
ull    float64
 33  Normalized Income                                        539 non-n
ull    float64
 34  Interest Income                                          300 non-n
ull    float64
 35  Interest Expense                                         457 non-n
ull    float64
 36  Net Interest Income                                      471 non-n
ull    float64
 37  EBIT                                                     474 non-n
ull    float64
 38  Reconciled Cost of Revenue                               458 non-n
ull    float64
 39  Reconciled Depreciation                                  525 non-n
ull    float64
 40  Net Income from Continuing Operation Net Minority Interest  539 non-n
ull    float64
 41  Total Unusual Items Excluding Goodwill                   492 non-n
ull    float64
 42  Total Unusual Items                                      492 non-n
ull    float64
 43  Normalized EBITDA                                        474 non-n
ull    float64
 44  Tax Rate for Calcs                                       539 non-n
ull    float64
 45  Tax Effect of Unusual Items                              539 non-n
ull    float64
 46  Operating Cash Flow                                      448 non-n
ull    float64
 47  Investing Cash Flow                                      448 non-n
ull    float64
 48  Financing Cash Flow                                      445 non-n
ull    float64
 49  End Cash Position                                        448 non-n
ull    float64
 50  Issuance of Debt                                         201 non-n
ull    float64
 51  Repayment of Debt                                        218 non-n
ull    float64
 52  Free Cash Flow                                           448 non-n
ull    float64
 53  Preferred Stock Equity                                   4 non-nul
l      float64
 54  Income from Associates & Other Participating Interests   22 non-nu
ll     float64
 55  Special Income Charges                                   62 non-nu
ll     float64
 56  Capital Expenditure                                      424 non-n
ull    float64
 57  Capital Lease Obligations                                89 non-nu
```

```
ll      float64
 58  Issuance of Capital Stock                                  129 non-n
ull     float64
 59  Repurchase of Capital Stock                                23 non-nu
ll      float64
 60  Other Income Expense                                       8 non-nul
l       float64
 61  Treasury Shares Number                                     2 non-nul
l       float64
 62  Non Interest Expense                                       5 non-nul
l       float64
 63  INTEREST_INCOME_AFTER_PROVISION_FOR_LOAN_LOSS              5 non-nul
l       float64
 64  Total Money Market Investments                             5 non-nul
l       float64
dtypes: float64(60), int64(2), object(3)
memory usage: 278.9+ KB
```

In [168]:

```
#fillter only required data
final_data = data[data['Date']=='3/31/2020']
```

In [169]:

```
len(final_data)
```

Out[169]:

```
127
```

In [170]:

```
final_data.shape
```

Out[170]:

```
(127, 65)
```

In [171]:

```
final_data.columns
```

Out[171]:

```
Index(['Unnamed: 0', 'Unnamed: 0.1', 'index', 'Symbol', 'Date', 'Total Ass
ets',
       'Total Liabilities Net Minority Interest',
       'Total Equity Gross Minority Interest', 'Total Capitalization',
       'Common Stock Equity', 'Net Tangible Assets', 'Working Capital',
       'Invested Capital', 'Tangible Book Value', 'Total Debt', 'Net Deb
t',
       'Share Issued', 'Ordinary Shares Number', 'Total Revenue',
       'Cost of Revenue', 'Gross Profit', 'Operating Expense',
       'Operating Income', 'Net Non Operating Interest Income Expense',
       'Pretax Income', 'Tax Provision', 'Net Income Common Stockholders',
       'Diluted NI Available to Com Stockholders', 'Basic Average Shares',
       'Diluted Average Shares', 'Rent Expense Supplemental', 'Total Expen
ses',
       'Net Income from Continuing & Discontinued Operation',
       'Normalized Income', 'Interest Income', 'Interest Expense',
       'Net Interest Income', 'EBIT', 'Reconciled Cost of Revenue',
       'Reconciled Depreciation',
       'Net Income from Continuing Operation Net Minority Interest',
       'Total Unusual Items Excluding Goodwill', 'Total Unusual Items',
       'Normalized EBITDA', 'Tax Rate for Calcs',
       'Tax Effect of Unusual Items', 'Operating Cash Flow',
       'Investing Cash Flow', 'Financing Cash Flow', 'End Cash Position',
       'Issuance of Debt', 'Repayment of Debt', 'Free Cash Flow',
       'Preferred Stock Equity',
       'Income from Associates & Other Participating Interests',
       'Special Income Charges', 'Capital Expenditure',
       'Capital Lease Obligations', 'Issuance of Capital Stock',
       'Repurchase of Capital Stock', 'Other Income Expense',
       'Treasury Shares Number', 'Non Interest Expense',
       'INTEREST_INCOME_AFTER_PROVISION_FOR_LOAN_LOSS',
       'Total Money Market Investments'],
      dtype='object')
```

In [172]:

```
final_data=final_data.reset_index()
```

In [174]:

```
#Current Ratio= Current assets/Current liabilities
```

In [175]:

```
final_data['current ratio'] = final_data['Total Assets']/final_data['Total Liabilities
 Net Minority Interest']
```

In [176]:

```
#Debt-To-Equity Ratio (D/E) Formula and Calculation
#Debt/Equity= Total Liabilities/Total Shareholders’ Equity
#working_capital = current_assets - current_liabilities
```

In [297]:

```
final_data['working_capital'] = final_data['Total Assets']  - final_data['Total Liabili
ties Net Minority Interest']
```

In [ ]:

```
final_data['Net icome'] = final_data['Total Revenue']  - final_data['Total Expenses']
```

In [177]:

```
#firt need to calcalte the net income
#net inome = totltevenves - total expense
final_data['Net icome'] = final_data['Total Revenue']  - final_data['Total Expenses']
```

In [178]:

```
final_data['Debt to equity ratio'] = final_data['Total Liabilities Net Minority Interes
t']/final_data['Total Equity Gross Minority Interest']
```

In [179]:

```
#ROA (Return on assets) =Net income or total revens/ total assets
```

In [180]:

```
final_data['Return on assets'] = final_data['Net icome'] /final_data['Total Assets']
```

In [181]:

```
#ROI (Return on investment) = net income/cost of investment
```

In [182]:

```
final_data['Return on investment'] = final_data['Net icome'] /final_data['Cost of Reven
ue']
```

In [ ]:

```
# revenue to expenses ratio = operting expense/operting income
```

In [191]:

```
final_data['revenue to expenses ratio'] = final_data['Operating Expense']/final_data['O
perating Income']
```

In [192]:

```
#Net profit margin = netprofit/totalrevene *100

final_data['Net profit margin'] = (final_data['Net icome']/final_data['Total Revenue'])
*100
```

In [206]:

```
#Inventory turnover ratio= Total revenve/Average inventory
final_data['Average inventory'] = (final_data['Operating Cash Flow']+final_data['End Ca
sh Position'])/2
final_data['Inventory turnover ratio'] = final_data['Operating Expense']/final_data['Av
erage inventory']
```

In [193]:

```
final_data.isnull().sum()
```

Out[193]:

```
level_0                                                   0
Unnamed: 0                                                0
Unnamed: 0.1                                              0
index                                                     0
Symbol                                                    0
Date                                                      0
Total Assets                                              0
Total Liabilities Net Minority Interest                   0
Total Equity Gross Minority Interest                      0
Total Capitalization                                      0
Common Stock Equity                                       0
Net Tangible Assets                                       0
Working Capital                                          15
Invested Capital                                          0
Tangible Book Value                                       0
Total Debt                                                7
Net Debt                                                 36
Share Issued                                              0
Ordinary Shares Number                                    0
Total Revenue                                             0
Cost of Revenue                                          18
Gross Profit                                             18
Operating Expense                                        15
Operating Income                                         15
Net Non Operating Interest Income Expense                15
Pretax Income                                             0
Tax Provision                                             3
Net Income Common Stockholders                            0
Diluted NI Available to Com Stockholders                  0
Basic Average Shares                                     17
Diluted Average Shares                                   17
Rent Expense Supplemental                                39
Total Expenses                                           15
Net Income from Continuing & Discontinued Operation       0
Normalized Income                                         0
Interest Income                                          34
Interest Expense                                         19
Net Interest Income                                      15
EBIT                                                     15
Reconciled Cost of Revenue                               18
Reconciled Depreciation                                   0
Net Income from Continuing Operation Net Minority Interest    0
Total Unusual Items Excluding Goodwill                    5
Total Unusual Items                                       5
Normalized EBITDA                                        15
Tax Rate for Calcs                                        0
Tax Effect of Unusual Items                               0
Operating Cash Flow                                       0
Investing Cash Flow                                       0
Financing Cash Flow                                       1
End Cash Position                                         0
Issuance of Debt                                         71
Repayment of Debt                                        68
Free Cash Flow                                            0
Preferred Stock Equity                                  125
Income from Associates & Other Participating Interests  120
Special Income Charges                                  112
Capital Expenditure                                       7
Capital Lease Obligations                                99
```

```
Issuance of Capital Stock                          91
Repurchase of Capital Stock                       120
Other Income Expense                              125
Treasury Shares Number                            126
Non Interest Expense                              126
INTEREST_INCOME_AFTER_PROVISION_FOR_LOAN_LOSS     126
Total Money Market Investments                    126
current ratio                                       0
Net icome                                          15
Debt to equity ratio                                0
Return on assets                                   15
Return on investment                               18
Net profit margin                                  15
revenue to expenses ratio                          15
dtype: int64
```

In [185]:

```
final_data.shape
```

Out[185]:

```
(127, 72)
```

Selecting only reuired fetures as given in satament

In [233]:

```
final_data1 = final_data[['Symbol','Debt to equity ratio', 'Net icome','Return on asset
s','Return on investment','Net profit margin','Inventory turnover ratio','revenue to ex
penses ratio']]
```

In [234]:

```
final_data1.head()
```

Out[234]:

	Symbol	Debt to equity ratio	Net icome	Return on assets	Return on investment	Net profit margin	Inventory turnover ratio	e
0	SANWARIA.NS	-1.656972	-3241270.0	-0.871317	-0.103206	-11.314129	1.870033	-(
1	EDELWEISS.NS	6.531531	NaN	NaN	NaN	NaN	NaN	
2	PANACEABIO.NS	6.102463	420600.0	0.030252	0.240929	7.829690	65.767747	7
3	BHARATFORG.NS	1.201720	6016260.0	0.052031	0.128149	7.843789	2.587769	:
4	POWERGRID.BO	2.967105	210452300.0	0.081999	18.399236	57.176861	0.814825	(

In [235]:

```
#checking for null values
final_data1.isnull().sum()
```

Out[235]:

```
Symbol                      0
Debt to equity ratio        0
Net icome                  15
Return on assets           15
Return on investment       18
Net profit margin          15
Inventory turnover ratio   15
revenue to expenses ratio  15
dtype: int64
```

In [236]:

```
final_data1 = final_data1.dropna()
```

In [237]:

```
final_data1.shape
```

Out[237]:

```
(109, 8)
```

In [238]:

```
final_data1.head()
```

Out[238]:

	Symbol	Debt to equity ratio	Net icome	Return on assets	Return on investment	Net profit margin	Inventory turnover ratio	e
0	SANWARIA.NS	-1.656972	-3241270.0	-0.871317	-0.103206	-11.314129	1.870033	-(
2	PANACEABIO.NS	6.102463	420600.0	0.030252	0.240929	7.829690	65.767747	7
3	BHARATFORG.NS	1.201720	6016260.0	0.052031	0.128149	7.843789	2.587769	:
4	POWERGRID.BO	2.967105	210452300.0	0.081999	18.399236	57.176861	0.814825	(
5	SADHNANIQ.BO	0.853189	241674.0	0.112062	0.577615	23.503497	-3.596641	'

In [239]:

```
len(final_data1)
```

Out[239]:

```
109
```

In [240]:

```
final_data1['stress_nonstress'] = final_data1['revenue to expenses ratio'].apply(lambda
x: 1 if x > 1  else 0)
```

In [241]:

```
final_data1['stress_nonstress'].value_counts()
```

Out[241]:

```
1    65
0    44
Name: stress_nonstress, dtype: int64
```

In [242]:

```
final_data1.head()
```

Out[242]:

	Symbol	Debt to equity ratio	Net icome	Return on assets	Return on investment	Net profit margin	Inventory turnover ratio	e
0	SANWARIA.NS	-1.656972	-3241270.0	-0.871317	-0.103206	-11.314129	1.870033	-(
2	PANACEABIO.NS	6.102463	420600.0	0.030252	0.240929	7.829690	65.767747	7
3	BHARATFORG.NS	1.201720	6016260.0	0.052031	0.128149	7.843789	2.587769	:
4	POWERGRID.BO	2.967105	210452300.0	0.081999	18.399236	57.176861	0.814825	(
5	SADHNANIQ.BO	0.853189	241674.0	0.112062	0.577615	23.503497	-3.596641	'

Perform EDA(exploratory data analysis) on features.

In [337]:

```
# importing data science modules
import pandas as pd
import numpy as np
import sklearn
from sklearn import preprocessing
import scipy as sp
import pickleshare

# importing graphics modules
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from matplotlib.gridspec import GridSpec
cols= ['#00876c','#85b96f','#f7e382','#f19452','#d43d51']


from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
import os
pd.set_option('display.max_columns', 999)
pd.set_option('display.max_rows', 999)
pd.set_option('display.width', 1000)

import sklearn
import datetime

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
import pickle

from IPython.display import display
import plotly.offline as py
import plotly.graph_objs as go
import plotly.tools as tls
py.init_notebook_mode(connected=True)
from IPython.display import display
%matplotlib inline
```

In [243]:

```
final_data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109 entries, 0 to 126
Data columns (total 9 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Symbol                     109 non-null    object 
 1   Debt to equity ratio       109 non-null    float64
 2   Net icome                  109 non-null    float64
 3   Return on assets           109 non-null    float64
 4   Return on investment       109 non-null    float64
 5   Net profit margin          109 non-null    float64
 6   Inventory turnover ratio   109 non-null    float64
 7   revenue to expenses ratio  109 non-null    float64
 8   stress_nonstress           109 non-null    int64  
dtypes: float64(7), int64(1), object(1)
memory usage: 8.5+ KB
```

In [245]:

```
final_data1.describe().T
```

Out[245]:

	count	mean	std	min	25%	
Debt to equity ratio	109.0	1.575573e+00	5.147112e+00	-3.943854e+01	0.372121	1.160758
Net icome	109.0	1.036073e+07	6.000763e+07	-2.014225e+08	140592.000000	1.070692
Return on assets	109.0	4.051842e-02	1.915265e-01	-1.582324e+00	0.018001	6.81708
Return on investment	109.0	5.800518e-01	2.771235e+00	-5.218623e+00	0.027191	1.30679
Net profit margin	109.0	1.113119e+01	2.487822e+01	-1.390515e+02	2.582241	8.84929
Inventory turnover ratio	109.0	-9.829045e-01	4.013564e+01	-3.826648e+02	0.814825	2.540624
revenue to expenses ratio	109.0	1.079265e+01	7.940555e+01	-1.514637e+02	0.459976	1.523424
stress_nonstress	109.0	5.963303e-01	4.928989e-01	0.000000e+00	0.000000	1.00000

In [246]:

```
#for correlation
final_data1.corr()
```

Out[246]:

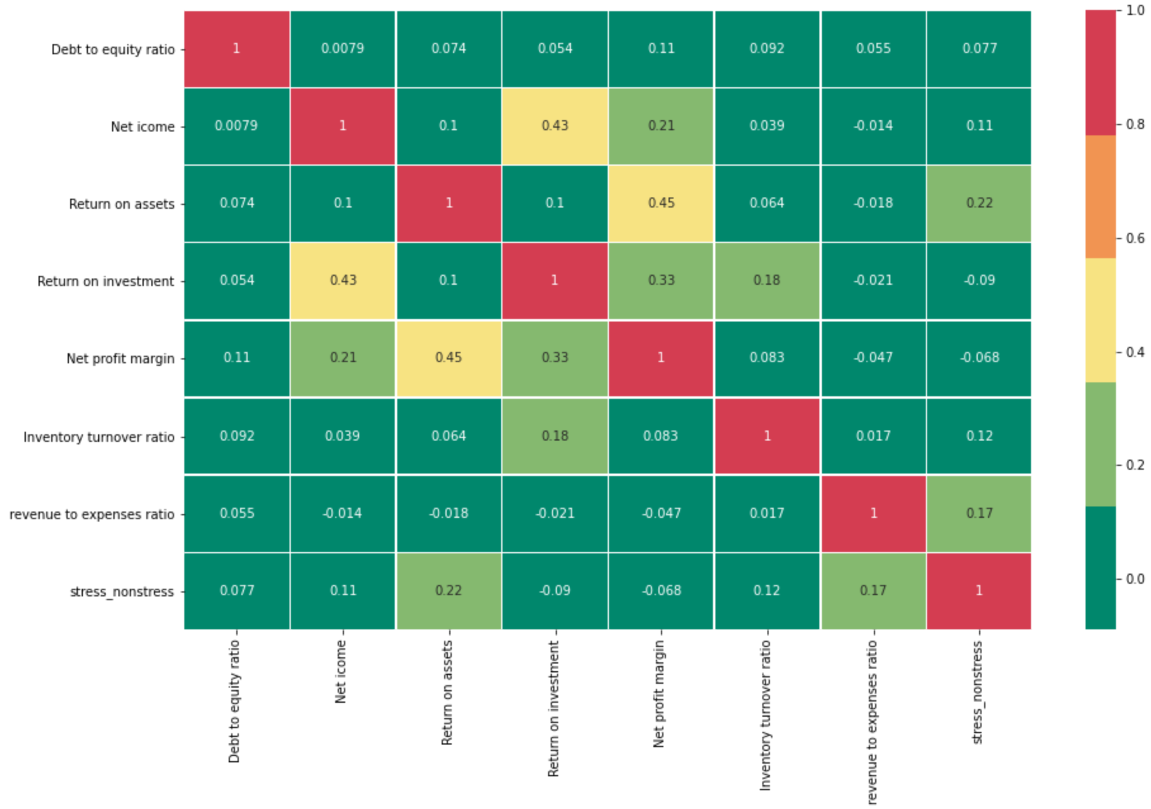
	Debt to equity ratio	Net icome	Return on assets	Return on investment	Net profit margin	Inventory turnover ratio	revenue to expenses ratio
Debt to equity ratio	1.000000	0.007882	0.074236	0.053508	0.109415	0.092486	0.054622
Net icome	0.007882	1.000000	0.103501	0.428963	0.214035	0.039120	-0.013582
Return on assets	0.074236	0.103501	1.000000	0.101120	0.447520	0.064113	-0.017708
Return on investment	0.053508	0.428963	0.101120	1.000000	0.326503	0.183994	-0.020590
Net profit margin	0.109415	0.214035	0.447520	0.326503	1.000000	0.083193	-0.047469
Inventory turnover ratio	0.092486	0.039120	0.064113	0.183994	0.083193	1.000000	0.016718
revenue to expenses ratio	0.054622	-0.013582	-0.017708	-0.020590	-0.047469	0.016718	1.000000
stress_nonstress	0.077439	0.113890	0.223541	-0.089710	-0.068199	0.121025	0.171239

In [249]:

```
correlation =final_data1.corr()
```

In [271]:

```
plt.figure(figsize=(15,9))
sns.heatmap(correlation,cmap=cols, annot=True, linewidths=0.5)
plt.show()
```

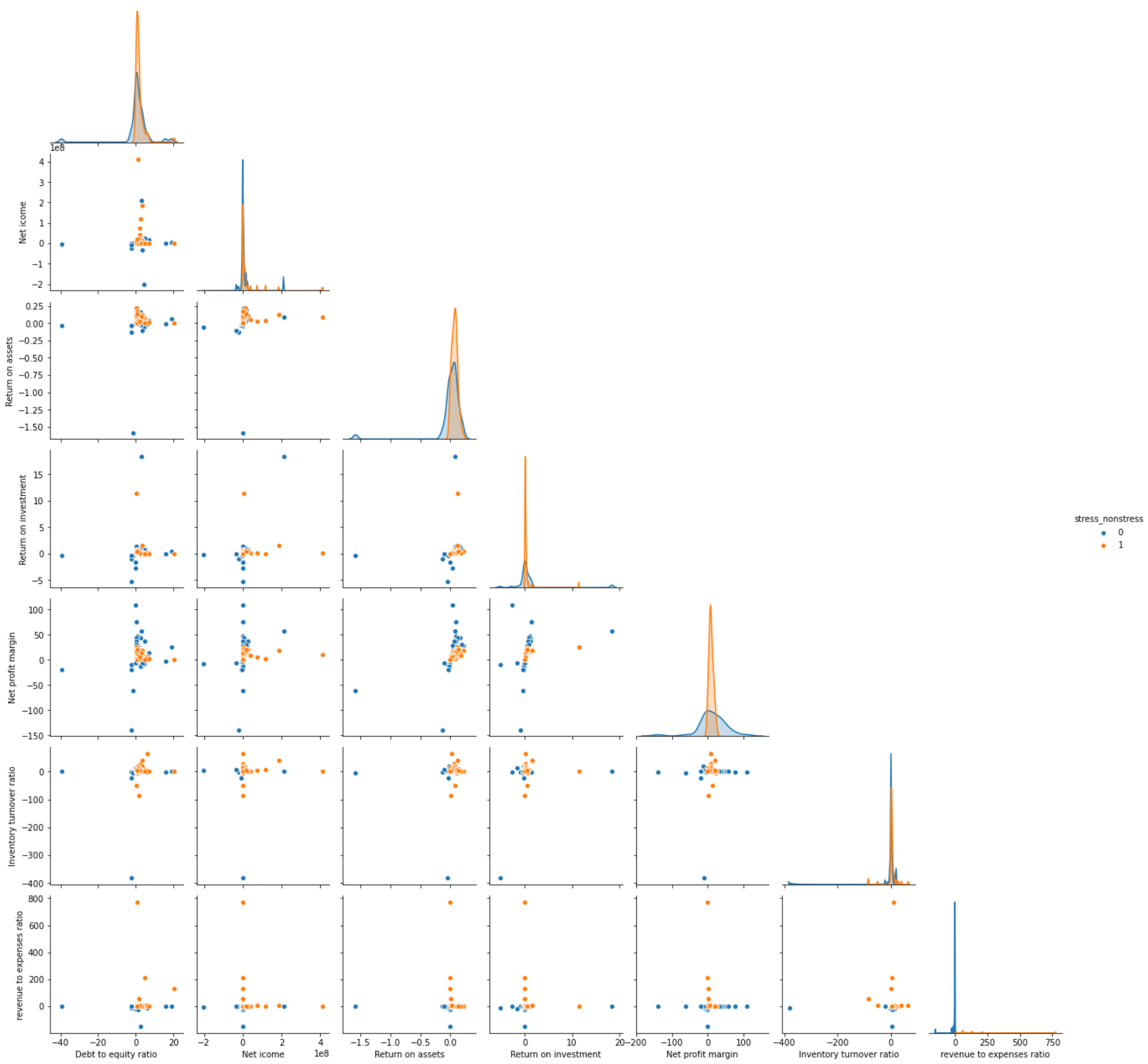


In [272]:

```
#pair plot
sns.pairplot(final_data1[1:], hue='stress_nonstress',corner=True)
```

Out[272]:

```
<seaborn.axisgrid.PairGrid at 0x1f7290b7c70>
```



In [263]:

```
#distrubution of data
def disbution_of_data(feture):
    plt.figure(figsize=(15,9))
    sns.distplot(feture,color='green',bins=100)
```

In [266]:

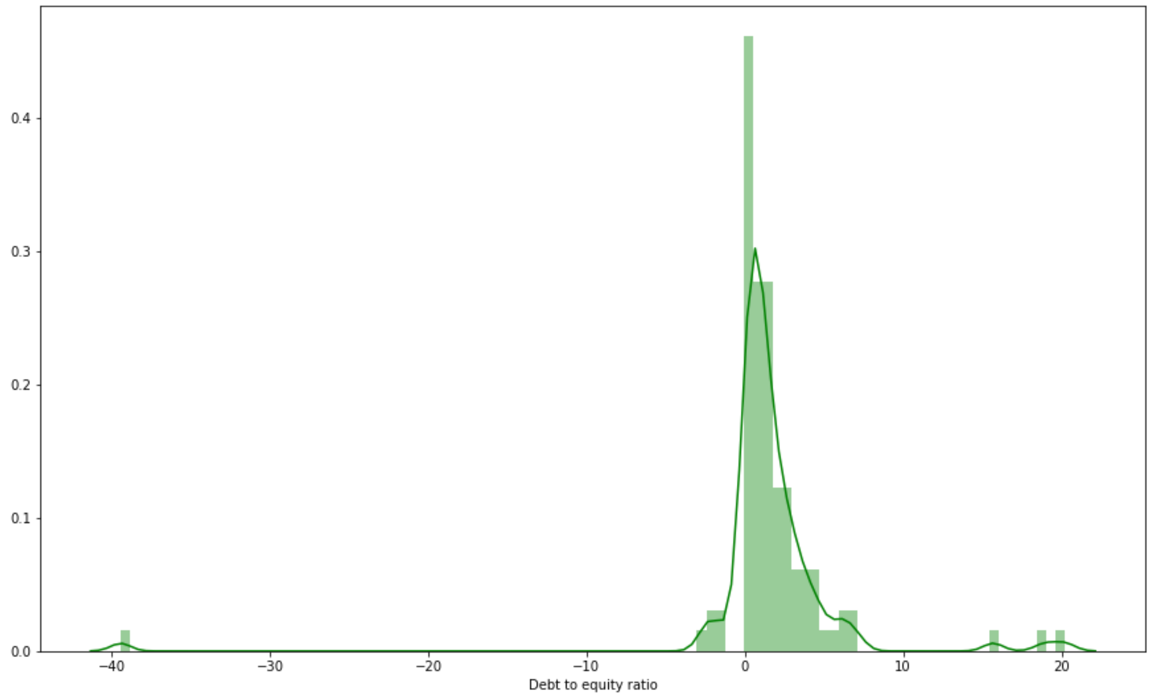
```
final_data1.columns
```

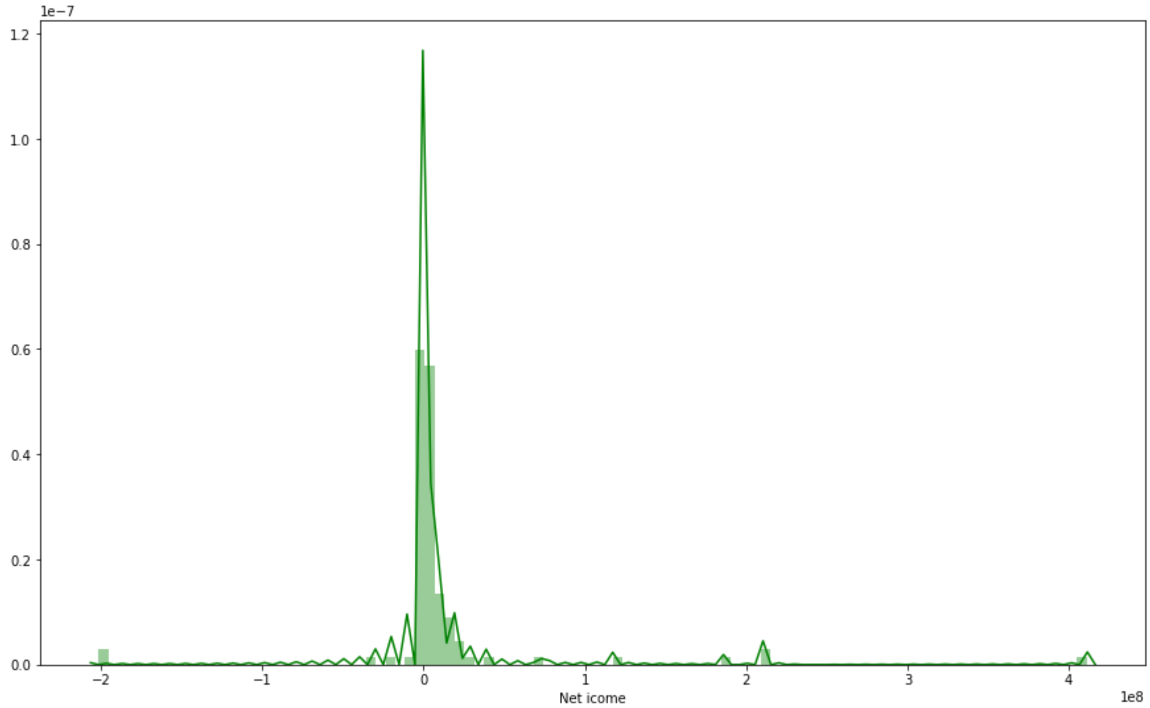
Out[266]:

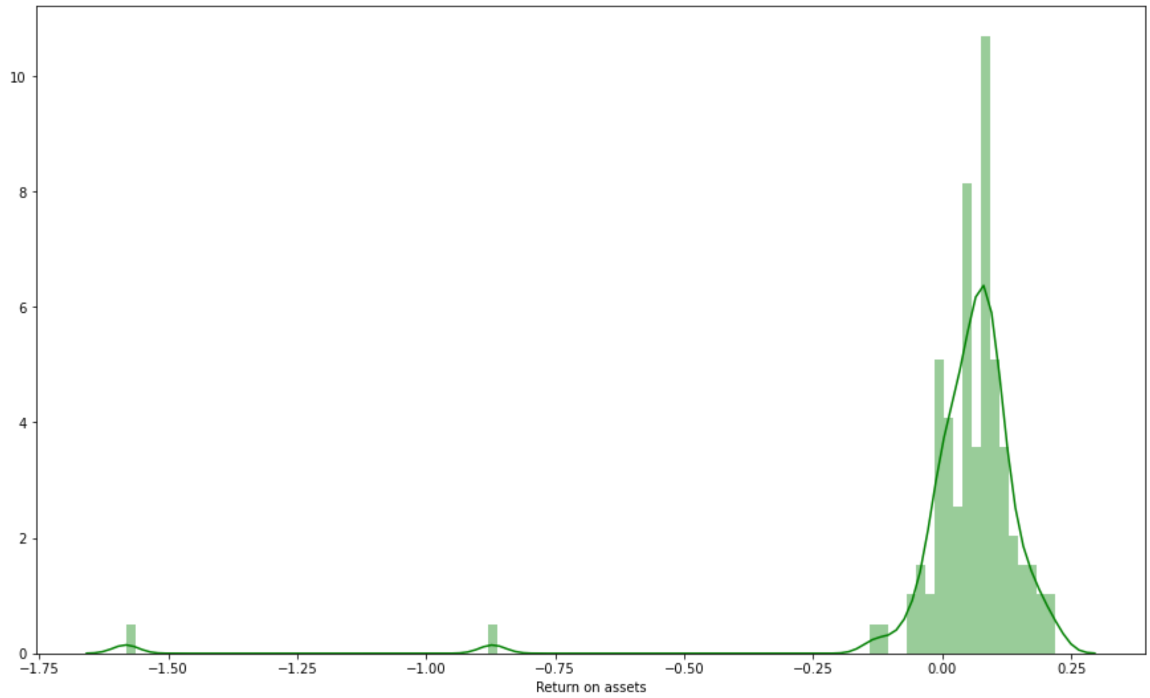
```
Index(['Symbol', 'Debt to equity ratio', 'Net icome', 'Return on assets',
       'Return on investment', 'Net profit margin', 'Inventory turnover ra
tio',
       'revenue to expenses ratio', 'stress_nonstress'],
      dtype='object')
```

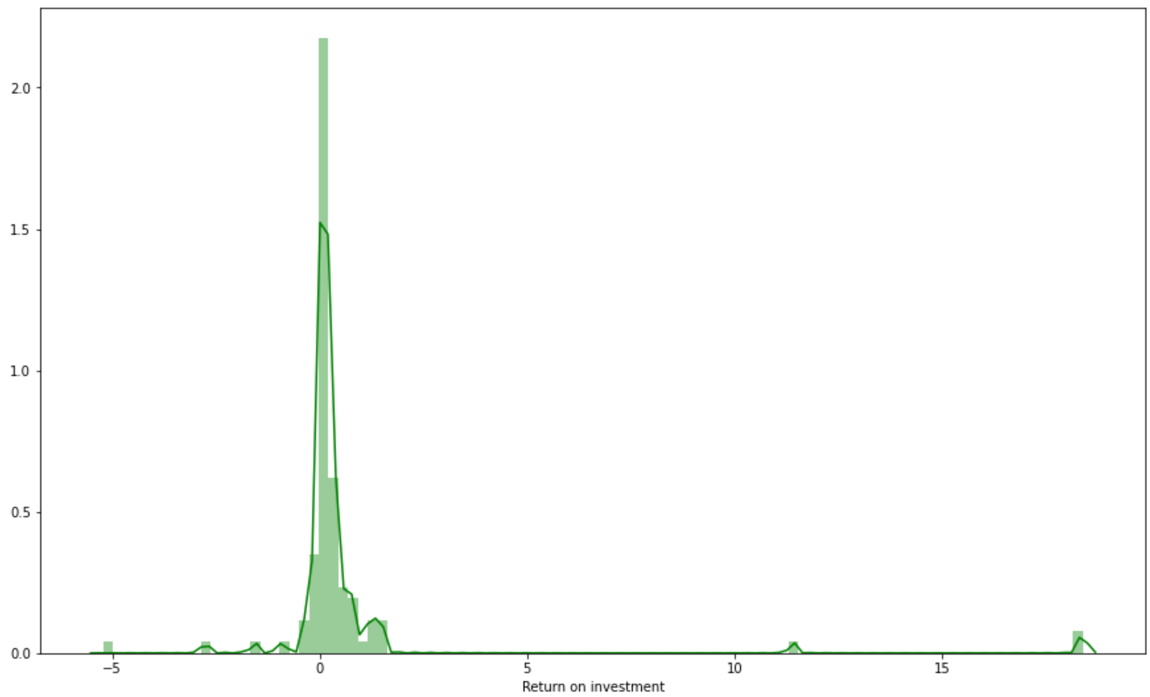
In [267]:

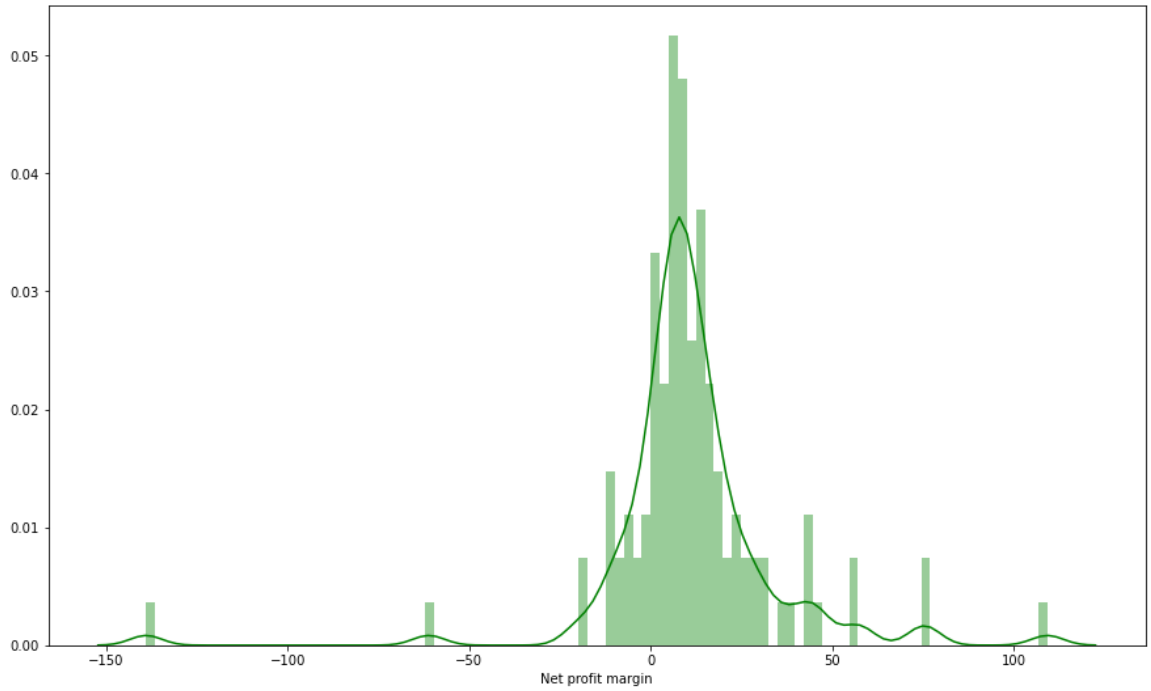
```
disbution_of_data(final_data1['Debt to equity ratio'])
disbution_of_data(final_data1['Net icome'])
disbution_of_data(final_data1['Return on assets'])
disbution_of_data(final_data1['Return on investment'])
disbution_of_data(final_data1['Net profit margin'])
disbution_of_data(final_data1['Inventory turnover ratio'])
disbution_of_data(final_data1['revenue to expenses ratio'])
```

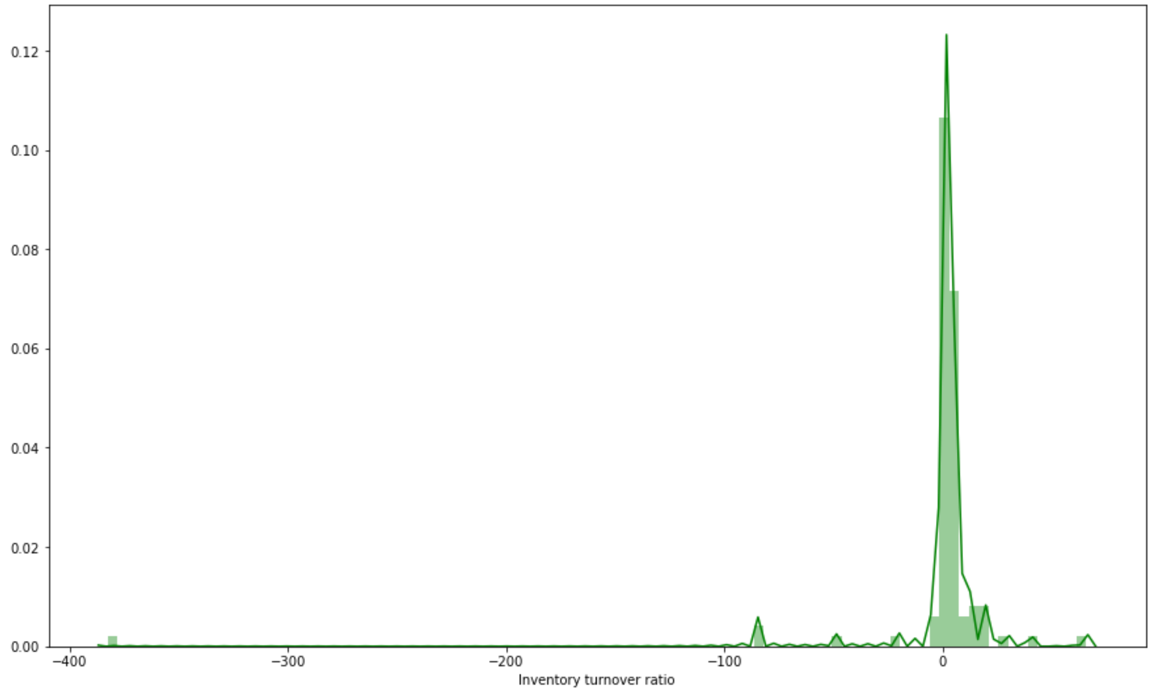

0.4
0.3
0.2
0.1
0.0
−40
−30
−20
−10
0
10
20
Debt to equity ratio

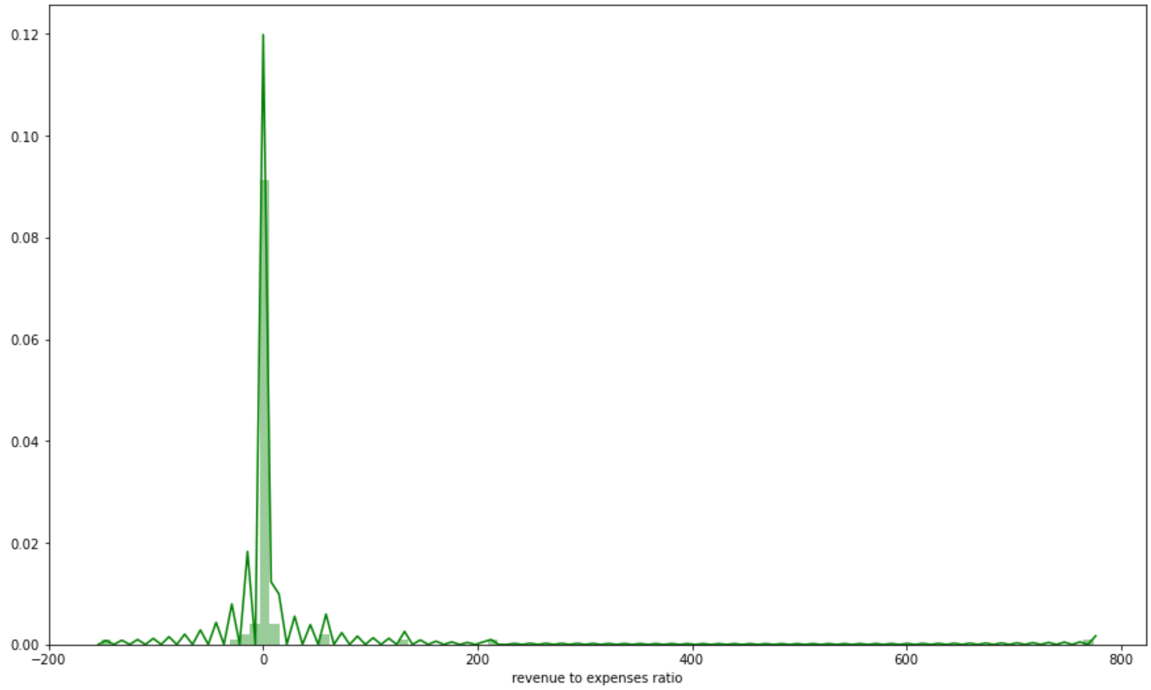

1e−7
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−2
−1
0
1
2
3
4
1e8
Net icome


10
8
6
4
2
0
−1.75
−1.50
−1.25
−1.00
−0.75
−0.50
−0.25
0.00
0.25
Return on assets


2.0
1.5
1.0
0.5
0.0
−5
0
5
10
15
Return on investment


0.05
0.04
0.03
0.02
0.01
0.00
−150
−100
−50
0
50
100
Net profit margin


0.12
0.10
0.08
0.06
0.04
0.02
0.00
−400
−300
−200
−100
0
Inventory turnover ratio

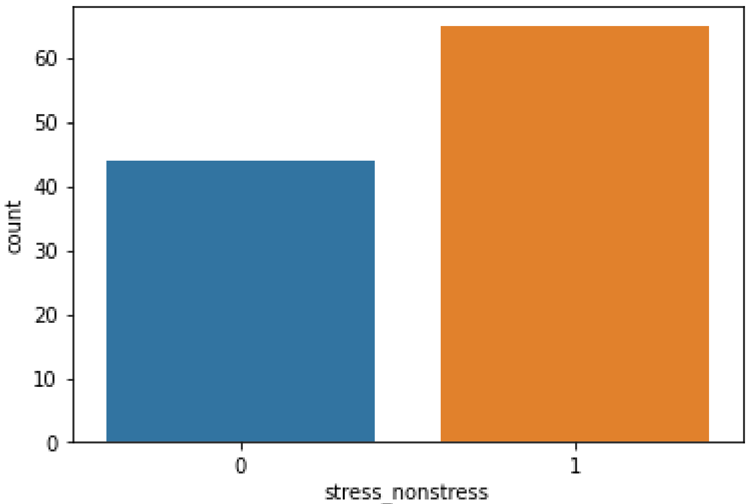


In [268]:

```
#countin of class feture
sns.countplot(final_data1['stress_nonstress'])
```

Out[268]:

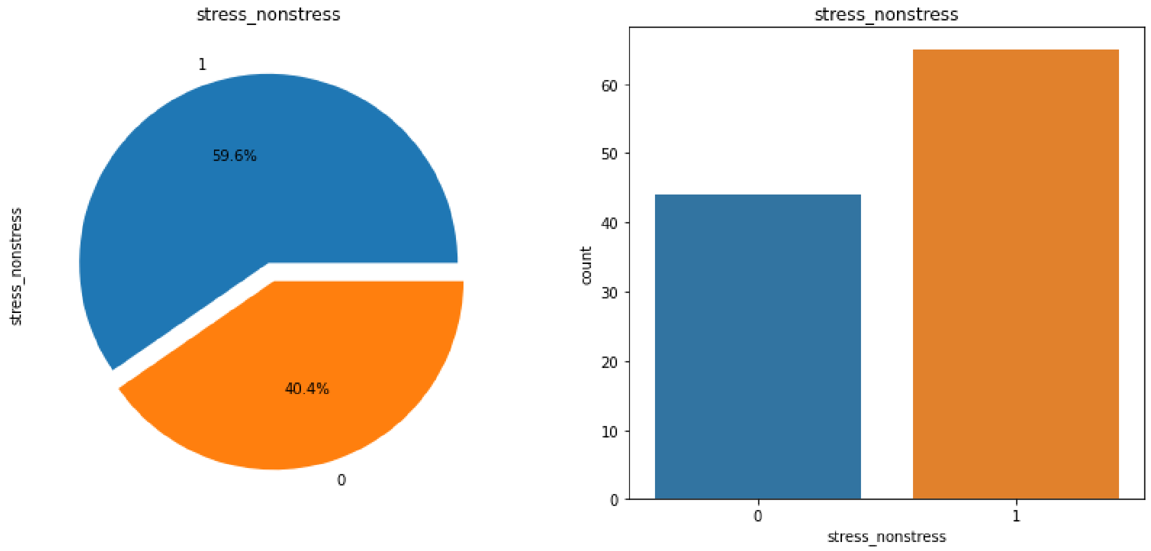
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f729626e50>
```



Univariate Analysis The objective of univariate analysis is to examine each of the variables one by one. The focus will be on the distribution of the variable. Let's start with dependent variable.

In [277]:

```
f,ax=plt.subplots(1,2,figsize=(14,6))
final_data1['stress_nonstress'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%
%',ax=ax[0],shadow=False)
ax[0].set_title('stress_nonstress')
ax[0].set_ylabel('stress_nonstress')
sns.countplot('stress_nonstress',data=final_data1,ax=ax[1])
ax[1].set_title('stress_nonstress')
plt.show()
```



In [282]:

```
final_data1.columns
```

Out[282]:

```
Index(['Symbol', 'Debt to equity ratio', 'Net icome', 'Return on assets',
       'Return on investment', 'Net profit margin', 'Inventory turnover ra
tio',
       'revenue to expenses ratio', 'stress_nonstress'],
      dtype='object')
```

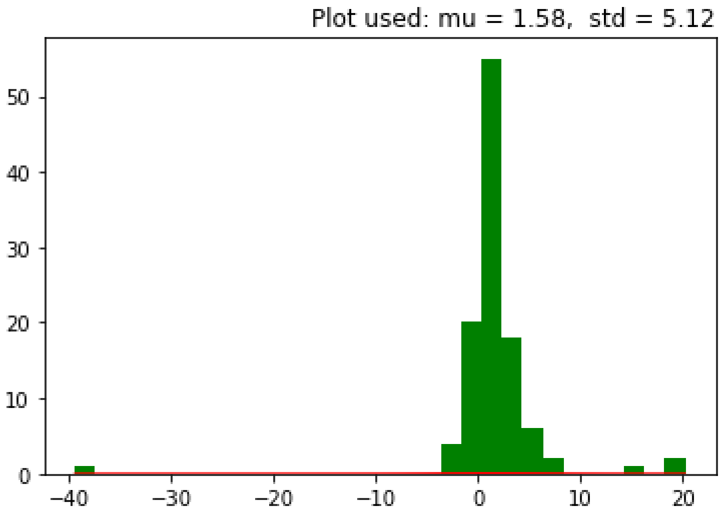
In [289]:

```
columns = ['Debt to equity ratio', 'Net icome', 'Return on assets',
       'Return on investment', 'Net profit margin', 'Inventory turnover ratio',
       'revenue to expenses ratio', 'stress_nonstress']
```

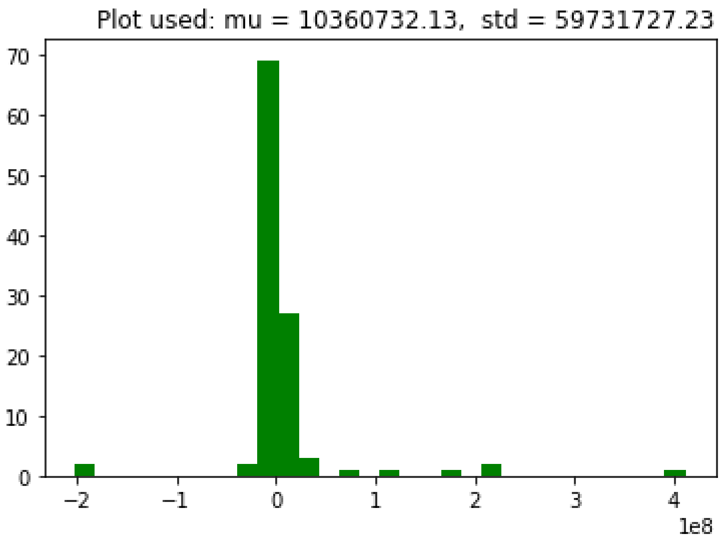
In [295]:

```
for column in columns:
    print(column)
    #s=df['column']
    s=final_data1[column]
    mu, sigma =norm.fit(s)
    count, bins, ignored = plt.hist(s, 30, color='g')
    plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *np.exp( - (bins - mu)**2 / (2 * sigm
a**2) ), linewidth=1, color='r')

    title = "Plot used: mu = %.2f,  std = %.2f" % (mu, sigma)
    plt.title(title, loc='right')

    plt.show()
```

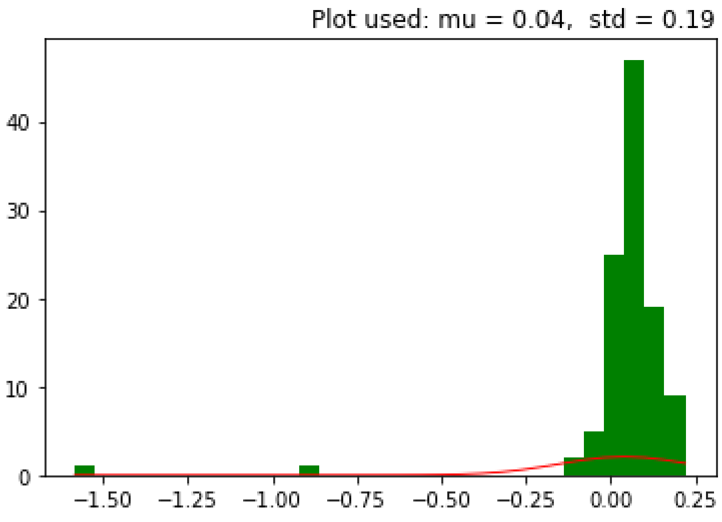
Debt to equity ratio



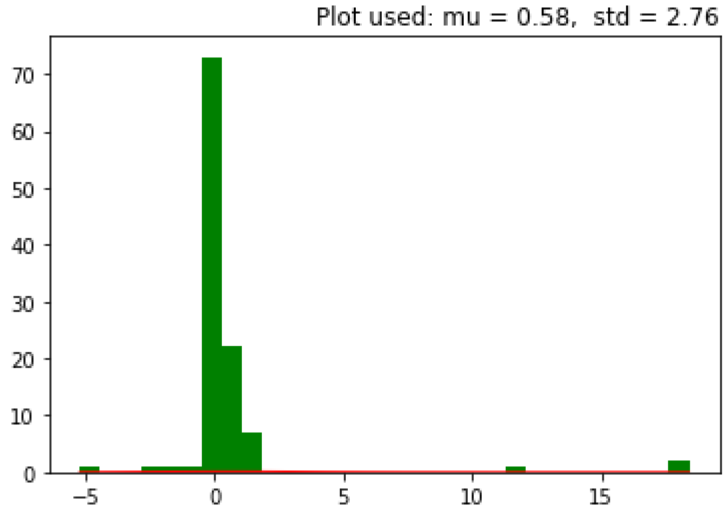
Net icome



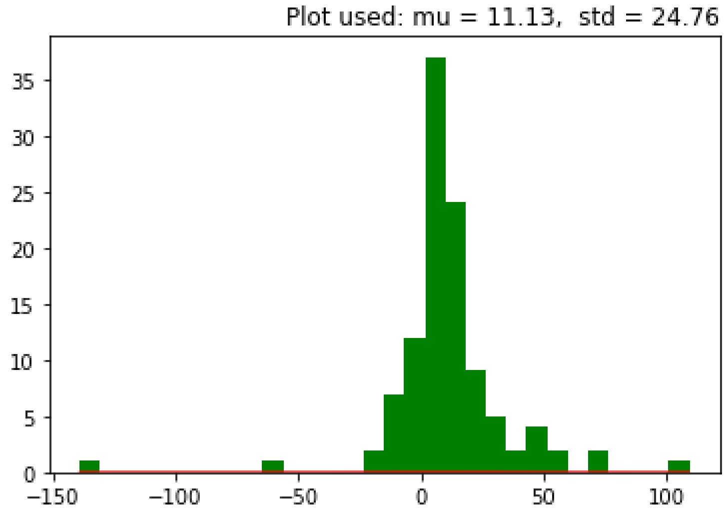
Return on assets



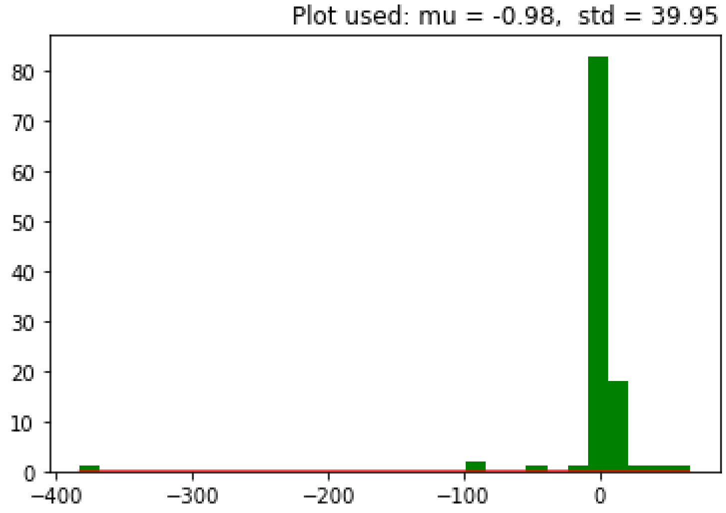
Return on investment



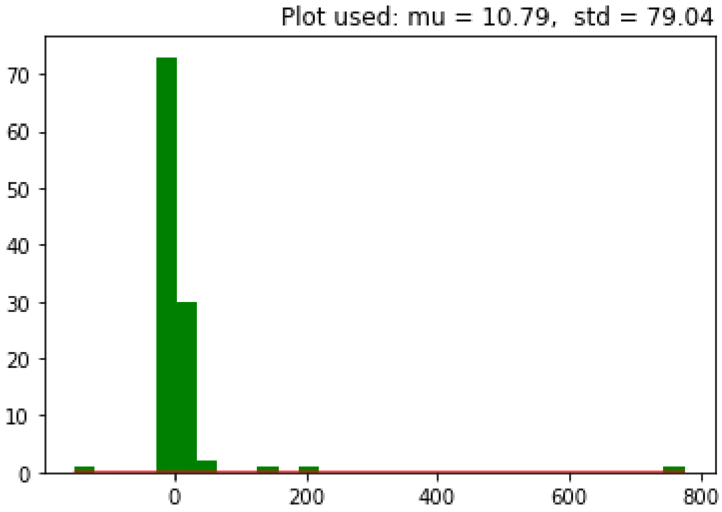
Net profit margin



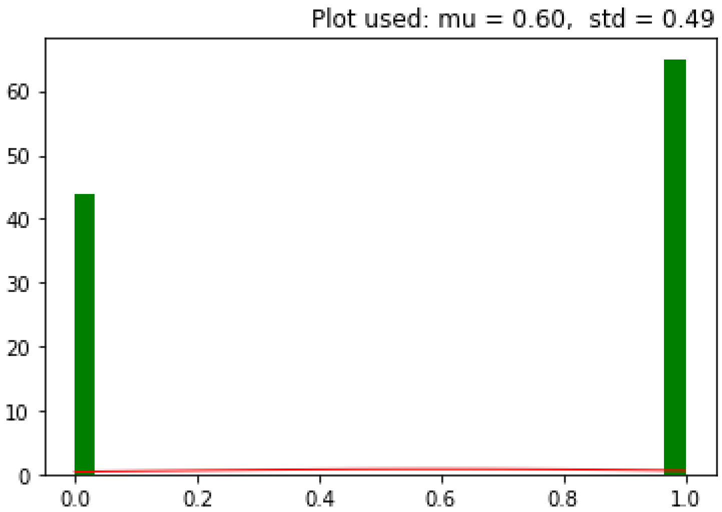
Inventory turnover ratio



revenue to expenses ratio

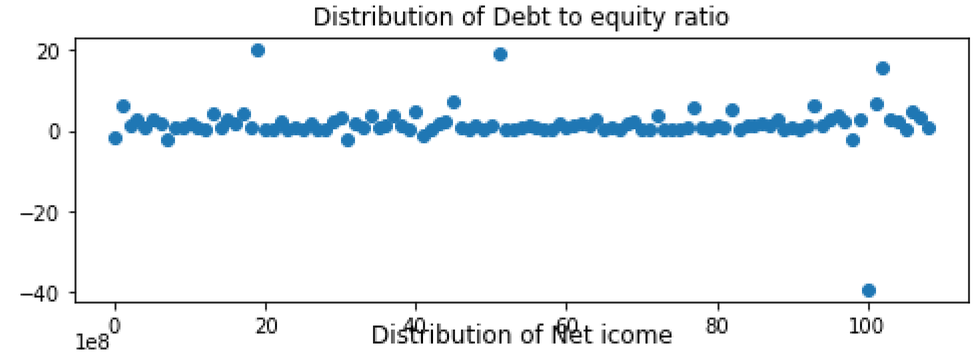


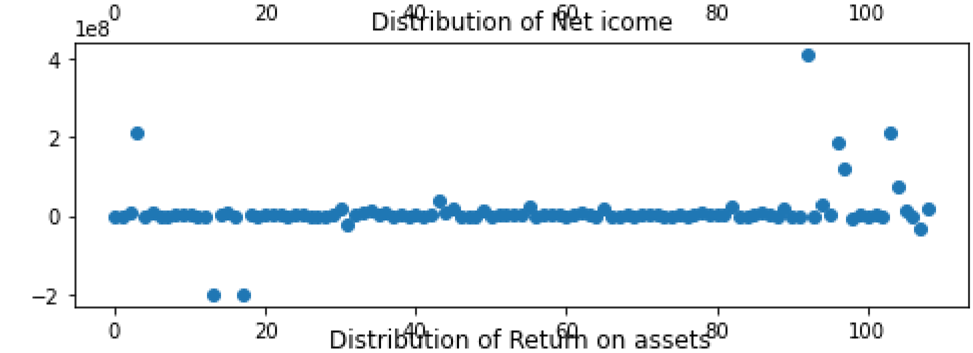
stress_nonstress

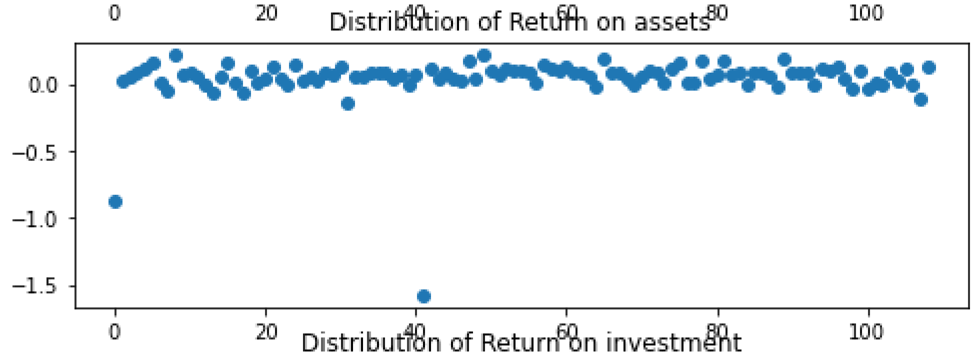


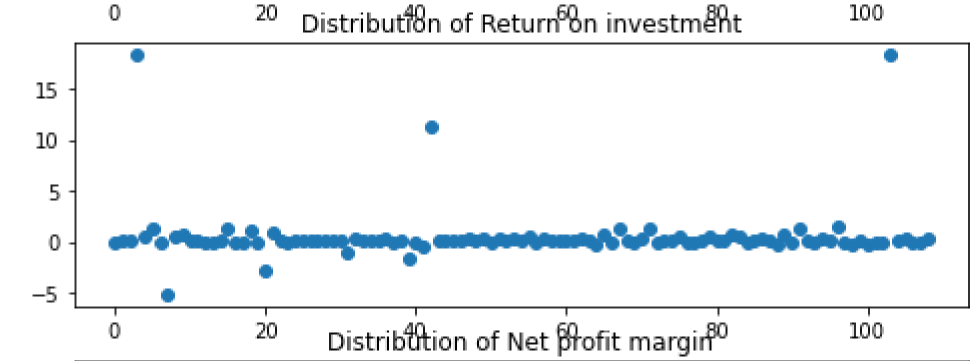
In [307]:

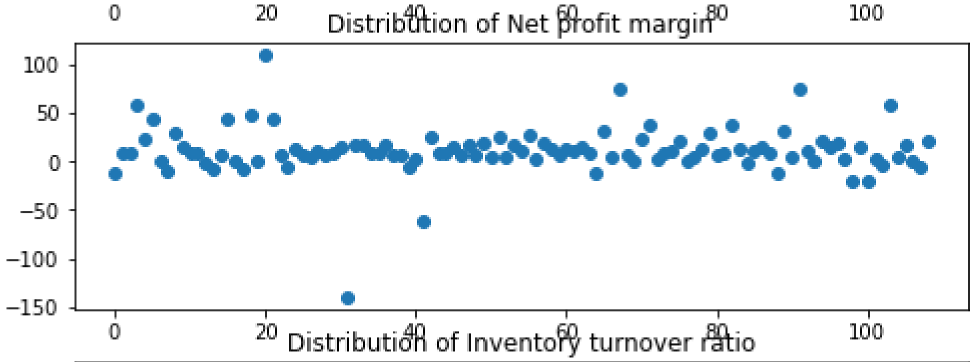
```
cols_to_use = ['Debt to equity ratio', 'Net icome', 'Return on assets',
       'Return on investment', 'Net profit margin', 'Inventory turnover ratio',
       'revenue to expenses ratio']
fig = plt.figure(figsize=(8, 20))
plot_count = 0
for col in cols_to_use:
    plot_count += 1
    plt.subplot(7, 1, plot_count)
    plt.scatter(range(final_data1.shape[0]), final_data1[col].values)
    plt.title("Distribution of "+col)
plt.show()
```

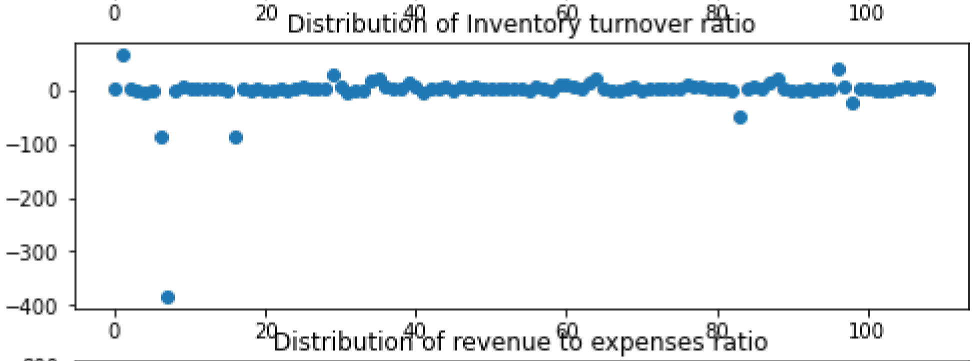

Distribution of Debt to equity ratio
20
0
−20
−40
0
20
40
60
80
100

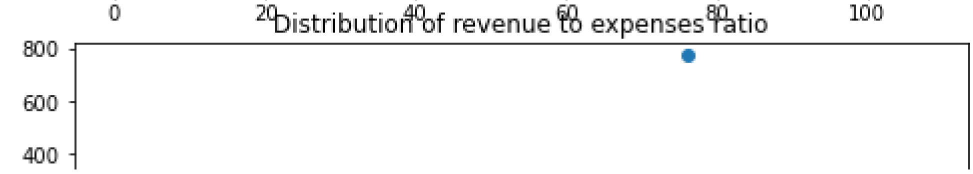

Distribution of Net icome
1e8
4
2
0
−2
0
20
40
60
80
100

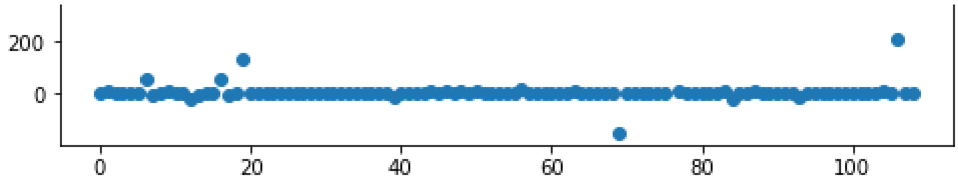

Distribution of Return on assets
0.0
−0.5
−1.0
−1.5
0
20
40
60
80
100


Distribution of Return on investment
15
10
5
0
−5
0
20
40
60
80
100


Distribution of Net profit margin
100
50
0
−50
−100
−150
0
20
40
60
80
100


Distribution of Inventory turnover ratio
0
−100
−200
−300
−400
0
20
40
60
80
100


Distribution of revenue to expenses ratio
800
600
400



In [ ]:

```
columns = ['Debt to equity ratio', 'Net icome', 'Return on assets',
       'Return on investment', 'Net profit margin', 'Inventory turnover ratio',
       'revenue to expenses ratio', 'stress_nonstress']
```

In [437]:

```
df_name=final_data1.columns
```

In [441]:

```
def OutLiersBox(df,nameOfFeature):

    trace0 = go.Box(
        y = df[nameOfFeature],
        name = "All Points",
        jitter = 0.3,
        pointpos = -1.8,
        boxpoints = 'all',
        marker = dict(
            color = 'rgb(7,40,89)'),
        line = dict(
            color = 'rgb(7,40,89)')
    )

    trace1 = go.Box(
        y = df[nameOfFeature],
        name = "Only Whiskers",
        boxpoints = False,
        marker = dict(
            color = 'rgb(9,56,125)'),
        line = dict(
            color = 'rgb(9,56,125)')
    )

    trace2 = go.Box(
        y = df[nameOfFeature],
        name = "Suspected Outliers",
        boxpoints = 'suspectedoutliers',
        marker = dict(
            color = 'rgb(8,81,156)',
            outliercolor = 'rgba(219, 64, 82, 0.6)',
            line = dict(
                outliercolor = 'rgba(219, 64, 82, 0.6)',
                outlierwidth = 2)),
        line = dict(
            color = 'rgb(8,81,156)')
    )

    trace3 = go.Box(
        y = df[nameOfFeature],
        name = "Whiskers and Outliers",
        boxpoints = 'outliers',
        marker = dict(
            color = 'rgb(107,174,214)'),
        line = dict(
            color = 'rgb(107,174,214)')
    )

    data = [trace0,trace1,trace2,trace3]

    layout = go.Layout(
        title = "{} Outliers".format(nameOfFeature)
    )

    fig = go.Figure(data=data,layout=layout)
    py.iplot(fig, filename = "Outliers")
```

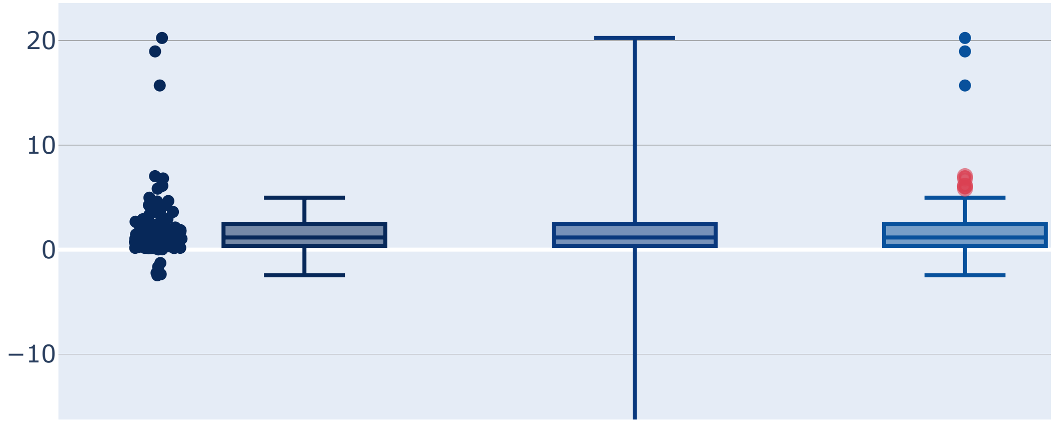
In [ ]:

```
#outliners investgation
```

In [442]:

```
OutLiersBox(final_data1,df_name[1])
```

Debt to equity ratio Outliers



In [443]:

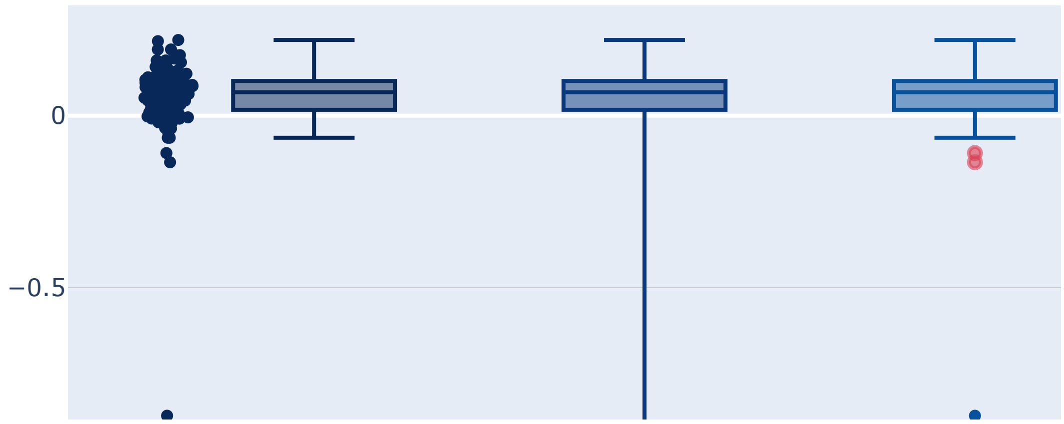
```
OutLiersBox(final_data1,df_name[2])
```

Net icome Outliers

In [445]:

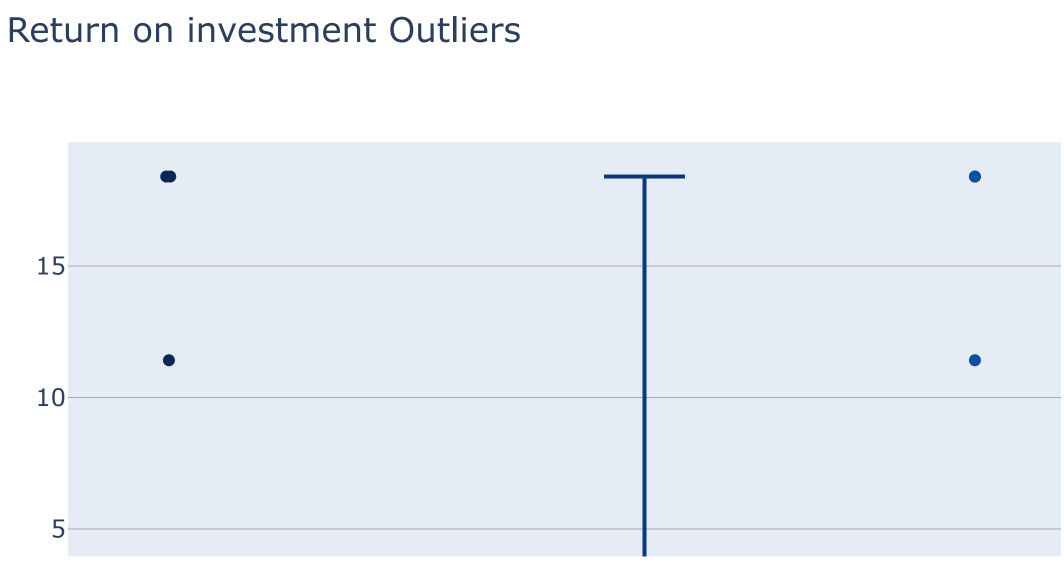
```
OutLiersBox(final_data1,df_name[3])
```

Return on assets Outliers



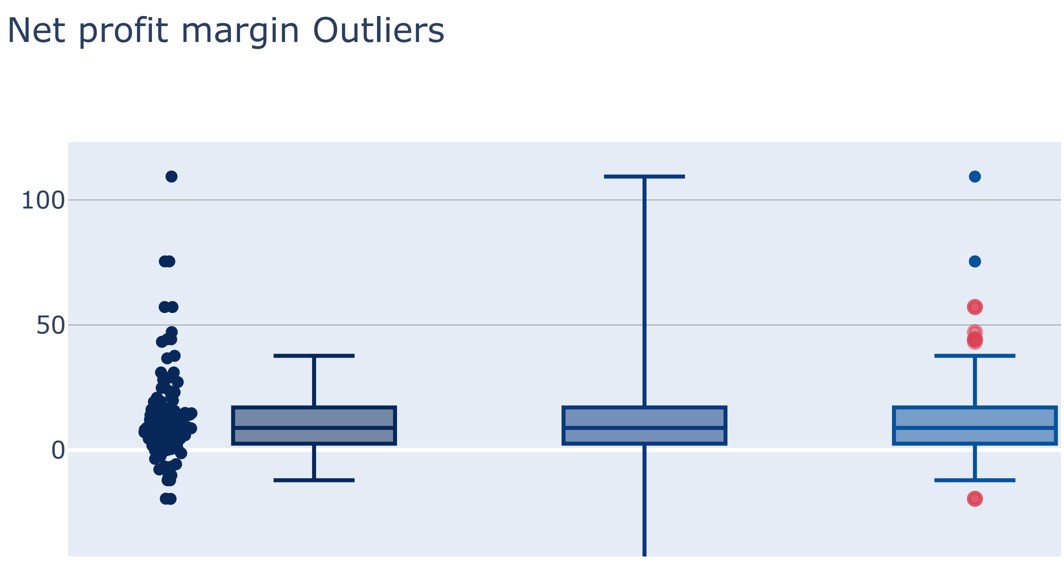
In [446]:

```
OutLiersBox(final_data1,df_name[4])
```



In [447]:

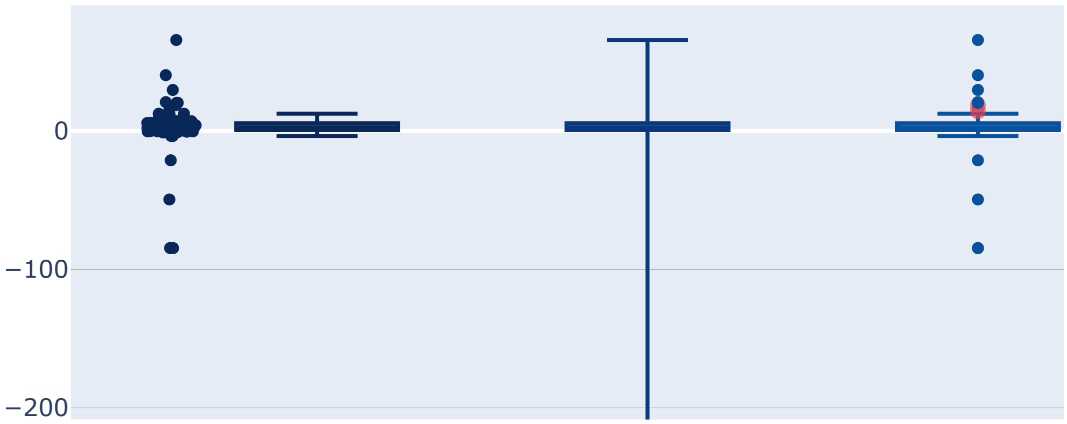
```
OutLiersBox(final_data1,df_name[5])
```



In [448]:

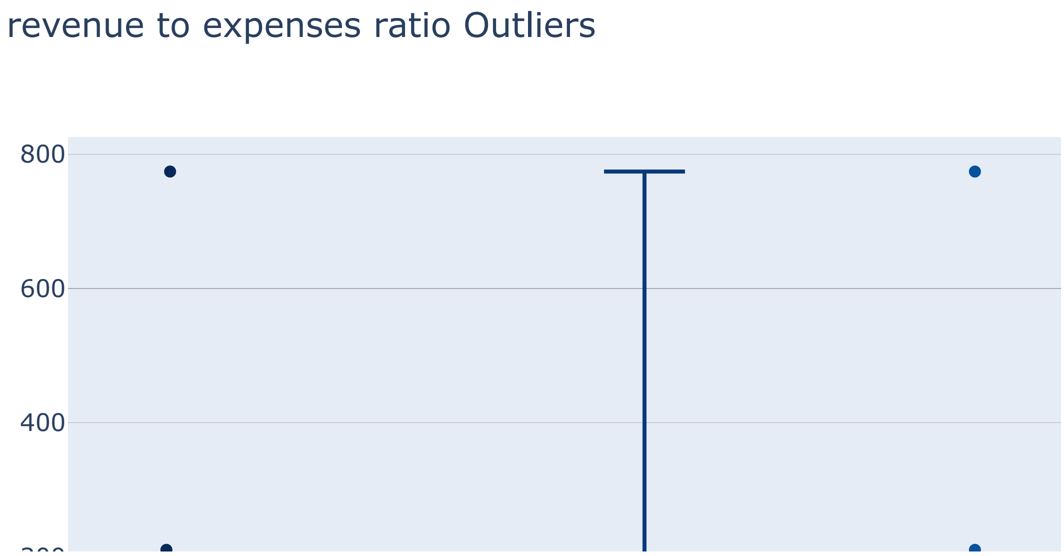
```
OutLiersBox(final_data1,df_name[6])
```

Inventory turnover ratio Outliers



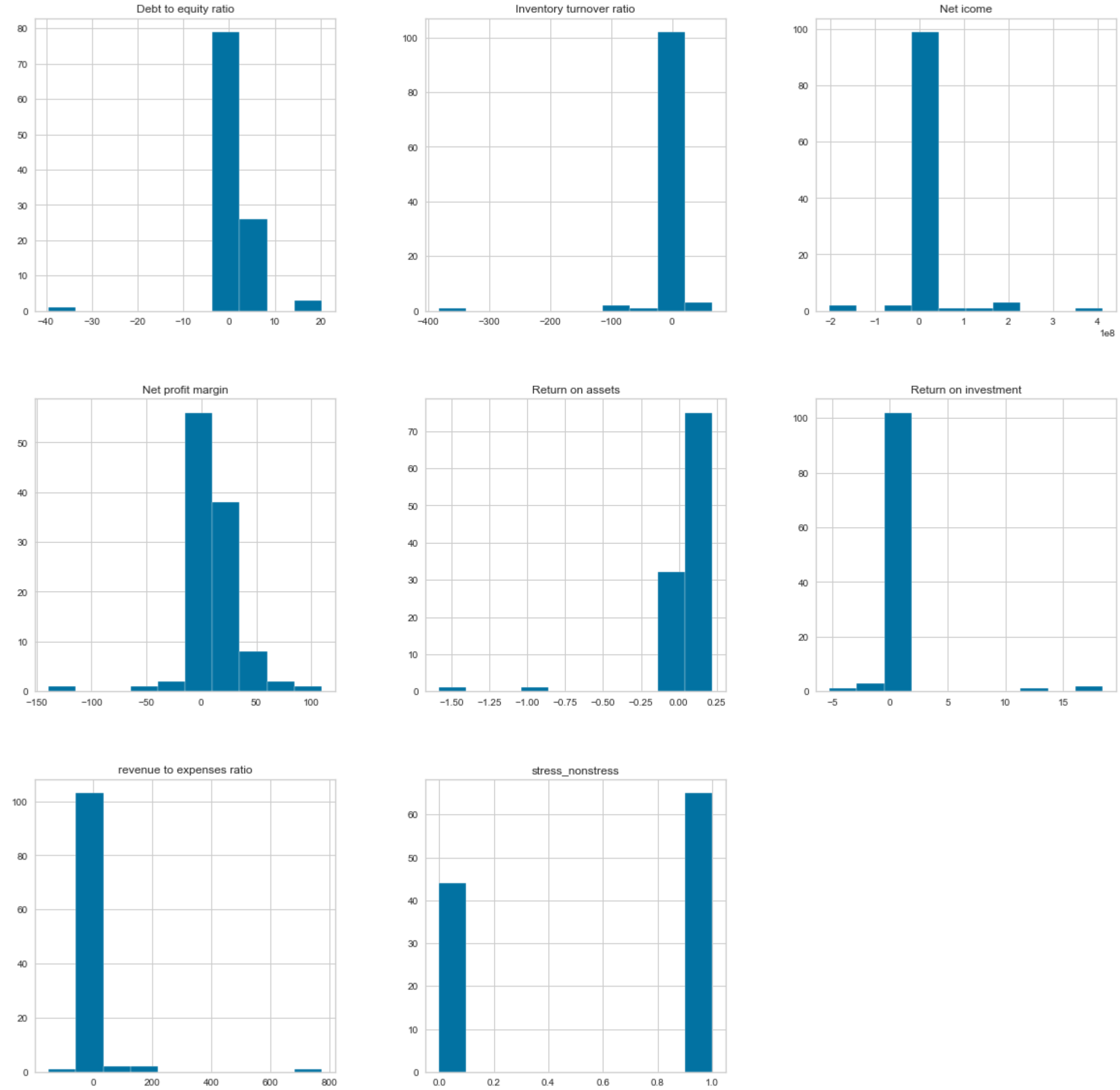
In [449]:

```
OutLiersBox(final_data1,df_name[7])
```



In [451]:

```
#histogram of all fetures
p =final_data1.hist(figsize = (20,20))
```



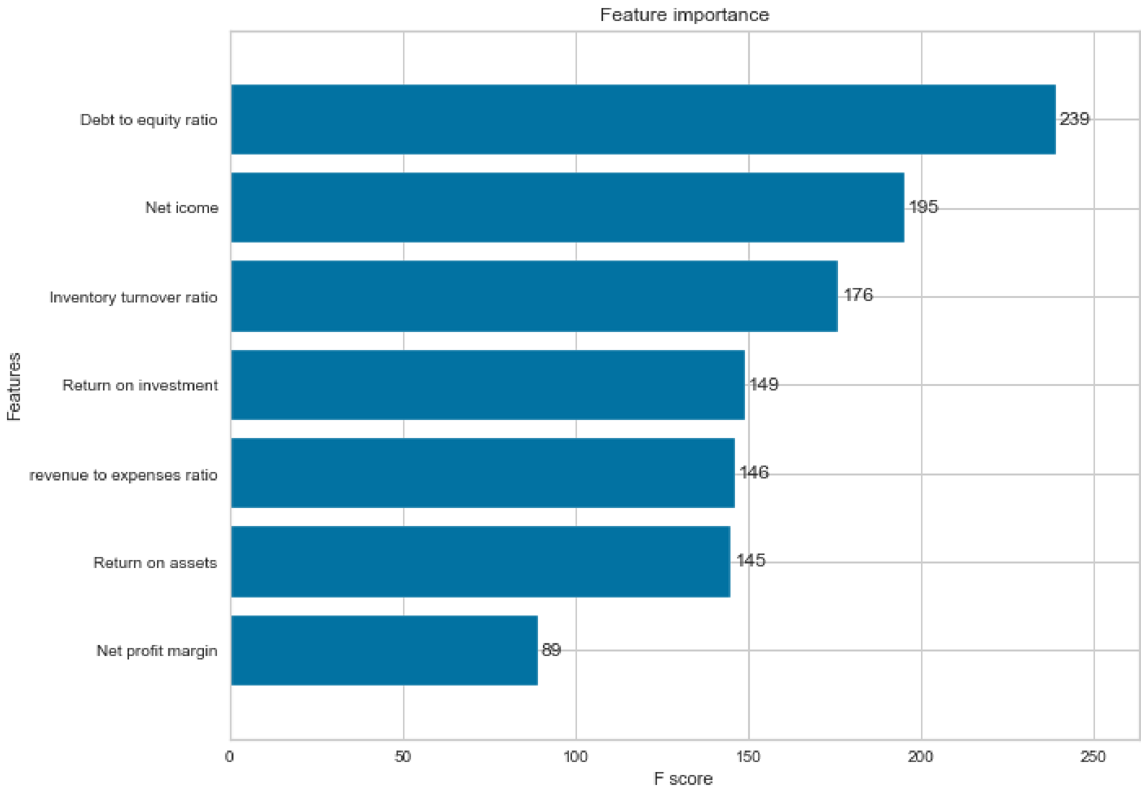
In [454]:

```
#fininding the feture importance using XGboost
import xgboost as xgb

train_y = final_data1['stress_nonstress']
train_X = final_data1.drop(['Symbol','stress_nonstress'], axis=1)

xgb_params = {
    'eta': 0.05,
    'max_depth': 10,
    'subsample': 1.0,
    'colsample_bytree': 0.7,
    'objective': 'reg:linear',
    'eval_metric': 'rmse',
    'silent': 1
}
```

In [455]:

```
import warnings
warnings.filterwarnings('ignore')
dtrain = xgb.DMatrix(train_X, train_y, feature_names=train_X.columns.values)
model = xgb.train(dict(xgb_params, silent=0), dtrain, num_boost_round=100)
remain_num = 99

fig, ax = plt.subplots(figsize=(10,8))
xgb.plot_importance(model, max_num_features=remain_num, height=0.8, ax=ax)
plt.show()
```

```
[09:42:00] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release
_1.2.0/src/objective/regression_obj.cu:174: reg:linear is now deprecated i
n favor of reg:squarederror.
[09:42:00] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release
_1.2.0\src\learner.cc:516:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in languag
e bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip t
hrough this
  verification. Please open an issue if you find above cases.


[09:42:00] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release
_1.2.0/src/objective/regression_obj.cu:174: reg:linear is now deprecated i
n favor of reg:squarederror.
```



Train a machine learning classifier(Logistic regression, support vector machine, naive bays, decision tree) to predict the classes and get

the correlation of coefficients (weights) with respect to the degree of stress.

In [406]:

```
# Time for Classification Models
import time


from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
```

In [407]:

```
dict_classifiers = {
    "Logistic Regression": LogisticRegression(),

    "Linear SVM": SVC(),

    "Decision Tree": tree.DecisionTreeClassifier(),


    "Naive Bayes": GaussianNB()
}
```

In [408]:

```
X=final_data1.drop(['Symbol','stress_nonstress'], axis=1)
y=final_data1['stress_nonstress']
```

In [409]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42
)
```

In [410]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train=scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [411]:

```
X_train
```

Out[411]:

```
array([[-1.36536567e-01, -6.56909417e-02,  5.53351070e-02,
        -1.24002642e-01,  1.60773066e-01,  7.11379511e-02,
        -1.50380988e-01],
       [-8.66794157e-02,  1.59909810e-01,  2.51944821e-01,
        -3.18327791e-02,  6.52175809e-01,  9.02603258e-02,
        -1.32703609e-01],
       [ 1.09804515e-01, -1.70197731e-01, -2.16295054e-01,
        -1.87823567e-01, -5.73149572e-01,  1.91650620e-01,
        -4.53311892e+00],
       [-2.17354652e-01, -1.61437489e-01,  2.33601400e-01,
        -1.54033653e-01, -2.47380278e-01,  1.13385678e-01,
        -5.84147922e-02],
       [ 4.91279275e-01, -1.57240324e-01,  1.76725609e-01,
        -1.80042630e-01, -4.74440583e-01,  2.16982624e-01,
        -1.17455040e-01],
       [ 1.63920223e-01, -6.51861961e-02,  6.14095893e-01,
         2.25205486e-01,  1.41946948e+00,  6.20271966e-02,
        -1.35634764e-01],
       [-2.43085413e-01, -1.48548457e-01,  3.06846094e-01,
        -6.43450332e-02,  2.08249367e-01,  1.40693890e-01,
        -8.77741927e-02],
       [-2.49500309e-02, -2.32629636e-02,  2.13813940e-01,
        -9.07575213e-02,  9.97191704e-02,  1.43010063e-01,
        -8.16910651e-02],
       [-2.74091086e-01, -1.69779262e-01, -2.25864251e-01,
        -7.06327536e-01, -8.59211367e-01,  3.74992694e-01,
        -5.99712380e-01],
       [-1.84755937e-01, -1.51931867e-01,  2.17237521e-01,
        -1.46571923e-01, -1.64985393e-01,  5.12902866e-01,
        -8.92224800e-02],
       [-1.35425404e-02, -1.69403226e-01, -1.83415449e-01,
        -1.85239328e-01, -5.46803056e-01, -1.71302135e+00,
         1.52919125e+00],
       [ 2.12628068e-01,  2.91817444e+00,  2.13798298e-01,
         5.86640403e+00,  1.99720194e+00,  8.95827147e-02,
        -1.28843210e-01],
       [ 3.17976052e-01, -8.34362202e-03,  2.31319393e-01,
        -1.43971831e-01, -1.45955418e-01,  4.68218021e-01,
        -8.97065667e-02],
       [-2.40645753e-01, -1.68787436e-01,  2.50628585e-01,
         2.84053352e-01,  2.81431873e+00,  6.75542360e-02,
        -1.59729811e-01],
       [-6.69370989e-01, -1.79214689e-01, -4.59137201e-01,
        -1.90389853e+00, -1.01696549e+00, -8.00493563e+00,
        -4.59156045e-01],
       [ 2.94494336e-02,  4.21896266e-01,  2.45798392e-02,
        -1.46518848e-01, -1.56509908e-01,  1.03456081e-01,
        -9.56484763e-02],
       [-1.58803000e-01, -1.67445136e-01,  1.87001463e-01,
        -1.71297671e-01, -3.61094998e-01,  7.23472740e-02,
        -1.48877673e-01],
       [-2.05905338e-01, -1.67269077e-01, -1.57704908e-01,
        -1.48167155e-01, -2.13188332e-01,  1.13033060e-01,
        -5.44340005e-02],
       [-2.59596263e-01, -1.60103904e-01, -9.90930899e-02,
        -1.32353848e-01, -1.09858047e-01,  1.41384945e-01,
        -6.71596742e-02],
       [ 3.07501801e+00, -1.68907754e-01, -1.92585971e-01,
        -1.84710449e-01, -5.48713717e-01,  1.20019081e-01,
```

```
 3.68344660e+00],
[ 1.49343803e-01, -1.78056371e-01, -3.03354432e-01,
 -2.33640313e-01, -1.10602955e+00,  5.02838855e-01,
 -2.13479769e-01],
[ 7.26745665e-01, -1.80890327e-01, -2.64958232e-01,
 -1.88037009e-01, -5.78842273e-01,  5.70027359e-02,
 -4.98338140e-01],
[-2.41463773e-01, -1.55994805e-01, -1.12899514e-01,
 -1.58724858e-01, -2.88416631e-01,  2.33504224e-01,
 -4.57662628e-02],
[-1.65749362e-01, -8.99290027e-02,  9.16768283e-01,
 -1.98462836e-02,  7.35717016e-01,  9.27718794e-02,
 -1.35147127e-01],
[ 8.48801348e-01, -1.48794345e-01, -1.46613923e-01,
 -1.80671199e-01, -4.85007716e-01,  8.09143442e-02,
 -6.45452010e-02],
[-2.12879344e-01,  3.77320236e-02,  8.99728219e-01,
 -5.17233358e-02,  3.07628319e-01,  1.26303206e-01,
 -9.96394059e-02],
[ 4.25511149e-01, -3.12519774e+00, -5.32171157e-01,
 -2.24552549e-01, -9.12651346e-01,  1.69040294e-01,
 -2.97340461e-01],
[ 6.87211745e-01, -1.39872115e-01, -1.55199722e-01,
 -1.68580981e-01, -3.81407030e-01,  1.95310740e-01,
  1.14884775e-02],
[-1.59068287e-01, -1.25730874e-01,  5.22481150e-01,
 -1.25915022e-01, -3.97283302e-03,  1.36975065e-01,
 -1.02602431e-01],
[ 2.12628068e-01,  2.91817444e+00,  2.13798298e-01,
  5.86640403e+00,  1.99720194e+00,  8.95827147e-02,
 -1.28843210e-01],
[ 4.25511149e-01, -3.12519774e+00, -5.32171157e-01,
 -2.24552549e-01, -9.12651346e-01,  1.69040294e-01,
 -2.97340461e-01],
[-6.06578936e-02, -1.46379665e-01,  1.04081466e-01,
 -1.60723425e-01, -2.63061914e-01,  1.01260679e-01,
 -1.10891637e-01],
[-8.31230879e-02, -1.55262733e-01,  2.32532514e-01,
 -1.17585076e-01, -9.27753168e-02,  1.97987911e-01,
 -4.03597223e-02],
[-1.35425404e-02, -1.69403226e-01, -1.83415449e-01,
 -1.85239328e-01, -5.46803056e-01, -1.71302135e+00,
  1.52919125e+00],
[ 1.00092956e-01,  9.18649596e-01, -5.35785217e-02,
 -1.52555871e-01, -3.27034139e-01,  1.65398319e-01,
  9.09324282e-02],
[ 2.02324636e-01, -1.46373943e-01,  3.27411839e-01,
 -1.22002512e-01,  6.09214094e-02,  1.00977192e-01,
 -1.17195662e-01],
[-2.40645753e-01, -1.68787436e-01,  2.50628585e-01,
  2.84053352e-01,  2.81431873e+00,  6.75542360e-02,
 -1.59729811e-01],
[-1.46163126e-01, -1.19352826e-01,  3.16456114e-01,
 -1.44890730e-01, -1.34558387e-01,  1.32714884e-01,
 -1.02748416e-01],
[-7.70897188e-02, -1.68205658e-01,  2.77734807e-01,
 -1.66796098e-01, -3.59142294e-01,  1.58971195e-01,
 -1.92693364e-02],
[-6.85805150e-01, -2.63449984e-01, -3.86812144e-01,
 -2.80475429e-01, -1.43597072e+00, -3.74528774e-01,
 -2.24923804e-01],
```

```
       [-1.91376232e-01, -1.67695161e-01, -9.29427117e-02,
        -1.32285777e-01, -3.02494319e-01,  1.79716023e-01,
         1.42477430e-01],
       [-2.26584103e-01, -1.65459373e-01,  2.33204963e-01,
        -3.02799869e-02,  4.06765279e-02, -9.72205013e-01,
        -2.44502270e-02],
       [-8.31230879e-02, -1.55262733e-01,  2.32532514e-01,
        -1.17585076e-01, -9.27753168e-02,  1.97987911e-01,
        -4.03597223e-02],
       [-2.25544081e-01,  3.24630075e-02,  3.63792514e-01,
        -6.85302275e-02,  1.94832631e-01,  2.26103135e-01,
        -8.75550220e-02],
       [-6.81388871e+00, -2.10343114e-01, -3.94828760e-01,
        -2.92247671e-01, -1.43944338e+00,  1.25846156e-01,
        -2.35720119e-01],
       [-4.91265806e-01, -1.72833207e-01, -8.28231706e+00,
        -3.25535773e-01, -3.30363350e+00, -3.14587375e-03,
        -1.56524806e-01],
       [-2.56778063e-01, -1.16030514e-01,  3.43887164e-01,
        -1.29935668e-01,  1.28234549e-02,  8.60926658e-02,
        -1.20325293e-01],
       [-5.27854945e-02, -1.64966673e-01,  1.55020085e-02,
        -1.42988041e-01, -2.46595828e-01,  1.94793933e-01,
         1.32082716e-02],
       [-4.46942506e-02,  2.18046529e-01,  3.49338858e-01,
        -6.45369840e-02,  3.72419104e-01,  1.01504533e-01,
        -1.17287332e-01],
       [-2.27875646e-01, -1.58808718e-01,  5.19019284e-01,
        -1.00256498e-01,  2.49681418e-01,  1.05361423e-01,
        -1.28474559e-01],
       [-2.50584756e-01, -1.28611666e-01,  5.84829881e-01,
         2.42640543e-02,  3.74754067e-01,  1.65808630e-01,
        -8.49305625e-02],
       [ 1.37007929e-02, -1.53033504e-01,  3.43020084e-02,
        -9.65133795e-02,  1.82156995e-01,  8.20752670e-02,
        -1.09532923e-01],
       [-1.30932645e-01,  1.18522136e-01,  4.65769037e-01,
        -4.45569622e-02,  3.23510789e-01,  1.38031101e-01,
        -9.87542001e-02],
       [ 3.36343575e-02, -1.55795152e-01,  2.81461538e-01,
        -1.61244966e-01, -2.99931694e-01,  2.78545967e-01,
        -5.83986752e-02],
       [-6.65645445e-02, -1.47853963e-01,  6.08855917e-02,
        -1.30301031e-01, -2.01363322e-01,  3.36726990e-01,
         1.09511362e-02],
       [ 4.83488854e-01, -1.66431712e-01, -2.03014997e-01,
        -1.86418547e-01, -5.59246126e-01,  1.28193772e-01,
         5.94650704e+00],
       [ 3.43780358e-01, -1.68510663e-01, -3.44006357e-02,
        -1.67215190e-01, -3.19245609e-01,  1.25727043e-01,
        -1.31792738e-01],
       [ 1.06585631e-01, -1.55191306e-01,  1.10111537e-01,
        -1.55575926e-01, -2.31827888e-01,  6.99253150e-01,
        -9.17127316e-02],
       [ 1.22798750e-01,  1.55728927e+00, -2.25686405e-02,
        -1.78019457e-01, -4.54465819e-01,  2.12791888e-01,
        -6.64395609e-02],
       [ 7.32149632e-01, -1.63586649e-01, -5.03583696e-02,
        -1.07699451e-01, -2.13926957e-01,  1.46061407e+00,
         7.16524269e-02],
       [-2.27843173e-01, -1.40363948e-01,  3.84259998e-01,
```

```
 -1.65467335e-01, -3.29564825e-01,  1.41736093e-01,
 -6.94123987e-02],
[-2.54331935e-01, -1.23067460e-01,  4.13064344e-01,
  1.09710974e-01,  1.37417129e+00,  7.98726187e-02,
 -1.43102199e-01],
[-7.98920420e-02, -8.14824368e-02,  6.08177629e-02,
 -1.44804384e-01, -2.13295220e-01,  1.27006157e-01,
 -3.47446898e-02],
[-2.57770218e-01, -1.69793920e-01, -2.33781722e-01,
 -2.07250634e-01, -8.18649185e-01,  4.94918303e-02,
 -2.19253247e-01],
[-6.65645445e-02, -1.47853963e-01,  6.08855917e-02,
 -1.30301031e-01, -2.01363322e-01,  3.36726990e-01,
  1.09511362e-02],
[ 2.02324636e-01, -1.46373943e-01,  3.27411839e-01,
 -1.22002512e-01,  6.09214094e-02,  1.00977192e-01,
 -1.17195662e-01],
[-2.17084983e-01, -1.63384663e-01,  3.82471042e-01,
 -1.21258013e-01, -7.37781801e-02,  1.38059192e-01,
 -5.86646527e-02],
[-2.49500309e-02, -2.32629636e-02,  2.13813940e-01,
 -9.07575213e-02,  9.97191704e-02,  1.43010063e-01,
 -8.16910651e-02],
[ 5.44289209e-01,  1.98875146e-01,  1.16735664e-01,
  6.71754962e-02,  1.07810360e+00,  8.34547179e-02,
 -1.36419833e-01],
[-2.75656415e-01, -1.54047955e-01,  1.96914629e-02,
 -1.09524950e+00,  4.33459652e+00,  4.17663439e-02,
 -1.40937295e-01],
[-1.88257111e-01, -1.64726641e-01,  4.59442386e-01,
 -1.02985169e-01, -2.36637504e-02,  3.07072582e-01,
 -5.15956246e-02],
[-2.57924557e-01, -1.57318134e-01,  2.96524949e-01,
  2.32839939e-01,  1.12318713e+00,  1.08686707e-01,
 -1.23741316e-01],
[-1.58059627e-01, -1.51208643e-01,  6.42285741e-02,
 -1.57074986e-01, -2.41859895e-01,  1.16938535e-01,
 -9.48246271e-02],
[-7.32618500e-02,  5.86582550e+00,  2.11928291e-01,
 -1.43381584e-01, -9.99916467e-02,  9.82598556e-02,
 -1.17333889e-01],
[ 2.86187361e+00, -8.70761628e-02,  1.04544905e-01,
 -4.15799732e-02,  5.47346504e-01,  1.25748956e-01,
 -1.26772992e-01],
[ 2.32222700e+00, -1.95554361e-01, -2.29523819e-01,
 -1.97863934e-01, -7.22899085e-01,  6.60884125e-02,
 -1.24224933e-01]])
```

In [412]:

```
from time import time
no_classifiers = len(dict_classifiers.keys())

def batch_classify(X_train, Y_train, verbose = True):
    df_results = pd.DataFrame(data=np.zeros(shape=(no_classifiers,3)), columns = ['clas
sifier', 'train_score', 'training_time'])
    count = 0
    for key, classifier in dict_classifiers.items():
        t_start = time()
        classifier.fit(X_train, Y_train)
        t_end = time()
        t_diff = t_end - t_start
        train_score = classifier.score(X_train, Y_train)
        df_results.loc[count,'classifier'] = key
        df_results.loc[count,'train_score'] = train_score
        df_results.loc[count,'training_time'] = t_diff
        if verbose:
            print("trained {c} in {f:.2f} s".format(c=key, f=t_diff))
        count+=1
    return df_results
```

In [413]:

```
df_results = batch_classify(X_train, y_train)
print(df_results.sort_values(by='train_score', ascending=False))
```

```
trained Logistic Regression in 0.01 s
trained Linear SVM in 0.00 s
trained Decision Tree in 0.00 s
trained Naive Bayes in 0.00 s
            classifier  train_score  training_time
2        Decision Tree         1.00           0.00
3          Naive Bayes         0.87           0.00
0  Logistic Regression         0.83           0.01
1           Linear SVM         0.83           0.00
```

In [414]:

```
#Avoiding Overfitting:
# Use Cross-validation.
from sklearn.model_selection import cross_val_score

# Logistic Regression
log_reg = LogisticRegression()
log_scores = cross_val_score(log_reg, X_train, y_train, cv=3)
log_reg_mean = log_scores.mean()

# SVC
svc_clf = SVC()
svc_scores = cross_val_score(svc_clf, X_train, y_train, cv=3)
svc_mean = svc_scores.mean()

# KNearestNeighbors
knn_clf = KNeighborsClassifier()
knn_scores = cross_val_score(knn_clf, X_train, y_train, cv=3)
knn_mean = knn_scores.mean()

# Decision Tree
tree_clf = tree.DecisionTreeClassifier()
tree_scores = cross_val_score(tree_clf, X_train, y_train, cv=3)
tree_mean = tree_scores.mean()

# Gradient Boosting Classifier
grad_clf = GradientBoostingClassifier()
grad_scores = cross_val_score(grad_clf, X_train, y_train, cv=3)
grad_mean = grad_scores.mean()

# Random Forest Classifier
rand_clf = RandomForestClassifier(n_estimators=18)
rand_scores = cross_val_score(rand_clf, X_train, y_train, cv=3)
rand_mean = rand_scores.mean()

# NeuralNet Classifier
neural_clf = MLPClassifier(alpha=1)
neural_scores = cross_val_score(neural_clf, X_train, y_train, cv=3)
neural_mean = neural_scores.mean()

# Naives Bayes
nav_clf = GaussianNB()
nav_scores = cross_val_score(nav_clf, X_train, y_train, cv=3)
nav_mean = neural_scores.mean()

# Create a Dataframe with the results.
d = {'Classifiers': ['Logistic Reg.', 'SVC', 'KNN', 'Dec Tree', 'Grad B CLF', 'Rand FC'
, 'Neural Classifier', 'Naives Bayes'],
    'Crossval Mean Scores': [log_reg_mean, svc_mean, knn_mean, tree_mean, grad_mean, ra
nd_mean, neural_mean, nav_mean]}

result_df = pd.DataFrame(data=d)
```

In [415]:

```
# All our models perform well but I will go with GradientBoosting.
result_df = result_df.sort_values(by=['Crossval Mean Scores'], ascending=False)
result_df
```

Out[415]:

	Classifiers	Crossval Mean Scores
3	Dec Tree	0.99
4	Grad B CLF	0.99
5	Rand FC	0.95
2	KNN	0.83
6	Neural Classifier	0.82
7	Naives Bayes	0.82
0	Logistic Reg.	0.80
1	SVC	0.74

In [416]:

```
# Cross validate our Gradient Boosting Classifier
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(grad_clf, X_train, y_train, cv=3)
```

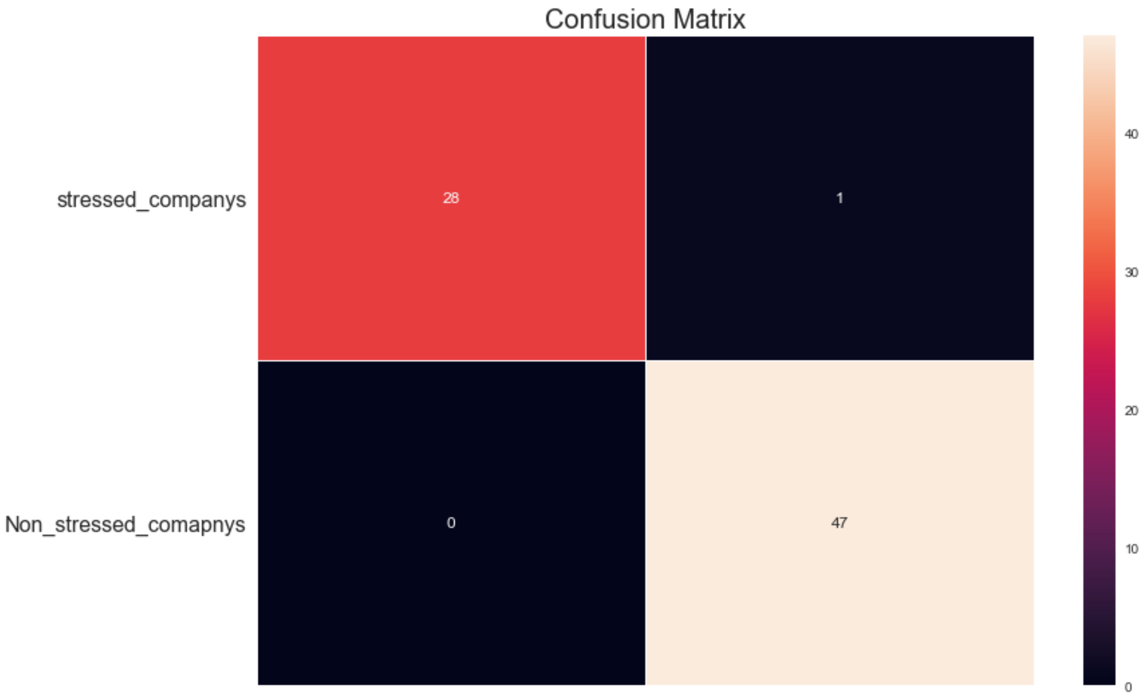
In [417]:

```
from sklearn.metrics import accuracy_score
grad_clf.fit(X_train, y_train)
print ("Gradient Boost Classifier accuracy is %2.2f" % accuracy_score(y_train, y_train_
pred))
```

```
Gradient Boost Classifier accuracy is 0.99
```

In [418]:

```
f#evalation Matrix
rom sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(y_train, y_train_pred)
f, ax = plt.subplots(figsize=(12, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", linewidths=.5, ax=ax)
plt.title("Confusion Matrix", fontsize=20)
plt.subplots_adjust(left=0.15, right=0.99, bottom=0.15, top=0.99)
ax.set_yticks(np.arange(conf_matrix.shape[0]) + 0.5, minor=False)
ax.set_xticklabels("")
ax.set_yticklabels(['stressed_companys', 'Non_stressed_comapnys'], fontsize=16, rotatio
n=360)
plt.show()
```



In [419]:

```
# Let's find the scores  for precision and recall.
from sklearn.metrics import precision_score, recall_score

print('Precision Score: ', precision_score(y_train, y_train_pred))

print('Recall Score: ', recall_score(y_train, y_train_pred))
```

```
Precision Score:  0.9791666666666666
Recall Score:  1.0
```

In [420]:

```
from sklearn.metrics import f1_score

f1_score(y_train, y_train_pred)
```

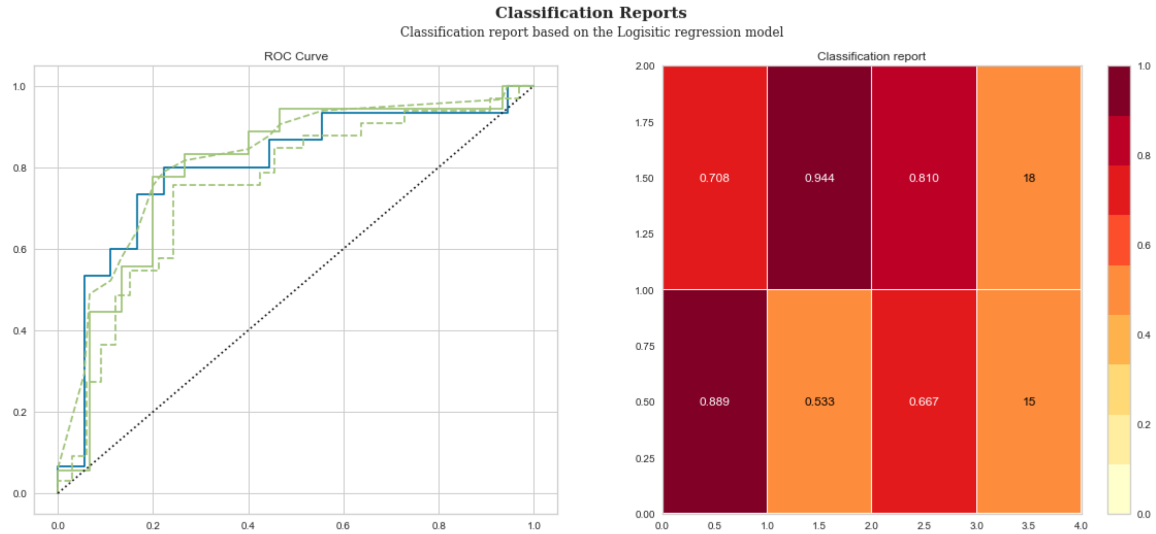
Out[420]:

```
0.9894736842105264
```

In [421]:

```
from sklearn.linear_model import LogisticRegression
from yellowbrick.classifier import ROCAUC, ClassificationReport, ClassificationScoreVis
ualizer
model = LogisticRegression()
model.fit(X_train, y_train)
model.score(X_test ,y_test)
pred=model.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
print(accuracy_score(y_test,pred))
print(classification_report(y_test,pred))

## Yellow brick reports
fig = plt.figure(figsize=(20,8))
gs=GridSpec(nrows=1, ncols=2)
plt.suptitle("Classification Reports", family='Serif', size=15, ha='center', weight='bo
ld')
plt.figtext(0.5,0.93,"Classification report based on the Logisitic regression model", f
amily='Serif', size=12, ha='center')
ax1=plt.subplot(gs[0,0])
ax1.set(title='ROC Curve')
visual = ROCAUC(model, classes=[0,1])
visual.fit(X_train,y_train)
ax1=visual.score(X_test,y_test)

ax2=plt.subplot(gs[0,1])
ax2.set(title='Classification report')
ax2=ClassificationReport(model,classes=[0,1], support=True).fit(X_train,y_train).score(
X_test,y_test)
```

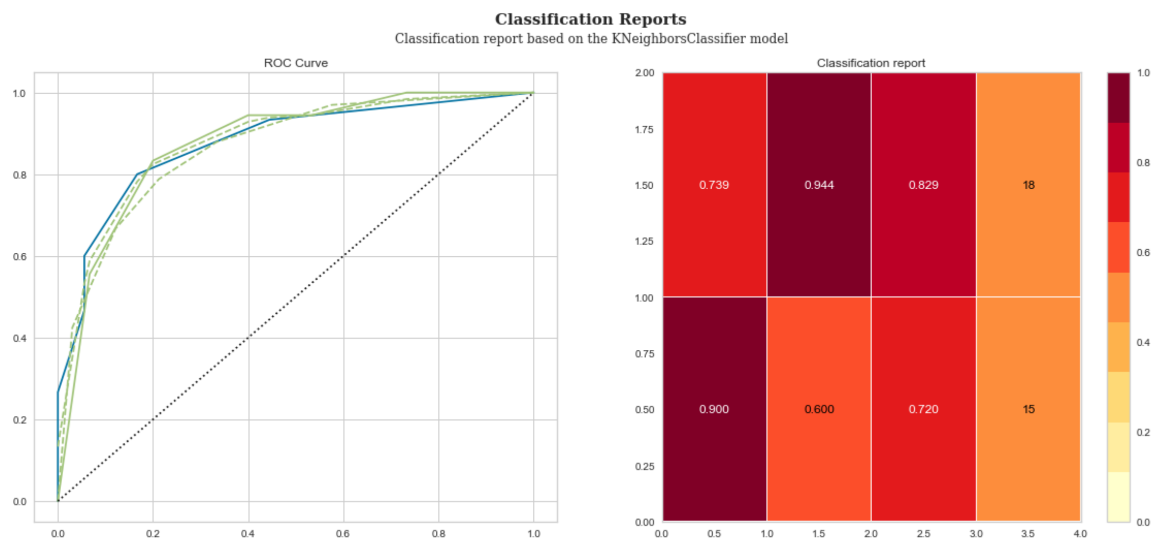
```
0.7575757575757576
              precision    recall  f1-score   support

           0       0.89      0.53      0.67        15
           1       0.71      0.94      0.81        18

    accuracy                           0.76        33
   macro avg       0.80      0.74      0.74        33
weighted avg       0.79      0.76      0.74        33
```



In [422]:

```
from sklearn.neighbors import KNeighborsClassifier
from yellowbrick.classifier import ROCAUC, ClassificationReport, ClassificationScoreVis
ualizer
model = KNeighborsClassifier()
model.fit(X_train, y_train)
model.score(X_test ,y_test)
pred=model.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
print(accuracy_score(y_test,pred))
print(classification_report(y_test,pred))

## Yellow brick reports
fig = plt.figure(figsize=(20,8))
gs=GridSpec(nrows=1, ncols=2)
plt.suptitle("Classification Reports", family='Serif', size=15, ha='center', weight='bo
ld')
plt.figtext(0.5,0.93,"Classification report based on the KNeighborsClassifier model", f
amily='Serif', size=12, ha='center')
ax1=plt.subplot(gs[0,0])
ax1.set(title='ROC Curve')
visual = ROCAUC(model, classes=[0,1])
visual.fit(X_train,y_train)
ax1=visual.score(X_test,y_test)

ax2=plt.subplot(gs[0,1])
ax2.set(title='Classification report')
ax2=ClassificationReport(model,classes=[0,1], support=True).fit(X_train,y_train).score(
X_test,y_test)
```

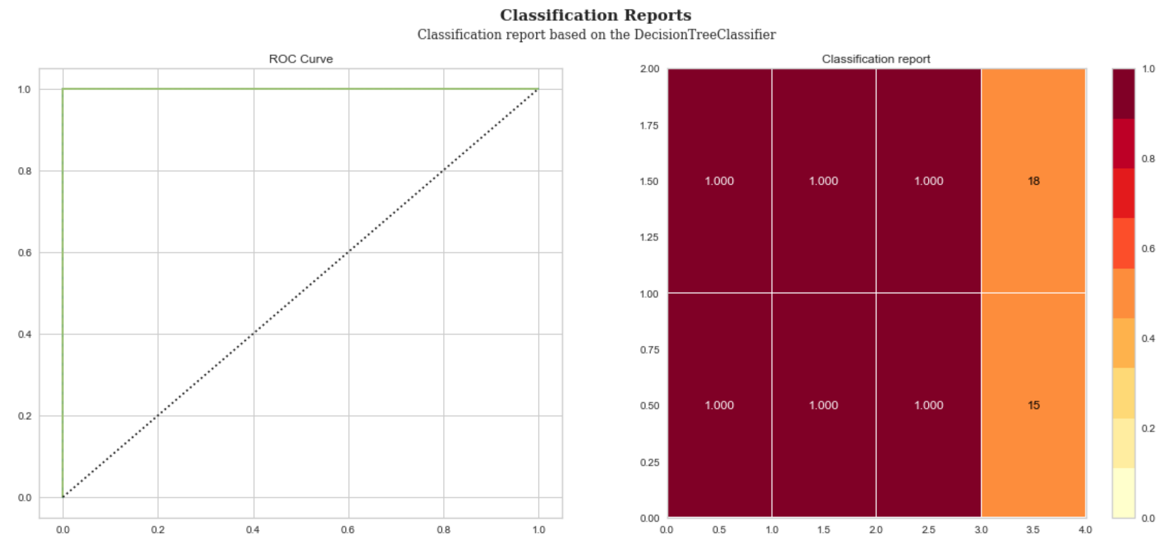
```
0.7878787878787878
              precision    recall  f1-score   support

           0       0.90      0.60      0.72        15
           1       0.74      0.94      0.83        18

    accuracy                           0.79        33
   macro avg       0.82      0.77      0.77        33
weighted avg       0.81      0.79      0.78        33
```



In [423]:

```
from sklearn.tree import DecisionTreeClassifier
from yellowbrick.classifier import ROCAUC, ClassificationReport, ClassificationScoreVisualizer
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
model.score(X_test ,y_test)
pred=model.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
print(accuracy_score(y_test,pred))
print(classification_report(y_test,pred))

## Yellow brick reports
fig = plt.figure(figsize=(20,8))
gs=GridSpec(nrows=1, ncols=2)
plt.suptitle("Classification Reports", family='Serif', size=15, ha='center', weight='bold')
plt.figtext(0.5,0.93,"Classification report based on the DecisionTreeClassifier", family='Serif', size=12, ha='center')
ax1=plt.subplot(gs[0,0])
ax1.set(title='ROC Curve')
visual = ROCAUC(model, classes=[0,1])
visual.fit(X_train,y_train)
ax1=visual.score(X_test,y_test)

ax2=plt.subplot(gs[0,1])
ax2.set(title='Classification report')
ax2=ClassificationReport(model,classes=[0,1], support=True).fit(X_train,y_train).score(X_test,y_test)
```

```
1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        18

    accuracy                           1.00        33
   macro avg       1.00      1.00      1.00        33
weighted avg       1.00      1.00      1.00        33
```



In [424]:

```
#Use the ensemble model to predict the final degree of stress.
```

In [425]:

```
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from scipy.stats import zscore
from time import time

bag_clf = BaggingClassifier(base_estimator=SVC(),n_estimators=10, random_state=0)
extra_tree_forest = ExtraTreesClassifier(n_estimators = 5, criterion ='entropy', max_fe
atures = 2)


ran_clf = RandomForestClassifier(max_depth=2, random_state=0)
ada_clf = AdaBoostClassifier(n_estimators=100, random_state=0)
gra_clf = GradientBoostingClassifier(learning_rate=0.05,max_depth=3,max_features=0.5,ra
ndom_state=0)
xgb_clf = XGBClassifier()
```

In [426]:

```
start_time=time()
model_list=[bag_clf,extra_tree_forest,ran_clf,ada_clf,xgb_clf,gra_clf ]
Score=[]
for i in model_list:
    i.fit(X_train,y_train)
    y_pred=i.predict(X_test)
    score=accuracy_score(y_test,y_pred)
    Score.append(score)
print(pd.DataFrame(zip(model_list,Score),columns=['Model Used','accuracy_score']))
end_time=time()
print(round(end_time-start_time,2),'sec')
```

```
                                          Model Used  accuracy_score
0  (SVC(C=1.0, break_ties=False, cache_size=200, ...            0.64
1  (ExtraTreeClassifier(ccp_alpha=0.0, class_weig...            0.88
2  (DecisionTreeClassifier(ccp_alpha=0.0, class_w...            1.00
3  (DecisionTreeClassifier(ccp_alpha=0.0, class_w...            1.00
4  XGBClassifier(base_score=0.5, booster='gbtree'...            1.00
5  ([DecisionTreeRegressor(ccp_alpha=0.0, criteri...            1.00
1.63 sec
```

In [427]:

```
#i wuill  select random classifer  and perform hypertuning
```

In [428]:

```
#like gridserach CV , hyper tuning randomserach CV
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 300, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 1000,10)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10,14]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4,6,8]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features':max_features,
               'max_depth':max_depth,
               'min_samples_split':min_samples_split,
               'min_samples_leaf':min_samples_leaf,
               'criterion':['gini','entropy']}
```

In [429]:

```
rf=RandomForestClassifier()
rf_randomcv=RandomizedSearchCV(estimator=rf,param_distributions=random_grid,n_iter=100,
cv=3,verbose=2,
                               random_state=100,n_jobs=-1)
### fit the randomized model
rf_randomcv.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   49.4s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  3.6min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  6.0min finished
```

Out[429]:

```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=
0.0,
                                                    min_impurity_split=Non
e,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_le
af=0.0,
                                                    n_estimators=100,
                                                    n_jobs...
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [10, 120, 230, 340, 4
50,
                                                      560, 670, 780, 890,
                                                      1000],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'min_samples_leaf': [1, 2, 4, 6,
8],
                                        'min_samples_split': [2, 5, 10, 1
4],
                                        'n_estimators': [300, 488, 677, 86
6,
                                                         1055, 1244, 1433,
1622,
                                                         1811, 2000]},
                   pre_dispatch='2*n_jobs', random_state=100, refit=True,
                   return_train_score=False, scoring=None, verbose=2)
```

In [430]:

```
best_random_grid=rf_randomcv.best_estimator_
```

In [431]:

```
from sklearn.metrics import accuracy_score
y_pred=best_random_grid.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print("Accuracy Score:{}" .format(accuracy_score(y_test,y_pred)))
print("Classification report: {}".format(classification_report(y_test,y_pred)))
```

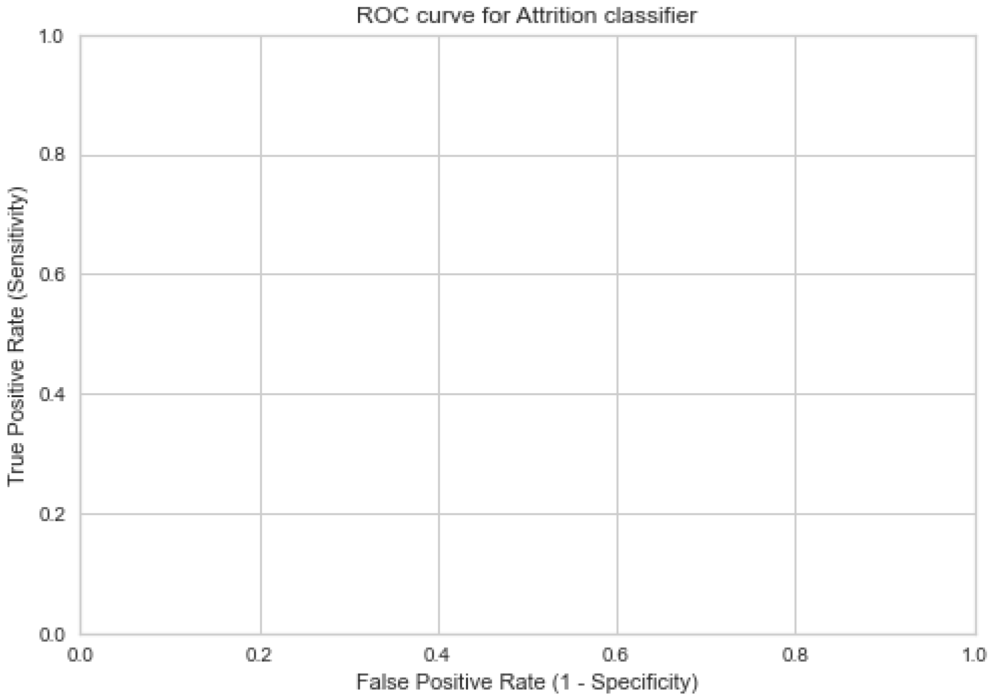
```
[[15  0]
 [ 0 18]]
Accuracy Score:1.0
Classification report:               precision    recall  f1-score   suppo
rt

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        18

    accuracy                           1.00        33
   macro avg       1.00      1.00      1.00        33
weighted avg       1.00      1.00      1.00        33
```

In [432]:

```
#after hyper tuning it seems accuracy_score got improved
```

In [433]:

```
from sklearn import metrics
#IMPORTANT: first argument is true values, second argument is predicted probabilities

# we pass y_test and y_pred_prob
# we do not use y_pred_class, because it will give incorrect results without generating
an error
# roc_curve returns 3 objects fpr, tpr, thresholds
# fpr: false positive rate
# tpr: true positive rate
fpr, tpr, thresholds = metrics.roc_curve(y_test,y_pred)

plt.plot(fpr, tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.rcParams['font.size'] = 12
plt.title('ROC curve for Attrition classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
```



In [434]:

```
#by lazyclassifer we can see all the models at time
from lazypredict.Supervised import LazyClassifier
lazy_clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
models,predictions = lazy_clf.fit(X_train, X_test, y_train, y_test)
models
```

```
100%|██████████| 30/30 [00:17<00:00,  1.74it/s]
```

Out[434]:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
AdaBoostClassifier	1.00	1.00	1.00	1.00	0.02
DecisionTreeClassifier	1.00	1.00	1.00	1.00	0.00
XGBClassifier	1.00	1.00	1.00	1.00	8.16
RandomForestClassifier	1.00	1.00	1.00	1.00	0.42
BaggingClassifier	1.00	1.00	1.00	1.00	0.02
LGBMClassifier	1.00	1.00	1.00	1.00	7.61
ExtraTreesClassifier	0.91	0.91	0.91	0.91	0.12
ExtraTreeClassifier	0.82	0.82	0.82	0.82	0.00
LabelPropagation	0.82	0.81	0.81	0.82	0.66
LabelSpreading	0.82	0.81	0.81	0.82	0.02
PassiveAggressiveClassifier	0.82	0.81	0.81	0.81	0.00
Perceptron	0.79	0.78	0.78	0.78	0.02
KNeighborsClassifier	0.79	0.77	0.77	0.78	0.00
LinearSVC	0.76	0.74	0.74	0.74	0.02
LogisticRegression	0.76	0.74	0.74	0.74	0.02
GaussianNB	0.70	0.68	0.68	0.69	0.02
QuadraticDiscriminantAnalysis	0.70	0.68	0.68	0.69	0.01
NuSVC	0.70	0.68	0.68	0.68	0.02
NearestCentroid	0.70	0.68	0.68	0.68	0.00
LinearDiscriminantAnalysis	0.64	0.61	0.61	0.58	0.00
CalibratedClassifierCV	0.64	0.61	0.61	0.58	0.02
RidgeClassifier	0.64	0.61	0.61	0.58	0.02
SGDClassifier	0.64	0.61	0.61	0.58	0.02
SVC	0.61	0.58	0.58	0.56	0.02
RidgeClassifierCV	0.61	0.57	0.57	0.54	0.00
DummyClassifier	0.58	0.55	0.55	0.54	0.02
BernoulliNB	0.55	0.52	0.52	0.51	0.02
CheckingClassifier	0.45	0.50	0.50	0.28	0.02

BY Harsha

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: