Analyzing Company Financials to Classify Executive Titles

Kevin W.S. Baum and Stephen F. Reagin

Shiley-Marcos School of Engineering, University of San Diego

Introduction

Top executives at publicly-traded companies are among the most highly compensated workers in American society, with total compensation packages often reaching 7 figures or more on an annual basis. The subject of executive pay is also contentious in popular media, where it is often addressed in the context of pay or wealth inequality.

Calls for greater transparency of executive pay have led to disclosure requirements, including both the amounts and performance conditions tied to the pay packages for top executive officers. Public companies must disclose, on an annual basis, the amounts paid to their top executives and the justifications for doing so. And as recently as August 2022 the SEC has adopted new rules requiring disclosure of the connection between company performance and executive pay (United States SEC, 2022).

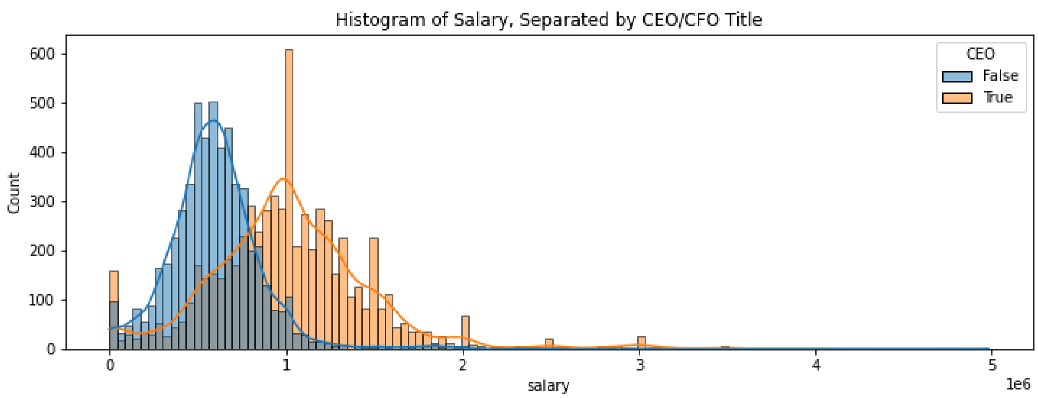
Looking through the *proxy statements* of public companies, which is the document in which this legally-required information is disclosed, one of the most common justifications for large executive pay packages is that higher pay should be tied to higher performance. However, performance can be defined differently for semiconductor stalwarts like Intel and AMD versus beverage companies like Keurig Dr Pepper and Coca Cola. That said, generally speaking, the most commonly agreed-upon metrics for overall company performance tend to focus on revenue and income growth.

A review of the academic and business literature in this space reveals common research themes: (a) the growth of executive pay over time, (Bebchuk & Grinstein, 2005), (b) the effect of increased transparency (Bruce & Skovoroda, 2015), and (c) consultancy services advising companies on how to tie executive pay to company performance (Abbott et al., 2019).

We would like to investigate the relationship, if any, between executive pay and company performance using a data-driven approach. However, it’s not clear at the outset that the techniques of machine learning and statistical mining are the appropriate tools for such an investigation. For example, we know CEOs tend to earn more than their CFO counterparts at most public companies with rare exceptions, and this is evidenced by the distribution of salaries (among other pay components) as seen in Figure 1.

Figure 1

Histogram of Salary, Separated by CEO/CFO Title



If we have a list of pay packages given to CEOs and CFOs for the S&P 500 companies, the classification algorithm is relatively simple for a human observer: (1) look at the two people listed for each company, (2) identify which person is paid more, and (3) label the person with higher pay as the CEO. We want to know whether computers can perform comparably using the standard Python tools of machine learning.

Therefore, the subject of our report is to investigate the feasibility of using mathematical tools to answer such questions. In particular, we ask: “can machines tell the difference between CEOs and CFOs on the basis of their pay packages?”

Overview

With the course's focus on classification and the encouragement in the instructions to do a project involving classification, we decided to create a classification model that would predict a CEO (Chief Executive Officer) versus a CFO (Chief Financial Officer), using pay components and financial data as predictor variables. This is not a dataset that we found on Kaggle or a site that specifically curated it for the purposes of using and discovering classification techniques. Rather, it is a completely unedited data scrub using SEC data. The process to discover our question was iterative, and we relied on exploratory data analysis and combining our intuitions in order to carve our path. In doing so, our overall goal was to demonstrate knowledge gained in ADS502 by using key learnings from the course to apply to "wild data."

With this in mind, we started with two datasets, one for executive pay, and the other for company financial data, extracting 10 years of quarterly "trailing 12 month" data from current S&P 500 companies into a Pandas data frame. We then grouped the data into average values for every year based on its ticker. For example, Apple will have average values specific to its financials for each year over the last 10 years. Next, we moved to the executive pay dataset, and extracted the executive information for those individuals within the S&P 500 ticker list over the last 10 years. After this, we combined the two separate data frames, left-joining them via their tickers so that financial information was duplicated for multiple executives by company and year. We then performed cleaning, pre-processing, and exploratory data analysis (EDA) before sub-setting the data frames by CEO and CFO, and finally concatenating them together.

Once this data frame was ready, we partitioned the data into training and testing datasets. Next, we next scaled the data because of the large numbers involved when summing over ten years. We proceeded to run CART, C5.0, random forest, logistic regression, naïve bayes, K Neighbors, and support vector machine models and scored them with evaluation metrics.

Initial Data

The ultimate source of all executive compensation information is the United States Securities and Exchange Commission (SEC). Every publicly-traded company must file an annual proxy statement with the SEC for shareholders to review, among other things, the total compensation packages paid to the CEO, CFO, and next three most highly-paid officers, a group is often referred to as the “Top 5 executives.” Of particular importance is the “Summary Compensation Table” which is also required by the SEC, and for all Top 5 officers the company must disclose three years of:

- Salary
- Performance bonus (“Non-Equity Incentive Compensation”)
- Stock and Option awards
- Off-cycle or one-time bonuses
- Other services like personal security, air travel, etc.

However, this data is inconvenient to source directly from .html files on the SEC because of the manual labor involved. We instead pulled data from sec-api.io, a third-party source, using API calls under a paid academic license. This raw dataset includes more than 170,000 observations of Summary Compensation Table data rows for companies in the Russell 3000, dating from 2006 to 2022. We also feature engineered Boolean title columns (CEO, CFO, Interim) to flag specific roles. We further narrowed our universe to include only the 485 members of the S&P 500 for which we have approximately 28,000 reliable data observations.

We sourced company financial information from stockrow.com using their free Excel-based worksheets of Income Statement information, specifically targeting Revenue and Net Income as the most common metrics of financial performance. Table 1 shows the descriptive statistics of the S&P 500 subset.

Table 1

Descriptive Statistics of Raw Dataset

Feature	Observations	Median	Mean
Salary	28440	$618K	$708K
Bonus	28440	$0	$247K
stockAwards	28440	$1,425K	$2,788K
optionAwards	28440	$93K	$5,133K
nonEquityIncentiveCompensation	28440	$612K	$1,054K
otherCompensation	28440	$61K	$370K
Total	28440	$4,083K	$6,460K
Revenue	28391	$8.79B	$22.0B
Net Income Common	28391	$787M	$1.99B

Data Preparation and Exploratory Analysis

The first step was to get the data we needed from the SEC API for executive pay, and from Stockrow.com for the company financials. After gathering the data from both sources, we decided we wanted to use only S&P 500 data and so we also loaded a data source containing the 503 tickers. We matched the data from the company financials to only return data for each ticker in the list of 503. For each ticker in the list, we created CSV files and then combined all the CSV files into a Pandas data frame to perform manipulations on. Now that our data frame was ready, we narrowed the fields into just ones we wanted: Ticker, Date, Revenue, Gross Profit, Operating Income, Income after Tax, and Net Income Common. Additionally, we created a datetime object from the “date” feature and extracted the year out of it. This let us ultimately break down the information by yearly basis so that we could establish yearly trends.

Next, we grouped the data by ticker (representative of company) and year and got the mean values for each ticker per year. To make sure this was done correctly, we manually checked “AAPL,” Apple, and found that the numbers matched when we did an independent search online. From here, we were ready to join in the executive pay data set and did so based on the ticker; only executive pay information for the S&P 500 companies were included. Note that not all companies were in the executive pay dataset, and after merging them together only 485 companies remained with 18 not present in the executive pay from the SEC API.

For data cleaning, we noted some duplicate records and performed a standard drop_duplicates() method to eliminate these duplicates. The company Agilent had duplicate records that were duplicates in reality, but had only one slight difference that caused them to get past the drop_duplicates method. We eliminated these duplicates by deleting the column that was supposed to contain full names but instead contained the words “human resources” and “group.” We noted about 200 outliers in the executive pay data and dropped these as well. One such outlier was a salary of 18 billion US dollars in one year.

Additionally, there were also some missing values in the company financials. For instance, 73 of the 485 companies did not have revenue data in 2012 and 14 did not have revenue data in 2021. We decided not to impute values or drop these companies because our classification model was classifying executives as either CEOs or CFOs. The more important metrics, in our eyes, were individual pay statistics.

For exploratory data analysis (EDA), we aimed to find patterns in the pay of CEOs over the last 10 years.

Figure 2

CEO Salary Percentiles 2012-2021

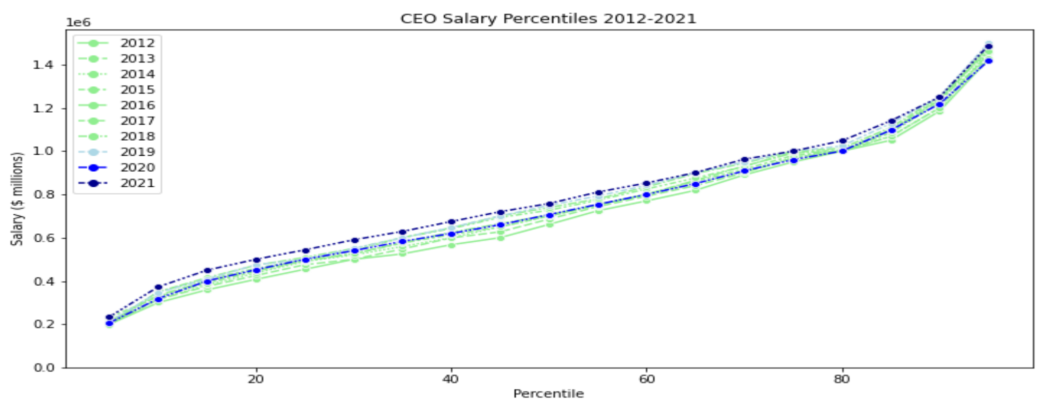


Figure 2 is constructed with the percentiles of CEO salary on the x-axis with the actual salary number on the y-axis. This means if a salary is the highest in the dataset, it will be in the 99th percentile, and if a salary is the lowest in the dataset, it will be in the 1st percentile. From this visualization, we can see that there is a consistent slope until roughly the 80th percentile mark is reached. This means that the increase in income from say the 40th percentile to the 60th percentile is reliable and predictable. This is not the case after about the 80th percentile however, where we see a steepening of the slope of the line for all years. This indicates that those CEOs above the 80th percentile earn much than those below them. A comparable increase in earner percentile, say 20-percentile increase, from the 40th to 60th is much less in real dollar value than a 20-percentile increase from the 70th percentile earner to the 90th percentile earner. Additionally, we highlight the lines for 2020 and 2021 to show the difference between these two years and note that 2021 is much higher for salary than 2020 for any given percentile chosen.

We have also created boxplots, violin plots, kernel density estimate (KDE) plots, a color mapped kernel density estimate (KDE) plot, and bar plots to investigate a multitude of factors. Our focus was on the changing CEO pay landscape, and we used visualizations to investigate changes in salary, stock awards, non-equity incentive compensation, and total compensation. We found a trend of CEO salaries rising in the last 16 years (since 2006), and the most visually telling representation of this trend was the KDE plot in Figure 3.

Figure 3

KDE Plot for CEO Salary Under $3M

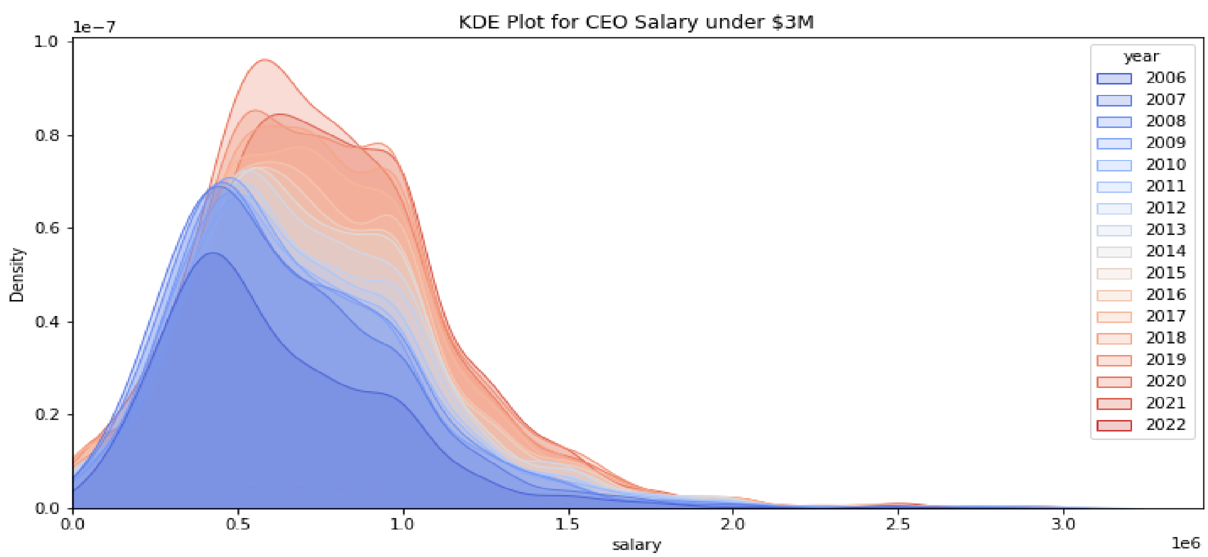


Figure 3 displays the density of incomes from 2006-2022 with blue marking the earlier years and red marking the more recent years, showing a clear trend of rising salaries over time.

When analyzing stock awards via box plot in Figure 4, we see a surprising widening of the interquartile range (IQR) spread in 2021, following a trend of increased IQR over time.

Figure 4

Boxplot Distribution of Stock Awards Less Than $30M

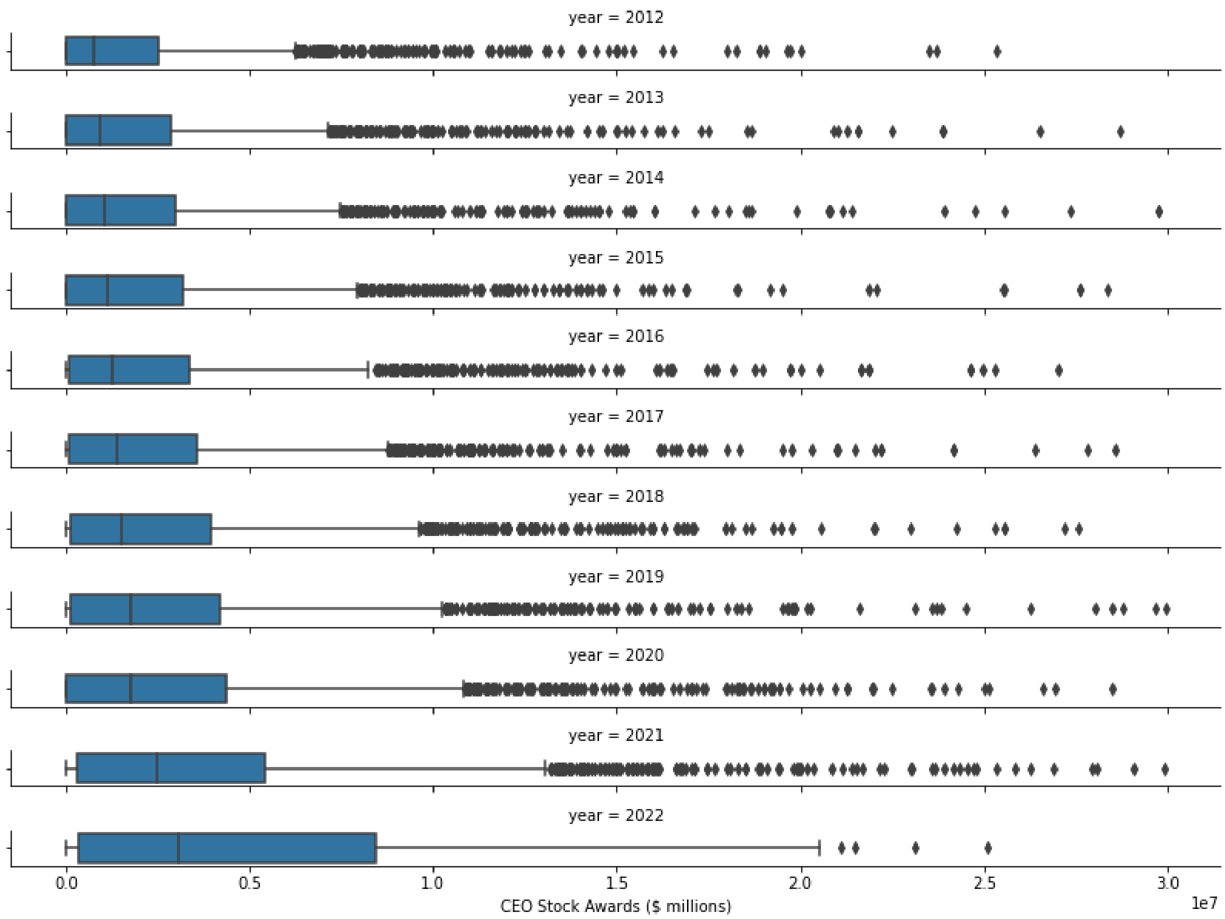


Figure 4 displays the overall trend of widening IQR ranges between 2012 and 2021, which may be due to stock awards being a more common form of compensation now compared with years past. Such a short box with extreme positive skewness in 2012 indicates that a high proportion of CEOs had smaller stock awards as part of their compensation package. In fact, for all years the data is very positively skewed, which indicates that the mean is larger than the median as evidenced in Table 1 and is heavily impacted by outlying very large values.

This is a logical conclusion based on the observation that the top 20% of CEO earners are making much more than the lower 80%. The large stock awards going to these top earners drags up the mean and results in a high positive skewness.

Finally, it is worth noting that companies vary wildly in how they pay their top earners. For instance, Microsoft has a very exaggerated change the year that they elected to have Satya Nadella become the new CEO in 2014. In years prior, the CEO position received relatively low compensation from the company. This is because Bill Gates and Steve Ballmer, the prior CEOs, were original founders and had already earned many tens of billions of dollars, thus any annual compensation was considered unlikely to motivate increased performance. But when it was time to make a change, the chief executive position started to make thousands of times the amount as before. Microsoft is only one example, and it is conceivable that other CEOs that founded their companies likewise have decided to have very low compensation as well.

To address this inherent variability in executive pay and financial performance, our final training dataset consisted of the *sum* of compensation components and financial metrics over the previous 10 years for each company. In this way, even if transitions occurred, we were still capturing the total amount which the company paid to *the CEO role* over the relevant period.

Data Mining Preliminary Results

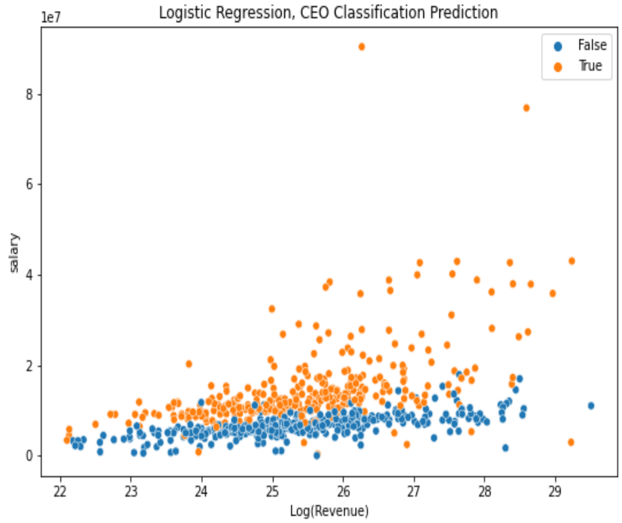
Our basic goal for model selection was to follow the textbook's advice, "to select the model that shows lowest generalization error rate" (Tan et al., 2019). We considered other possibilities for optimal selection, such as minimizing the count of false positives or false negatives. Ultimately, we felt that a generalized accuracy measure would work well for this type of data, as there is not a need to be extra conservative like there would be for, say, diagnosing whether a patient's health indicators classify them as having a disease or not having it. In such a case, it would be better to have false positives (patient does not really have the disease but tests say they do) rather than false negatives (patient really has the disease but tests say they do not).

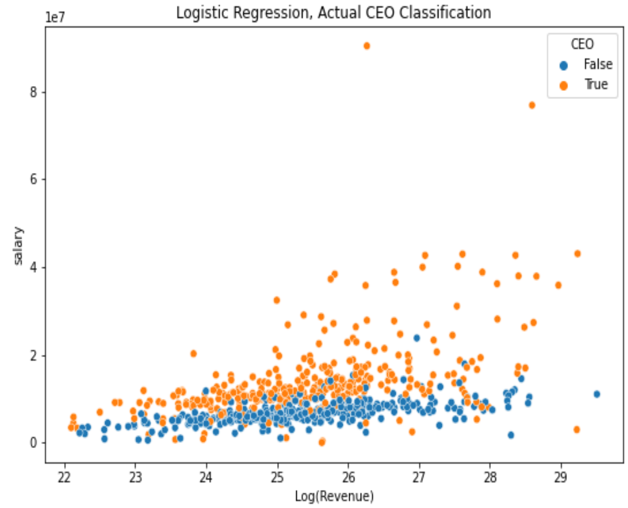
To evaluate our models, we partitioned the data to "assess the performance of a learned model on a labeled test set [which] has not been used at any stage of model selection" (Tan et al., 2019). We felt that the data set was sufficiently small to follow the books advice for smaller or less complex datasets, where "one should retain sufficient records for accurate assessment, so that the training sets would contain only 50–67% of the original data" (Tan et al., 2019). Our main concern was balancing the proportion so that the training model could effectively learn the behavior but not be so large that the testing model's error metric was unreliable because it was not able to compute over enough instances. Ultimately, we decided on a 70% partition into the training set and 30% into the test set.

We tried different models to see which would produce the best results for reducing our error rate, producing the best accuracy for true positives. The model which we felt was best for this purpose was the logistic regression model.

Figure 5

Comparison of Classification Prediction Versus Actual





Many of the models tended to overfit the training data. For instance, random forest produced a higher overall accuracy, but we felt it overfit the training data with an accuracy of 95% on the training data with 84% on the testing data. The CART and C5.0 models had lower accuracy for both training and testing than the logistic regression model, while also being slightly overfit in the training data. Naïve Bayes, KNeighbors, and SVM produced good results, but also had more of a difference in accuracy between the training and testing datasets than the logistic regression model. Ultimately, logistic regression works well for the general purpose for our model because it is a "discriminative model for classification that directly computes the poster[ior] probabilities without making any assumption about the class conditional probabilities" making it "generic" and "applicable in diverse applications" (Tan et al., 2019).

The baseline condition was 52%, representing the proportion of CEOs to CFOs in the dataset. While all of the models in Table 2 represent a significant improvement above the baseline 52% accuracy, logistic regression was our preferred choice.

Table 2

Comparison of Model Accuracy Results

Model	Train Data Accuracy	Test Data Accuracy
CART	87%	83%
C5.0	87%	80%
Random Forest	95%	84%
Logistic Regression	90%	89%
Naïve Bayes	87%	83%
KNeighbors	91%	87%
Support Vector Machine	91%	88%

Further Discussion

Classifying CEOs and CFOs is only scratching the surface of the type of information this data source is capable of. Our initial goal was to solve the more politically hard-hitting question of, “do executives actually get paid based on their companies do or do they reward themselves no matter what the company does?” We wanted to create a classification model that would predict if a CEOs or CFOs compensation trends would correctly classify if a company was a rapid grower, flatliner, or decliner. Even though the 2010s was a very prosperous time in the American economy, we found that about 70 of the companies in the S&P 500 had declining revenues during the period. One of the beautiful elements of the economy though is that growth is limitless, while the maximum amount of decline possible is just 100%. Thus, due to the fantastic success of companies like Tesla and Amazon, the growth of the overall economy far outweighed the companies bringing down the overall index.

Beyond classification, the data set also has potential for linear regression projects, as well as unsupervised algorithms like clustering. For example, a linear regression project could investigate which factors are most important to overall compensation for executives. Variables such as company revenue, company net income, position title, age, location, and many more could provide indicators as to which factors affect executive pay statistics the most. Clustering algorithms could be used to find trends that are worth exploring for supervised algorithms that are not known from an initial view of the data.

References

Abbott, S., Aksoy, M., Groysberg, B., & Marino, M.R. (2019, February). Compensation Packages That Actually Drive Performance. Principles for designing executive pay. *Harvard Business Review*. https://hbr.org/2021/01/compensation-packages-that-actually-drive-performance

Bebchuk, L., & Grinstein, Y. (2005). The Growth of Executive Pay. *Oxford Review of Economic Policy*, *21(2)*, 283–303. https://doi.org/10.1093/oxrep/gri017

Bruce, A., & Skovoroda, R. (2015). *The Empirical Literature of Executive Pay: Context, the Pay-Performance Issueand Future Directions.* Nottingham University Business School. http://irep.ntu.ac.uk/id/eprint/35402/1/12805_Bruce.pdf

Tan, P. N., Steinbach, M., Karpatne, A., & Kumar, V. (2019). *Introduction to Data Mining* (2nd ed.). Pearson.

United States Securities and Exchange Commission. (2022, August 25). *SEC Adopts Pay Versus Performance Disclosure Rules.* https://www.sec.gov/news/press-release/2022-149

In [3]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import csv
import requests
import os
import time
import json
import math
import pandas as pd
from sklearn.model_selection import train_test_split
import random
import statsmodels.tools.tools as stattools
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

First task is to create the dataframe of S&P 500 relevant financials and executives. We will constrain analysis to just these 503 companies to narrow the scope of this project.

In [4]:
```python
#List of S&P 500 tickers from: https://stockmarketmba.com/stocksinthesp500.php
ticker_list = pd.read_csv('SP500_index.csv')['Symbol']
ticker_list
```

Out[4]:
```
0       AAPL
1       MSFT
2       GOOG
3      GOOGL
4       AMZN
       ...
498     LUMN
499      ALK
500      NWL
501      VNO
502      TAP
Name: Symbol, Length: 503, dtype: object
```

In [5]:
```python
# Function to create excel files based on company financials
def income_puller(ticker):
    url=f'https://stockrow.com/api/companies/{ticker}/financials.xlsx?dimension=T&section=Income%20Statement&so
    response = requests.get(url)
    with open(os.path.join("StockRow_financials/Excel", f"{ticker}_inc.xlsx"), 'wb') as f:
        f.write(response.content)
    time.sleep(0.25)

# Function to create CSV files based on company files
# Pulling based on ticker
def csv_maker(ticker):
    #this section reads the Excel file and turns into a CSV
    ticker_df = pd.read_excel(f'StockRow_financials/Excel/{ticker}_inc.xlsx').T
    # change the first row to the columns
    ticker_df.columns = ticker_df.iloc[0]
    #remove the first row
    ticker_df = ticker_df.iloc[1:]

    ticker_df['Ticker'] = f"{ticker}"

    ticker_df.to_csv(f'StockRow_financials/CSV/{ticker}_inc.csv',index_label="Date")
```

This code pulls financial information from StockRow.com and we have left it in as a commented out cell to display the process of excel and CSV creation.

In [6]:
```python
#This was a one-time request to create the CSVs for each ticker.
#start_time = time.time()
#
#for ticker in ticker_list:
#    try:
#        income_puller(ticker)
#    except:
#        print(f"Ticker {ticker} throws an error")
#
#print(f"Code took {np.round(time.time() - start_time,2)} seconds to run")
```

This code converts the StockRow workbooks from Excel files to CSV files

In [7]:
```python
# start_time = time.time()

# for ticker in ticker_list:
#     try:
#         csv_maker(ticker)
#     except:
#         print(f"Ticker {ticker} throws an error")

# print(f"Code took {np.round(time.time() - start_time,2)} seconds to run")
```

This code combines all downloaded CSV files (for which we have a ticker) into one Pandas DataFrame

In [8]:
```python
starter_df = pd.read_csv(f'StockRow_financials/CSV/{ticker_list[0]}_inc.csv')

start_time = time.time()

for ticker in ticker_list[1:]:
    try:
        ticker_df = pd.read_csv(f'StockRow_financials/CSV/{ticker}_inc.csv')
        starter_df = pd.concat([starter_df,ticker_df])
    except:
        print(f'Ticker {ticker} threw an error')



print(f"This code took {time.time() - start_time} seconds")
```

```
Ticker META threw an error
Ticker ELV threw an error
Ticker BALL threw an error
Ticker TRGP threw an error
Ticker GEN threw an error
This code took 5.213965177536011 seconds
```

In [9]:
```python
#save the DataFrame as one large CSV
starter_df.to_csv('SP500_allFinancials.csv',index=False)
```

Narrow the financial columns, convert "Date" to a DateTime object and create a "Year" field as we ultimately want to analyze the data on a year over year basis.

In [10]:
```python
starter_df = starter_df[['Ticker', 'Date', 'Revenue', 'Gross Profit',
                         'Operating Income', 'Income after Tax', 'Net Income Common']].reset_index(drop=True)

starter_df['Date'] = pd.to_datetime(starter_df['Date'])
starter_df['Year'] = pd.DatetimeIndex(starter_df['Date']).year
```

In [11]:
```python
#example of grouping the data to find the average financial values for each calendar year
starter_df[starter_df['Ticker']=='AAPL'].groupby('Year').mean()
```

Out[11]:

Year	Revenue	Gross Profit	Operating Income	Income after Tax	Net Income Common
2012	1.646870e+11	6.901900e+10	5.511100e+10	4.174700e+10	4.174700e+10
2013	1.708525e+11	6.525800e+10	5.011225e+10	3.787200e+10	3.787200e+10
2014	1.841935e+11	7.101400e+10	5.336100e+10	4.005850e+10	4.005850e+10
2015	2.263010e+11	9.029050e+10	6.853200e+10	5.141750e+10	5.141750e+10
2016	2.203950e+11	8.646250e+10	6.224650e+10	4.734475e+10	4.734475e+10
2017	2.280935e+11	8.771550e+10	6.122750e+10	4.781425e+10	4.781425e+10
2018	2.574745e+11	9.855900e+10	6.820600e+10	5.710000e+10	5.710000e+10
2019	2.613452e+11	9.912250e+10	6.499925e+10	5.641200e+10	5.641200e+10
2020	2.776220e+11	1.064325e+11	6.831750e+10	5.924500e+10	5.924500e+10
2021	3.541752e+11	1.469285e+11	1.036732e+11	8.958700e+10	8.958700e+10
2022	3.892957e+11	1.686247e+11	1.190483e+11	1.004570e+11	1.004570e+11

In [12]:
```python
#grouping the data to find average values per year for every ticker
condensed_finances_df = starter_df.groupby(['Ticker','Year']).mean()
condensed_finances_df
```

Out[12]:

Ticker	Year	Revenue	Gross Profit	Operating Income	Income after Tax	Net Income Common
A	2013	6.854500e+09	3.569250e+09	9.845000e+08	9.442500e+08	9.442500e+08
	2014	5.252250e+09	2.680750e+09	6.885000e+08	4.702500e+08	6.605000e+08
	2015	4.619250e+09	2.308250e+09	5.257500e+08	3.070000e+08	3.390000e+08
	2016	4.116000e+09	2.130250e+09	5.877500e+08	4.672500e+08	4.650000e+08
	2017	4.357750e+09	2.319750e+09	7.500000e+08	6.020000e+08	6.020000e+08
...	...	...	...	...	...	...
ZTS	2018	5.644000e+09	3.819000e+09	1.673500e+09	1.166000e+09	1.171250e+09
	2019	6.092500e+09	4.097500e+09	1.727250e+09	1.430500e+09	1.431000e+09
	2020	6.474000e+09	4.500250e+09	1.967500e+09	1.631250e+09	1.632250e+09
	2021	7.454000e+09	5.169500e+09	2.354000e+09	1.922500e+09	1.925500e+09
	2022	7.964333e+09	5.451667e+09	2.553333e+09	2.074000e+09	2.076667e+09

5330 rows × 5 columns

We compared the following Apple cell with a manual check of Apple financial data via search engine to ensure the trustworthiness of the method used to group the data.

In [13]:
```python
#sanity check using AAPL
condensed_finances_df.loc['AAPL']
```

Out[13]:

Year	Revenue	Gross Profit	Operating Income	Income after Tax	Net Income Common
2012	1.646870e+11	6.901900e+10	5.511100e+10	4.174700e+10	4.174700e+10
2013	1.708525e+11	6.525800e+10	5.011225e+10	3.787200e+10	3.787200e+10
2014	1.841935e+11	7.101400e+10	5.336100e+10	4.005850e+10	4.005850e+10
2015	2.263010e+11	9.029050e+10	6.853200e+10	5.141750e+10	5.141750e+10
2016	2.203950e+11	8.646250e+10	6.224650e+10	4.734475e+10	4.734475e+10
2017	2.280935e+11	8.771550e+10	6.122750e+10	4.781425e+10	4.781425e+10
2018	2.574745e+11	9.855900e+10	6.820600e+10	5.710000e+10	5.710000e+10
2019	2.613452e+11	9.912250e+10	6.499925e+10	5.641200e+10	5.641200e+10
2020	2.776220e+11	1.064325e+11	6.831750e+10	5.924500e+10	5.924500e+10
2021	3.541752e+11	1.469285e+11	1.036732e+11	8.958700e+10	8.958700e+10
2022	3.892957e+11	1.686247e+11	1.190483e+11	1.004570e+11	1.004570e+11

Now that the financials dataframe is ready, we commence preprocessing for the executive pay dataset that we ultimate want to join with the financials dataset via shared ticker.

In [14]:
```python
execpay_df = pd.read_csv('main_SCT_pay_file.csv')

execpay_df = execpay_df[execpay_df['ticker'].isin(ticker_list)]

execpay_df
```

Out[14]:

	id	cik	ticker	name	position	year	salary	bonus	stockAwards	optionA
0	73b3a60ba203743c008330c96b7d8b66	1090872	A	Sam Raha	Senior Vice President, President Diagnostics a...	2021	563500	0	1541332	
1	97393f60cd9f321650e472673daaa70c	1090872	A	Michael R. McMullen	Chief Executive Officer	2021	1280000	0	9165390	
2	83b9cc2bca477fe8ce23e0ab56e70c66	1090872	A	Jacob Thaysen	Senior Vice President, President Life Sciences...	2021	625000	0	1812285	
3	7c60bb804071675ae15ec930f6dea190	1090872	A	Robert McMahon	Senior Vice President, Chief Financial Officer	2021	663500	0	2291271	
4	259b710a8befe67c61a2c3dec14f344a	1090872	A	Padraig McDonnell	Senior Vice President, President Cross-Lab Group	2021	495000	0	1249771	
...	...	...	...	...	...	...	...	...	...	
140505	509ce14e56d662f93a33c221556bee61	829224	SBUX	Clifford Burrows	president, Starbucks Coffee Americas and US	2011	678942	0	2050337	9
140509	ca5ec5821d6a445f90716a3d9695b490	1393612	DFS	Harit Talwar	EVP, President— US Cards	2010	1750000	875000	0	
140517	5182311313ab849b40a9a5063ec9b2da	1175454	FLT	Todd W. House	President, US Direct Business & Chief Operatin...	2010	275000	40000	787500	12
140520	1c4e649f102fc4f3523b36f94390bd72	14272	BMY	Anthony C. Hooper	SVP Commercial Operations & President US, Japa...	2010	800000	0	2649962	
140535	8a25b731a4a44ab8020d41f1beda5ba5	1175454	FLT	Todd W. House	President, US Direct Business & Chief Operatin...	2009	192877	63906	0	3

44457 rows × 17 columns

In [15]:
```python
# Merging the company financials plus the executive pay datasets together to create
# the main data frame.
main_df = execpay_df.merge(condensed_finances_df,
                           left_on=['ticker','year'], right_on=['Ticker','Year'])
```

In [16]:
```python
main_df
```

Out[16]:

	id	cik	ticker	name	position	year	salary	bonus	stockAwards	optionA
0	73b3a60ba203743c008330c96b7d8b66	1090872	A	Sam Raha	Senior Vice President, President Diagnostics a...	2021	563500	0	1541332	
1	97393f60cd9f321650e472673daaa70c	1090872	A	Michael R. McMullen	Chief Executive Officer	2021	1280000	0	9165390	
2	83b9cc2bca477fe8ce23e0ab56e70c66	1090872	A	Jacob Thaysen	Senior Vice President, President Life Sciences...	2021	625000	0	1812285	
3	7c60bb804071675ae15ec930f6dea190	1090872	A	Robert McMahon	Senior Vice President, Chief Financial Officer	2021	663500	0	2291271	
4	259b710a8befe67c61a2c3dec14f344a	1090872	A	Padraig McDonnell	Senior Vice President, President Cross-Lab Group	2021	495000	0	1249771	
...	...	...	...	...	...	...	...	...	...	
28435	7b803e9ff0f142d5b1c36b5fdbaced15	1564708	NWS	Gerson Zweifach	General Counsel	2014	1500000	0	0	
28436	473bf52703cdbffdc9b311d8dee9101a	1564708	NWS	Bedi Ajay Singh	Chief Financial Officer	2013	655769	713425	496499	
28437	c36a1d9990a2218f8d2a09b25fb7cd65	1564708	NWS	Robert J. Thomson	Chief Executive Officer	2013	992308	1000000	0	
28438	7d64a65ecf544ffd0ce4d4ada1607c73	1564708	NWS	Gerson Zweifach	General Counsel	2013	0	0	0	
28439	abc0667932c203c041cae8baae64f178	1564708	NWS	K. Rupert Murdoch	Executive Chairman	2013	0	0	0	

28440 rows × 22 columns

NOTE: The following 10 cells (until the cell that starts with the comment "Start CEO/CFO Classification Model") were set up for another classification purpose to classify a company as growing, stagnating, or shrinking based on executive pay trends. The goal here was, "are changes in executive pay appropriate in line company performance?" We decided to not do this, but we wish to keep the code as it is an interesting project for the future, and it was a large factor in our discussion process of how we wanted to work with this data set to begin with.

In [17]:
```python
main_df.ticker.value_counts()
```

Out[17]:
```
A       112
MTCH     99
JCI      92
APA      90
EXC      88
       ...
CARR     16
CME      14
AMP       5
VTRS      5
OGN       5
Name: ticker, Length: 485, dtype: int64
```

In [18]:
```python
apple_df = main_df[main_df['ticker']=='AAPL']
apple_df
apple_df_2021 = apple_df[apple_df['year']==2020]
apple_df_2021
rev_2021 = apple_df_2021['Revenue']
rev_2021
# Making sure we can perform calculations with this, looking for 'float' or 'int'
type(rev_2021)

apple_df_2012 = apple_df[apple_df['year']==2012]
rev_2012 = max(apple_df_2012['Revenue'])

subtracted_rev = rev_2021-rev_2012

percent_rev_change = (subtracted_rev/rev_2012)*100

print("Revenue percentage growth over 10 years: ", percent_rev_change)
```

```
Revenue percentage growth over 10 years:  117    68.57554
118    68.57554
119    68.57554
120    68.57554
121    68.57554
Name: Revenue, dtype: float64
```

In [19]:
```python
main_df_ticker_list = main_df['ticker'].unique()
main_df_ticker_list
data_2012 = []
# data_2013 = []
# data_2014 = []
# data_2015 = []
# data_2016 = []
# data_2017 = []
# data_2018 = []
# data_2019 = []
# data_2020 = []
data_2021 = []
data_change_dollars = []
data_change_percentage = []


data = {"Ticker": main_df_ticker_list,
       "Revenue 2012": data_2012,
#         "Revenue 2013": data_2013,
#         "Revenue 2014": data_2014,
#         "Revenue 2015": data_2015,
#         "Revenue 2016": data_2016,
#        "Revenue 2017": data_2017,
#         "Revenue 2018": data_2018,
#         "Revenue 2019": data_2019,
#         "Revenue 2020": data_2020,
        "Revenue 2021": data_2021,
       "Revenue Change $": data_change_dollars,
       "Revenue Change %": data_change_percentage}

for ticker in main_df_ticker_list:
    ticker_df = main_df[main_df["ticker"]==ticker]
    try:
        ticker_df_2012 = ticker_df[ticker_df["year"]==2012]
        rev_2012 = max(ticker_df_2012["Revenue"])
        data_2012.append(rev_2012)
    except:
        data_2012.append("No Data Available")
#     try:
#         ticker_df_2013 = ticker_df[ticker_df["year"]==2013]
#         rev_2013 = max(ticker_df_2013["Revenue"])
#         data_2013.append(rev_2013)
#     except:
#         data_2013.append("No Data Available")
#     try:
#         ticker_df_2014 = ticker_df[ticker_df["year"]==2014]
#         rev_2014 = max(ticker_df_2014["Revenue"])
#         data_2014.append(rev_2014)
#     except:
#         data_2014.append("No Data Available")
#     try:
#         ticker_df_2015 = ticker_df[ticker_df["year"]==2015]
#         rev_2015 = max(ticker_df_2015["Revenue"])
#         data_2015.append(rev_2015)
#     except:
#         data_2015.append("No Data Available")
#     try:
#         ticker_df_2016 = ticker_df[ticker_df["year"]==2016]
#         rev_2016 = max(ticker_df_2016["Revenue"])
#         data_2016.append(rev_2016)
#     except:
#         data_2016.append("No Data Available")
#     try:
#         ticker_df_2017 = ticker_df[ticker_df["year"]==2017]
#         rev_2017 = max(ticker_df_2017["Revenue"])
#         data_2017.append(rev_2017)
#     except:
#         data_2017.append("No Data Available")
#     try:
#         ticker_df_2018 = ticker_df[ticker_df["year"]==2018]
#         rev_2018 = max(ticker_df_2018["Revenue"])
#         data_2018.append(rev_2018)
#     except:
#         data_2018.append("No Data Available")
#     try:
#         ticker_df_2019 = ticker_df[ticker_df["year"]==2019]
#         rev_2019 = max(ticker_df_2019["Revenue"])
#         data_2019.append(rev_2019)
#     except:
#         data_2019.append("No Data Available")
#     try:
#         ticker_df_2020 = ticker_df[ticker_df["year"]==2020]
#         rev_2020 = max(ticker_df_2020["Revenue"])
#         data_2020.append(rev_2020)
#     except:
#         data_2020.append("No Data Available")
    try:
        ticker_df_2021 = ticker_df[ticker_df['year']==2021]
        rev_2021 = max(ticker_df_2021['Revenue'])
        data_2021.append(rev_2021)
    except:
        data_2021.append("No Data Available")
    try:
        rev_change_dollars = rev_2021 - rev_2012
        data_change_dollars.append(rev_change_dollars)
    except:
        print("No calculation performed")
    try:
        rev_change_percentage = (rev_change_dollars / rev_2012)*100
        data_change_percentage.append(rev_change_percentage)
    except:
        print("No calculation performed")



new_df = pd.DataFrame(data, index = main_df_ticker_list)
new_df
no_data_2012 = new_df["Revenue 2012"] == "No Data Available"
no_data_2021 = new_df["Revenue 2021"] == "No Data Available"
no_calc_2012 = new_df.loc[no_data_2012, ["Revenue Change $", "Revenue Change %"]] = "No Calculation Performed"
no_calc_2022 = new_df.loc[no_data_2021, ["Revenue Change $", "Revenue Change %"]] = "No Calculation Performed"
print(new_df.head(15))
print(new_df.shape)
```

```
      Ticker         Revenue 2012         Revenue 2021  \
A          A    No Data Available         5952000000.0
AAPL    AAPL       164687000000.0       354175250000.0
MSFT    MSFT        72930000000.0       172302750000.0
AMZN    AMZN        61093000000.0  447553750000.000061
TSLA    TSLA          413256000.0        44618250000.0
GOOGL  GOOGL    No Data Available    No Data Available
GOOG    GOOG        46039000000.0       228366750000.0
UNH      UNH       110618000000.0       275483250000.0
JNJ      JNJ        67224000000.0        89659750000.0
XOM      XOM  480681000000.000122    No Data Available
JPM      JPM       108074000000.0       127989000000.0
NVDA    NVDA    No Data Available        20525500000.0
PG        PG        81646000000.0        76618000000.0
V          V        10720000000.0        23395750000.0
CVX      CVX       241909000000.0       129147500000.0

               Revenue Change $          Revenue Change %
A      No Calculation Performed  No Calculation Performed
AAPL             189488250000.0                115.059628
MSFT              99372750000.0                136.257713
AMZN        386460750000.000061                632.577791
TSLA              44204994000.0              10696.757942
GOOGL  No Calculation Performed  No Calculation Performed
GOOG             182327750000.0                 396.02891
UNH              164865250000.0                149.040165
JNJ               22435750000.0                 33.374613
XOM    No Calculation Performed  No Calculation Performed
JPM               19915000000.0                 18.427189
NVDA   No Calculation Performed  No Calculation Performed
PG                -5028000000.0                 -6.158293
V                 12675750000.0                118.243937
CVX             -112761500000.0                -46.613189
(485, 5)
```

The following code block shows that we have 84 of our 485 companies that do not have data in either 2021 or 2012. Some have both that are missing but overall, we can see that 2012 is the bigger culprit of missing revenue data.

In [20]:
```python
print(f"Missing {new_df['Revenue 2012'].value_counts()['No Data Available']} values in 2012")
print(f"Missing {new_df['Revenue 2021'].value_counts()['No Data Available']} values in 2021")
print(f"Missing {new_df['Revenue Change $'].value_counts()['No Calculation Performed']} change history records
```

```
Missing 73 values in 2012
Missing 14 values in 2021
Missing 84 change history records overall
```

In [21]:
```python
index_no_calc = new_df[new_df['Revenue Change $'] == 'No Calculation Performed'].index
new_df.drop(index_no_calc , inplace=True)
new_df
```

Out[21]:

	Ticker	Revenue 2012	Revenue 2021	Revenue Change $	Revenue Change %
AAPL	AAPL	164687000000.0	354175250000.0	189488250000.0	115.059628
MSFT	MSFT	72930000000.0	172302750000.0	99372750000.0	136.257713
AMZN	AMZN	61093000000.0	447553750000.000061	386460750000.000061	632.577791
TSLA	TSLA	413256000.0	44618250000.0	44204994000.0	10696.757942
GOOG	GOOG	46039000000.0	228366750000.0	182327750000.0	396.02891
...	...	...	...	...	...
ALK	ALK	4657000000.0	4455250000.0	-201750000.0	-4.332188
DVA	DVA	8186280000.0	11573402000.0	3387122000.0	41.375594
VNO	VNO	2649217100.0	1524119500.0	-1125097600.0	-42.46906
DISH	DISH	13181334000.0	17680414750.0	4499080750.0	34.132211
RL	RL	6924400000.0	5318200000.0	-1606200000.0	-23.196234

401 rows × 5 columns

In [22]:
```python
# Performing a rank on who grew the most and least, by percentage growth not overall growth
new_df['Growth_Rank'] = new_df['Revenue Change %'].rank(ascending=False)
new_df

# Sorting the rank
rslt_df = new_df.sort_values(by = 'Growth_Rank')
# There are three rows with NaN for 2021 Revenue data to drop
rslt_df = rslt_df.dropna()
```

In [23]:
```python
rslt_df.head(30)
```

Out[23]:

	Ticker	Revenue 2012	Revenue 2021	Revenue Change $	Revenue Change %	Growth_Rank
TSLA	TSLA	413256000.0	44618250000.0	44204994000.0	10696.757942	1.0
FANG	FANG	74962000.0	4948250000.0	4873288000.0	6501.011179	2.0
DXCM	DXCM	99900000.0	2241049950.0	2141149950.0	2143.293243	3.0
NOW	NOW	243712000.0	5357500000.0	5113788000.0	2098.291426	4.0
CNC	CNC	8110000000.0	120285750000.0	112175750000.0	1383.178175	5.0
INCY	INCY	297059000.0	2830670425.0	2533611425.0	852.898389	6.0
REGN	REGN	1378477100.0	12799075000.0	11420597900.0	828.493843	7.0
NFLX	NFLX	3609281800.0	28076899750.0	24467617950.0	677.908218	8.0
EPAM	EPAM	433799000.0	3239886500.0	2806087500.0	646.863524	9.0
AMZN	AMZN	61093000000.0	447553750000.000061	386460750000.000061	632.577791	10.0
CBOE	CBOE	512338000.0	3483075150.0	2970737150.0	579.839315	11.0
CHTR	CHTR	7504000000.0	50411000000.0	42907000000.0	571.78838	12.0
ALGN	ALGN	560041000.0	3499667425.0	2939626425.0	524.89486	13.0
ABMD	ABMD	151794000.0	939890500.0	788096500.0	519.188176	14.0
LEN	LEN	4105132000.0	25099789500.0	20994657500.0	511.424663	15.0
CI	CI	29119000000.0	168514000000.0	139395000000.0	478.70806	16.0
FTNT	FTNT	533639000.0	3026574925.0	2492935925.0	467.157746	17.0
DHI	DHI	4722500000.0	26726099000.0	22003599000.0	465.931159	18.0
SIVB	SIVB	984264100.0	5356748000.0	4372483900.0	444.238889	19.0
ZBRA	ZBRA	996168000.0	5249750000.0	4253582000.0	426.994443	20.0
CSGP	CSGP	349936000.0	1839697325.0	1489761325.0	425.72394	21.0
ENPH	ENPH	216678000.0	1137001250.0	920323250.0	424.74236	22.0
ICE	ICE	1363000000.0	7075250000.0	5712250000.0	419.09391	23.0
MPWR	MPWR	213813000.0	1071375250.0	857562250.0	401.0805	24.0
GOOG	GOOG	46039000000.0	228366750000.0	182327750000.0	396.02891	25.0
TMUS	TMUS	15961590000.0	79039250000.0	63077660000.0	395.184064	26.0
LRCX	LRCX	3168549000.0	15038472000.0	11869923000.0	374.616993	27.0
VRTX	VRTX	1527042000.0	6950734500.0	5423692500.0	355.17638	28.0
MOH	MOH	5914209000.0	24585500000.0	18671291000.0	315.702252	29.0
MCHP	MCHP	1490390000.0	5904900000.0	4414510000.0	296.198311	30.0

In [24]:
```python
rslt_df[rslt_df['Revenue Change %']< 0].count()
```

Out[24]:
```
Ticker              70
Revenue 2012        70
Revenue 2021        70
Revenue Change $    70
Revenue Change %    70
Growth_Rank         70
dtype: int64
```

In [25]:
```python
print(f"Median of Percent Revenue Change is {rslt_df['Revenue Change %'].median()}")
print(f"Mean of Percent Revenue Change is {rslt_df['Revenue Change %'].mean()}")
```

```
Median of Percent Revenue Change is 52.98032445273718
Mean of Percent Revenue Change is 148.36562357985858
```

In [26]:
```python
rslt_df['Label'] = ['Growing Company' if x >= 0
                    else 'Declining Company'
                    for x in rslt_df['Revenue Change %']]
rslt_df['Label'].value_counts()
```

Out[26]:
```
Growing Company      328
Declining Company     70
Name: Label, dtype: int64
```

Below is the first cell of the classification model predicting CEO or CFO.

In [27]:
```python
# Start CEO/CFO Classification Model
main_df.head(15)
main_df = main_df.drop_duplicates()
main_df.loc[main_df['ticker']=='A']
```

Out[27]:

	id	cik	ticker	name	position	year	salary	bonus	stockAwards	optionAw
0	73b3a60ba203743c008330c96b7d8b66	1090872	A	Sam Raha	Senior Vice President, President Diagnostics a...	2021	563500	0	1541332	
1	97393f60cd9f321650e472673daaa70c	1090872	A	Michael R. McMullen	Chief Executive Officer	2021	1280000	0	9165390	
2	83b9cc2bca477fe8ce23e0ab56e70c66	1090872	A	Jacob Thaysen	Senior Vice President, President Life Sciences...	2021	625000	0	1812285	
3	7c60bb804071675ae15ec930f6dea190	1090872	A	Robert McMahon	Senior Vice President, Chief Financial Officer	2021	663500	0	2291271	
4	259b710a8befe67c61a2c3dec14f344a	1090872	A	Padraig McDonnell	Senior Vice President, President Cross-Lab Group	2021	495000	0	1249771	
10	6a9fd7caafd5ff9409e40f8c289e1cd3	1090872	A	Jacob Thaysen	Senior Vice President, President Life Sciences...	2020	614904	0	2117779	
11	ccef820f858971ef639e1266f4b96840	1090872	A	Michael R. McMullen	Chief Executive Officer	2020	1172853	0	11190749	
12	0afb57350eef3b644a4f9802490ce25f	1090872	A	Robert McMahon	Senior Vice President, Chief Financial Officer	2020	638333	0	2614535	
13	5c5621b65ca11ff037e274c80b191cd2	1090872	A	Sam Raha	Senior Vice President, President Diagnostics and	2020	538782	0	1673282	
14	4deeb4d5b673e6f653a86a1eddda5990	1090872	A	Genomics Group	Michael Tang Senior Vice President, General Co...	2020	561026	0	1568688	
15	9637116a014dc78e135a1d65a0831e78	1090872	A	Mark Doak	Senior Vice President, President Agilent Cross...	2020	316538	500000	2614535	
22	e22445c4e02f5347b2e40cbee5789855	1090872	A	Michael R. McMullen	Chief Executive Officer	2019	1220833	0	9560588	
23	ca8648d0544fa3180a6edc2d02a14c82	1090872	A	Robert McMahon	Senior Vice President, Chief Financial Officer	2019	610000	1000000	2221067	
24	56ef5c9c8b7f5e16ed1c7b2fe79b59d6	1090872	A	Genomics Group	Michael Tang Senior Vice President, General Co...	2019	545833	0	1351856	
25	d41442da4b3801c2139f69932b802b1e	1090872	A	Mark Doak	Senior Vice President, President Agilent Cross...	2019	597917	0	2240337	
26	93f8d0fe6316e6a144439bc3c141bc9d	1090872	A	Jacob Thaysen	Senior Vice President, President Life Sciences...	2019	597917	0	1858905	
27	8830ad897377808bae9e8833e0265300	1090872	A	Michael Tang	Senior Vice President, General Counsel	2019	545833	0	1351856	
34	55f648eec48f4dcede4ca6a5994f0481	1090872	A	Jacob Thaysen	Senior Vice President, President Life Sciences...	2018	556250	0	1845713	
35	392901ea2763dd2e201a44dde00f7da9	1090872	A	Robert McMahon	Senior Vice President, Chief Financial Officer	2018	147879	2000000	4131798	
36	99164580887f667c5d56319a065a07e1	1090872	A	Michael R. McMullen	Chief Executive Officer	2018	1168750	0	8476147	
37	93f6834cefea5c3dc174d07a9d617864	1090872	A	Human Resources	Didier Hirsch Former Senior Vice President, Ch...	2018	648333	0	2808278	
38	8da1e032270355f6a9421f9c86c78437	1090872	A	Mark Doak	Senior Vice President, President Cross-Lab Group	2018	570833	0	2088406	
39	a8ee964000ed2eb96c1e9cee384182e8	1090872	A	Dominique Grau	Senior Vice President,	2018	470625	0	1276453	
46	e048daf0786cf01a1b260e2d48fc6b99	1090872	A	Human Resources	Didier Hirsch Former Senior Vice President, Ch...	2017	627500	0	2676250	
47	2a318ae1fb2ac866af17486e49563820	1090872	A	Michael R. McMullen	Chief Executive Officer	2017	1095833	0	7205347	
48	b846615fbe785357bd5a5c48b468696f	1090872	A	Patrick Kaltenbach	Senior Vice President, President Life Sciences...	2017	532083	0	1787948	
49	9d5fef9b9e9e99e038903c1255f80895	1090872	A	Didier Hirsch	Senior Vice President, Chief Financial Officer	2017	627500	0	2676250	
50	8f0626122dd54a0ed7e4dde3d5181b14	1090872	A	Mark Doak	Senior Vice President, President Cross-Lab Group	2017	520833	0	1899068	
51	16f1ec86b31b1ebced16e98aa3b3b71d	1090872	A	Jacob Thaysen	Senior Vice President, President Diagnostics and	2017	495000	0	1338104	
58	a13b28e1a078d7d7abcbaa143b74bc66	1090872	A	Didier Hirsch	Senior Vice President, Chief Financial Officer	2016	600000	0	2747816	
59	afeb892081434400721bc17149de2de4	1090872	A	Jacob Thaysen	Senior Vice President, President Diagnostics and	2016	436667	0	845472	
60	2d37dac047db8fd03cfcacfe73d701a3	1090872	A	Applied Markets Group	Jacob Thaysen Senior Vice President, President...	2016	436667	0	845472	
61	365622904851a1079530120efd7db657	1090872	A	Human Resources	Didier Hirsch Former Senior Vice President, Ch...	2016	600000	0	2747816	
62	6e525b30af9d2f4e44b7c1d67f072833	1090872	A	Michael R. McMullen	Chief Executive Officer	2016	1041667	0	6341255	
63	ece95b49afb2267f963e22254bb3749d	1090872	A	Mark Doak	Senior Vice President, President Cross-Labs Group	2016	470833	0	1638114	
64	733eaae28284cb81dd47193f5edf955f	1090872	A	Patrick Kaltenbach	Senior Vice President, President Life Sciences...	2016	487976	0	1479571	
72	bd787dd65f1458a8a34a4d8eb1edf173	1090872	A	Patrick Kaltenbach	Senior Vice President, President Life Sciences...	2015	407262	0	505138	48
73	88ba9193971115c64b4b338c80ae2369	1090872	A	Didier Hirsch	Senior Vice President, Chief Financial Officer	2015	600000	0	1049012	100
74	73bacd1b846e148680eb4a9becc62344	1090872	A	Chain	Patrick Kaltenbach Senior Vice President, Pres...	2015	407262	0	505138	48
75	1fa34d6f5935e9de3d36e3fa398b8ce3	1090872	A	Michael R. McMullen	President Chief Executive Officer	2015	845833	0	2520205	243
76	8338bde68a45b23cb89207f7f0ca48ad	1090872	A	Mark Doak	Senior Vice President, President Cross-Labs	2015	422917	0	549457	52
77	e96906309e246a7c5bb160c798988614	1090872	A	Group	William P. Sullivan Former President and Chief...	2015	630000	0	3057116	194
84	05a042698e63b1f1d6e5e326c5791dfd	1090872	A	Didier Hirsch	Senior Vice President, Chief Financial Officer	2014	600000	0	1044826	104
85	605f6d5e14e301ffec5c6ba2041d28b4	1090872	A	Marie Oh Huber	Senior Vice President, General Counsel and Sec...	2014	525000	0	1007380	75
86	74d8234da776693b7fb0888c50653825	1090872	A	Michael R. McMullen	President and Chief Operating Officer	2014	606250	0	2332309	99
87	d3af951ba9570439f8e44affe16bc27f	1090872	A	Ronald S. Nersesian	Executive Vice President and Keysight Presiden...	2014	795833	0	2109864	225
88	56fe7f35486e52fd83e1dded8482b048	1090872	A	Group	William P. Sullivan Former President and Chief...	2014	1050000	0	6632830	456
89	a6ad6be0d2312415b499e2961d5f1589	1090872	A	William P. Sullivan	Chief Executive Officer	2014	1050000	0	6632830	456
96	649b6806fbe4a654140ba6e0aa7c2c25	1090872	A	Group	William P. Sullivan Former President and Chief...	2013	1045000	0	3789936	414
97	b4add3c41ed5c86ed22bfce5309856fa	1090872	A	Didier Hirsch	Senior Vice President, Chief Financial Officer	2013	597917	0	869456	95
98	5ce6068d033adac551087700211e1918	1090872	A	Designate, Keysight	Nicolas Roelofs Former Senior Vice President, ...	2013	550000	0	668802	73
99	9bd549134abbd8a364e660c272f6daa7	1090872	A	Michael R. McMullen	President and Chief Operating Officer	2013	575000	0	802571	87
100	4ca37fb051392e805eb9d6436b55eb75	1090872	A	William P. Sullivan	President and Chief Executive Officer	2013	1045000	0	3789936	414
101	494a3f0983f34c1b769175d70f229d69	1090872	A	and Diagnostics	Michael R. McMullen Senior Vice President, Che...	2013	575000	0	802571	87
102	c4d72990c76de80b5f428196ad9686b9	1090872	A	Lars Holmkvist	Former Senior Vice President, Life Sciences	2013	938588	0	891721	
103	8f98917e1baf986ed5115bcf8041b331	1090872	A	Ronald S. Nersesian	Executive Vice President and President and Chi...	2013	741667	0	1560567	170

56 rows × 22 columns

In [28]:
```python
# Need to drop duplicate Agilent ones with the name "Group" or "human resources"
main_df = main_df.drop(main_df[(main_df.name == 'Group') |
                       (main_df.name == 'Human Resources')].index)
# Creating the separate dataframes for CEO/CFO so that we can perform aggregate
# functions separately.
df_CEO = main_df[(main_df['CEO'] == True)]
df_CFO = main_df[(main_df['CFO'] == True)]

# Getting sum of values per ticker.
df_CEO = df_CEO.groupby('ticker').sum()
df_CFO = df_CFO.groupby('ticker').sum()

# Subsetting for the fields we want
df_CEO = df_CEO[['CEO','salary', 'bonus', 'stockAwards',
                 'optionAwards','nonEquityIncentiveCompensation',
                 'total', 'Revenue', 'Net Income Common']]
df_CEO.loc[df_CEO["CEO"] > 0, "CEO"] = True

df_CFO = df_CFO[['CEO','salary', 'bonus', 'stockAwards',
                 'optionAwards','nonEquityIncentiveCompensation',
                 'total', 'Revenue', 'Net Income Common']]
df_CFO.loc[df_CFO["CEO"] == 0, "CEO"] = False
```

In [29]:
```python
# Preparing to join the data frames back together
frames = [df_CEO, df_CFO]

# Joining the two data frames together.
new_main = pd.concat(frames)
new_main = new_main.sort_values(by=['ticker'])


# NOTE: The following commented out code gets percentages of executive pay as part of revenue.
# It is commented out because it is not used in the final classification model but is
# interesting as a potential variable in future projects with this dataset.
# new_main['salary/revenue'] = new_main['salary'] / new_main['Revenue']
# new_main['bonus/revenue'] = new_main['bonus'] / new_main['Revenue']
# new_main['stockAwards/revenue'] = new_main['stockAwards'] / new_main['Revenue']
# new_main['optionAwards/revenue'] = new_main['optionAwards'] / new_main['Revenue']
# new_main['nonEquityIncentiveCompensation/revenue'] = new_main['nonEquityIncentiveCompensation'] / new_main['R
# new_main['all_Equity_Awards'] = new_main['stockAwards'] + new_main['optionAwards']

## Dropping Where CEO is not equal to True or False
indexNumbers = new_main[ (new_main['CEO'] == 3) |
                (new_main['CEO'] == 2) | (new_main['CEO'] == 5)].index
new_main.drop(indexNumbers , inplace=True)

## Converting True/False CEO column from object to boolean because the model
## was not recognizing the y-predictor type.
new_main['CEO'] = new_main['CEO'].astype(bool)
print(new_main.dtypes)
new_main
```

```
CEO                                  bool
salary                              int64
bonus                               int64
stockAwards                         int64
optionAwards                        int64
nonEquityIncentiveCompensation      int64
total                               int64
Revenue                           float64
Net Income Common                 float64
dtype: object
```

Out[29]:

	CEO	salary	bonus	stockAwards	optionAwards	nonEquityIncentiveCompensation	total	Revenue	Net Inco Comm
ticker									
A	True	11458269	0	68552878	15107596	16133895	115188782	5.834650e+10	7.675250e
A	False	5085129	3000000	19646031	2999887	4580051	36893668	4.623975e+10	6.070500e
AAL	True	2441140	0	130227654	0	10203331	159413533	4.842888e+11	1.304125e
AAL	False	7218184	0	38471191	0	11200278	62707472	4.588178e+11	1.355525e
AAP	True	9668318	3485000	44487383	14649081	8633582	83061159	9.050150e+10	4.663842e
...	...	...	...	...	...	...	...	...	...
ZBRA	True	12679100	0	59775901	12967242	19148913	105531711	4.446959e+10	3.304858e
ZION	False	6419692	3652500	5839982	985436	1808345	17642731	2.865551e+10	5.211822e
ZION	True	21152988	11702000	15629354	3350878	7071579	58692242	7.047798e+10	1.245729e
ZTS	True	10643533	0	51539576	20646680	16059793	101710673	5.388575e+10	9.552250e
ZTS	False	7704090	2600000	18872778	7109989	6657597	45597108	8.033900e+10	1.365375e

959 rows × 9 columns

We have now finished preprocessing for our data to run through classification models

In [30]:
```python
new_main.to_csv('new_main_df.csv',index=False)
```

CEO pay changes YOY

This workbook brings in the raw dataset, removes known mistakes, creates DataFrames for CEOs and CFOs only, and creates a series of charts analyzing pay components

In [3]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [4]:
```
starter_df = pd.read_csv('raw_SCT_pay_file.csv')

starter_df.head(3)
```

Out[4]:

	id	cik	ticker	name	position	year	salary	bonus	stockAwards	optionAwards	r
0	73b3a60ba203743c008330c96b7d8b66	1090872	A	Sam Raha	Senior Vice President, President Diagnostics a...	2021	563500	0	1541332	0	
1	97393f60cd9f321650e472673daaa70c	1090872	A	Michael R. McMullen	Chief Executive Officer	2021	1280000	0	9165390	0	
2	83b9cc2bca477fe8ce23e0ab56e70c66	1090872	A	Jacob Thaysen	Senior Vice President, President Life Sciences...	2021	625000	0	1812285	0	

In [5]:
```
#removing tickers with known mistakes
mistakes_df = pd.read_csv('mistakes.csv').drop(columns='Unnamed: 0')
mistakes_df['ticker'].value_counts()

starter_df = starter_df[starter_df['ticker'].isin(set(mistakes_df['ticker'])) == False].reset_index(drop=True)

starter_df.head(3)
```

Out[5]:

	id	cik	ticker	name	position	year	salary	bonus	stockAwards	optionAwards	r
0	73b3a60ba203743c008330c96b7d8b66	1090872	A	Sam Raha	Senior Vice President, President Diagnostics a...	2021	563500	0	1541332	0	
1	97393f60cd9f321650e472673daaa70c	1090872	A	Michael R. McMullen	Chief Executive Officer	2021	1280000	0	9165390	0	
2	83b9cc2bca477fe8ce23e0ab56e70c66	1090872	A	Jacob Thaysen	Senior Vice President, President Life Sciences...	2021	625000	0	1812285	0	

In [6]:
```
#this code not needed but it flags CEOs, CFOs, and Interim appointments
#ceo_title_list = ['CHIEF EXECUTIVE','CEO']
#cfo_title_list = ['CHIEF FINANC','CFO']
#interim_title_list = ['INTERIM','ACTING']
#
#ceo_regstr = '|'.join(ceo_title_list)
#cfo_regstr = '|'.join(cfo_title_list)
#interim_regstr = '|'.join(interim_title_list)
#
## https://stackoverflow.com/questions/49508281/check-if-multiple-substrings-are-in-pandas-dataframe
#starter_df['CEO'] = starter_df['position'].str.upper().str.contains(ceo_regstr)
#starter_df['CFO'] = starter_df['position'].str.upper().str.contains(cfo_regstr)
#starter_df['Interim'] = starter_df['position'].str.upper().str.contains(interim_regstr)
```

Create separate DataFrames for CEOs and CFOs

In [7]:
```
ceo_df = starter_df[(starter_df['CEO']==True)]
ceo_df.reset_index(inplace=True)

ceo_2021_df = starter_df[(starter_df['CEO']==True) & (starter_df['year']==2021)]
ceo_2021_df.reset_index(inplace=True)

cfo_2021_df = starter_df[(starter_df['CFO']==True) & (starter_df['year']==2021)]
cfo_2021_df.reset_index(inplace=True)
```

In [8]:
```
ceo_df = ceo_df[(ceo_df['salary'] > 0) & (ceo_df['year'] > 2005)]
```

In [9]:
```
#https://www.statology.org/deciles-in-python/
percentile_ceo_2021 = np.percentile(ceo_df[ceo_df['year'] == 2021]['salary'], np.arange(0,100,5))
percentile_ceo_2020 = np.percentile(ceo_df[ceo_df['year'] == 2020]['salary'], np.arange(0,100,5))
percentile_ceo_2019 = np.percentile(ceo_df[ceo_df['year'] == 2019]['salary'], np.arange(0,100,5))
percentile_ceo_2018 = np.percentile(ceo_df[ceo_df['year'] == 2018]['salary'], np.arange(0,100,5))
percentile_ceo_2017 = np.percentile(ceo_df[ceo_df['year'] == 2017]['salary'], np.arange(0,100,5))
percentile_ceo_2016 = np.percentile(ceo_df[ceo_df['year'] == 2016]['salary'], np.arange(0,100,5))
percentile_ceo_2015 = np.percentile(ceo_df[ceo_df['year'] == 2015]['salary'], np.arange(0,100,5))
percentile_ceo_2014 = np.percentile(ceo_df[ceo_df['year'] == 2014]['salary'], np.arange(0,100,5))
percentile_ceo_2013 = np.percentile(ceo_df[ceo_df['year'] == 2013]['salary'], np.arange(0,100,5))
percentile_ceo_2012 = np.percentile(ceo_df[ceo_df['year'] == 2012]['salary'], np.arange(0,100,5))

data_list = [percentile_ceo_2012,percentile_ceo_2013,percentile_ceo_2014,percentile_ceo_2015,percentile_ceo_201
         percentile_ceo_2018,percentile_ceo_2019,percentile_ceo_2020,percentile_ceo_2021]
index_list = [2012,2013,2014,2015,2016,2017,2018,2019,2020,2021]

percentile_df = pd.DataFrame(data_list,index=index_list,columns=[0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,
percentile_df
```

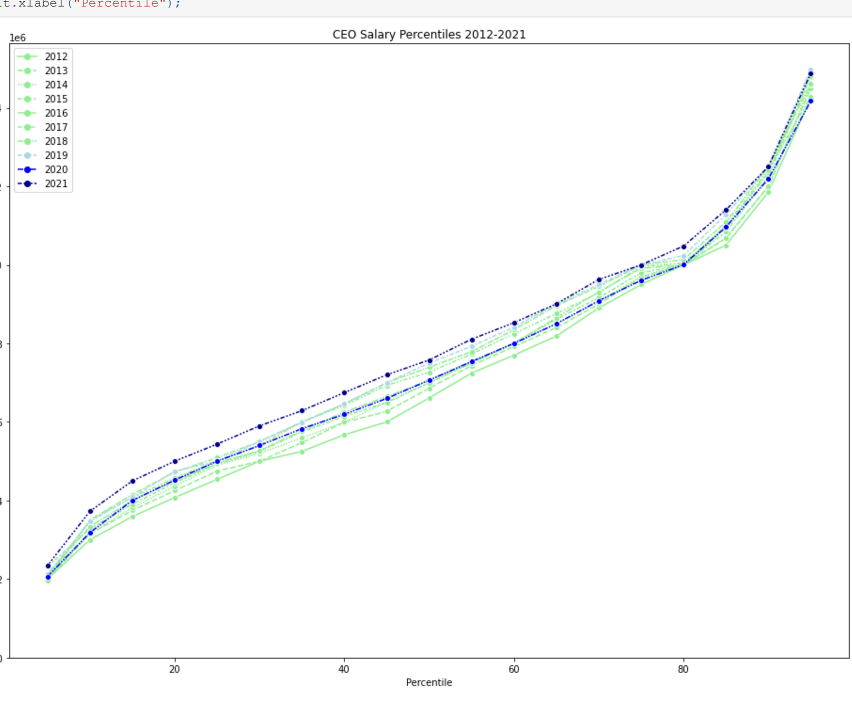
Out[9]:

	0	5	10	15	20	25	30	35	40	45	50	55
2012	1.0	200000.00	300000.0	358992.55	407444.0	453811.25	500000.0	525000.00	567424.8	600000.00	661009.5	724403.05
2013	1.0	200135.75	315000.0	375000.00	425000.0	475000.00	500000.0	547562.00	599039.0	626936.00	685802.0	743442.00
2014	1.0	207711.00	316202.0	382027.00	436539.0	490000.00	520000.0	559259.00	600000.0	650000.00	700000.0	750000.00
2015	1.0	231530.25	327805.0	390538.75	445000.0	493204.00	525000.0	575000.00	614054.0	650019.75	700000.0	750000.00
2016	1.0	213498.75	333800.0	399753.25	450000.0	495000.00	526929.9	577854.25	625000.0	664657.20	705480.5	753548.00
2017	1.0	196319.20	350000.0	407345.95	457569.4	500000.00	540000.0	600000.00	642000.0	693245.30	726789.5	772888.50
2018	1.0	204953.60	350000.0	415230.40	472576.8	508865.00	550000.0	600000.00	645833.0	700000.00	739231.0	778910.20
2019	1.0	212228.00	346211.6	408730.60	474700.2	500000.75	550000.0	599520.20	646538.6	700000.00	750000.0	792589.60
2020	1.0	206250.00	318462.0	400000.00	452083.0	500000.00	540000.0	582950.00	619500.0	659860.00	706410.0	753846.00
2021	1.0	234518.85	372587.1	449922.70	500000.0	543625.75	590074.7	628834.10	674645.6	720000.00	757750.0	810687.40

In [10]:
```
sns.crayon_palette
```

Out[10]: <function seaborn.palettes.crayon_palette(colors)>

In [11]:
```
my_palette = ['lightgreen','lightgreen','lightgreen','lightgreen','lightgreen','lightgreen','lightgreen','light
my_size = [0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5]
my_marker = ['o','o','o','o','o','o','o','o','o','o']

plt.figure(figsize=(16,12))
sns.lineplot(data=percentile_df.drop(columns=[0]).T,
             palette=my_palette,
             markers=my_marker,)
#            sizes=my_size)
plt.title('CEO Salary Percentiles 2012-2021')
plt.ylim(0,)
plt.xlabel("Percentile");
```

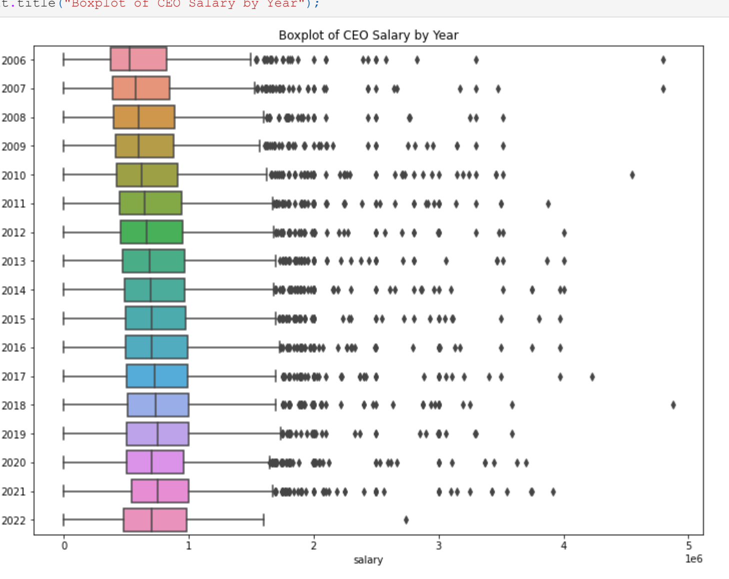


Plots for CEO Pay Components

Salary

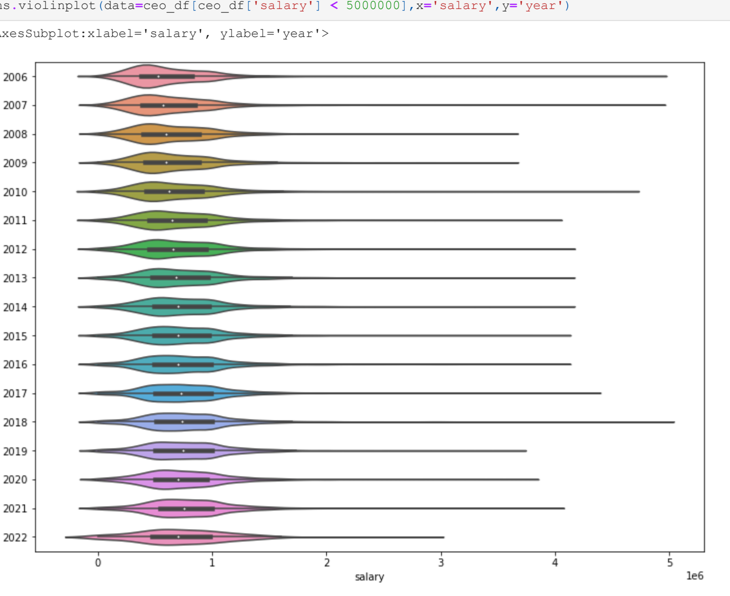
In [12]:
```
ceo_df['year'] = ceo_df['year'].astype('category')
```

In [13]:
```
plt.figure(figsize=(12,9))
sns.boxplot(data=ceo_df[ceo_df['salary'] < 5000000],x='salary',y='year')
plt.title("Boxplot of CEO Salary by Year");
```



In [14]:
```
plt.figure(figsize=(12,9))
sns.violinplot(data=ceo_df[ceo_df['salary'] < 5000000],x='salary',y='year')
```

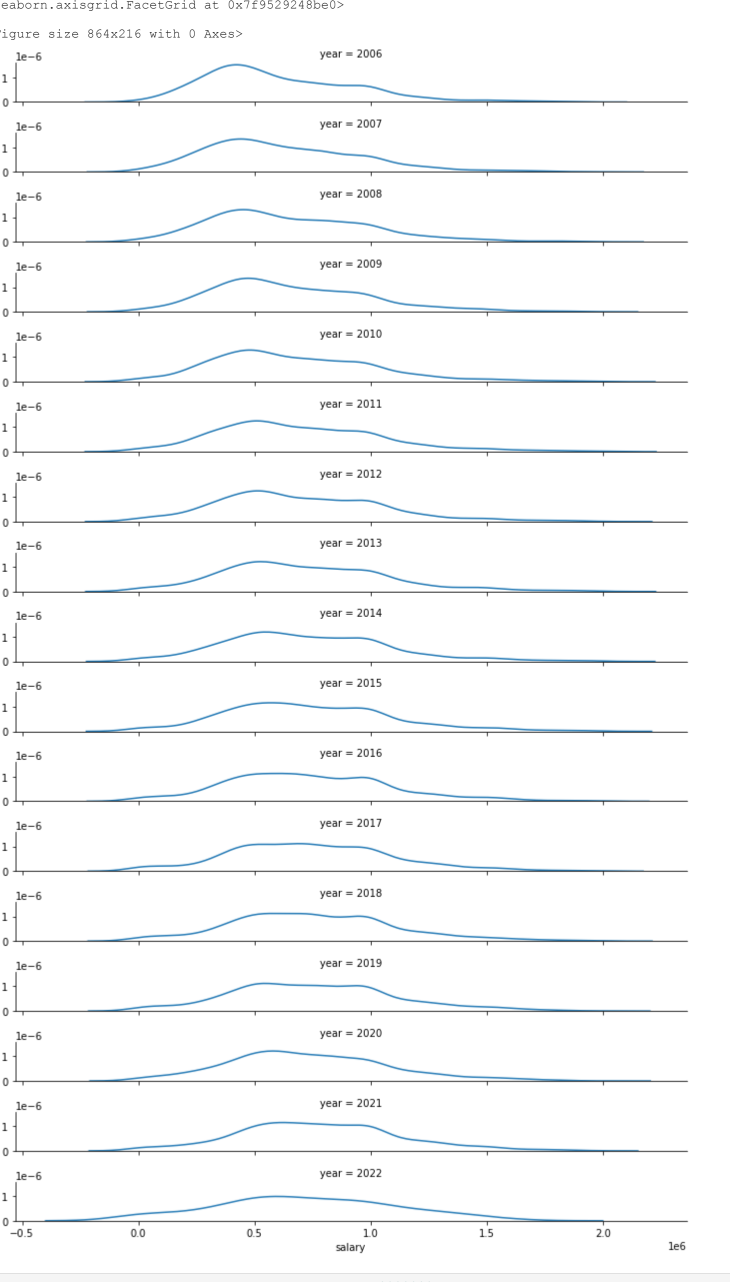
Out[14]: <AxesSubplot:xlabel='salary', ylabel='year'>



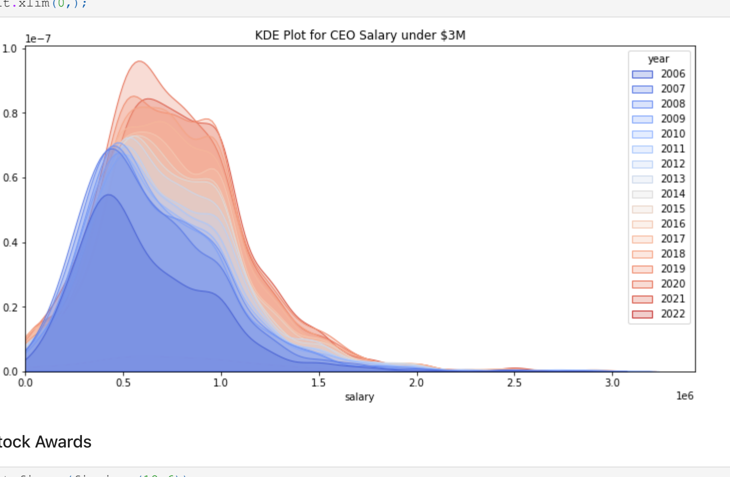
In [15]:
```
plt.figure(figsize=(12,3))
g = sns.FacetGrid(data=ceo_df[ceo_df['salary'] < 2000000], row='year',aspect=10,height=1)
g.map(sns.kdeplot, 'salary')#,bins=50,kde=True,stat='density')
```

Out[15]: <seaborn.axisgrid.FacetGrid at 0x7f9529248be0>

<Figure size 864x216 with 0 Axes>

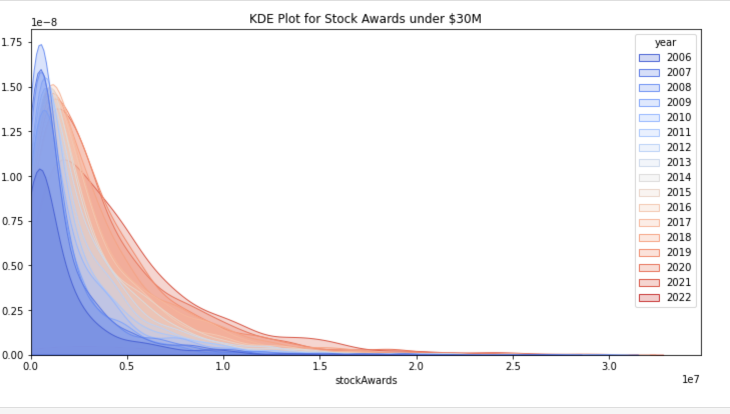


In [16]:
```
#sns.boxplot(data=ceo_df[ceo_df['stockAwards'] < 50000000],x='stockAwards',y='year')

plt.figure(figsize=(12,6))
sns.kdeplot(data=ceo_df[ceo_df['salary'] < 3000000], x='salary',hue='year',palette='coolwarm',shade=True)
plt.title('KDE Plot for CEO Salary under $3M')
plt.xlim(0,);
```

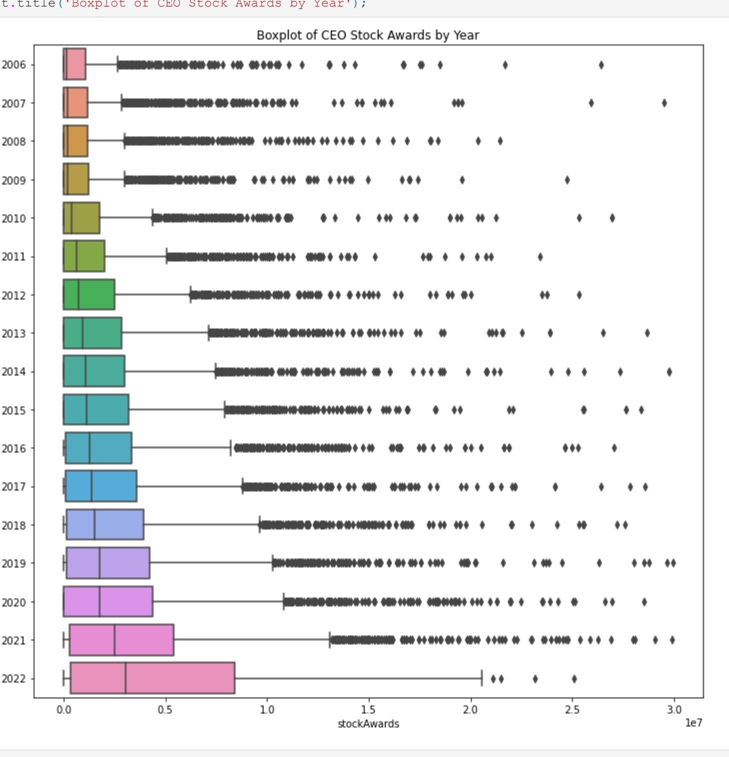


Stock Awards

In [17]:
```
plt.figure(figsize=(12,6))
sns.kdeplot(data=ceo_df[(ceo_df['stockAwards'] < 30000000) & (ceo_df['stockAwards']>0)], x='stockAwards',hue='y
plt.title('KDE Plot for Stock Awards under $30M')
plt.xlim(0,);
```



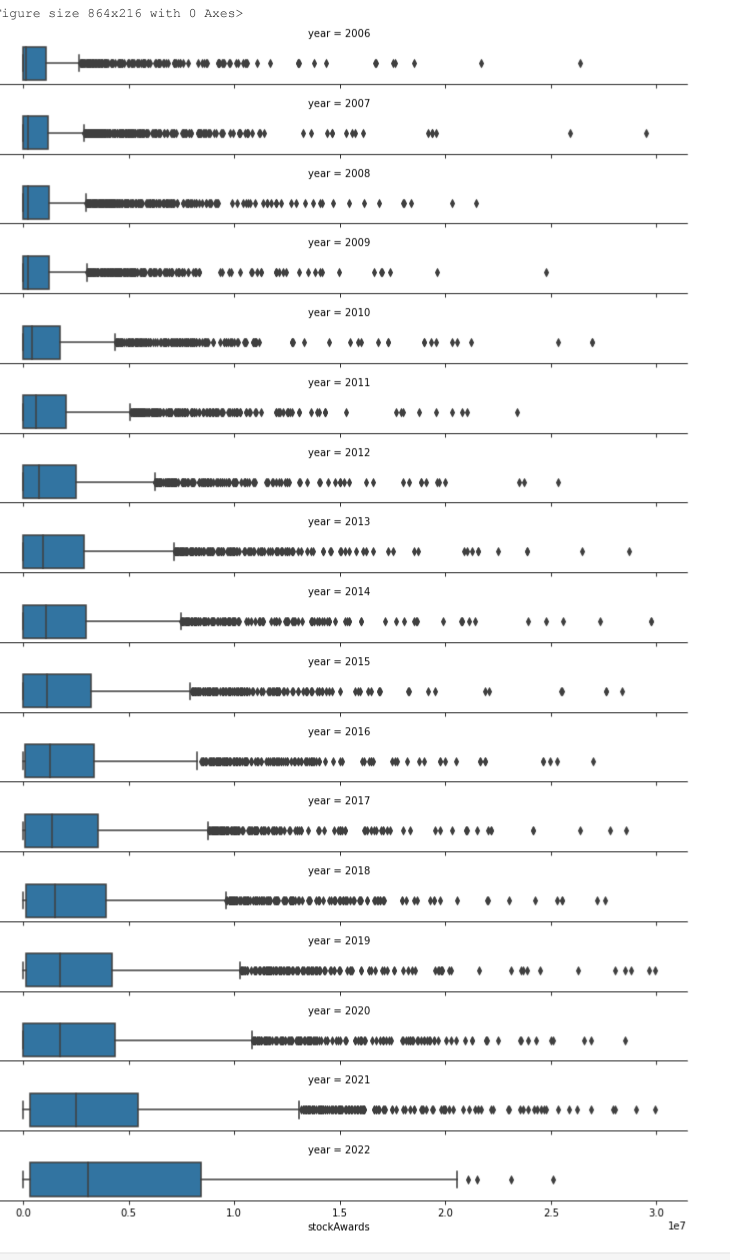
In [18]:
```
plt.figure(figsize=(12,12))
sns.boxplot(data=ceo_df[ceo_df['stockAwards'] < 30000000],x='stockAwards',y='year')
plt.title('Boxplot of CEO Stock Awards by Year');
```



In [19]:
```
plt.figure(figsize=(12,3))
g = sns.FacetGrid(data=ceo_df[ceo_df['stockAwards'] < 30000000], row='year',aspect=10,height=1)
g.map(sns.boxplot, 'stockAwards')
```

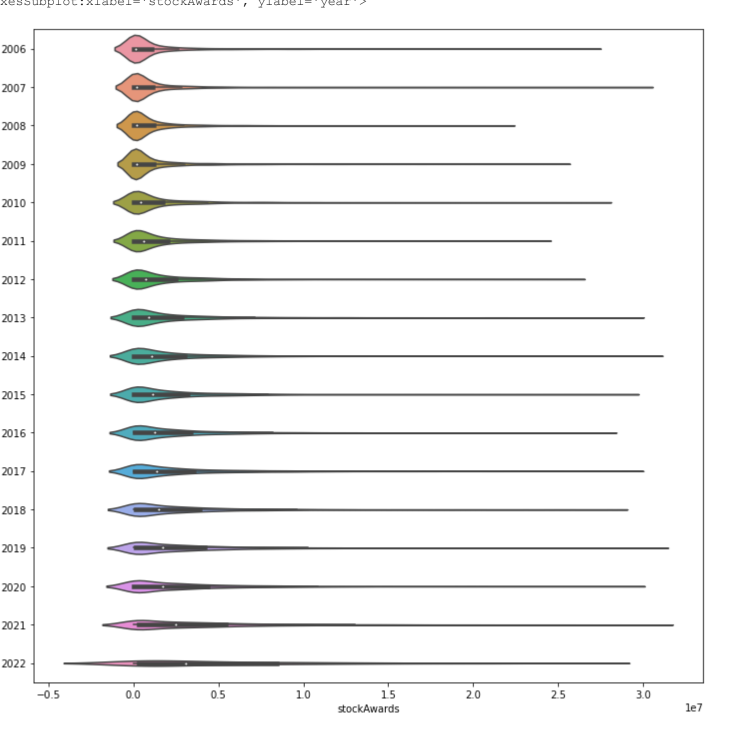
Out[19]: <seaborn.axisgrid.FacetGrid at 0x7f952a30e9b0>

<Figure size 864x216 with 0 Axes>



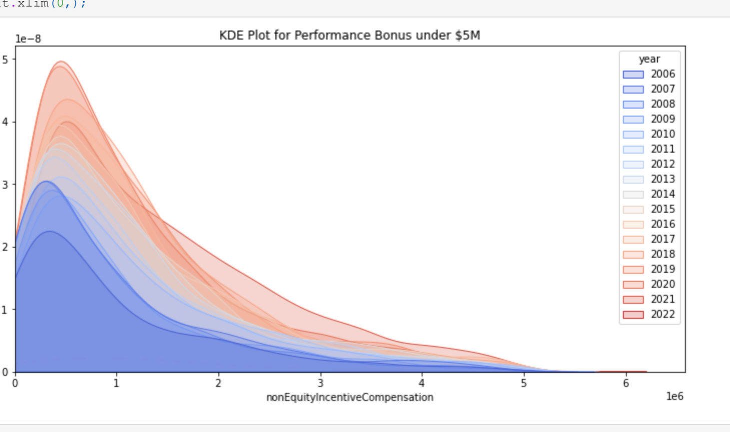
In [20]:
```
plt.figure(figsize=(12,12))
sns.violinplot(data=ceo_df[ceo_df['stockAwards'] < 30000000],x='stockAwards',y='year')
```

Out[20]: <AxesSubplot:xlabel='stockAwards', ylabel='year'>

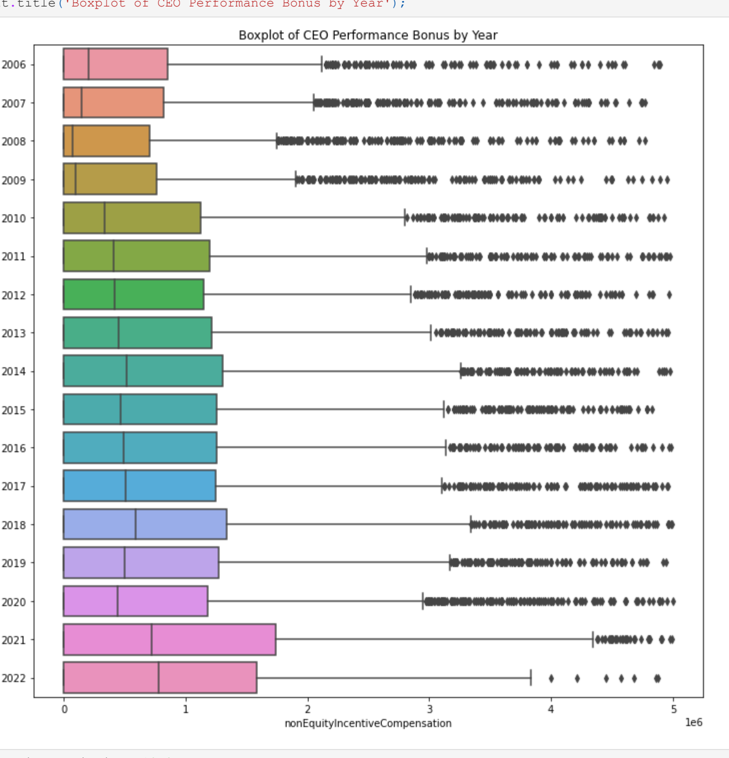


Performance Bonus Awards

In [21]:
```
plt.figure(figsize=(12,6))
sns.kdeplot(data=ceo_df[(ceo_df['nonEquityIncentiveCompensation'] < 5000000) & (ceo_df['nonEquityIncentiveCompe
            x='nonEquityIncentiveCompensation',hue='year',palette='coolwarm',shade=True)
plt.title('KDE Plot for Performance Bonus under $5M')
plt.xlim(0,);
```



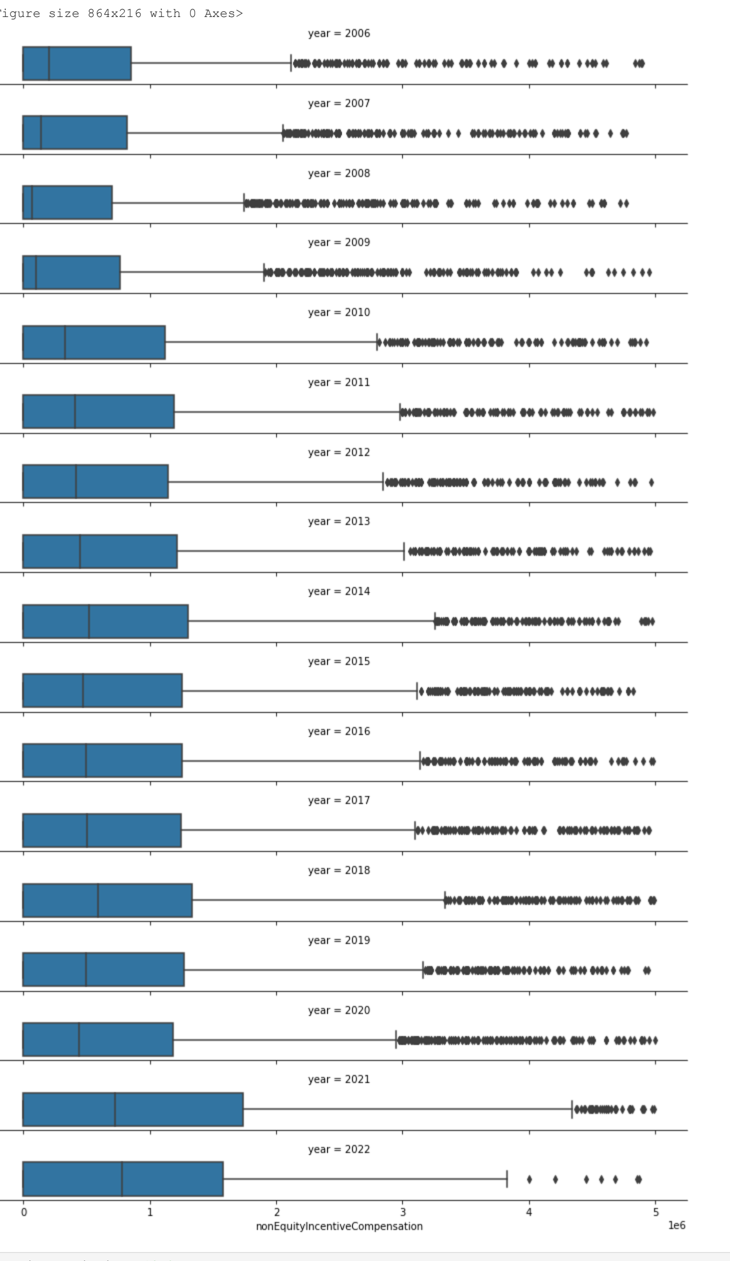
In [22]:
```
plt.figure(figsize=(12,12))
sns.boxplot(data=ceo_df[ceo_df['nonEquityIncentiveCompensation'] < 5000000],x='nonEquityIncentiveCompensation',
plt.title('Boxplot of CEO Performance Bonus by Year');
```



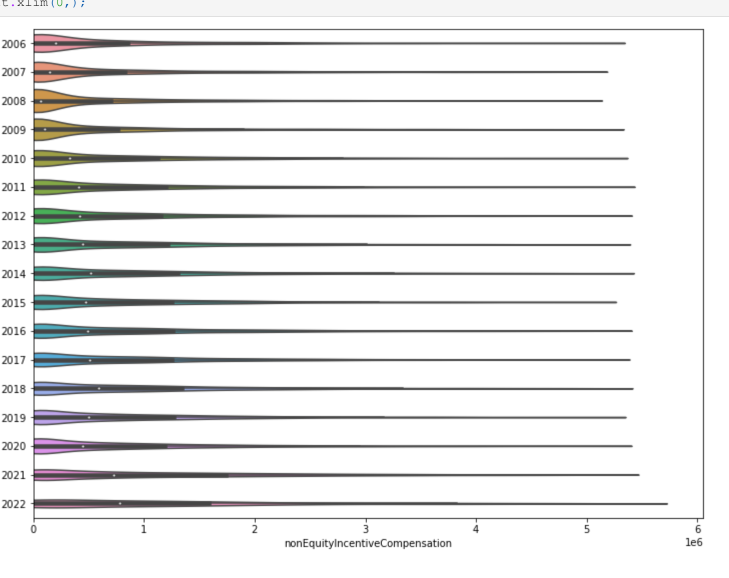
In [23]:
```
plt.figure(figsize=(12,3))
g = sns.FacetGrid(data=ceo_df[ceo_df['nonEquityIncentiveCompensation'] < 5000000], row='year',aspect=10,height=
g.map(sns.boxplot, 'nonEquityIncentiveCompensation')
```

Out[23]: <seaborn.axisgrid.FacetGrid at 0x7f952b7219f0>

<Figure size 864x216 with 0 Axes>

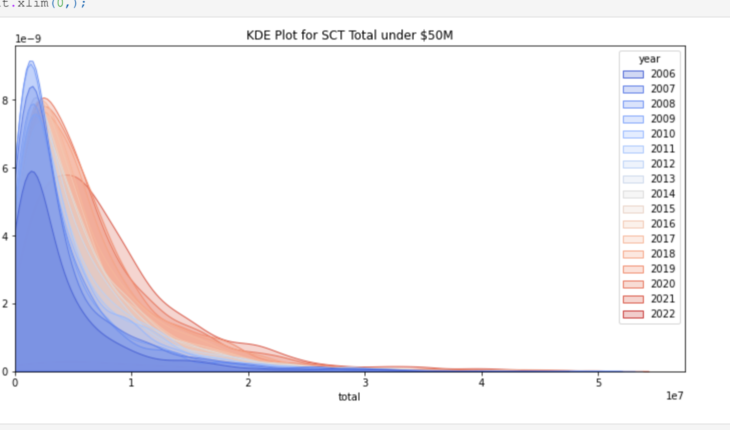


In [24]:
```
plt.figure(figsize=(12,9))
sns.violinplot(data=ceo_df[ceo_df['nonEquityIncentiveCompensation'] < 5000000],x='nonEquityIncentiveCompensatio
plt.xlim(0,);
```

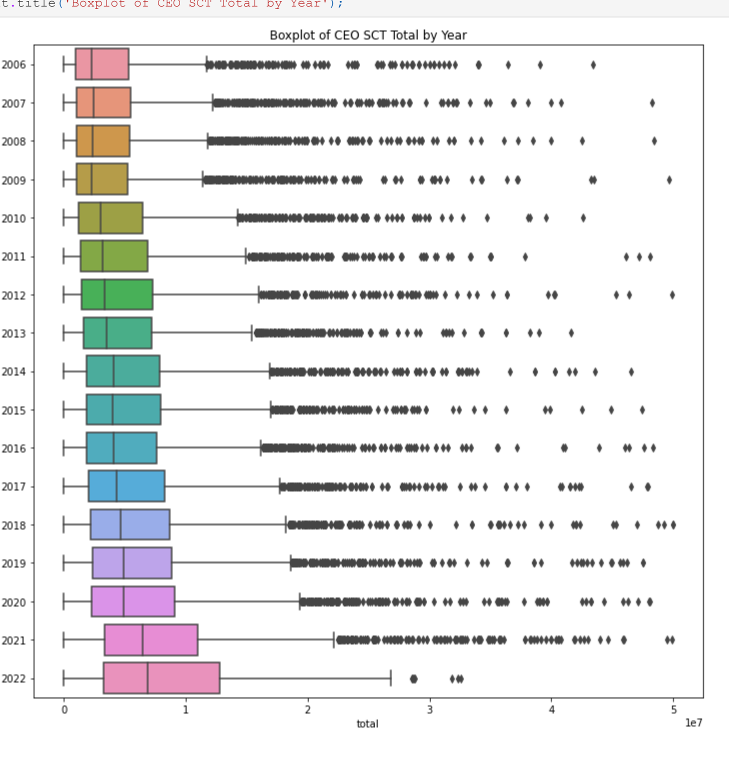


SCT Total

In [25]:
```
plt.figure(figsize=(12,6))
sns.kdeplot(data=ceo_df[(ceo_df['total'] < 50000000) & (ceo_df['total']>0)],
            x='total',hue='year',palette='coolwarm',shade=True)
plt.title('KDE Plot for SCT Total under $50M')
plt.xlim(0,);
```



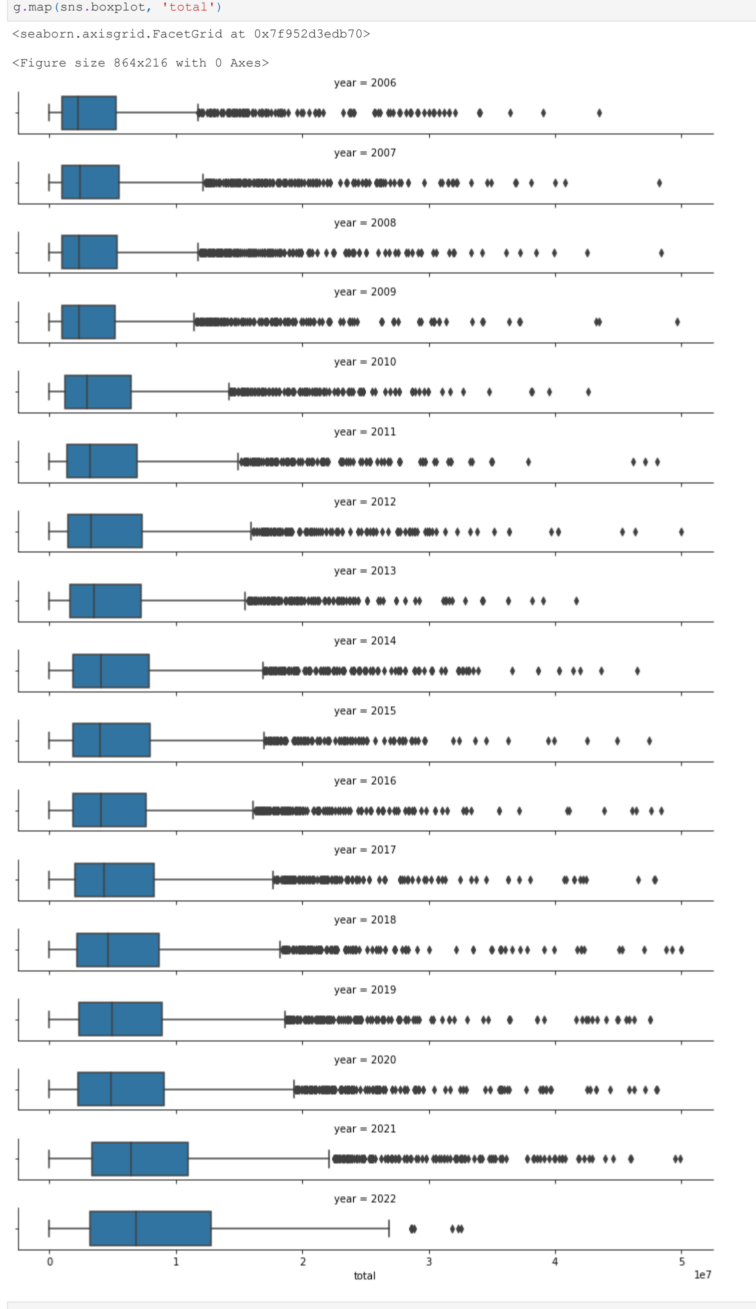
In [26]:
```
plt.figure(figsize=(12,12))
sns.boxplot(data=ceo_df[ceo_df['total'] < 50000000],x='total',y='year')
plt.title('Boxplot of CEO SCT Total by Year');
```



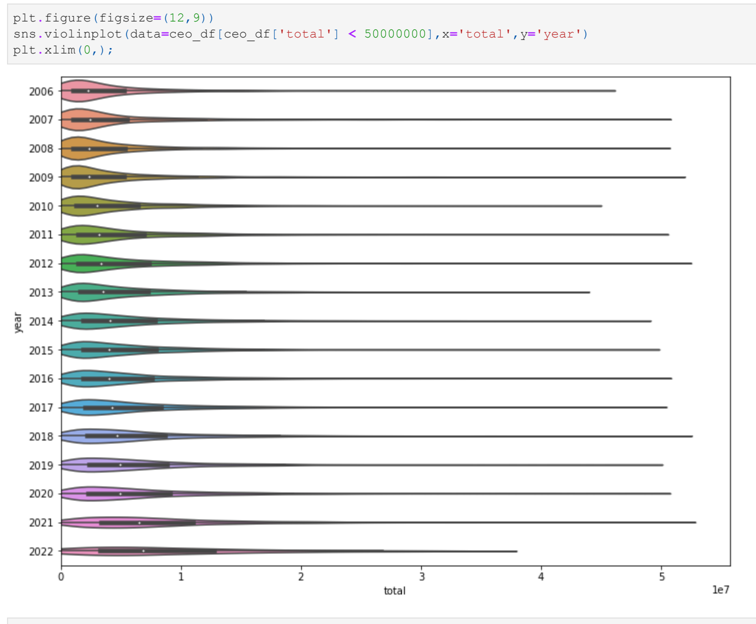
In [27]:
```
plt.figure(figsize=(12,3))
g = sns.FacetGrid(data=ceo_df[ceo_df['total'] < 50000000], row='year',aspect=10,height=1)
g.map(sns.boxplot, 'total')
```

Out[27]: `<seaborn.axisgrid.FacetGrid at 0x7f952d3edb70>`

`<Figure size 864x216 with 0 Axes>`



In [28]:
```
plt.figure(figsize=(12,9))
sns.violinplot(data=ceo_df[ceo_df['total'] < 50000000],x='total',y='year')
plt.xlim(0,);
```



In [29]:
```
print("Count of CEOs by year:")
print(starter_df[(starter_df['CEO'] == True) & (starter_df['Interim'] == False)].groupby('year')['CEO'].sum())
```

```
Count of CEOs by year:
year
2004       1
2005      53
2006    1435
2007    2001
2008    2063
2009    2026
2010    2061
2011    2115
2012    2187
2013    2330
2014    2382
2015    2446
2016    2531
2017    2674
2018    2797
2019    2949
2020    3040
2021    2813
2022     182
Name: CEO, dtype: int64
```

In [30]:
```
ceo_df['salary'].quantile(0.99)
```

Out[30]: `2000000.0`

In [31]:
```
ticker_series = starter_df.groupby('year')['ticker'].nunique()
ceo_series = starter_df[(starter_df['CEO'] == True) & (starter_df['Interim'] == False)].groupby('year')['CEO'].

pd.DataFrame([ticker_series, ceo_series]).T
```

Out[31]:

	ticker	CEO
year		
2004	1	1
2005	49	53
2006	1196	1435
2007	1536	2001
2008	1576	2063
2009	1603	2026
2010	1634	2061
2011	1680	2115
2012	1742	2187
2013	1819	2330
2014	1890	2382
2015	1948	2446
2016	2018	2531
2017	2099	2674
2018	2205	2797
2019	2339	2949
2020	2519	3040
2021	2473	2813
2022	161	182

In [32]:
```
pd.set_option('display.float_format', lambda x: '%.0f' % x)
starter_df.drop(columns=['year','changeInPensionValueAndDeferredEarnings']).describe()
```

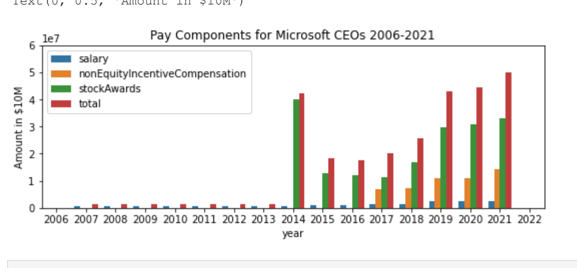
Out[32]:

	cik	salary	bonus	stockAwards	optionAwards	nonEquityIncentiveCompensation	otherCompensation	to
count	172774	172774	172774	172774	172774	172774	172774	1727
mean	907963	481363	931896	1493095	1425116	966832	441096	32924
std	509158	367194	149921671	55942801	184662312	83971598	91501722	170409
min	1750	0	0	0	0	0	0	
25%	723125	288000	0	0	0	0	8560	7856
50%	940942	412946	0	337822	0	180000	26206	16569
75%	1307954	600000	9002	1170596	332156	547858	80606	35326
max	1915657	20000000	52320562950	17980451374	49999911100	18771084337	37361610730	56821463

In [ ]:

In [33]:
```
plt.figure(figsize=(9,3))
sns.barplot(data=pd.melt(ceo_df[ceo_df['ticker']=='MSFT'][['year','salary','nonEquityIncentiveCompensation','st
                    id_vars='year', var_name='component', value_name="amount"),
               x='year',y='amount', hue='component',ci=None)
plt.ylim(0,60000000)
plt.legend(loc='upper left')
plt.title('Pay Components for Microsoft CEOs 2006-2021')
plt.ylabel('Amount in $10M')
```

Out[33]: `Text(0, 0.5, 'Amount in $10M')`



In [34]:
```
pd.melt(ceo_df[ceo_df['ticker']=='F'][['year','salary','nonEquityIncentiveCompensation','stockAwards','total']]
                    id_vars='year', var_name='component', value_name="amount")
```

Out[34]:

	year	component	amount
0	2021	salary	1700000
1	2020	salary	1425000
2	2020	salary	1800000
3	2019	salary	1800000
4	2019	salary	1800000
...	...	...	...
91	2008	total	13565378
92	2007	total	21670674
93	2007	total	8677747
94	2006	total	28183476
95	2006	total	2747100

96 rows × 3 columns

In [ ]:

```
In [85]:
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import confusion_matrix, classification_report
```

```
In [87]:
new_main = pd.read_csv('./datasets/new_main_df.csv')

new_main = new_main[['CEO', 'salary', 'stockAwards', 'optionAwards', 'nonEquityIncentiveCompensation',
                     'Revenue', 'Net Income Common','all_Equity_Awards']]

new_main['Revenue'] = np.log(new_main['Revenue'])

new_main.head()
```

Out[87]:

	CEO	salary	stockAwards	optionAwards	nonEquityIncentiveCompensation	Revenue	Net Income Common	all_Equity_Awards
0	True	11458269	68552878	15107596	16133895	24.789665	7.675250e+09	83660474
1	False	5085129	19646031	2999887	4580051	24.557106	6.070500e+09	22645918
2	True	2441140	130227654	0	10203331	26.905947	1.304125e+10	130227654
3	False	7218184	38471191	0	11200278	26.851919	1.355525e+10	38471191
4	True	9668318	44487383	14649081	8633582	25.228632	4.663842e+09	59136464

Given that we have a small data set here, I am going to go off the guidance of the book in partitioning. "On the other hand, for smaller or less complex data sets, one should retain sufficient records for accurate assessment, so that the training sets would contain only 50–67% of the original data." I think we should sufficient records to train on though, so I posit that we do a 60% training, 40% testing split (if you think we should do otherwise, by all means I am all ears).

```
In [88]:
#Partitioning the data
fin_train, fin_test = train_test_split(new_main,
                                       test_size = 0.3, random_state = 7)

# Testing to see if we have partitioned correctly.
print("Shape of entire dataset: ",new_main.shape,"\n")
print("Value counts of CEO vs CFO:\n",new_main['CEO'].value_counts(),"\n")
print("Shape of training dataset: ",fin_train.shape,"\n")
print("Shape of testing dataset: ",fin_test.shape)
```

```
Shape of entire dataset:  (959, 8)

Value counts of CEO vs CFO:
 True     497
False    462
Name: CEO, dtype: int64

Shape of training dataset:  (671, 8)

Shape of testing dataset:  (288, 8)
```

Baseline model: We call the "positive" class "True" for CEO and see that a baseline model would predict correctly with (497/959) = 52% accuracy. If our model beats 52% accuracy, then it beats the baseline and is considered useful. I am not going to choose the X variables as proportions of revenue, although I can see there being multicollinearity concerns with this approach if we are incorporating the Y variable into the X variable in this way. I am curious to hear your thoughts on this. Note that I am not taking the "total comp" as this would definitely be a multicollinearity concern if we are using the other 5 variables.

Setting up train/test datasets and scaling

```
In [89]:
#setting up X and y dataframes
y_train = fin_train[['CEO']]
X_train = fin_train.drop(columns='CEO')

# Setting up the variables from the test dataset.
y_test = fin_test['CEO']
X_test = fin_test.drop(columns='CEO')

######################
#scale the data
sc = StandardScaler()
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.transform(X_test)
######################
```

CART Decision Tree Model

```
In [90]:
#instantiate and fit DT classifier
cart01 = DecisionTreeClassifier(criterion = "gini", max_leaf_nodes=5).fit(X_train_sc,y_train)

#make predictions with training dataset
y_predCart_train = cart01.predict(X_train_sc)

print("Confusion matrix for training data:")
print(confusion_matrix(y_true=y_train, y_pred=y_predCart_train))
print(classification_report(y_true=y_train, y_pred=y_predCart_train))

#make predictions with test dataset
y_predCart_test = cart01.predict(X_test_sc)

print("\nConfusion matrix for testing data:")
print(confusion_matrix(y_true=y_test, y_pred=y_predCart_test))
print(classification_report(y_true=y_test, y_pred=y_predCart_test))
```

```
Confusion matrix for training data:
[[292  31]
 [ 57 291]]
              precision    recall  f1-score   support

       False       0.84      0.90      0.87       323
        True       0.90      0.84      0.87       348

    accuracy                           0.87       671
   macro avg       0.87      0.87      0.87       671
weighted avg       0.87      0.87      0.87       671


Confusion matrix for testing data:
[[121  18]
 [ 33 116]]
              precision    recall  f1-score   support

       False       0.79      0.87      0.83       139
        True       0.87      0.78      0.82       149

    accuracy                           0.82       288
   macro avg       0.83      0.82      0.82       288
weighted avg       0.83      0.82      0.82       288
```

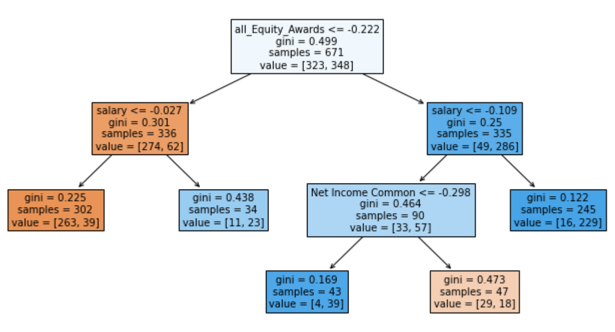
Setting up a function which creates a confusion matrix based on model and train/test datasets

```
In [91]:
def predictor_function(model, X_train, X_test, y_train, y_test):

    #make predictions with training dataset
    y_pred_train = model.predict(X_train)

    print("Confusion matrix for training data:")
    print(confusion_matrix(y_true=y_train, y_pred=y_pred_train))
    print(classification_report(y_true=y_train, y_pred=y_pred_train))

    #make predictions with test dataset
    y_pred_test = model.predict(X_test)

    print("\nConfusion matrix for testing data:")
    print(confusion_matrix(y_true=y_test, y_pred=y_pred_test))
    print(classification_report(y_true=y_test, y_pred=y_pred_test))
```

```
In [92]:
predictor_function(cart01,X_train_sc, X_test_sc, y_train, y_test)
```

```
Confusion matrix for training data:
[[292  31]
 [ 57 291]]
              precision    recall  f1-score   support

       False       0.84      0.90      0.87       323
        True       0.90      0.84      0.87       348

    accuracy                           0.87       671
   macro avg       0.87      0.87      0.87       671
weighted avg       0.87      0.87      0.87       671


Confusion matrix for testing data:
[[121  18]
 [ 33 116]]
              precision    recall  f1-score   support

       False       0.79      0.87      0.83       139
        True       0.87      0.78      0.82       149

    accuracy                           0.82       288
   macro avg       0.83      0.82      0.82       288
weighted avg       0.83      0.82      0.82       288
```

```
In [93]:
plt.figure(figsize=(12,6))
plot_tree(cart01, filled=True, feature_names=X_train.columns);
```

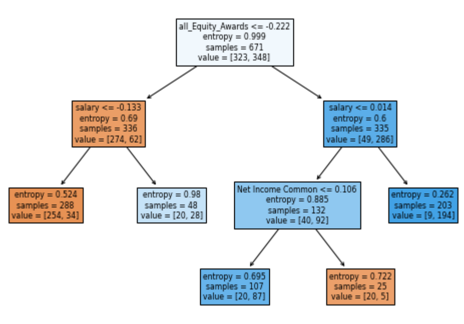


C5.0 Decision Tree Model

```
In [94]:
#C5.0 Model
c50_01 = DecisionTreeClassifier(criterion="entropy", max_leaf_nodes=5).fit(X_train_sc,y_train)

predictor_function(c50_01,X_train_sc, X_test_sc, y_train, y_test)
```

```
Confusion matrix for training data:
[[274  49]
 [ 39 309]]
              precision    recall  f1-score   support

       False       0.88      0.85      0.86       323
        True       0.86      0.89      0.88       348

    accuracy                           0.87       671
   macro avg       0.87      0.87      0.87       671
weighted avg       0.87      0.87      0.87       671


Confusion matrix for testing data:
[[113  26]
 [ 31 118]]
              precision    recall  f1-score   support

       False       0.78      0.81      0.80       139
        True       0.82      0.79      0.81       149

    accuracy                           0.80       288
   macro avg       0.80      0.80      0.80       288
weighted avg       0.80      0.80      0.80       288
```

```
In [95]:
plt.figure(figsize=(9,6))
plot_tree(c50_01, filled=True, feature_names=X_train.columns);
```



Random Forest Model

```
In [96]:
#Random Forest Model
rfy = np.ravel(y_train)
rf01 = RandomForestClassifier(n_estimators = 100, criterion="gini",max_depth=5)
rf01.fit(X_train_sc,rfy)

predictor_function(rf01,X_train_sc, X_test_sc, y_train, y_test)
```

```
Confusion matrix for training data:
[[314   9]
 [ 28 320]]
              precision    recall  f1-score   support

       False       0.92      0.97      0.94       323
        True       0.97      0.92      0.95       348

    accuracy                           0.94       671
   macro avg       0.95      0.95      0.94       671
weighted avg       0.95      0.94      0.94       671


Confusion matrix for testing data:
[[119  20]
 [ 25 124]]
              precision    recall  f1-score   support

       False       0.83      0.86      0.84       139
        True       0.86      0.83      0.85       149

    accuracy                           0.84       288
   macro avg       0.84      0.84      0.84       288
weighted avg       0.84      0.84      0.84       288
```

```
In [97]:
rf01.decision_path(X_train_sc)
```

```
Out[97]:
(<671x4852 sparse matrix of type '<class 'numpy.int64'>'
 	with 389344 stored elements in Compressed Sparse Row format>,
 array([   0,   51,   98,  153,  202,  251,  302,  349,  398,  447,  498,
         537,  588,  625,  672,  715,  764,  813,  862,  909,  956, 1009,
        1062, 1109, 1162, 1213, 1254, 1293, 1336, 1383, 1420, 1465, 1518,
        1565, 1618, 1667, 1720, 1773, 1824, 1867, 1918, 1977, 2028, 2071,
        2120, 2173, 2218, 2259, 2310, 2365, 2424, 2473, 2526, 2573, 2610,
        2653, 2704, 2753, 2800, 2847, 2898, 2945, 2994, 3037, 3088, 3137,
        3186, 3237, 3298, 3349, 3400, 3445, 3492, 3541, 3592, 3641, 3682,
        3737, 3786, 3839, 3888, 3939, 3988, 4041, 4096, 4139, 4186, 4239,
        4282, 4329, 4372, 4425, 4478, 4529, 4570, 4619, 4664, 4717, 4762,
        4809, 4852]))
```

```
In [98]:
rf01.get_params()
```

```
Out[98]:
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': 5,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

Logistic Regression Model (default settings)

```
In [99]:
lry = np.ravel(y_train)
logreg01 = LogisticRegression()
logreg01.fit(X_train_sc,lry)

predictor_function(logreg01,X_train_sc, X_test_sc, y_train, y_test)
```

```
Confusion matrix for training data:
[[300  23]
 [ 46 302]]
              precision    recall  f1-score   support

       False       0.87      0.93      0.90       323
        True       0.93      0.87      0.90       348

    accuracy                           0.90       671
   macro avg       0.90      0.90      0.90       671
weighted avg       0.90      0.90      0.90       671


Confusion matrix for testing data:
[[127  12]
 [ 20 129]]
              precision    recall  f1-score   support

       False       0.86      0.91      0.89       139
        True       0.91      0.87      0.89       149

    accuracy                           0.89       288
   macro avg       0.89      0.89      0.89       288
weighted avg       0.89      0.89      0.89       288
```

Naive Bayes Model (default settings)

https://scikit-learn.org/stable/modules/naive_bayes.html

```
In [100…
gnby = np.ravel(y_train)

gnb = GaussianNB()
gnb.fit(X_train_sc,gnby)

predictor_function(cart01,X_train_sc, X_test_sc, y_train, y_test)
```

```
Confusion matrix for training data:
[[292  31]
 [ 57 291]]
              precision    recall  f1-score   support

       False       0.84      0.90      0.87       323
        True       0.90      0.84      0.87       348

    accuracy                           0.87       671
   macro avg       0.87      0.87      0.87       671
weighted avg       0.87      0.87      0.87       671


Confusion matrix for testing data:
[[121  18]
 [ 33 116]]
              precision    recall  f1-score   support

       False       0.79      0.87      0.83       139
        True       0.87      0.78      0.82       149

    accuracy                           0.82       288
   macro avg       0.83      0.82      0.82       288
weighted avg       0.83      0.82      0.82       288
```

KNeighbors Model

Notice how changing the weights from "uniform" to "distance" results in a *very* overfit modeL

```
In [101…
from sklearn.neighbors import KNeighborsClassifier
```

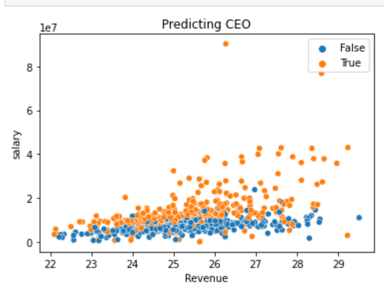
```
In [102…
knn = KNeighborsClassifier(n_neighbors=5, weights='uniform')

knny = np.ravel(y_train)
knn.fit(X_train_sc,knny)

predictor_function(knn,X_train_sc, X_test_sc, y_train, y_test)
```

```
Confusion matrix for training data:
[[306  17]
 [ 44 304]]
              precision    recall  f1-score   support

       False       0.87      0.95      0.91       323
        True       0.95      0.87      0.91       348

    accuracy                           0.91       671
   macro avg       0.91      0.91      0.91       671
weighted avg       0.91      0.91      0.91       671


Confusion matrix for testing data:
[[125  14]
 [ 24 125]]
              precision    recall  f1-score   support

       False       0.84      0.90      0.87       139
        True       0.90      0.84      0.87       149

    accuracy                           0.87       288
   macro avg       0.87      0.87      0.87       288
weighted avg       0.87      0.87      0.87       288
```

```
In [103…
knn = KNeighborsClassifier(n_neighbors=5, weights='distance')

knny = np.ravel(y_train)
knn.fit(X_train_sc,knny)

predictor_function(knn,X_train_sc, X_test_sc, y_train, y_test)
```

```
Confusion matrix for training data:
[[323   0]
 [  0 348]]
              precision    recall  f1-score   support

       False       1.00      1.00      1.00       323
        True       1.00      1.00      1.00       348

    accuracy                           1.00       671
   macro avg       1.00      1.00      1.00       671
weighted avg       1.00      1.00      1.00       671


Confusion matrix for testing data:
[[125  14]
 [ 24 125]]
              precision    recall  f1-score   support

       False       0.84      0.90      0.87       139
        True       0.90      0.84      0.87       149

    accuracy                           0.87       288
   macro avg       0.87      0.87      0.87       288
weighted avg       0.87      0.87      0.87       288
```

```
In [104…
sns.scatterplot(data=X_train, x='Revenue',y='salary', hue=knn.predict(X_train_sc))
plt.title('Predicting CEO');
```



Suppor Vector Machine

```
In [105…
from sklearn.svm import SVC
```

```
In [106…
svcy = np.ravel(y_train)

svc = SVC()
svc.fit(X_train_sc,svcy)

predictor_function(svc,X_train_sc, X_test_sc, y_train, y_test)
```

```
Confusion matrix for training data:
[[303  20]
 [ 44 304]]
              precision    recall  f1-score   support

       False       0.87      0.94      0.90       323
        True       0.94      0.87      0.90       348

    accuracy                           0.90       671
   macro avg       0.91      0.91      0.90       671
weighted avg       0.91      0.90      0.90       671


Confusion matrix for testing data:
[[128  11]
 [ 25 124]]
              precision    recall  f1-score   support

       False       0.84      0.92      0.88       139
        True       0.92      0.83      0.87       149

    accuracy                           0.88       288
   macro avg       0.88      0.88      0.87       288
weighted avg       0.88      0.88      0.87       288
```

```
In [107…
svc.get_params()
```

```
Out[107]:
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

```
In [108…
svc.support_vectors_
```

```
Out[108]:
array([[ 0.50443643,  0.59707948, -0.19249866, ...,  2.26253538,
         3.96678542,  0.13051731],
       [-0.53998382, -0.60718545, -0.15549085, ..., -0.69358501,
        -0.48241272, -0.43420506],
       [-0.32642185, -0.3594091 , -0.13628877, ..., -0.22029892,
        -0.35876796, -0.29499724],
       ...,
       [ 3.7341026 ,  0.82377441,  0.70739544, ...,  1.22142338,
         0.68566547,  1.01518815],
       [ 4.07664129,  2.56042895, -0.19249866, ...,  1.24806762,
         1.74842383,  1.10298719],
       [ 0.06781806, -0.72883012, -0.19249866, ..., -0.49593355,
        -0.22353882, -0.52622117]])
```