# Fiscal Stress Score for Local Governments

## Basic computational skills, information search and reporting


## Abstract

Local governments and school districts throughout the State and country faced new challenges that threatened their fiscal health after the great recession. Fiscal stress monitoring system has hence been used since then for early actions to prevent these entities from ending up in severe fiscal stress. This project imitates the computation of fiscal stress score developed by Office of New York State Comptroller (OSC) to provide scores earlier than them for each town in New York state for years 2000-2017. The score is calculated based on financial indicators. The scores are used for various trend analysis. It is found that most of the towns came in the category of no fiscal stress while very few lied in the significant fiscal stress. Moreover, it is found that population has some effect on the fiscal stress score of towns when grouped as county.


## Introduction

Since the onset of the economic recession in December 2007, local governments and school districts throughout the State and country faced new challenges that threatened their fiscal health. To help the local governments and school districts deal with these fiscal challenges, it is important to identify clearly those local governments and school districts that are moving towards, or are already in, fiscal stress. Such monitoring of the fiscal health of them should allow for early actions to prevent these entities from ending up in severe fiscal stress. There are various researches and studies on measuring and assessing fiscal scores. They also talk about its prediction, responses, causes, consequences and so on. These indicators also give opportunities to policymakers to adjust policies before extreme fiscal stress event.

A public fiscal stress monitoring has been developed by OSC that helps identifying local governments and school districts that are in fiscal stress, as well as those showing susceptibility to fiscal stress. They just provide the stress score to reflect the ability of a local government or school district to generate enough revenues to meet expenditures but does not assess how a local government or school district is being managed. The fiscal score is based on various financial and environmental indicators which are measured separately. In this project, I focus my attention on calculating the stress score for all towns (that comes under Local Government) for the years 2000-2015 extended to year 2017. Local governments file audited financial reports to the New York State Office of State Comptroller. These reports are made available to the public in various formats. The OSC releases the stress score based on these reports, for towns in the 3$^{rd}$ quarter of the year after the ending of last year i.e. it takes 9 months to release the data. I focus on financial indicators only to get the stress score. They basically test critical areas such as fund balance levels, operating deficits, cash-on-hand, fixed costs and short-term cashflow borrowing. The examination focuses on the major operating funds of each entity, depending on class (county, city, town, village, or school). The goal is to write a code that imitates the procedure of calculating the stress score and provide insights about these scores using EDA that can help us understand why towns face stress. This will help local government to calculate the scores early resulting in identifying and improving the fiscal stress.

## Data

The accounting data for all towns in NY to calculate the stress score is taken from the OSC website. The data ranges from year 1998-2017. Reason for extending the data years range below year 2000 is that some financial indicators require data of previous 2 years to be used too. For extension beyond 2015, there was change in system for calculating the stress score in 2017. Since I am imitating the procedure to calculate score, it is necessary to check the results with actual results. The website provides the actual stress score using the new updated system from years 2016-2019. And therefore, to check if my code works correctly, I have extended the years. Detailed account level data for Revenue, expenditure, balance sheet and debt data are taken which is divided into two parts.

First file contains data for Revenue, expenditure and balance sheet which has details such as 'Calendar_Year', 'Municipal_Code', 'Entity_Name', Class_Description', 'County', 'Fiscal_Year_End', 'Account_Code', 'Account_Code_Narrative', 'Financial_Statement', 'Financial_Statement_Segment', 'Level_1_Category', 'Level_2_Category', 'Object_Of_Expenditure', 'Amount', 'Snapshot_Date'. It gives detailed accounting information for example for the expense 'Clerk fees', it tells about town, years, broad category under financial statements and it's sub category, account code, amount and so on. A snapshot date is the date scores are calculated and finalized.

Similarly, for debt data file, the details are given as 'Calendar_Year', 'Municipal_Code', 'Entity_Name', 'Sub_Govt_Type', 'County', Fiscal_Year_End', 'Debt_Type_Desc', 'Beginning_Year_Balance_Sum', 'Prior_Year_Adjustment_Sum', 'Debt_Issued_In_Current_Year', 'Debt_Paid_Current_Year', 'Accreted_Interest_Sum', 'Ending_Year_Balance_Sum', 'Snapshot_Date'. This data is helpful for some of the indicators to calculate stress score.

The debt data files are first filtered for Towns only. Unnecessary columns are removed. Both the files are then merged for further analysis. The table below (Fig. 1) shows the descriptive statistics of the merged sample dataset. The 'Unique' column for example gives lot of useful information such as the sample data set if for 919 towns. There are 4 kinds of financial statements, a total of 57 counties and 1583 types of accounting codes to be used for various calculations. This helps when dataset is large.

| | coun | unique | top | freq | mean | std | min | 25% | 50% | 75% |
|---|---|---|---|---|---|---|---|---|---|---|
| ACCOUNT_CODE | 3819900 | 6015 | A8021 | 18482 | NaN | NaN | NaN | NaN | NaN | NaN |
| ACCOUNT_CODE_NARRATIVE | 3819900 | 1583 | Fund Balance - End of Year | 130479 | NaN | NaN | NaN | NaN | NaN | NaN |
| ACCRETED_INTEREST_SUM | 1 | NaN | NaN | NaN | 34.7669 | 139.068 | 0 | 0 | 0 | 0 |
| AMOUNT | 3.61439e+0 | NaN | NaN | NaN | 368605 | 1.61224e+07 | -5.8592e+08 | 1633 | 12315 | 74954 |
| BEGINNING_YEAR_BALANCE_SUM | 2273 | NaN | NaN | NaN | 4.0916e+06 | 2.27658e+07 | 0 | 86437.5 | 360179 | 1.691e+06 |
| CALENDAR_YEAR | 3.84362e+0 | NaN | NaN | NaN | 2007.62 | 5.74995 | 1998 | 2003 | 2008 | 2013 |
| COUNTY | 384361 | 57 | Erie | 160766 | NaN | NaN | NaN | NaN | NaN | NaN |
| DEBT_ISSUED_IN_CURRENT_YEAR | 920 | NaN | NaN | NaN | 2.14381e+06 | 8.85004e+06 | 0 | 85668.2 | 260000 | 1.17291e+06 |
| DEBT_PAID_CURRENT_YEAR | 2218 | NaN | NaN | NaN | 765600 | 4.71412e+06 | 0 | 23000 | 58786.5 | 220000 |
| DEBT_TYPE_DESC | 2371 | 10 | Bond | 11533 | NaN | NaN | NaN | NaN | NaN | NaN |
| ENDING_YEAR_BALANCE_SUM | 2319 | NaN | NaN | NaN | 4.13266e+06 | 2.32917e+07 | 0 | 82687.8 | 359031 | 1.71e+06 |
| ENTITY_NAME | 384361 | 919 | Town of North Hempstead | 15511 | NaN | NaN | NaN | NaN | NaN | NaN |
| FINANCIAL_STATEMENT | 3819900 | 4 | STATEMENT OF EXPENDITURES AND OTHER USES | 1707722 | NaN | NaN | NaN | NaN | NaN | NaN |
| FINANCIAL_STATEMENT_SEGMENT | 381950 | 30 | EXPENDITURES | 1674086 | NaN | NaN | NaN | NaN | NaN | NaN |
| OBJECT_OF_EXPENDITURE | 170772 | 7 | Contractual | 728976 | NaN | NaN | NaN | NaN | NaN | NaN |
| PRIOR_YEAR_ADJUSTMENT_SUM | 2052 | NaN | NaN | NaN | 4157.02 | 434646 | -4.0835e+07 | 0 | 0 | 0 |

Fig. 1 Descriptive Statistics of Sample Dataset

**Methodology**

To compute the fiscal stress score, OCI uses 9 financial indicators. They are:

1. Assigned and Unassigned Fund Balance (25%) , 2. Total Fund Balance (25%) , 3. Operating deficit (10%) , 4. Cash Ratio (10%) , 5. Cash % of Monthly Expenditure (10%) , 6. Short term debt issuance (5%) , 7. Short term debt issuance trend (5%) , 8. Personal Services and Employee Benefits % Revenues (5%) & 9. Debt service % Revenues (5%). Brief for the indicators are as follows:

Year-End Fund Balance (Indicators 1 & 2) It is an important measure of financial condition and affects the ability to deal with revenue shortfalls and expenditure overruns.

Operating Deficits (Indicator 3) Annual operating results are a measure of recent financial operations and the direction that finances are headed. Multiple years of operating deficits can face financial hardship. This suggests that a budget is not structurally balanced − recurring revenues are insufficient to support recurring expenditures.

Cash Position (Indicators 4 & 5) Whether an entity has enough cash on hand to pay its bills. These have ability to liquidate current liabilities as well as fund the ensuing fiscal year's operations.

Use of Short-Term Cash-Flow Debt (Indicators 6 & 7) They reflect borrowing for cash flow purposes, to pay for normal operating costs. They evaluate the amount of short-term cash-flow debt that was issued in the last fiscal year as well as the trend, considering the prior three-year period.

Fixed Costs (Indicators 8 & 9) They evaluate the amount of revenues used for personal services and employee benefits, and debt service. The level of fixed costs affects flexibility in responding to economic changes.

To calculate stress score, I make various functions and class. Each of the indicator has its own calculation formula (refer to code appendix and reference (2) for details), so I have made function to calculate the value of each of the 9 indicators. The calculated value can be of 3 types: general funds value, combined fund value and all fund value. Moreover, some indicators require true/false as an input and therefore I gave the input as a bool. Example for 'Town of Adams' for year '2017' can be seen below (Fig 2). All the calculated values for each indicator are ready to be scored.

```
year=2017
        town=Town of Adams
                Indicator_1: 60.68622088884778 12.11432056282397 None
                Indicator_2: 78.94675338515799 21.086662946471392 None
                Indicator_3: 0.07737283951495103 None [False, False, True]
                Indicator_4: inf None None
                Indicator_5: 984.2967460905403 None None
                Indicator_6: 0.0 None None
                Indicator_7: 0.0 False [False, False, False]
                Indicator_8: 33.41972886337395 None [33.41972886337395, 32.87410332361644, 34.04532876936165]
                Indicator_9: 5.939531203320173 None [5.939531203320173, 5.921817227553644, 5.945450207626446]
```

Fig. 2 Calculated value of each indicators for 1 town for 1 year

The above values are sent to next group of functions that contain conditions to give final scores. As mentioned above each of the indicators has its own contribution to the final score out of 100. Each has its own unique condition for scoring. Some conditions take inputs for the same year and some take values for same and previous 2 years. Giving all the conditions with various edge cases like denominator NaN or infinity, data not available, etc. The table (Appendix 1) provided by OSC shows the conditions

to calculate the score for each indicator. Scores for each of the indicators are added to give the final stress score out of 100 (i.e. 100%).

After getting the final stress score, next step is to classify them. The classification of score is based on ranges of score in 4 parts. 65% - 100% is referred as Significant Fiscal Stress, 55% - 64.9% as Moderate Fiscal Stress, 45% - 54.9% as Susceptible of Fiscal stress and 0% - 44.9% as No designation. The categories are given different weights to reflect their relative importance in measuring financial stress. Importantly, No designation classification does not imply that the entity is free of all fiscal stress conditions. This can also be a result of not filing. Score will not be generated properly if data is missing. For entities that either did not submit their annual financial reports (each of the past three years) or did so very late (after the snapshot date)—well after the opportunity to be properly evaluated and scored has passed. This is perhaps the most troubling classification as not filing deprives citizens in these communities the full transparency they deserve from their local leaders. Entities in this category are not retroactively scored even after they submit their information. All the information is saved in the final data frame that contains stress scores for each town for years 1998-2017 which is then filtered for years 2000-2017. Sample for 'Town of Adams' for 2017 (Fig. 3) shows the same. It is also verified with the actual scores data (Fig. 4) from the OSC website after the change in system.

| Year | Town_Name | Indicator_1 (25%) | Indicator_2 (25%) | Indicator_3 (10%) | Indicator_4 (10%) | Indicator_5 (10%) | Indicator_6 (5%) | Indicator_7 (5%) | Indicator_8 (5%) | Indicator_9 (5%) |
|---|---|---|---|---|---|---|---|---|---|---|
| 2017 | Town of Adams | 0.0 | 0.0 | 3.333333 | 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

| Max_score | Stress_Score | Score_Classifications |
|---|---|---|
| 100 | 3.333333 | No Designation |

Fig. 3 Stress Score through code for 'Town of Adams' for 2017

**Name:** Town of Adams
**MuniCode:** 220300100000
**County:** Jefferson

| Financial Indicators | Fiscal Stress Financial Indicators | Public Scores Post System Enhancements 2017 | 2018 |
|---|---|---|---|
| 1 | Assigned and Unassigned Fund Balance as a Percentage (%) of Gross | 0 | 0 |
| 2 | Total Fund Balance as a Percentage (%) of Gross Expenditures | 0 | 0 |
| 3 | Operating Deficits | 3.33 | 0 |
| 4 | Cash Ratio - Cash and Investments as a Percentage (%) of Current Liabilities | 0 | 0 |
| 5 | Cash as a Percentage (%) of Monthly Gross Expenditures | 0 | 0 |
| 6 | Short-Term Cash-Flow Debt Issuance as a Percentage (%) of Total Revenues | 0 | 0 |
| 7 | Short-Term Cash-Flow Debt Issuance Trend | 0 | 0 |
| 8 | Personal Service and Employee Benefits as a Percentage (%) of Total | 0 | 0 |
| 9 | Debt Service as a Percentage (%) of Total Revenues (3 year avg) | 0 | 0 |
| | **Total Points*** | 3.3 | 0.0 |
| | **Score Classification** | No Designation | No Designation |

Fig. 4 Actual Stress Score for 'Town of Adams' for 2017 & 2018

## Exploratory Data Analysis & Results

After gathering the stress scores for all the towns for the years 2000-2017, analysis is done using graphs and charts to see the trends. From macro level to micro, EDA helps in in-depth research. Initially the trend is analyzed based on all the years (Fig. 5) which clearly shows strange behavior of the trend. It

suddenly has a sharp break after 2010. When I go back to check the source data, turns out we don't have account 915 and 917 for the years before 2010, both of which are needed for Indicator 1. As we have a lot of missing values, it doesn't make sense trying to impute them. The reason for the same is that New York state actually changed its fund balance reporting method in 2010. That is the reason for the missing 915 and 917 before 2010. This can be further used to improve the method of calculation. Moreover, the results cannot be checked with actual data as the OSC website does not have data for actual score for years before 2012. Therefore, for now, I have looked further deep into years from 2011 for my analysis (Fig. 6).

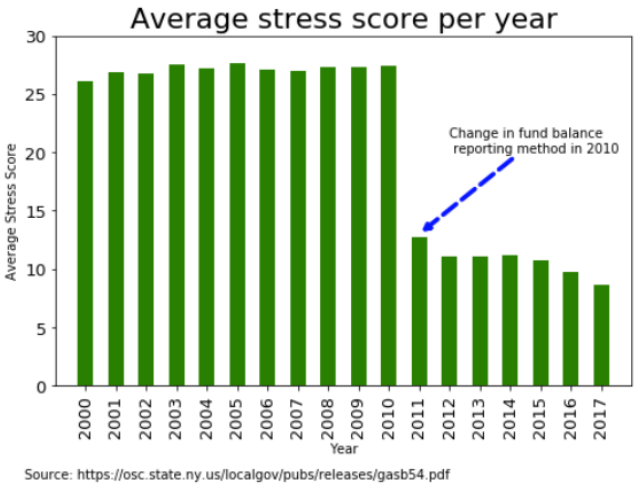


Fig. 5 Avg. stress score per year from 2000

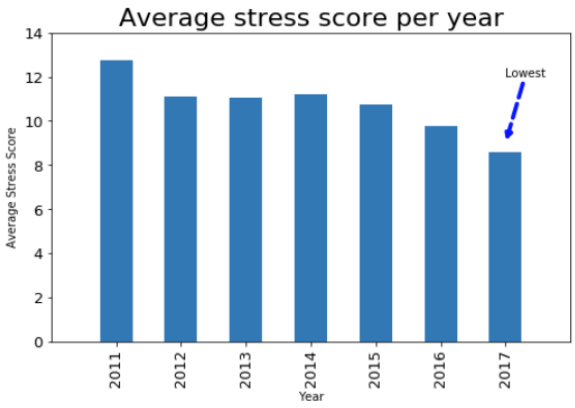


Fig. 6 Avg. stress score per year from 2011

From the chart (Fig. 6) above, it can be analyzed that the average stress score has been reducing. The reduction can be based on both a positive and a negative change. Towns might have cut spending on activities related to recreation and have put off capital improvements as well as maintenance for roads, building and the like. While such delays may be an effective short-run strategy, it may result in more extensive costs in the longer-run. Ability to raise sufficient revenue to satisfy expenditure needs is the most important concern in local government financing such as towns in this case. Local governments under fiscal stress often need to dig deep into their tax base to generate enough financing because of inflexible revenue resources.

Further I analyze the classifications of stress score (Fig.7) and it can be seen that most of the towns i.e. 96.8% of the scores came out to be No Designation. But again, No designation classification does not imply that the entity is free of all fiscal stress conditions as discussed above. Moreover, very few entries in the range of 17 years were of significant fiscal stress.

| | Stress_Score_Classifications | Count |
|---|---|---|
| 0 | No Designation | 15907 |
| 1 | Susceptible of Fiscal Stress | 353 |
| 2 | Moderate Fiscal Stress | 108 |
| 3 | Significant Fiscal Stress | 63 |

Fig. 7 Classifications of stress score count

Further I analyze the trend for a random town over the years. I have taken the 'Town of Rochester (Fig. 8) for the span of 7 years and it can be seen that initially the score was really low (No Designation) but in the middle it rose to more than 50 i.e. there was moderate fiscal stress and further there is decline is the score again leading to No Designation. There can be different explanations for this possibly mis management of funds or lack of funds during the high score years.

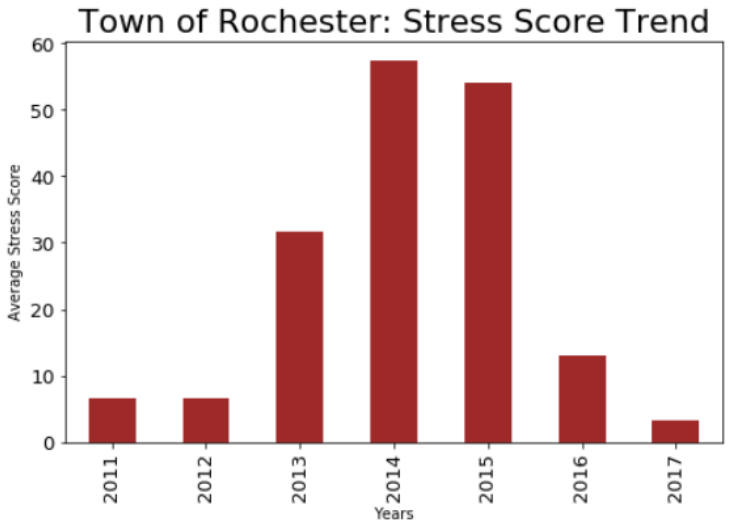


Fig. 8 Trend of stress score for Town of Rochester

To further analyze, I look at the demographics as suggested in research (M.Skidmorea, E.Scorsone 2011) that said, for their research on fiscal stress score, the region with the largest population base, has experienced the greatest level of stress. To know the relation, I find the population data of NY state, county wise for 2017 and group the stress score too county wise for 2017. Further I find the correlation between the score and the population to see if there is high score with high population. The result showed that the correlation between Stress Score by County and Population of County is: =0.3731792663581337. Moreover, it can be seen through the scatter plot (Fig. 9) that most of the points lie at low population and score averaging between 5-10%. A few points can be seen at high population and high score but they are present at low populations too. So, it can be concluded that population has not very strong relationship but can be a factor contributing to fiscal stress score.

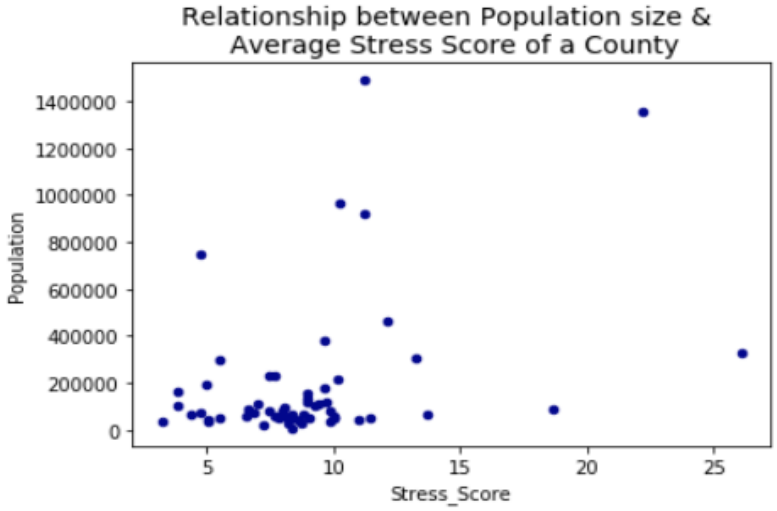


Fig. 9 Relationship between Population size & Average Stress Score of a County

## Conclusion

The scores can be used for various trend analysis. It is found that most of the towns came in the category of no fiscal stress while very few lied in the significant fiscal stress. A drawback of these approaches in the context of the indicator's calculation is that it combines both external forces with internal local government management decisions. For example, a local decision maker, in the interest of maintaining fiscal responsibility, may decide to significantly cut spending in order to avoid fiscal imbalance and short-term operating loans, etc. By measures such as these, this community may appear to be fiscally sound, even though it is experiencing fiscal distress. These measures, though useful for many purposes, will not provide an effective measure of distress if one seeks to understand how local decision makers have responded in terms of allocating increasingly limited resources to the various functional expenditure categories. It is also seen that there has been a decline in overall average fiscal score from 2011 which can be good sign for local governments as well as the society. Early signal warning using fiscal score can eventually help better management and distribution of funds. Further in depth analysis can be made using various factors such population of town/county/states of US, analyzing different changed in system to make scores more accurate for comparison, etc.

## Resources


1. Input Data Download: https://www.osc.state.ny.us/localgov/datanstat/findata/index_choice.htm#

2. Methodology: https://www.osc.state.ny.us/localgov/fiscalmonitoring/pdf/fiscalstressmonitoring.pdf

3. Others:

- Measuring Fiscal Vulnerability and Fiscal Stress: A Proposed Set of Indicators; Mr. James McHugh, Iva Petrova, Mr. Emanuele Baldacc; 2011
- Causes and consequences of fiscal stress in Michigan cities; Mark Skidmorea, Eric Scorsone;2011
- Fiscal stress in local government: A case study of the tri-cities in the Commonwealth of Virginia; Yaw Agyeman Badu & Sheng Yung Li; 1994
- https://www.newyork-demographics.com/counties_by_population
- https://www.osc.state.ny.us/localgov/fiscalmonitoring/pdf/system-enhancements-local-governments.pdf
- http://wwe1.osc.state.ny.us/localgov/fiscalmonitoring/fsms.cfm
- https://www.osc.state.ny.us/localgov/fiscalmonitoring/pdf/system-manual.pdf
- https://osc.state.ny.us/localgov/pubs/releases/gasb54.pdf

## Appendix 1- Indicator Scoring conditions

| Local Government Financial Indicators Scoring | | | |
|---|---|---|---|
| **Financial Indicator** | **Scoring - Points** | **Max. Points** | **Scoring - Weighted Average** |
| 1. Assigned and Unassigned Fund Balance | **General Fund Result**<br>3 Points = Less Than or Equal to 3.33% Last Fiscal Year<br>2 Points = Greater Than 3.33% But Less Than or Equal to 6.67% Last Fiscal Year<br>1 Point = Greater Than 6.67% But Less Than or Equal to 10% Last Fiscal Year<br>0 Points = Greater Than 10% Last Fiscal Year<br>**Combined Funds Result Minus General Fund Result**<br>1 Point = Negative % When the General Fund % is Subtracted from the Combined Funds % for the Last Fiscal Year | 4 | 50% |
| 2. Total Fund Balance | **General Fund Result**<br>3 Points = Less Than or Equal to 10% Last Fiscal Year<br>2 Points = Greater Than 10% But Less Than or Equal to 15% Last Fiscal Year<br>1 Point = Greater Than 15% But Less Than or Equal to 20% Last Fiscal Year<br>0 Points = Greater Than 20% Last Fiscal Year<br>**Combined Funds Result Minus General Fund Result**<br>1 Point = Negative % When the General Fund % is Subtracted from the Combined Funds % for the Last Fiscal Year | 4 | |
| 3. Operating Deficit | **Combined Funds Result**<br>3 Points = Deficits in Three of Last Three Fiscal Years or a Deficit in the Last Fiscal Year Less Than or Equal to -10%<br>2 Points = Deficits in Two of Last Three Fiscal Years<br>1 Point = Deficit in One of Last Three Fiscal Years<br>0 Points = No Deficits in Last Three Fiscal Years | 3 | 10% |
| 4. Cash Ratio | **Combined Funds Result**<br>3 Points = Less Than or Equal to 50% Last Fiscal Year<br>2 Points = Greater Than 50% But Less Than or Equal to 75% Last Fiscal Year<br>1 Point = Greater Than 75% But Less Than or Equal to 100% Last Fiscal Year<br>0 Points = Greater Than 100% Last Fiscal Year | 3 | 20% |
| 5. Cash % of Monthly Expenditures | **Combined Funds Result (Villages and Towns)**<br>3 Points = Less Than or Equal to 33.3% Last Fiscal Year<br>2 Points = Greater Than 33.3% But Less Than or Equal to 66.7% Last Fiscal Year<br>1 Point = Greater Than 66.7% But Less Than or Equal to 100% Last Fiscal Year<br>0 Points = Greater Than 100% Last Fiscal Year<br>**Combined Funds Result (Cities and Counties)**<br>3 Points = Less Than or Equal to 50% Last Fiscal Year<br>2 Points = Greater Than 50% But Less Than or Equal to 100% Last Fiscal Year<br>1 Point = Greater Than 100% But Less Than or Equal to 150% Last Fiscal Year<br>0 Points = Greater Than 150% Last Fiscal Year | 3 | |
| 6. Short-Term Debt Issuance | **All Funds Result**<br>3 Points = Greater Than 15% Last Fiscal Year<br>2 Points = Greater Than 5% But Less Than or Equal to 15% Last Fiscal Year<br>1 Point = Greater Than 0% But Less Than or Equal to 5% Last Fiscal Year<br>0 Points = 0% Last Fiscal Year | 3 | 10% |
| 7. Short-Term Debt Issuance Trend | **All Funds Result**<br>3 Points = Issuance in Each of Last Three Fiscal Years or Issued a Budget Note in Last Fiscal Year<br>2 Points = Issuance in Each of Last Two Fiscal Years<br>1 Point = Issuance in Last Fiscal Year<br>0 Points = No Issuance | 3 | |
| 8. Personal Services and Employee Benefits % Revenues | **All Funds Result**<br>3 Points = Last Three Fiscal Years Average Greater Than or Equal to 75%<br>2 Points = Last Three Fiscal Years Average Greater Than or Equal to 70% But Less Than 75%<br>1 Point = Last Three Fiscal Years Average Greater Than or Equal to 65% But Less Than 70%<br>0 Points = Last Three Fiscal Years Average Less Than 65% | 3 | 10% |
| 9. Debt Service % Revenues | **All Funds Result**<br>3 Points = Last Three Fiscal Years Average Greater Than or Equal to 20%<br>2 Points = Last Three Fiscal Years Average Greater Than or Equal to 15% But Less Than 20%<br>1 Point = Last Three Fiscal Years Average Greater Than or Equal to 10% But Less Than 15%<br>0 Points = Last Three Fiscal Years Average Less Than 10% | 3 | |
| | **Total** | **29** | **100%** |

**Appendix 2- Coding**

```
# project 2

#importing relevant libraries

import pandas as pd

import numpy as np

import warnings

import matplotlib.pyplot as plt

from typing import List, Tuple, Callable

import math

import os

warnings.filterwarnings('ignore')

# importing accounting data for towns from 1998-2017

colNames = ['CALENDAR_YEAR', 'MUNICIPAL_CODE', 'ENTITY_NAME', 'CLASS_DESCRIPTION', 'COUNTY',

        'FISCAL_YEAR_END', 'ACCOUNT_CODE', 'ACCOUNT_CODE_NARRATIVE', 'FINANCIAL_STATEMENT',

        'FINANCIAL_STATEMENT_SEGMENT', 'LEVEL_1_CATEGORY', 'LEVEL_2_CATEGORY',

        'OBJECT_OF_EXPENDITURE', 'AMOUNT', 'SNAPSHOT_DATE']

acct_data_df  = pd.DataFrame(columns = colNames)


for subdir, dirs, files in os.walk("../data/accounting_data/"):

    for file in files:

        df = pd.read_csv(f'{subdir}{file}')

        acoounting_data_df = acoounting_data_df.merge(df, how='outer')

# importing debt data for towns from 1998-2017

colNames = ['CALENDAR_YEAR', 'MUNICIPAL_CODE', 'ENTITY_NAME', 'SUB_GOVT_TYPE', 'COUNTY',

        'FISCAL_YEAR_END', 'DEBT_TYPE_DESC', 'BEGINNING_YEAR_BALANCE_SUM',

        'PRIOR_YEAR_ADJUSTMENT_SUM', 'DEBT_ISSUED_IN_CURRENT_YEAR', 'DEBT_PAID_CURRENT_YEAR',
```

```
        'ACCRETED_INTEREST_SUM', 'ENDING_YEAR_BALANCE_SUM', 'SNAPSHOT_DATE']
debt_data_df = pd.DataFrame(columns = colNames)


for subdir, dirs, files in os.walk("../data/debt_data/"):
    for file in files:
        df1 = pd.read_csv(f'{subdir}{file}')
        debt_data_df = debt_data_df.merge(df1, how='outer')
# filter debt data for towns
debt_data_df = debt_data_df[debt_data_df.SUB_GOVT_TYPE == 'Town']
# dropping unnecessary columns
acct_data_df = acct_data_df.drop(['MUNICIPAL_CODE', 'CLASS_DESCRIPTION', 'FISCAL_YEAR_END', 'LEVEL_1_CATEGORY', 'LEVEL_2_CATEGORY', 'SNAPSHOT_DATE'], axis=1)
debt_data_df = debt_data_df.drop(['MUNICIPAL_CODE', 'SUB_GOVT_TYPE', 'FISCAL_YEAR_END', 'SNAPSHOT_DATE'], axis=1)
#merging the two df
accounting_data_df = pd.concat([acct_data_df, debt_data_df], ignore_index=True)
# descriptive statistics of sample data
accounting_data_df.describe(include='all').transpose()
# declaring constant
AMOUNT = 'AMOUNT'
# function to filter df based on some conditions
def get_filtered_df(raw_df: pd.DataFrame,
                    exact_conditions: List[Tuple[str, List[str]]],
                    prefix_conditions: List[Tuple[str, List[str]]] = []):
    filtered_df = raw_df.copy()
    for column_name, values in exact_conditions:
        filtered_df = filtered_df[filtered_df[column_name].isin(values)]
    for column_name, values in prefix_conditions:
        filtered_df = filtered_df[filtered_df.apply(lambda row: any([row[column_name].startswith(value) for value in values]), axis=1)]
```

```
    return filtered_df
    # function to accounting code to filter data for calculating indicators
def get_codes(prefixes: List[str], code_numbers: List[int]):
    codes = []
    for prefix in prefixes:
        for code_number in code_numbers:
            codes.append(f'{prefix}{code_number}')
    return codes
# function for scoring each of the 9 indicators based on its own conditions
def indicator_1_scoring(value_general: float, value_combined: float):
    general_score = None
    combined_score = None
    if math.isnan(value_general) or math.isnan(value_combined):
        return np.nan
    if math.isinf(value_general) or math.isinf(value_combined):
        return 0
    if value_general <= 3.33:
        general_score = 3
    elif value_general > 3.33 and value_general <= 6.67:
        general_score = 2
    elif value_general > 6.67 and value_general <= 10:
        general_score = 1
    elif value_general > 10:
        general_score = 0


    if value_combined < 0:
        combined_score = 1
    else:
        combined_score = 0
```

```
    return ((general_score + combined_score) * 25)/4

def indicator_2_scoring(value_general: float, value_combined: float):
    if math.isnan(value_general) or math.isnan(value_combined):
        return np.nan
    if math.isinf(value_general) or math.isinf(value_combined):
        return 0
    general_score = None
    combined_score = None
    if value_general <= 10:
        general_score = 3
    elif value_general > 10 and value_general <= 15:
        general_score = 2
    elif value_general > 15 and value_general <= 20:
        general_score = 1
    elif value_general > 20:
        general_score = 0

    if value_combined < 0:
        combined_score = 1
    else:
        combined_score = 0

    return ((general_score + combined_score) * 25)/4

def indicator_3_scoring(value_combined: List[bool]):
    if any(math.isnan(value) or value is None for value in value_combined):
        return np.nan
    if any(math.isinf(value) for value in value_combined):
        return 0
```

```
    combined_score = 0
    for value in value_combined:
        if value:
            combined_score += 1
    return combined_score * 10/3


def indicator_4_scoring(value_combined: float):
    if math.isnan(value_combined):
        return np.nan
    if math.isinf(value_combined):
        return 0
    combined_score = None
    if value_combined <= 50:
        combined_score = 3
    elif value_combined > 50 and value_combined <= 75:
        combined_score = 2
    elif value_combined > 75 and value_combined <= 100:
        combined_score = 1
    elif value_combined > 100:
        combined_score = 0

    return combined_score * 10/3


def indicator_5_scoring(value_combined: float):
    if math.isnan(value_combined):
        return np.nan
    if math.isinf(value_combined):
        return 0
    combined_score = None
    if value_combined <= 33.33:
```

```
        combined_score = 3
    elif value_combined > 33.33 and value_combined <= 66.67:
        combined_score = 2
    elif value_combined > 66.67 and value_combined <= 100:
        combined_score = 1
    elif value_combined > 100:
        combined_score = 0

    return combined_score * 10/3

def indicator_6_scoring(value_general: float):
    if math.isnan(value_general):
        return np.nan
    if math.isinf(value_general):
        return 0
    general_score = None
    if value_general > 15:
        general_score = 3
    elif value_general > 5 and value_general <= 15:
        general_score = 2
    elif value_general > 0 and value_general <= 5:
        general_score = 1
    elif value_general == 0:
        general_score = 0

    return general_score * 5/3

def indicator_7_scoring(value_allFunds: List[bool], is_issued_budget_in_last_yr: bool):
    assert len(value_allFunds) == 3
    if any(math.isnan(value) or value is None for value in value_allFunds):
```

```
        return np.nan
    score = None
    if is_issued_budget_in_last_yr:
        score = 3
    elif value_allFunds[0] and value_allFunds[1] and value_allFunds[2]:
        score = 3
    elif value_allFunds[0] and value_allFunds[1]:
        score = 2
    elif value_allFunds[0]:
        score = 1
    else:
        score = 0

    return score * 5/3

def indicator_8_scoring(value_allFunds: List[float]):
    if any(math.isnan(value) for value in value_allFunds):
        return np.nan
    if any(math.isinf(value) for value in value_allFunds):
        return 0
    value_allFunds_average = sum(value_allFunds) / len(value_allFunds)
    allFunds_score = None
    if value_allFunds_average >= 75:
        allFunds_score = 3
    elif value_allFunds_average >= 70 and value_allFunds_average < 75:
        allFunds_score = 2
    elif value_allFunds_average >= 65 and value_allFunds_average < 70:
        allFunds_score = 1
    elif value_allFunds_average < 65:
        allFunds_score = 0
```

```
    return allFunds_score * 5/3


def indicator_9_scoring(value_allFunds: List[float]):
    if any(math.isnan(value) for value in value_allFunds):
        return np.nan
    if any(math.isinf(value) for value in value_allFunds):
        return 0
    value_allFunds_average = sum(value_allFunds) / len(value_allFunds)
    allFunds_score = None
    if value_allFunds_average >= 20:
        allFunds_score = 3
    elif value_allFunds_average >= 15 and value_allFunds_average < 20:
        allFunds_score = 2
    elif value_allFunds_average >= 10 and value_allFunds_average < 15:
        allFunds_score = 1
    elif value_allFunds_average < 10:
        allFunds_score = 0


    return allFunds_score * 5/3
# prefix for general and combined codes for calculating indicator's value
general_fund_prefix = ['A', 'DA']
combined_fund_prefix = ['A', 'B', 'DA', 'DB', 'FX', 'G', 'ES', 'EW', 'SS', 'SW']
# functions for calculation of value for each of the 9 indicators to be sent for scoring

# indicator 1 = Assigned and Unassigned Fund Balance
# indicator 2 = Total Fund Balance
def calculator_indicator_1(df: pd.DataFrame):
    return calculator_indicator_1_or_2(df, [915, 917], [915, 917, 924])
```

```
def calculator_indicator_2(df: pd.DataFrame):
    return calculator_indicator_1_or_2(df, [8029], [8029])


def calculator_indicator_1_or_2(df: pd.DataFrame, general_target_codes, combined_target_codes):
    num_codes = get_codes(general_fund_prefix, general_target_codes)
    num_df = get_filtered_df(df, [('ACCOUNT_CODE', num_codes)])
    den_df = get_filtered_df(df,
                             [('FINANCIAL_STATEMENT', ['STATEMENT OF EXPENDITURES AND OTHER USES'])],
                             [('ACCOUNT_CODE', general_fund_prefix)])
    num_total = num_df.AMOUNT.sum()
    den_total = den_df.AMOUNT.sum()
    value_general = (num_total/den_total)*100


    num_codes = get_codes(combined_fund_prefix, combined_target_codes)
    num_df = get_filtered_df(df, [('ACCOUNT_CODE', num_codes)])
    den_df = get_filtered_df(df,
                             [('FINANCIAL_STATEMENT', ['STATEMENT OF EXPENDITURES AND OTHER USES'])],
                             [('ACCOUNT_CODE', combined_fund_prefix)])
    num_total = num_df.AMOUNT.sum()
    den_total = den_df.AMOUNT.sum()
    value_combined = (num_total/den_total)*100
    return value_general, (value_combined - value_general)


# indicator 3 = Operating Deficits
def calculator_indicator_3(df: pd.DataFrame):
    gross_revenue_df = get_filtered_df(df,
                             [('FINANCIAL_STATEMENT', ['STATEMENT OF REVENUES AND OTHER SOURCES'])],
                             [('ACCOUNT_CODE', combined_fund_prefix)])
```

```
    revenue_total = gross_revenue_df.AMOUNT.sum()

    gross_expenditure_df = get_filtered_df(df,
                           [('FINANCIAL_STATEMENT', ['STATEMENT OF EXPENDITURES AND OTHER USES'])],
                           [('ACCOUNT_CODE', combined_fund_prefix)])

    expenditure_total = gross_expenditure_df.AMOUNT.sum()

    return (revenue_total-expenditure_total)/expenditure_total, None


# indicator 4 = Cash Ratio
def calculator_indicator_4(df: pd.DataFrame):
    combined_cash_and_invest__codes = list(range(200, 223)) + [450, 451]
    combined_current_liab_codes = list(range(600, 626)) + list(range(631, 637)) + list(range(639, 668))
    combined_tax_rec_codes = [280,290, 295]

    num_codes = get_codes(combined_fund_prefix, combined_cash_and_invest__codes)
    den_1_codes = get_codes(combined_fund_prefix, combined_current_liab_codes)
    den_2_codes = get_codes(combined_fund_prefix, combined_tax_rec_codes)
    cash_and_invest_df = get_filtered_df(df, [('ACCOUNT_CODE', num_codes)])
    current_liab_df = get_filtered_df(df, [('ACCOUNT_CODE', den_1_codes)])
    tax_rec_df = get_filtered_df(df, [('ACCOUNT_CODE', den_2_codes)])

    num_total = cash_and_invest_df.AMOUNT.sum()
    den_1_total = current_liab_df.AMOUNT.sum()
    den_2_total = tax_rec_df.AMOUNT.sum()
    value_combined = (num_total/(den_1_total-den_2_total))*100
    return value_combined, None
```

```
#indicator 5 = Cash as a Percentage of Monthly Expenditures
def calculator_indicator_5(df: pd.DataFrame):
    combined_target_codes = [200, 201, 450, 451]
    num_codes = get_codes(combined_fund_prefix, combined_target_codes)
    num_df = get_filtered_df(df, [('ACCOUNT_CODE', num_codes)])
    den_df = get_filtered_df(df,
                             [('FINANCIAL_STATEMENT', ['STATEMENT OF EXPENDITURES AND OTHER USES'])],
                             [('ACCOUNT_CODE', combined_fund_prefix)])
    num_total = num_df.AMOUNT.sum()
    den_total = den_df.AMOUNT.sum()
    value_combined = (num_total/(den_total/12))*100
    return value_combined, None


# indicator 6 = Short-Term Debt Issuance
def calculator_indicator_6(df: pd.DataFrame):
    short_term_debt_df = get_filtered_df(df,
                               [('DEBT_TYPE_DESC', ['Revenue Anticipation Note', 'Tax Anticipation Note',
                                                    'Budget Note', 'Deficiency Note'])])


    total_revenue_df = get_filtered_df(df,
                              [('FINANCIAL_STATEMENT', ['STATEMENT OF REVENUES AND OTHER SOURCES']),
                               ('FINANCIAL_STATEMENT_SEGMENT', ['REVENUES'])],
                              [('ACCOUNT_CODE', general_fund_prefix)])
    num_total = short_term_debt_df.ENDING_YEAR_BALANCE_SUM.sum()
    den_total = total_revenue_df.AMOUNT.sum()
    value_general = (num_total/den_total)*100
```

```
    return value_general, None


# indicator 7 = Short-Term Debt Issuance Trend
def calculator_indicator_7(df: pd.DataFrame):
    short_term_debt_df = get_filtered_df(df,
                                         [('DEBT_TYPE_DESC', ['Revenue Anticipation Note', 'Tax Anticipation Note',
                                                              'Budget Note', 'Deficiency Note'])])
    budget_note_issued = get_filtered_df(df,
                                         [('DEBT_TYPE_DESC', ['Budget Note'])])
    debt_issuance = short_term_debt_df.AMOUNT.sum()
    is_issued_budget_in_last_yr = budget_note_issued.DEBT_ISSUED_IN_CURRENT_YEAR.sum() > 0


    return debt_issuance, is_issued_budget_in_last_yr


# indicator 8 = Personal Services and Employee Benefits % Revenues
def calculator_indicator_8(df: pd.DataFrame):
    personal_service_employee_benefits_df = get_filtered_df(df,
                                                            [('OBJECT_OF_EXPENDITURE', ['Personal Services', 'Employee Benefits'])])


    total_revenue_df = get_filtered_df(df,
                                       [('FINANCIAL_STATEMENT', ['STATEMENT OF REVENUES AND OTHER SOURCES']),
                                        ('FINANCIAL_STATEMENT_SEGMENT', ['REVENUES'])])
    num_total = personal_service_employee_benefits_df.AMOUNT.sum()
    den_total = total_revenue_df.AMOUNT.sum()
    value_allFunds = (num_total/den_total)*100
    return value_allFunds, None


# indicator 9 = Debt Service % Revenues
```

```
def calculator_indicator_9(df: pd.DataFrame):
    debt_service_df = get_filtered_df(df,
                                      [('OBJECT_OF_EXPENDITURE', ['Debt Interest', 'Debt Principal'])])
    current_refunding_bond_proceeds_df = get_filtered_df(df,
                                           [('ACCOUNT_CODE', ['V5729'])])



    total_revenue_df = get_filtered_df(df,
                                       [('FINANCIAL_STATEMENT', ['STATEMENT OF REVENUES AND OTHER SOURCES']),
                                        ('FINANCIAL_STATEMENT_SEGMENT', ['REVENUES'])])
    num_total_1 = debt_service_df.AMOUNT.sum()
    num_total_2 = current_refunding_bond_proceeds_df.AMOUNT.sum()
    den_total = total_revenue_df.AMOUNT.sum()
    value_allFunds = ((num_total_1-num_total_2)/den_total)*100
    return value_allFunds, None


# list of all indicators and list of indicators currently running (for test)
indicator_scoring_functions = [indicator_1_scoring, indicator_2_scoring, indicator_3_scoring, indicator_4_scoring,
                               indicator_5_scoring, indicator_6_scoring, indicator_7_scoring, indicator_8_scoring,
                               indicator_9_scoring]
indicator_calculators = [calculator_indicator_1, calculator_indicator_2, calculator_indicator_3, calculator_indicator_4,
                         calculator_indicator_5, calculator_indicator_6, calculator_indicator_7, calculator_indicator_8,
                         calculator_indicator_9]
# class of indicators to store intermediate data i.e. value of each of the 9 indicators of all towns in all years
class Indicator:
    def __init__(self,
                 indicator_number: str,
```

```
            indicator_weight: int,
            calculator: Callable,
            score_condition: Callable):
    self.indicator_number = indicator_number
    self.indicator_weight = indicator_weight
    self.calculator = calculator
    self.score_condition = score_condition
    self.score = None
    self.value_1 = None
    self.value_2 = None
    self.year_dependent_values = None

def run_calculator(self, df: pd.DataFrame):
    if self.calculator is None:
        self.score = np.nan
    else:
        value_1, value_2 = self.calculator(df)
        self.value_1 = value_1
        self.value_2 = value_2

def calculate_score(self):
    if self.value_1 is None:
        return
    if self.year_dependent_values is not None:
        if type(self.year_dependent_values) is not list:
            self.score = np.nan
        else:
            if self.value_2 is None:
                # Indicator 3, 8, 9
                self.score = self.score_condition(self.year_dependent_values)
```

```
            else:
                # Indicator 7
                self.score = self.score_condition(self.year_dependent_values, self.value_2)
        return
    if self.value_2 is None:
        # Indicator 4,5,6
        self.score = self.score_condition(self.value_1)
    else:
        # Indicator 1,2
        self.score = self.score_condition(self.value_1, self.value_2)

def get_score(self):
    return self.score

def get_number(self):
    return self.indicator_number

def get_number_with_weight(self):
    return f'{self.indicator_number} ({self.indicator_weight}%)'

def get_value_1(self):
    return self.value_1

def get_value_2(self):
    return self.value_2

def set_year_dependent_values(self, values):
    self.year_dependent_values = values

def get_year_dependent_values(self):
```

```
        return self.year_dependent_values


# list of unique values of all years
years_list = acoounting_data_df.CALENDAR_YEAR.unique().tolist()
indicators_weight = {1: 25,    2: 25,    3: 10,    4: 10,    5: 10,    6: 5,    7: 5,    8: 5,    9: 5}
# creating the final result df which will have scores for all towns, all years, all 9 indicators, their resultant stress score and its classification
col_Names = ('Year', 'Town_Name', 'Indicator_1 (25%)', 'Indicator_2 (25%)', 'Indicator_3 (10%)', 'Indicator_4 (10%)', 'Indicator_5 (10%)',
             'Indicator_6 (5%)', 'Indicator_7 (5%)', 'Indicator_8 (5%)', 'Indicator_9 (5%)', 'Max_score', 'Stress_Score', 'Score_Classifications')
 stress_score_df = pd.DataFrame(columns=col_Names)
# intermediate calculation of indicators requiring either bool condition or >1 years data
def calculate_year_dependent_values_for_indicator_3(year, town, data_map):
    if year-1 not in data_map or year-2 not in data_map:
        return np.nan
    if town not in data_map[year-1] or town not in data_map[year-2]:
        return np.nan
    this_year_deficit = data_map[year][town][2].get_value_1() < 0
    prev_year_deficit = data_map[year-1][town][2].get_value_1() < 0
    prev_2_year_deficit = data_map[year-2][town][2].get_value_1() < 0
    return [this_year_deficit, prev_year_deficit, prev_2_year_deficit]



def calculate_year_dependent_values_for_indicator_7(year, town, data_map):
    if year-1 not in data_map or year-2 not in data_map:
        return np.nan
    if town not in data_map[year-1] or town not in data_map[year-2]:
        return np.nan
    this_year_issuance = data_map[year][town][6].get_value_1() > 0
    prev_year_issuance = data_map[year-1][town][6].get_value_1() > 0
```

```
    prev_2_year_issuance = data_map[year-2][town][6].get_value_1() > 0
    return [this_year_issuance, prev_year_issuance, prev_2_year_issuance]


def calculate_year_dependent_values_for_indicator_8_or_9(year, town, data_map, index):
    if year-1 not in data_map or year-2 not in data_map:
        return np.nan
    if town not in data_map[year-1] or town not in data_map[year-2]:
        return np.nan
    this_year_percent_revenue = data_map[year][town][index].get_value_1()
    prev_year_percent_revenue = data_map[year-1][town][index].get_value_1()
    prev_2_year_percent_revenue = data_map[year-2][town][index].get_value_1()
    return [this_year_percent_revenue, prev_year_percent_revenue, prev_2_year_percent_revenue]
# Run calculator for each year for each town
data_map = {}
print('Running all calculators')
for year in years_list:
    print(f'Running calculator for year={year}')
    data_map[year] = {}
    filtered_year_df = accounting_data_df[accounting_data_df.CALENDAR_YEAR == year]
    towns = filtered_year_df.ENTITY_NAME.unique().tolist()
    for town in towns:
        print(f'Running calculator for town={town}')
        indicators = []
        count = 1
        for scoring_function, calculator in zip(indicator_scoring_functions, indicator_calculators):
            indicators.append(Indicator(f'Indicator_{count}', indicators_weight[count], calculator, scoring_function))
            count += 1
        filtered_town_df = filtered_year_df[filtered_year_df.ENTITY_NAME == town]
        for indicator in indicators:
```

```
            indicator.run_calculator(filtered_town_df)
        data_map[year][town] = indicators
print('Running all calculators finished')
# indicators requiring year_dependent_values
for year in data_map.keys():
    for town in data_map[year].keys():
        for indicator in data_map[year][town]:
            if indicator.get_number() == 'Indicator_3':
                indicator.set_year_dependent_values(calculate_year_dependent_values_for_indicator_3(year, town, data_map))
            elif indicator.get_number() == 'Indicator_7':
                indicator.set_year_dependent_values(calculate_year_dependent_values_for_indicator_7(year, town, data_map))
            elif indicator.get_number() == 'Indicator_8':

indicator.set_year_dependent_values(calculate_year_dependent_values_for_indicator_8_or_9(year, town, data_map, 7))
            elif indicator.get_number() == 'Indicator_9':

indicator.set_year_dependent_values(calculate_year_dependent_values_for_indicator_8_or_9(year, town, data_map, 8))


# printing intermediate results to keep check
for year in data_map.keys():
    print(f'year={year}')
    for town in data_map[year].keys():
        print(f'\ttown={town}')
        for indicator in data_map[year][town]:
            print(f'\t\t{indicator.get_number()}:      {indicator.get_value_1()}      {indicator.get_value_2()}      {indicator.get_year_dependent_values()}')
# Caching data map to avoid re-computation
f = open("../data/cache/data_map.txt", "w")
for year in data_map.keys():
```

```
    f.write(f'year={year}\n')
    for town in data_map[year].keys():
        f.write(f'\ttown={town}\n')
        for indicator in data_map[year][town]:
            f.write(f'\t\t{indicator.get_number()}:     {indicator.get_value_1()}     {indicator.get_value_2()}
{indicator.get_year_dependent_values()}\n')
f.close()
# Calculate score
print('Calculating scores')
for year in data_map.keys():
    print(f'Running for year={year}')
    towns = filtered_year_df.ENTITY_NAME.unique().tolist()
    for town in data_map[year].keys():
        print(f'Calculating score for town={town}')
        all_scores = {}
        for indicator in data_map[year][town]:
            indicator.calculate_score()
            all_scores.update({indicator.get_number_with_weight(): indicator.get_score()})
        all_scores.update({'Year'               : year,
                           'Town_Name'          : town,
                           'Max_score'          : 100,
                           'Stress_Score'       : np.nan,
                           'Score_Classifications': np.nan})
        print(all_scores)
        stress_score_df = stress_score_df.append(all_scores, ignore_index=True)
print('Calculating all scores finished')
stress_score_df.to_csv(f'../answer.csv', index=False)
# after getting final df with stress scores
# drop data for years before 2000
stress_score_df = stress_score_df.drop(stress_score_df[stress_score_df.Year <2000].index)
# adding values of all indicators to get stress score
```

```
stress_score_df['Stress_Score'] = stress_score_df.iloc[:,2:10].sum(axis=1)
# adding score classifications
def get_score_classification(stress_score):
    if stress_score >= 0 and  stress_score <= 44.9:
        return 'No Designation'
    elif stress_score >= 45 and  stress_score <= 54.9:
        return 'Susceptible of Fiscal Stress'
    elif stress_score >= 55 and  stress_score <= 64.9:
        return 'Moderate Fiscal Stress'
    elif stress_score >= 65 and  stress_score <= 100:
        return 'Significant Fiscal Stress'
    else:
        raise Exception(f'Invalid Stress Score = {stress_score}')
# calling the above function to get classifications
stress_score_df['Score_Classifications'] =
stress_score_df['Stress_Score'].apply(get_score_classification)
stress_score_df.head()
stress_score_df.to_csv(f'../stress_score_df.csv', index=False)
# EDA
# stress score trend per year
average_stress_score_per_year = stress_score_df.groupby(['Year'])['Stress_Score'].mean()


ax = average_stress_score_per_year.plot(kind='bar', figsize=(8,5), color="green", fontsize=13);
ax.set_alpha(0.8)
ax.set_title("Average stress score per year", fontsize=22)
ax.set_ylabel("Average Stress Score", fontsize=10);
ax.annotate('Change in fund balance \n reporting method in 2010', xy=(11, 13),
            xycoords='data',
            xytext=(12, 20),
            arrowprops=dict(arrowstyle= '-|>',color='blue', lw=3.5,ls='--'))
```

```
ax.set_xlim(-1,18)
ax.set_ylim(0,30)
plt.text(-2, -8, 'Source: https://osc.state.ny.us/localgov/pubs/releases/gasb54.pdf')
plt.show()
# stress score trend per year from 2011
stress_score_df_from_2011 = stress_score_df[stress_score_df.Year >2010]
average_stress_score_from_2011 =
stress_score_df_from_2011.groupby(['Year'])['Stress_Score'].mean()
ax = average_stress_score_from_2011.plot(kind='bar', figsize=(8,5), fontsize=13);
ax.set_alpha(0.8)
ax.set_title("Average stress score per year", fontsize=22)
ax.set_ylabel("Average Stress Score", fontsize=10);
ax.annotate('Lowest', xy=(6, 9),
            xycoords='data',
            xytext=(6, 12),
            arrowprops=dict(arrowstyle= '-|>',color='blue', lw=3.5,ls='--'))

ax.set_xlim(-1,7)
ax.set_ylim(0,14)
plt.show()
# stress score trend based on classification
score_classification =stress_score_df[['Score_Classifications']]
score_classification.head()
score_classification.Score_Classifications.value_counts()
data = [['No Designation', 15907], ['Susceptible of Fiscal Stress', 353],
        ['Moderate Fiscal Stress', 108], ['Significant Fiscal Stress', 63]]
score_classification_df = pd.DataFrame(data, columns = ['Stress_Score_Classifications', 'Count'])
# stress score for Town of Rochester
town_of_rochester = stress_score_df[stress_score_df.Town_Name == 'Town of Rochester']
town_of_rochester = town_of_rochester[town_of_rochester.Year >2010]
stress_score_per_year = town_of_rochester['Stress_Score']
```

```
ax = stress_score_per_year.plot(kind='bar', figsize=(8,5), color="brown", fontsize=13);
ax.set_alpha(0.8)
ax.set_title("Town of Rochester: Stress Score Trend", fontsize=22)
ax.set_ylabel("Average Stress Score", fontsize=10);
ax.set_xlabel("Years", fontsize=10);
ax.set_xticklabels(['2011', '2012', '2013', '2014', '2015', '2016', '2017'])
plt.show()
# creating a df for population based analysis
stress_score_subset = stress_score_df[['Town_Name','Stress_Score','Year']]
accounting_data_subset = accounting_data_df[['ENTITY_NAME', 'COUNTY']]
accounting_data_subset = accounting_data_subset.drop_duplicates(subset='ENTITY_NAME')
accounting_data_subset = accounting_data_subset.rename({'ENTITY_NAME': 'Town_Name'}, axis=1)
county_wise_df = pd.merge(stress_score_subset,accounting_data_subset,on='Town_Name',how='left')
# stress score trend for population
county_wise_df_1 = county_wise_df[county_wise_df.Year == 2017]
x = county_wise_df_1[['COUNTY', 'Stress_Score']]
x = county_wise_df_1.groupby(['COUNTY'])['Stress_Score'].mean().to_frame().reset_index()
# importing population df
population_df = pd.read_csv('../data/population_data.csv')
merged_df = pd.merge(x, population_df, on='COUNTY',how='inner')
# correlation & scatter plot
correlation = merged_df['Stress_Score'].corr(merged_df['Population'])
print(f'Correlation between Stress Score by County and Population of County is: ={correlation}')
scatter_plot = merged_df.plot.scatter(x='Stress_Score', y='Population', c='DarkBlue')
scatter_plot.plot()
plt.title("Relationship between Population size & \n Average Stress Score of a County", fontsize=14)
```