in this coursework, we performed forllowing task

- perform the exploratory data analysis
  - summary statistics(max,min, count for catrgorical variables)
  - correlation matrix between variables
  - visualization for distribution
  - winsorize the independent variable
- here we also conduct some text analysis
  - text cleaning
  - tokenization, lemmatization
  - find most common n-grams

but as the work for text analyisi is too large so that we are still developing them

Here we really did a lot of work in below session, here we firsly answer the questions:

- data Exploration [2 points]: Extract and present the following information:
- how many quantitative and qualitative variables you have in the data.
  - all 70+ financial ratios are numeric so that they are quantitative
  - So far there is no quantitave data

Feature Engineering [2 points]:

- decide and present which variables need cleaning (any out of range values, outliers)
  - as we are using really complete database, there are only few missing values.
  - to deal with potential outliers, we will winsorize the data at 1% and 99% level
- state how you handle missing cases (e.g., eliminate, impute)
  - there are no more than 1% missing data, we will eliminate the data with missing value
- state if you conduct any feature selection (e.g., using a subset the columns/variables)
  - here we will use either PCA or LASSO for feature selection

for questions not answered above, they are code related and will be presented in following parts

In [2]:

```
import pandas as pd
import numpy as np
from os.path import join
import statsmodels.api as sm
import seaborn as sns
import warnings
from sec_api import QueryApi
from sec_api import ExtractorApi
import matplotlib.pyplot as plt


warnings.filterwarnings("ignore")
RAW_DATA_PATH = 'C:/Users/Lenovo/OneDrive - The Chinese University of Hong Kong/lrl_study
```

In [7]:

```
# import s&p rating data
df_SP_rating = pd.read_csv(join(RAW_DATA_PATH, 'rating_2010_2011_sp1.csv'))
# eliminate forecasts with missing announcement dates, gvkey and ranking

df_SP_rating = df_SP_rating.dropna(subset=['gvkey','splticrm','datadate','cik'], how='an

df_SP_rating[['year']]=pd.to_datetime(df_SP_rating['datadate']).dt.year
df_SP_rating=df_SP_rating.drop_duplicates(['gvkey', 'year'], keep='last')#here we keep t
df_SP_rating=df_SP_rating.drop(['cusip','tic','spsdrm'],axis=1)
df_SP_rating.info()
df_SP_rating.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2042 entries, 23 to 137006
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   gvkey      2042 non-null   int64
 1   splticrm   2042 non-null   object
 2   datadate   2042 non-null   object
 3   cik        2042 non-null   float64
 4   conm       2041 non-null   object
 5   year       2042 non-null   int64
dtypes: float64(1), int64(2), object(3)
memory usage: 111.7+ KB
```

Out[7]:

	gvkey	splticrm	datadate	cik	conm	year
23	1004	BB	2010-12-31	1750.0	AAR CORP	2010
26	1010	B+	2010-03-31	910627.0	ACF INDUSTRIES INC	2010
74	1045	B-	2010-12-31	6201.0	AMERICAN AIRLINES GROUP INC	2010
77	1048	A-	2010-03-31	65695.0	ANR PIPELINE CO	2010
125	1075	BBB-	2010-12-31	764622.0	PINNACLE WEST CAPITAL CORP	2010

In [8]:

```
#import financial ratio data
# import s&p rating data
df_financial_ratio = pd.read_csv(join(RAW_DATA_PATH, 'financial ratio.csv'))
# eliminate forecasts with missing announcement dates, gvkey and ranking
df_financial_ratio = df_financial_ratio.dropna(subset=['gvkey','public_date'], how='any'
#creat a variable called year for later merging data
df_financial_ratio[['year']]=pd.to_datetime(df_financial_ratio['public_date']).dt.year
df_financial_ratio=df_financial_ratio.drop_duplicates(['gvkey', 'year'], keep='last')
df_financial_ratio=df_financial_ratio.drop(['cusip','TICKER','permno'],axis=1)#drop some
df_financial_ratio.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8036 entries, 11 to 51274
Data columns (total 76 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   gvkey            8036 non-null   int64
 1   adate            8014 non-null   object
 2   qdate            8036 non-null   object
 3   public_date      8036 non-null   object
 4   CAPEI            7873 non-null   float64
 5   bm               7774 non-null   float64
 6   evm              7993 non-null   float64
 7   pe_op_basic      7709 non-null   float64
 8   pe_op_dil        7713 non-null   float64
 9   pe_exi           7730 non-null   float64
 10  pe_inc           7730 non-null   float64
 11  ps               7901 non-null   float64
 12  pcf              8006 non-null   float64
 13  dpr              5639 non-null   float64
 14  npm              7900 non-null   float64
 15  opmbd            7900 non-null   float64
 16  opmad            7900 non-null   float64
 17  gpm              7890 non-null   float64
 18  ptpm             7900 non-null   float64
 19  cfm              7880 non-null   float64
 20  roa              8008 non-null   float64
 21  roe              7726 non-null   float64
 22  roce             7971 non-null   float64
 23  efftax           4920 non-null   float64
 24  aftret_eq        8008 non-null   float64
 25  aftret_invcapx   7872 non-null   float64
 26  aftret_equity    8008 non-null   float64
 27  pretret_noa      6475 non-null   float64
 28  pretret_earnat   6475 non-null   float64
 29  GProf            8013 non-null   float64
 30  equity_invcap    7957 non-null   float64
 31  debt_invcap      7929 non-null   float64
 32  totdebt_invcap   7917 non-null   float64
 33  capital_ratio    7989 non-null   float64
 34  int_debt         5272 non-null   float64
 35  int_totdebt      5650 non-null   float64
 36  cash_lt          8015 non-null   float64
 37  invt_act         6428 non-null   float64
 38  rect_act         6469 non-null   float64
 39  debt_at          7977 non-null   float64
 40  debt_ebitda      7955 non-null   float64
 41  short_debt       6751 non-null   float64
 42  curr_debt        6482 non-null   float64
 43  lt_debt          7989 non-null   float64
 44  profit_lct       6482 non-null   float64
 45  ocf_lct          6479 non-null   float64
 46  cash_debt        7947 non-null   float64
 47  fcf_ocf          6766 non-null   float64
 48  lt_ppent         7925 non-null   float64
 49  dltt_be          7746 non-null   float64
 50  debt_assets      8015 non-null   float64
 51  debt_capital     7959 non-null   float64
 52  de_ratio         8015 non-null   float64
 53  intcov           5898 non-null   float64
 54  intcov_ratio     5896 non-null   float64
 55  cash_ratio       6484 non-null   float64
```

```
 56  quick_ratio      6482 non-null   float64
 57  curr_ratio       6484 non-null   float64
 58  cash_conversion  6013 non-null   float64
 59  inv_turn         6114 non-null   float64
 60  at_turn          7898 non-null   float64
 61  rect_turn        7808 non-null   float64
 62  pay_turn         7812 non-null   float64
 63  sale_invcap      7846 non-null   float64
 64  sale_equity      7648 non-null   float64
 65  sale_nwc         5715 non-null   float64
 66  rd_sale          8017 non-null   float64
 67  adv_sale         7881 non-null   float64
 68  staff_sale       7881 non-null   float64
 69  accrual          8008 non-null   float64
 70  ptb              7774 non-null   float64
 71  PEG_trailing     3290 non-null   float64
 72  divyield         2630 non-null   object
 73  PEG_1yrforward   5413 non-null   float64
 74  PEG_ltgforward   4016 non-null   float64
 75  year             8036 non-null   int64
dtypes: float64(70), int64(2), object(4)
memory usage: 4.7+ MB
```

In [9]:

```
#creat veariable for corresponding year and merge with respect to year and compnay gvkey
#check duplicate
df_merged=df_financial_ratio.merge(df_SP_rating, on =['gvkey','year'])
df_merged['cik']=df_merged['cik'].astype(int).astype(str)
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1106 entries, 0 to 1105
Data columns (total 80 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   gvkey            1106 non-null   int64  
 1   adate            1106 non-null   object 
 2   qdate            1106 non-null   object 
 3   public_date      1106 non-null   object 
 4   CAPEI            1102 non-null   float64
 5   bm               1048 non-null   float64
 6   evm              1101 non-null   float64
 7   pe_op_basic      1073 non-null   float64
 8   pe_op_dil        1073 non-null   float64
 9   pe_exi           1071 non-null   float64
 10  pe_inc           1072 non-null   float64
 11  ps               1106 non-null   float64
 12  pcf              1105 non-null   float64
 13  dpr              934 non-null    float64
 14  npm              1106 non-null   float64
 15  opmbd            1106 non-null   float64
 16  opmad            1106 non-null   float64
 17  gpm              1106 non-null   float64
 18  ptpm             1106 non-null   float64
 19  cfm              1103 non-null   float64
 20  roa              1105 non-null   float64
 21  roe              1042 non-null   float64
 22  roce             1103 non-null   float64
 23  efftax           830 non-null    float64
 24  aftret_eq        1105 non-null   float64
 25  aftret_invcapx   1097 non-null   float64
 26  aftret_equity    1105 non-null   float64
 27  pretret_noa      941 non-null    float64
 28  pretret_earnat   941 non-null    float64
 29  GProf            1106 non-null   float64
 30  equity_invcap    1104 non-null   float64
 31  debt_invcap      1104 non-null   float64
 32  totdebt_invcap   1102 non-null   float64
 33  capital_ratio    1106 non-null   float64
 34  int_debt         1033 non-null   float64
 35  int_totdebt      1034 non-null   float64
 36  cash_lt          1106 non-null   float64
 37  invt_act         933 non-null    float64
 38  rect_act         942 non-null    float64
 39  debt_at          1104 non-null   float64
 40  debt_ebitda      1099 non-null   float64
 41  short_debt       1089 non-null   float64
 42  curr_debt        943 non-null    float64
 43  lt_debt          1106 non-null   float64
 44  profit_lct       943 non-null    float64
 45  ocf_lct          942 non-null    float64
 46  cash_debt        1100 non-null   float64
 47  fcf_ocf          1073 non-null   float64
 48  lt_ppent         1086 non-null   float64
 49  dltt_be          1049 non-null   float64
 50  debt_assets      1106 non-null   float64
 51  debt_capital     1085 non-null   float64
 52  de_ratio         1106 non-null   float64
 53  intcov           1041 non-null   float64
 54  intcov_ratio     1041 non-null   float64
 55  cash_ratio       943 non-null    float64
```

```
 56  quick_ratio      943 non-null    float64
 57  curr_ratio       943 non-null    float64
 58  cash_conversion  900 non-null    float64
 59  inv_turn         888 non-null    float64
 60  at_turn          1105 non-null   float64
 61  rect_turn        1094 non-null   float64
 62  pay_turn         1061 non-null   float64
 63  sale_invcap      1104 non-null   float64
 64  sale_equity      1044 non-null   float64
 65  sale_nwc         806 non-null    float64
 66  rd_sale          1106 non-null   float64
 67  adv_sale         1106 non-null   float64
 68  staff_sale       1106 non-null   float64
 69  accrual          1105 non-null   float64
 70  ptb              1048 non-null   float64
 71  PEG_trailing     520 non-null    float64
 72  divyield         579 non-null    object
 73  PEG_1yrforward   964 non-null    float64
 74  PEG_ltgforward   845 non-null    float64
 75  year             1106 non-null   int64
 76  splticrm         1106 non-null   object
 77  datadate         1106 non-null   object
 78  cik              1106 non-null   object
 79  conm             1106 non-null   object
dtypes: float64(70), int64(2), object(8)
memory usage: 699.9+ KB
```

In [6]:

```
df_merged.head()
```

Out[6]:

	gvkey	adate	qdate	public_date	CAPEI	bm	evm	pe_op_basic	pe_op_dil	pe_exi
0	1004	2010-05-31	2010-08-31	2010-12-31	19.866	1.343	10.679	19.076	19.345	21.131
1	1045	2009-12-31	2010-09-30	2010-12-31	-3.303	NaN	13.901	-4.211	-4.211	-3.623
2	1075	2009-12-31	2010-09-30	2010-12-31	16.841	1.245	6.993	14.493	14.544	14.544
3	1078	2009-12-31	2010-09-30	2010-12-31	18.933	0.265	9.554	11.714	11.800	15.864
4	1161	2009-12-31	2010-09-30	2010-12-31	-5.853	0.127	9.139	-26.387	-16.360	5.049

5 rows × 80 columns

data exploration

here I will firsly start to look at the dependent variable

1.Description of the dataset

- splticrm: The Global Company Key or GVKEY is a unique six-digit number key assigned to each company (issue, currency, index) in the Capital IQ Compustat database.
- splticrm: S&P Domestic Long Term Issuer Credit Rating

Variable Name	Type	Description
PERMNO	double	PERMNO
GVKEY	string	Global Company Key (GVKEY)
CUSIP	string	CUSIP IDENTIFIER - HISTORICAL (CUSIP)
TICKER	string	EXCHANGE TICKER SYMBOL - HISTORICAL (TICKER)
PEG_1yrforward	double	Forward P/E to 1-year Growth (PEG) ratio (PEG_1yrforward)
CAPEI	double	Shillers Cyclically Adjusted P/E Ratio (CAPEI)
bm	double	Book/Market (bm)
PEG_ltgforward	double	Forward P/E to Long-term Growth (PEG) ratio (PEG_ltgforward)
evm	double	Enterprise Value Multiple (evm)
pe_op_basic	double	Price/Operating Earnings (Basic, Excl. EI) (pe_op_basic)
pe_op_dil	double	Price/Operating Earnings (Diluted, Excl. EI) (pe_op_dil)
pe_exi	double	P/E (Diluted, Excl. EI) (pe_exi)
pe_inc	double	P/E (Diluted, Incl. EI) (pe_inc)
ps	double	Price/Sales (ps)
pcf	double	Price/Cash flow (pcf)
dpr	double	Dividend Payout Ratio (dpr)
ptb	double	Price/Book (ptb)
PEG_trailing	double	Trailing P/E to Growth (PEG) ratio (PEG_trailing)
divyield	double	Dividend Yield (divyield)
efftax	double	Effective Tax Rate (efftax)
GProf	double	Gross Profit/Total Assets (GProf)
aftret_eq	double	After-tax Return on Average Common Equity (aftret_eq)
aftret_equity	double	After-tax Return on Total Stockholders Equity (aftret_equity)
aftret_invcapx	double	After-tax Return on Invested Capital (aftret_invcapx)
gpm	double	Gross Profit Margin (gpm)
npm	double	Net Profit Margin (npm)
opmad	double	Operating Profit Margin After Depreciation (opmad)
opmbd	double	Operating Profit Margin Before Depreciation (opmbd)
pretret_earnat	double	Pre-tax Return on Total Earning Assets (pretret_earnat)
pretret_noa	double	Pre-tax return on Net Operating Assets (pretret_noa)
ptpm	double	Pre-tax Profit Margin (ptpm)
roa	double	Return on Assets (roa)

Variable Name	Type	Description
roce	double	Return on Capital Employed (roce)
roe	double	Return on Equity (roe)
capital_ratio	double	Capitalization Ratio (capital_ratio)
equity_invcap	double	Common Equity/Invested Capital (equity_invcap)
debt_invcap	double	Long-term Debt/Invested Capital (debt_invcap)
totdebt_invcap	double	Total Debt/Invested Capital (totdebt_invcap)
invt_act	double	Inventory/Current Assets (invt_act)
rect_act	double	Receivables/Current Assets (rect_act)
fcf_ocf	double	Free Cash Flow/Operating Cash Flow (fcf_ocf)
ocf_lct	double	Operating CF/Current Liabilities (ocf_lct)
cash_debt	double	Cash Flow/Total Debt (cash_debt)
cash_lt	double	Cash Balance/Total Liabilities (cash_lt)
cfm	double	Cash Flow Margin (cfm)
short_debt	double	Short-Term Debt/Total Debt (short_debt)
profit_lct	double	Profit Before Depreciation/Current Liabilities (profit_lct)
curr_debt	double	Current Liabilities/Total Liabilities (curr_debt)
debt_ebitda	double	Total Debt/EBITDA (debt_ebitda)
dltt_be	double	Long-term Debt/Book Equity (dltt_be)
int_debt	double	Interest/Average Long-term Debt (int_debt)
int_totdebt	double	Interest/Average Total Debt (int_totdebt)
lt_debt	double	Long-term Debt/Total Liabilities (lt_debt)
lt_ppent	double	Total Liabilities/Total Tangible Assets (lt_ppent)
de_ratio	double	Total Debt/Equity (de_ratio)
debt_assets	double	Total Debt/Total Assets (debt_assets)
debt_at	double	Total Debt/Total Assets (debt_at)
debt_capital	double	Total Debt/Capital (debt_capital)
intcov	double	After-tax Interest Coverage (intcov)
intcov_ratio	double	Interest Coverage Ratio (intcov_ratio)
cash_conversion	double	Cash Conversion Cycle (Days) (cash_conversion)
cash_ratio	double	Cash Ratio (cash_ratio)
curr_ratio	double	Current Ratio (curr_ratio)
quick_ratio	double	Quick Ratio (Acid Test) (quick_ratio)
at_turn	double	Asset Turnover (at_turn)
inv_turn	double	Inventory Turnover (inv_turn)
pay_turn	double	Payables Turnover (pay_turn)
rect_turn	double	Receivables Turnover (rect_turn)
sale_equity	double	Sales/Stockholders Equity (sale_equity)
sale_invcap	double	Sales/Invested Capital (sale_invcap)
sale_nwc	double	Sales/Working Capital (sale_nwc)

Variable Name	Type	Description
Accrual	double	Accruals/Average Assets (Accrual)

Question: Is there any string value that actually numeric among those variables?

The numeric feature "divyield" is categorized as "object". We have to convert it to "float" type.

In [8]:

```
df_merged["divyield"] = pd.to_numeric(df_merged["divyield"].str.replace('%', ''), errors
```

In [9]:

```
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1106 entries, 0 to 1105
Data columns (total 80 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   gvkey            1106 non-null   int64  
 1   adate            1106 non-null   object 
 2   qdate            1106 non-null   object 
 3   public_date      1106 non-null   object 
 4   CAPEI            1102 non-null   float64
 5   bm               1048 non-null   float64
 6   evm              1101 non-null   float64
 7   pe_op_basic      1073 non-null   float64
 8   pe_op_dil        1073 non-null   float64
 9   pe_exi           1071 non-null   float64
 10  pe_inc           1072 non-null   float64
 11  ps               1106 non-null   float64
 12  pcf              1105 non-null   float64
 13  dpr              934 non-null    float64
 14  npm              1106 non-null   float64
 15  opmbd            1106 non-null   float64
 16  opmad            1106 non-null   float64
 17  gpm              1106 non-null   float64
 18  ptpm             1106 non-null   float64
 19  cfm              1103 non-null   float64
 20  roa              1105 non-null   float64
 21  roe              1042 non-null   float64
 22  roce             1103 non-null   float64
 23  efftax           830 non-null    float64
 24  aftret_eq        1105 non-null   float64
 25  aftret_invcapx   1097 non-null   float64
 26  aftret_equity    1105 non-null   float64
 27  pretret_noa      941 non-null    float64
 28  pretret_earnat   941 non-null    float64
 29  GProf            1106 non-null   float64
 30  equity_invcap    1104 non-null   float64
 31  debt_invcap      1104 non-null   float64
 32  totdebt_invcap   1102 non-null   float64
 33  capital_ratio    1106 non-null   float64
 34  int_debt         1033 non-null   float64
 35  int_totdebt      1034 non-null   float64
 36  cash_lt          1106 non-null   float64
 37  invt_act         933 non-null    float64
 38  rect_act         942 non-null    float64
 39  debt_at          1104 non-null   float64
 40  debt_ebitda      1099 non-null   float64
 41  short_debt       1089 non-null   float64
 42  curr_debt        943 non-null    float64
 43  lt_debt          1106 non-null   float64
 44  profit_lct       943 non-null    float64
 45  ocf_lct          942 non-null    float64
 46  cash_debt        1100 non-null   float64
 47  fcf_ocf          1073 non-null   float64
 48  lt_ppent         1086 non-null   float64
 49  dltt_be          1049 non-null   float64
 50  debt_assets      1106 non-null   float64
 51  debt_capital     1085 non-null   float64
 52  de_ratio         1106 non-null   float64
 53  intcov           1041 non-null   float64
 54  intcov_ratio     1041 non-null   float64
 55  cash_ratio       943 non-null    float64
```

```
 56  quick_ratio       943 non-null    float64
 57  curr_ratio        943 non-null    float64
 58  cash_conversion   900 non-null    float64
 59  inv_turn          888 non-null    float64
 60  at_turn           1105 non-null   float64
 61  rect_turn         1094 non-null   float64
 62  pay_turn          1061 non-null   float64
 63  sale_invcap       1104 non-null   float64
 64  sale_equity       1044 non-null   float64
 65  sale_nwc          806 non-null    float64
 66  rd_sale           1106 non-null   float64
 67  adv_sale          1106 non-null   float64
 68  staff_sale        1106 non-null   float64
 69  accrual           1105 non-null   float64
 70  ptb               1048 non-null   float64
 71  PEG_trailing      520 non-null    float64
 72  divyield          579 non-null    float64
 73  PEG_1yrforward    964 non-null    float64
 74  PEG_ltgforward    845 non-null    float64
 75  year              1106 non-null   int64
 76  splticrm          1106 non-null   object
 77  datadate          1106 non-null   object
 78  cik               1106 non-null   object
 79  conm              1106 non-null   object
dtypes: float64(71), int64(2), object(7)
memory usage: 699.9+ KB
```

In [115]:

```
df_merged.head()
```

Out[115]:

	gvkey	adate	qdate	public_date	CAPEI	bm	evm	pe_op_basic	pe_op_dil	pe_exi
0	1004	2010-05-31	2010-08-31	2010-12-31	19.866	1.343	10.679	19.076	19.345	21.131
1	1045	2009-12-31	2010-09-30	2010-12-31	-3.303	NaN	13.901	-4.211	-4.211	-3.623
2	1075	2009-12-31	2010-09-30	2010-12-31	16.841	1.245	6.993	14.493	14.544	14.544
3	1078	2009-12-31	2010-09-30	2010-12-31	18.933	0.265	9.554	11.714	11.800	15.864
4	1161	2009-12-31	2010-09-30	2010-12-31	-5.853	0.127	9.139	-26.387	-16.360	5.049

5 rows × 80 columns

3.Explore the Data

In [12]:

```
credit = df_merged
```

3.1 Explore the Target Variable

In [13]:

```
# Re-examine output/target/label (i.e., "HumanOrLightning") variable based on our previo
df_merged["splticrm"].unique()
# Examine the total cases for each value in our target variable
# fire_subset["HumanOrLightning"].value_counts()
print(df_merged["splticrm"].value_counts())

# Do you think it is a imbalanced variable? To be learned later :)

# Visualize output variable
sns.countplot(df_merged["splticrm"], label = "Count")
plt.xticks(rotation=45)
```

```
BBB     132
BBB-    125
B+      123
BB-     119
B       109
BB       92
BBB+     88
A        66
BB+      65
A-       62
B-       39
A+       30
CCC+     16
AA-      15
AA        7
D         6
AAA       4
AA+       3
CC        2
CCC-      2
CCC       1
Name: splticrm, dtype: int64
```

Out[13]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 1
6,
        17, 18, 19, 20]),
 [Text(0, 0, 'BB'),
  Text(1, 0, 'B-'),
  Text(2, 0, 'BBB-'),
  Text(3, 0, 'AA'),
  Text(4, 0, 'B+'),
  Text(5, 0, 'A-'),
  Text(6, 0, 'A'),
  Text(7, 0, 'BBB'),
  Text(8, 0, 'BBB+'),
  Text(9, 0, 'BB+'),
  Text(10, 0, 'A+'),
  Text(11, 0, 'B'),
  Text(12, 0, 'AAA'),
  Text(13, 0, 'BB-'),
  Text(14, 0, 'AA-'),
  Text(15, 0, 'AA+'),
  Text(16, 0, 'D'),
  Text(17, 0, 'CCC+'),
  Text(18, 0, 'CCC-'),
  Text(19, 0, 'CCC'),
  Text(20, 0, 'CC')])
```

3.2 Explore the Financial Ratios

- How many kinds of financial ratios in our dataset?

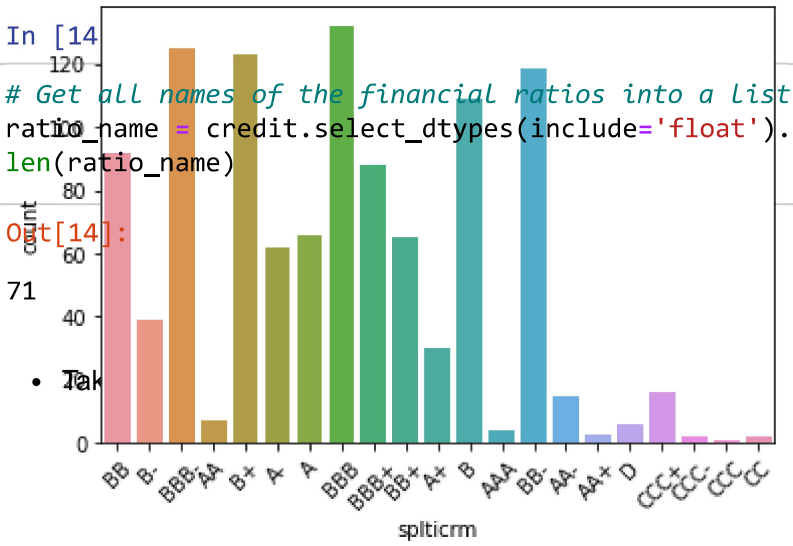
In [14]:

```
# Get all names of the financial ratios into a list
ratio_name = credit.select_dtypes(include='float').columns.tolist()
len(ratio_name)
```

Out[14]:

71

- Tak



In [15]:

```
for ratio in ratio_name:
    print(credit[ratio].agg([min,max]))
```

```
min   -7749.7
max    1519.5
Name: CAPEI, dtype: float64
min       0.007
max    5152.550
Name: bm, dtype: float64
min   -421.980
max    774.246
Name: evm, dtype: float64
min   -1750.8
max     666.0
Name: pe_op_basic, dtype: float64
min   -1800.000
max     671.333
Name: pe_op_dil, dtype: float64
min   -1602.8
max     597.0
Name: pe_exi, dtype: float64
min   -1679.400
```

We can see the scales of the ratios differ dramatically, which suggests a standardization process otherwise the influence of some ratios will be overestimated.

In [16]:

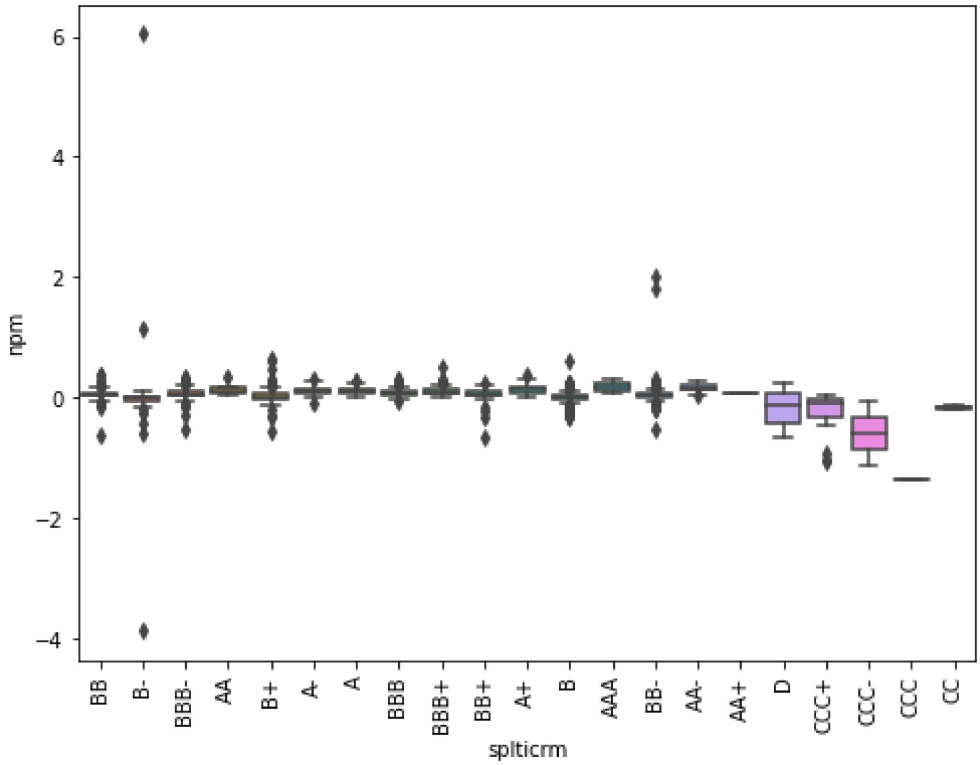
```
# Examine the relationship between features and target variable

# Examine summary statistics
print(credit[["npm","splticrm"]].groupby(["splticrm"]).mean())
# Examine a plot
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x="splticrm", y="npm", data = credit)
plt.xticks(rotation=90)
```

```
                npm
splticrm
A          0.103591
A+         0.136000
A-         0.102581
AA         0.131714
AA+        0.078667
AA-        0.153667
AAA        0.186750
B          0.005339
B+         0.029431
B-         0.039385
BB         0.057217
BB+        0.042877
BB-        0.064412
BBB        0.084402
BBB+       0.105125
BBB-       0.064312
CC        -0.172500
CCC       -1.356000
CCC+      -0.285438
CCC-      -0.588000
D         -0.181833
```

Out[16]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 1
6,
        17, 18, 19, 20]),
 [Text(0, 0, 'BB'),
  Text(1, 0, 'B-'),
  Text(2, 0, 'BBB-'),
  Text(3, 0, 'AA'),
  Text(4, 0, 'B+'),
  Text(5, 0, 'A-'),
  Text(6, 0, 'A'),
  Text(7, 0, 'BBB'),
  Text(8, 0, 'BBB+'),
  Text(9, 0, 'BB+'),
  Text(10, 0, 'A+'),
  Text(11, 0, 'B'),
  Text(12, 0, 'AAA'),
  Text(13, 0, 'BB-'),
  Text(14, 0, 'AA-'),
  Text(15, 0, 'AA+'),
  Text(16, 0, 'D'),
  Text(17, 0, 'CCC+'),
  Text(18, 0, 'CCC-'),
  Text(19, 0, 'CCC'),
  Text(20, 0, 'CC')])
```


6
4
2
0
−2
−4
npm
BB
B-
BBB-
AA
B+
A-
A
BBB
BBB+
BB+
A+
B
AAA
BB-
AA-
AA+
D
CCC+
CCC-
CCC
CC
splticrm

In [122]:

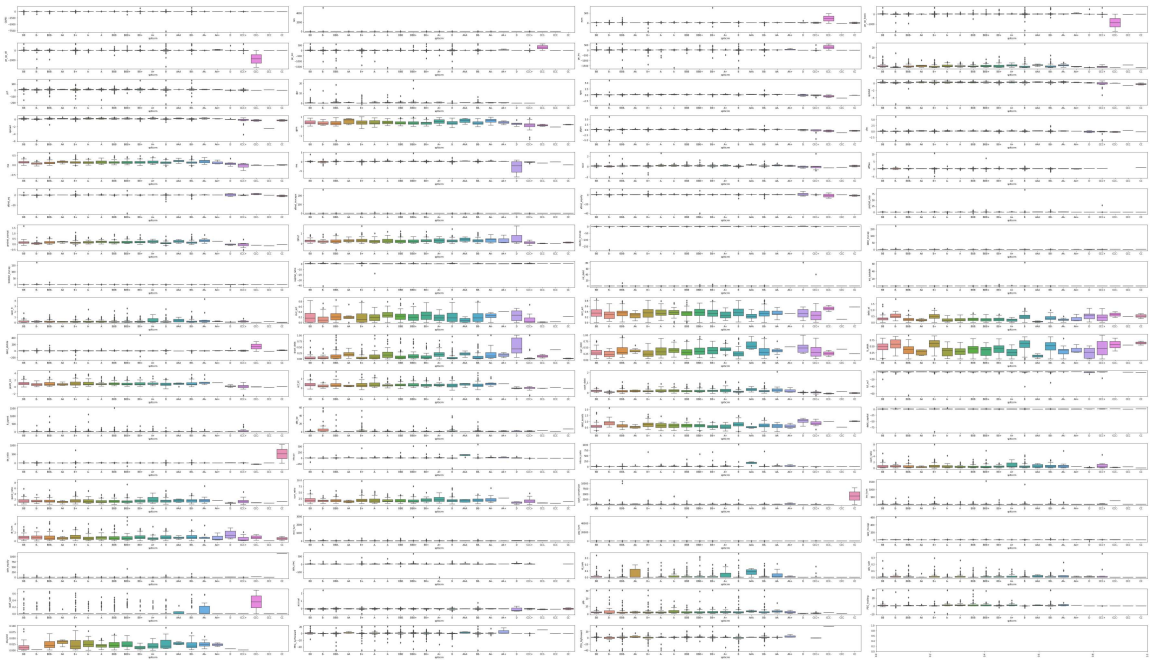
```
# Examine the relationship between features and target variable

# Define a function to show the relationship features and target variable
def plot_relat(num_feat, h, l): #num_feat is a df.columns containing all the numeric fea
    # Create a figure and axis objects for the subplots
    fig, axes = plt.subplots(h, l, figsize=(70, 40))
    # Flatten the axes array
    axes = axes.flatten()

    for i, column in enumerate(num_feat):
        sns.boxplot(x="splticrm", y=column, data = credit, ax=axes[i])

    # Adjust spacing between subplots
    plt.tight_layout()
    plt.xticks(rotation=90)

#covid_num_feat = covid.describe().iloc[:,1:].columns.tolist()
plot_relat(ratio_name, 18, 4)
```



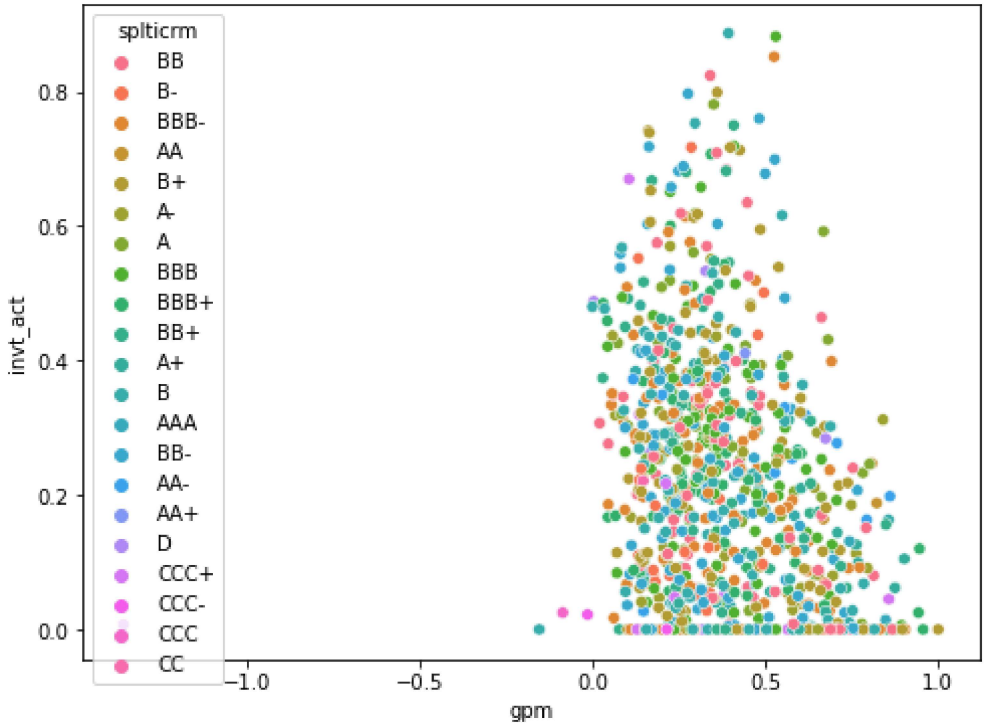
In [123]:

```
# We can also examine the relationship between multiple numerical input variables and ou

plt.figure(figsize=(8, 6))
sns.scatterplot(x = "gpm", y = "invt_act", hue = "splticrm", data = credit)
```

Out[123]:

```
<AxesSubplot:xlabel='gpm', ylabel='invt_act'>
```



- Check the distribution

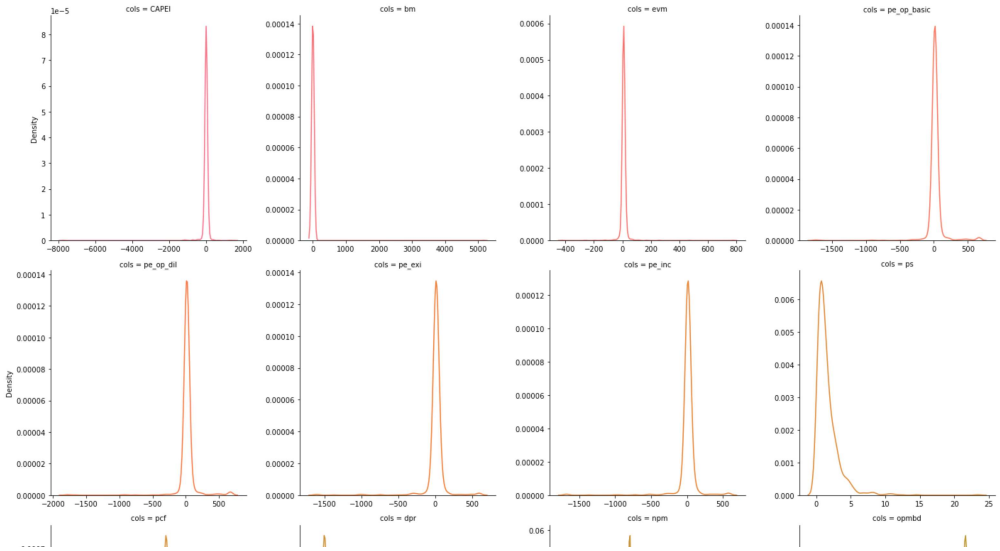
In [124]:

```
# Examine all numerical variables together
numericVal = ['float64']
NumFire = credit.select_dtypes(include = numericVal)

# Use melt to change data to long format
NumFireLong = NumFire.melt(var_name='cols',  value_name='Measure')
NumFireLong.shape
sns.displot(kind='kde', data=NumFireLong, col='cols', col_wrap=4, x='Measure', hue="cols
```

Out[124]:

```
<seaborn.axisgrid.FacetGrid at 0x1446fec07c0>
```



- Explore the correlations between all the financial ratios

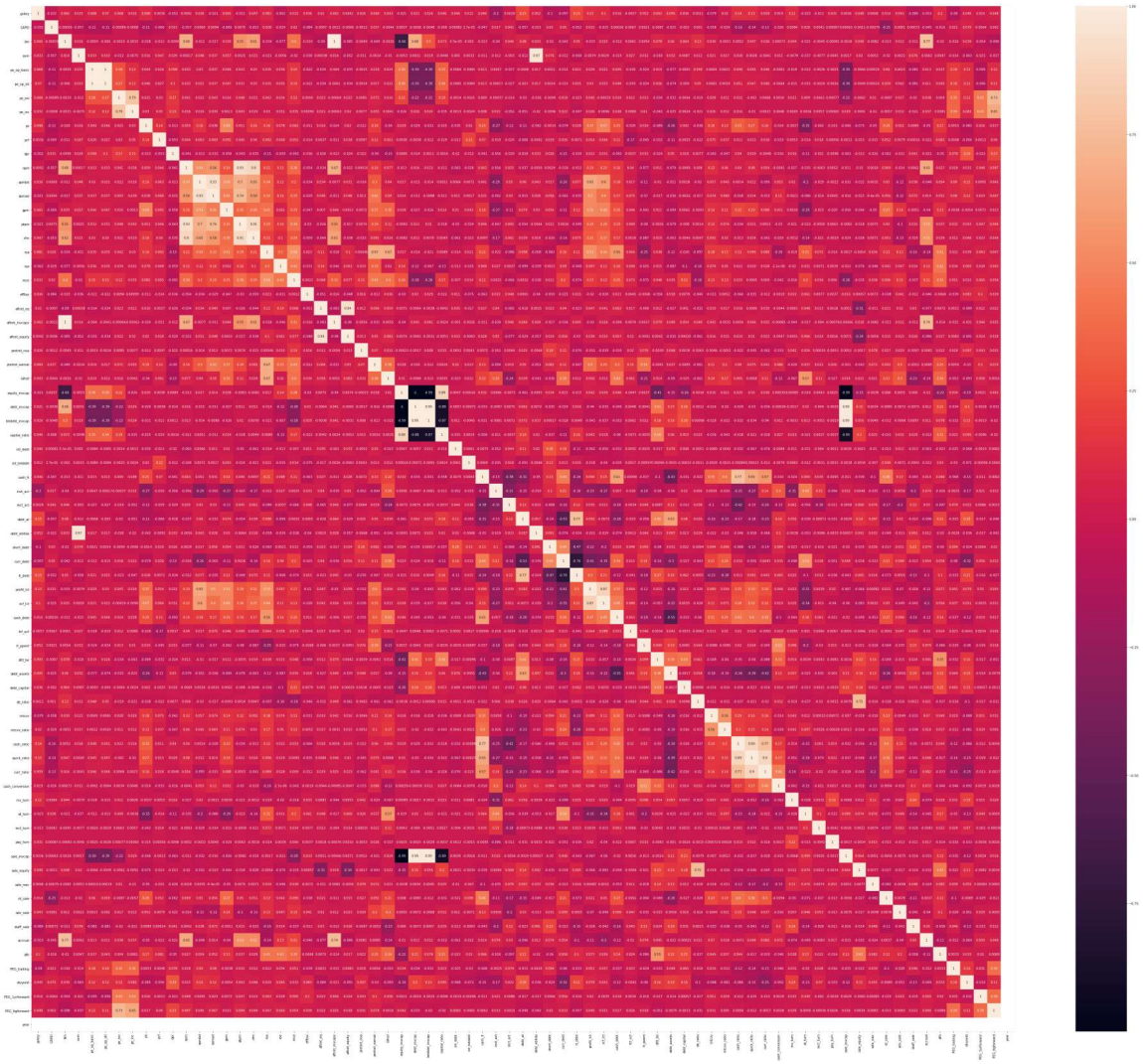
Correlation Maxtrix

In [125]:

```
# Create a Correlation matrix to explore relationship between
# Let's check the correlation between the variables
plt.figure(figsize=(70,60))
sns.heatmap(credit.corr(), annot=True)
```

Out[125]:

<AxesSubplot:>



We can see there are some variables have perfect collinearity. For example, the correlation between the variable "pe_op_basic" and the variable "pe_op_dil" equals 1. There are also many variables have quite high correlations. These results indicate there are some overlap among all the 71 financial ratios and we only need to use the ratios that can reveal the most information.

- Conduct some statistical test to evaluate the relationship

3.3 Handle Missing Values

In [17]:

```
credit.isnull().sum()
```

Out[17]:

```
gvkey          0
adate          0
qdate          0
public_date    0
CAPEI          4
              ..
year           0
splticrm       0
datadate       0
cik            0
conm           0
Length: 80, dtype: int64
```

In [18]:

```
# Store all the names of columns containing missing values
missing = credit[ratio_name].columns[credit[ratio_name].isnull().any()].tolist()
print(missing)
```

```
['CAPEI', 'bm', 'evm', 'pe_op_basic', 'pe_op_dil', 'pe_exi', 'pe_inc', 'pc
f', 'dpr', 'cfm', 'roa', 'roe', 'roce', 'efftax', 'aftret_eq', 'aftret_inv
capx', 'aftret_equity', 'pretret_noa', 'pretret_earnat', 'equity_invcap',
'debt_invcap', 'totdebt_invcap', 'int_debt', 'int_totdebt', 'invt_act', 'r
ect_act', 'debt_at', 'debt_ebitda', 'short_debt', 'curr_debt', 'profit_lc
t', 'ocf_lct', 'cash_debt', 'fcf_ocf', 'lt_ppent', 'dltt_be', 'debt_capita
l', 'intcov', 'intcov_ratio', 'cash_ratio', 'quick_ratio', 'curr_ratio',
'cash_conversion', 'inv_turn', 'at_turn', 'rect_turn', 'pay_turn', 'sale_i
nvcap', 'sale_equity', 'sale_nwc', 'accrual', 'ptb', 'PEG_trailing', 'divy
ield', 'PEG_1yrforward', 'PEG_ltgforward']
```

In [19]:

```
# Impute the missing values with mode
for i in credit[missing]:
    credit[i] = credit[i].fillna(credit[i].mode()[0])
```

In [20]:

```
# Check whether there are any missing values left
credit[ratio_name].columns[credit[ratio_name].isnull().any()].tolist()
```

Out[20]:

```
[]
```

3.4 Check Outliers

In [23]:

```
# Define a function to implement the examination of outliers
def winsorize_data(data):
    # Calculate the 1% and 99% percentiles
    p1 = np.percentile(data, 1)
    p99 = np.percentile(data, 99)

    # Winsorize the data
    winsorized_data = np.where(data < p1, p1, data)
    winsorized_data = np.where(data > p99, p99, winsorized_data)

    return winsorized_data



for column in ratio_name:
    column_data = credit[column]
    winsorized_data = winsorize_data(column_data)
    credit[column] = winsorized_data
```

3.5 Encoding the Credit Rating

Credit Rating is a kind of ordinal variable. We need to assign higher value to higer credit rating, which means AAA should be assigned to the biggest value and D should be assigned to the smallest value.

In [28]:

```
from sklearn.preprocessing import OrdinalEncoder
```

In [29]:

```
print(credit["splticrm"].unique())
print(len(credit["splticrm"].unique().tolist()))
```

```
['BB' 'B-' 'BBB-' 'AA' 'B+' 'A-' 'A' 'BBB' 'BBB+' 'BB+' 'A+' 'B' 'AAA'
 'BB-' 'AA-' 'AA+' 'D' 'CCC+' 'CCC-' 'CCC' 'CC']
21
```

In [30]:

```
rate_enco = OrdinalEncoder(categories=[["AAA","AA+","AA","AA-","A+","A","A-","BBB+","BBB
# Mapping from categories to the numbers
credit["splticrm"] = rate_enco.fit_transform(credit.loc[:, ["splticrm"]])
```

In [31]:

```
print(credit["splticrm"].unique())
```

```
[11. 15.  9.  2. 13.  6.  5.  8.  7. 10.  4. 14.  0. 12.  3.  1. 20. 16.
 18. 17. 19.]
```

4.Dimension Reduction

4.1 Practice LASSO

In [32]:

```
from sklearn.linear_model import LassoCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [33]:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(credit[ratio_name], credit["splticrm

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create a LassoCV object and fit the model
lasso_cv = LassoCV(cv=5)  # we can adjust the number of cross-validation folds with the
lasso_cv.fit(X_train, y_train)

# Get the best alpha value
best_alpha = lasso_cv.alpha_
print("Best Alpha:", best_alpha)

# Predict on the test set
y_pred = lasso_cv.predict(X_test)

# Evaluate the model
score = lasso_cv.score(X_test, y_test)
print("R^2 Score:", score)
```

```
Best Alpha: 0.046505008822677295
R^2 Score: -12.347400277550927
```

In [34]:

```
# Get the selected features (non-zero coefficients)
selected_feat = credit[ratio_name].columns[lasso_cv.coef_ != 0]
print("Selected Features:", selected_feat.tolist())
print("\nThe number of financial ratios is reduced to "+str(len(selected_feat))+" using
```

```
Selected Features: ['CAPEI', 'bm', 'pe_op_basic', 'pe_op_dil', 'pe_exi',
'pe_inc', 'ps', 'pcf', 'dpr', 'npm', 'opmad', 'gpm', 'cfm', 'roa', 'roe',
'roce', 'aftret_equity', 'pretret_noa', 'pretret_earnat', 'GProf', 'capita
l_ratio', 'int_debt', 'int_totdebt', 'cash_lt', 'debt_at', 'short_debt',
'curr_debt', 'lt_debt', 'ocf_lct', 'cash_debt', 'fcf_ocf', 'dltt_be', 'deb
t_capital', 'intcov', 'intcov_ratio', 'cash_ratio', 'quick_ratio', 'cash_c
onversion', 'inv_turn', 'at_turn', 'sale_equity', 'sale_nwc', 'rd_sale',
'adv_sale', 'staff_sale', 'ptb', 'PEG_trailing', 'divyield', 'PEG_ltgforwa
rd']

The number of financial ratios is reduced to 49 using LASSO.
```

4.2 Practice PCA

In [35]:

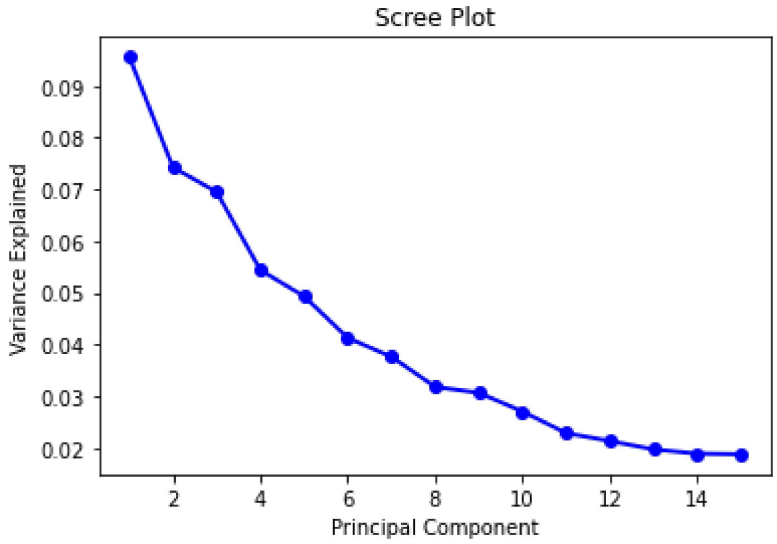
```
from sklearn.decomposition import PCA
```

In [36]:

```
scaler = StandardScaler()
X = scaler.fit_transform(credit[ratio_name])

#show PCA with all components
pca0 = PCA(n_components=15)
ori_pca = pca0.fit_transform(X)

#scree plot
PC_values = np.arange(pca0.n_components_) + 1
plt.plot(PC_values, pca0.explained_variance_ratio_, 'o-', linewidth=2, color='blue')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.show()
```



In [37]:

```
#select n=13
pca = PCA(n_components=13) #first create a PCA model

Xpca = pca.fit_transform(X) #fit the PCA model with X

pca_names = ["PC"+str(i+1) for i in range(13)]
pca_df = pd.DataFrame(
    data = Xpca,
    columns = pca_names)
pca_df
```

Out[37]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	
0	-0.859759	-0.406213	-0.753764	0.903540	0.179369	1.142941	0.352010	-1.180265	
1	-1.104884	-0.413861	-0.632842	1.156673	-0.115918	0.948291	0.267209	-1.125141	
2	-3.322414	-0.094079	1.132859	-0.088420	-0.588981	-0.137007	0.051854	2.076667	-
3	-3.323001	-0.095511	1.133852	-0.087447	-0.590546	-0.136164	0.049151	2.075132	-
4	-0.841511	-0.023263	1.810350	-0.453113	0.622613	-1.256160	-0.153846	-1.077054	
...	...	...	...	...	...	...	...	...	
2164	-0.946381	0.163124	1.510475	-2.338472	-0.074015	0.461924	1.111966	-1.226385	
2165	-0.945749	0.165169	1.509457	-2.339457	-0.073047	0.460242	1.115887	-1.224058	
2166	0.164215	-1.715167	3.515242	-3.786282	-2.275395	5.492840	-6.047621	4.724832	
2167	0.159342	-1.715776	3.517218	-3.782493	-2.279923	5.494564	-6.044957	4.719908	
2168	-6.585211	-1.120659	0.596201	-3.281221	0.883175	0.737008	-1.561588	9.973985	

2169 rows × 13 columns

text analysis

In [21]:

```
#start to load all 10k fillings
import os
from pathlib import Path
import re
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer


# Specify the directory path
directory = 'C:/Users/Lenovo/Desktop/10-X_C_2006-2010/QTR1'
os.chdir(directory)

# Get all file names in the directory
file_names =pd.DataFrame(os.listdir(directory))[:500]#here we only processed first 1000 
file_names.head()
```

Out[21]:

	0
0	20100104_10-K-A_edgar_data_1314772_0001104659-...
1	20100104_10-K-A_edgar_data_1435224_0001165527-...
2	20100104_10-K_edgar_data_1088787_0001448788-09...
3	20100104_10-K_edgar_data_12040_0000914317-10-0...
4	20100104_10-K_edgar_data_1385329_0001062993-10...

In [22]:

```
def getFillingInfo(file_name):#this function is to get the information part of the file
    name_parts = file_name.split('_')
    res=[name_parts[0],name_parts[1],name_parts[4]]
    return res
def year(date):#this function is to get the information part of the file title
    year=date[:4]
    return year
```

In [23]:

```
#here we want to creat a dataframe with all information about the document
df=pd.DataFrame(file_names.iloc[:,0].apply(getFillingInfo))
df[['date', 'type', 'CIK']] = df.iloc[:,0].apply(pd.Series)
df['year'] = df['date'].apply(year)
df=df.iloc[:,1:]
df['path'] = file_names
df=df[df['type']=="10-K"].reset_index(drop=True)#we just want the 10K file
print(df)
```

```
         date  type      CIK  year  \
0    20100104  10-K  1088787  2010
1    20100104  10-K    12040  2010
2    20100104  10-K  1385329  2010
3    20100104  10-K  1418196  2010
4    20100104  10-K  1446414  2010
..        ...   ...      ...   ...
121  20100114  10-K   733337  2010
122  20100114  10-K    77597  2010
123  20100114  10-K   785787  2010
124  20100114  10-K   886128  2010
125  20100114  10-K   893965  2010

                                                  path
0    20100104_10-K_edgar_data_1088787_0001448788-09...
1    20100104_10-K_edgar_data_12040_0000914317-10-0...
2    20100104_10-K_edgar_data_1385329_0001062993-10...
3    20100104_10-K_edgar_data_1418196_0001432093-10...
4    20100104_10-K_edgar_data_1446414_0001161697-09...
..                                                 ...
121  20100114_10-K_edgar_data_733337_0001144204-10-...
122  20100114_10-K_edgar_data_77597_0000950123-10-0...
123  20100114_10-K_edgar_data_785787_0001193125-10-...
124  20100114_10-K_edgar_data_886128_0000950123-10-...
125  20100114_10-K_edgar_data_893965_0000893965-10-...

[126 rows x 5 columns]
```

In [24]:

```
df=df[df['CIK'].isin(df_merged['cik'])].reset_index()
df
```

Out[24]:

	index	date	type	CIK	year	path
0	49	20100112	10-K	805305	2010	20100112_10-K_edgar_data_805305_0000950123-10-...
1	123	20100114	10-K	785787	2010	20100114_10-K_edgar_data_785787_0001193125-10-...

In [25]:

```
def readtxt(file_path):
    # Convert the file path to a Path object
    file_path = Path(file_path)

    # Read the contents of the file
    text = file_path.read_text()

    # Replace newline characters with a space
    text = text.replace("\n", " ")

    # Remove "'s" occurrences
    text = text.replace("\'s", "")

    # Remove multiple spaces with a single space
    text = re.sub(r'\s+', ' ', text)

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Remove hyphens
    text = text.replace("-", "")

    # Convert the text to lowercase
    text = text.lower()

    return text

def preprocess_text(text):

    # Tokenize the text
    tokens = word_tokenize(text)
    # Remove stop words
    filtered_tokens = [token for token in tokens if token not in stopwords.words('englis
    # Lemmatize the tokens
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens]
    # Join the tokens back into a string
    processed_text = ' '.join(lemmatized_tokens)
    return processed_text
```

In [26]:

```
df['txt']=df['path'].apply(readtxt)
df
```

Out[26]:

	index	date	type	CIK	year	path	
0	49	20100112	10-K	805305	2010	20100112_10-K_edgar_data_805305_0000950123-10-...	<header> <filename>_k_edga
1	123	20100114	10-K	785787	2010	20100114_10-K_edgar_data_785787_0001193125-10-...	<header> <filename>_k_edga

In [27]:

```
def getcontent(text):#here we want to filter the tags at the beginning of the document a
    try:
        text_new = text[text.index('table of content'):]
    except:
        text_new= text#this is becuase the form is not fully standardized so when the ke
    return text_new
df['txt'] = df['txt'].apply(getcontent)
df
```

Out[27]:

	index	date	type	CIK	year	path	txt
0	49	20100112	10-K	805305	2010	20100112_10-K_edgar_data_805305_0000950123-10-...	table of contents united states securities and...
1	123	20100114	10-K	785787	2010	20100114_10-K_edgar_data_785787_0001193125-10-...	table of contents united states securities and...

In [28]:

```
df['txt']=df['txt'].apply(preprocess_text)
df
```

Out[28]:

	index	date	type	CIK	year	path	txt
0	49	20100112	10-K	805305	2010	20100112_10-K_edgar_data_805305_0000950123-10-...	table content united state security exchange c...
1	123	20100114	10-K	785787	2010	20100114_10-K_edgar_data_785787_0001193125-10-...	table content united state security exchange c...

In [29]:

```
documents = df['txt'].tolist()
vectorizer = CountVectorizer()#get the word count
bag_of_words = vectorizer.fit_transform(documents)
feature_names = vectorizer.get_feature_names()

vectorizer2 = TfidfVectorizer()#get the tf-idf for word
tfidf = vectorizer2.fit_transform(documents)
feature_names2 = vectorizer2.get_feature_names()
```

In [30]:

```
# Convert bag_of_words to a pandas DataFrame
bag_of_words_df = pd.DataFrame(bag_of_words.toarray(), columns=feature_names)

# View the bag_of_words DataFrame
new_columns = ['w_' + col for col in bag_of_words_df]#here we add a w_ infront of all wo
bag_of_words_df.columns = new_columns
print(bag_of_words_df)
df=df.join(bag_of_words_df)
print(df)
```

```
   w_aa  w_abandonment  w_abercrombie  w_ability  w_abl  w_able  \
0     0              0              1         27      9       8
1     1              1              0         42      0      15

   w_abovereferenced  w_abrasion  w_abreast  w_absence  ...  w_zation  \
0                  0           0          0          0  ...         2
1                  1           1          1          1  ...         0

   w_zealand  w_zebraska  w_zero  w_zip  w_zipsafe  w_zoo  w_zqk  w_zumiez
\
0          5           1      11      1          0      2      1         1
1          9           0       4      1          1      0      0         0

   w_zurich
0         1
1         0

[2 rows x 4017 columns]
   index      date  type     CIK  year  \
0     49  20100112  10-K  805305  2010
1    123  20100114  10-K  785787  2010

                                              path  \
0  20100112_10-K_edgar_data_805305_0000950123-10-...
1  20100114_10-K_edgar_data_785787_0001193125-10-...

                                               txt  w_aa  w_abandonment
\
0  table content united state security exchange c...     0              0
1  table content united state security exchange c...     1              1

   w_abercrombie  ...  w_zation  w_zealand  w_zebraska  w_zero  w_zip  \
0              1  ...         2          5           1      11      1
1              0  ...         0          9           0       4      1

   w_zipsafe  w_zoo  w_zqk  w_zumiez  w_zurich
0          0      2      1         1         1
1          1      0      0         0         0

[2 rows x 4024 columns]
```

In [31]:

```
tfidf_df = pd.DataFrame(tfidf.toarray(),columns=feature_names2)
print(tfidf_df)
```

```
         aa  abandonment  abercrombie   ability       abl      able  \
0  0.000000     0.000000     0.000948  0.018206  0.008529  0.005394
1  0.001006     0.001006     0.000000  0.030052  0.000000  0.010733

   abovereferenced  abrasion   abreast   absence  ...    zation   zealand
\
0         0.000000  0.000000  0.000000  0.000000  ...  0.001895  0.003371
1         0.001006  0.001006  0.001006  0.001006  ...  0.000000  0.006440

   zebraska      zero       zip   zipsafe       zoo       zqk    zumiez  \
0  0.000948  0.007417  0.000674  0.000000  0.001895  0.000948  0.000948
1  0.000000  0.002862  0.000716  0.001006  0.000000  0.000000  0.000000

     zurich
0  0.000948
1  0.000000

[2 rows x 4017 columns]
```

In [32]:

```
def get_top_n_words(corpus, n=None):
    # Create a CountVectorizer object and fit it on the corpus
    vec = CountVectorizer(stop_words='english').fit(corpus)

    # Transform the corpus into a bag-of-words matrix
    bag_of_words = vec.transform(corpus)

    # Calculate the sum of word frequencies across all documents
    sum_words = bag_of_words.sum(axis=0)

    # Create a list of (word, frequency) tuples
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]

    # Sort the list in descending order based on word frequency
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)

    # Return the top n words
    return words_freq[:n]

# Call the function with the 'txt' column from the DataFrame and retrieve the top 30 com
common_words = get_top_n_words(df['txt'], 30)
print(common_words)
```

```
[('company', 784), ('million', 763), ('financial', 545), ('october', 509),
('operation', 504), ('income', 394), ('year', 389), ('fiscal', 386), ('ass
et', 385), ('stock', 382), ('tax', 377), ('cash', 360), ('agreement', 33
1), ('product', 316), ('net', 310), ('statement', 304), ('credit', 299),
('loss', 293), ('cost', 266), ('facility', 264), ('share', 260), ('resul
t', 260), ('value', 247), ('quiksilver', 231), ('consolidated', 227), ('ta
ble', 224), ('market', 220), ('expense', 220), ('related', 217), ('plan',
216)]
```

In [33]:

```
def get_top_n_bigram(corpus, n=None):#similar jobs as unigrame
    vec = CountVectorizer(ngram_range=(2, 2)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words = get_top_n_bigram(df['txt'], 40)
common_words
```

Out[33]:

```
[('table content', 183),
 ('financial statement', 177),
 ('ended october', 172),
 ('year ended', 142),
 ('income tax', 141),
 ('approximately million', 139),
 ('common stock', 134),
 ('fair value', 134),
 ('discontinued operation', 122),
 ('million million', 120),
 ('credit facility', 119),
 ('quiksilver inc', 118),
 ('cash flow', 116),
 ('continuing operation', 103),
 ('consolidated financial', 101),
 ('financial reporting', 101),
 ('foreign currency', 96),
 ('internal control', 94),
 ('result operation', 89),
 ('fiscal year', 85),
 ('control financial', 85),
 ('report form', 81),
 ('asset liability', 77),
 ('stock option', 77),
 ('asia pacific', 75),
 ('united state', 74),
 ('incorporated reference', 67),
 ('senior note', 66),
 ('interest rate', 62),
 ('reference exhibit', 60),
 ('balance sheet', 58),
 ('gross profit', 58),
 ('exhibit company', 57),
 ('purchase price', 55),
 ('income loss', 55),
 ('million fiscal', 54),
 ('stock purchase', 53),
 ('loss income', 53),
 ('tax benefit', 53),
 ('financial condition', 48)]
```

In [34]:

```
def get_top_n_trigram(corpus, n=None):#similar jobs as unigrame
    vec = CountVectorizer(ngram_range=(3, 3)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words = get_top_n_trigram(df['txt'], 20)
common_words
```

Out[34]:

```
[('year ended october', 142),
 ('consolidated financial statement', 85),
 ('internal control financial', 85),
 ('control financial reporting', 85),
 ('incorporated reference exhibit', 59),
 ('reference exhibit company', 56),
 ('share common stock', 47),
 ('foreign currency exchange', 43),
 ('aep industry inc', 43),
 ('discontinued operation net', 40),
 ('jp morgan europe', 39),
 ('annual report form', 31),
 ('cash collateral agreement', 31),
 ('bank jp morgan', 31),
 ('provision income tax', 30),
 ('income continuing operation', 30),
 ('fiscal year ended', 29),
 ('condition result operation', 29),
 ('ended october thousand', 29),
 ('disclosure control procedure', 29)]
```