Yilun Wang(yilun830@bu.edu) - ASSIGNMENT #2

- Use this template to start working on Assignment #2.
- Follow the insutructions listed in "Assignment #2" under the **Assignments** tab on the BA870 site on QuestromTools.

convert notebook to html then print as PDF

In [159]:

```
!jupyter nbconvert --to html /content/BA870_Assignment2_Yilun_Wang.ipynb
```

```
[NbConvertApp] WARNING | pattern '/content/BA870_Assignment2_Yilun_W
ang.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELE
ASES.

Options
=======
The options below are convenience aliases to configurable class-opti
ons,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an e
rror and include the error message in the cell output (the default b
ehaviour is to abort conversion). This flag is only relevant if '--e
xecute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting note
book with default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConve
rtApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConve
rtApp.export_format=notebook --FilesWriter.build_directory= --ClearO
utputPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --T
emplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --
TemplateExporter.exclude_input=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN',
'ERROR', 'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'not
ebook', 'pdf', 'python', 'rst', 'script', 'slides']
            or a dotted object name that represents the import path
for an
            `Exporter` class
    Default: 'html'
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template file to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_file]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a t
ime.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                  to output to the directory of each
notebook. To recover
                                  previous default behaviour (output
ting to the current
                                  working directory) use . as the fl
ag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url poin
ting to a copy
            of reveal.js.
            For speaker notes to work, this must be a relative path
```

```
to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory o
f the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#r
eveal-js-html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb

            which will convert mynotebook.ipynb to the default forma
t (probably HTML).

            You can specify the export format with `--to`.
            Options include ['asciidoc', 'custom', 'html', 'latex',
'markdown', 'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. L
aTeX includes
            'base', 'article' and 'report'.  HTML includes 'basic' a
nd 'full'. You
            can specify the flavor of the format used.

            > jupyter nbconvert --to html --template basic mynoteboo
k.ipynb

            You can also pipe the output to stdout, rather than a fi
le

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post se
rve

            Multiple notebooks can be given at the command line in a
couple of
            different ways:
```

```
            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file,
containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```

Import packages

In [160]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

import statsmodels.api as sm
from statsmodels.sandbox.regression.predstd import wls_prediction_std
import seaborn as sns
#import winsorize
from scipy.stats.mstats import winsorize
```

Model preparation

Basic Info about the Data

In [161]:

```
data = pd.read_csv('https://raw.githubusercontent.com/ChasteloveCNN/ba765-sessio
n02/main/assignment2.csv')
```

In [162]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 354 entries, 0 to 353
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   TICKER             354 non-null    object 
 1   CURRENT ASSETS     352 non-null    float64
 2   TOTAL ASSETS       354 non-null    float64
 3   EBIT               354 non-null    float64
 4   CURRENT LIABIL     353 non-null    float64
 5   TOTAL LIABILITIES  354 non-null    float64
 6   RETAINED EARNINGS  354 non-null    float64
 7   TOTAL SALES        354 non-null    float64
 8   CREDIT_RATING      354 non-null    int64  
dtypes: float64(7), int64(1), object(1)
memory usage: 25.0+ KB
```

In [163]:

```
data.head(5)
```

Out[163]:

	TICKER	CURRENT ASSETS	TOTAL ASSETS	EBIT	CURRENT LIABIL	TOTAL LIABILITIES	RETAINED EARNINGS	TOTAL SALES
0	ARXX	328.354	638.022	47.473	119.215	150.352	95.273	551.846
1	ABT	11281.883	36178.172	4860.219	11951.195	22123.986	9958.494	22476.322
2	AMD	3963.000	13147.000	401.000	2852.000	7072.000	464.000	5649.000
3	APD	2612.600	11180.700	1013.500	2323.400	6078.700	5521.800	8850.400
4	HON	12304.000	30941.000	3544.000	10135.000	21221.000	11256.000	31367.000

In [164]:

```
# import PRCC_C & CSHO data
data2 = pd.read_csv('https://raw.githubusercontent.com/ChasteloveCNN/ba765-session02/main/variables.csv')
```

In [165]:

```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   gvkey     344 non-null    int64  
 1   datadate  344 non-null    int64  
 2   fyear     344 non-null    int64  
 3   indfmt    344 non-null    object 
 4   consol    344 non-null    object 
 5   popsrc    344 non-null    object 
 6   datafmt   344 non-null    object 
 7   tic       344 non-null    object 
 8   curcd     344 non-null    object 
 9   ceq       344 non-null    float64
 10  csho      344 non-null    float64
 11  ni        344 non-null    float64
 12  costat    344 non-null    object 
 13  prcc_c    333 non-null    float64
dtypes: float64(4), int64(3), object(7)
memory usage: 37.8+ KB
```

In [166]:

```
data2.head(5)
```

Out[166]:

	gvkey	datadate	fyear	indfmt	consol	popsrc	datafmt	tic	curcd	ceq	csh
0	1056	20060630	2006	INDL	C	D	STD	ARXX	USD	487.670	75.27
1	1078	20061231	2006	INDL	C	D	STD	ABT	USD	14054.186	1537.24
2	1161	20061231	2006	INDL	C	D	STD	AMD	USD	5785.000	547.00
3	1209	20060930	2006	INDL	C	D	STD	APD	USD	4924.000	217.25
4	1300	20061231	2006	INDL	C	D	STD	HON	USD	9720.000	800.59

In [167]:

```
# rename column tic to TICKER
data2 = data2.rename(columns={'tic':'TICKER'})
```

In [168]:

```
# merge two datasets
df = pd.merge(data, data2, how='outer', on='TICKER')
```

In [169]:

```
# delete useless columns
del df['gvkey']
del df['datadate']
del df['fyear']
del df['indfmt']
del df['consol']
del df['popsrc']
del df['datafmt']
del df['curcd']
del df['costat']
```

In [170]:

```
df.head(5)
```

Out[170]:

	TICKER	CURRENT ASSETS	TOTAL ASSETS	EBIT	CURRENT LIABIL	TOTAL LIABILITIES	RETAINED EARNINGS	TOTAL SALES
0	ARXX	328.354	638.022	47.473	119.215	150.352	95.273	551.846
1	ABT	11281.883	36178.172	4860.219	11951.195	22123.986	9958.494	22476.322
2	AMD	3963.000	13147.000	401.000	2852.000	7072.000	464.000	5649.000
3	APD	2612.600	11180.700	1013.500	2323.400	6078.700	5521.800	8850.400
4	HON	12304.000	30941.000	3544.000	10135.000	21221.000	11256.000	31367.000

Deal with null values:

In [171]:

```
# check null value
df.isna().sum()
```

Out[171]:

```
TICKER                0
CURRENT ASSETS        2
TOTAL ASSETS          0
EBIT                  0
CURRENT LIABIL        1
TOTAL LIABILITIES     0
RETAINED EARNINGS     0
TOTAL SALES           0
CREDIT_RATING         0
ceq                  10
csho                 10
ni                   10
prcc_c               21
dtype: int64
```

In [172]:

```
df.shape
```

Out[172]:

```
(354, 13)
```

In [173]:

```
# fill na with mean and dropna
df["CURRENT ASSETS"].fillna(df["CURRENT ASSETS"].mean(),inplace=True)
df["CURRENT LIABIL"].fillna(df["CURRENT LIABIL"].mean(),inplace=True)
df.dropna(inplace=True)
```

In [174]:

```
# check null value again
df.isna().sum()
```

Out[174]:

```
TICKER               0
CURRENT ASSETS       0
TOTAL ASSETS         0
EBIT                 0
CURRENT LIABIL       0
TOTAL LIABILITIES    0
RETAINED EARNINGS    0
TOTAL SALES          0
CREDIT_RATING        0
ceq                  0
csho                 0
ni                   0
prcc_c               0
dtype: int64
```

In [175]:

```
df.describe()
```

Out[175]:

	CURRENT ASSETS	TOTAL ASSETS	EBIT	CURRENT LIABIL	TOTAL LIABILITIES	RETAINE EARNING
count	333.000000	333.000000	333.000000	333.000000	333.000000	333.00000
mean	4497.713730	13049.247916	1582.855138	3134.118965	7525.366003	4317.32429
std	8741.047669	30080.692937	4579.799671	6902.587269	20472.141607	14591.48347
min	77.343000	181.360000	-8167.000000	27.577000	48.123000	-7863.00000
25%	657.093000	1712.100000	130.325000	295.900000	911.200000	106.32500
50%	1504.000000	3618.431000	365.320000	835.569000	2032.000000	781.89100
75%	4351.700000	10021.000000	1004.201000	2636.584000	6158.000000	2426.60000
max	75777.000000	278554.000000	56939.000000	75352.000000	280860.000000	192445.00000

In [176]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 333 entries, 0 to 353
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   TICKER             333 non-null    object 
 1   CURRENT ASSETS     333 non-null    float64
 2   TOTAL ASSETS       333 non-null    float64
 3   EBIT               333 non-null    float64
 4   CURRENT LIABIL     333 non-null    float64
 5   TOTAL LIABILITIES  333 non-null    float64
 6   RETAINED EARNINGS  333 non-null    float64
 7   TOTAL SALES        333 non-null    float64
 8   CREDIT_RATING      333 non-null    int64  
 9   ceq                333 non-null    float64
 10  csho               333 non-null    float64
 11  ni                 333 non-null    float64
 12  prcc_c             333 non-null    float64
dtypes: float64(11), int64(1), object(1)
memory usage: 36.4+ KB
```

In [177]:

```
df.head(5)
```

Out[177]:

	TICKER	CURRENT ASSETS	TOTAL ASSETS	EBIT	CURRENT LIABIL	TOTAL LIABILITIES	RETAINED EARNINGS	TOTAL SALES
0	ARXX	328.354	638.022	47.473	119.215	150.352	95.273	551.846
1	ABT	11281.883	36178.172	4860.219	11951.195	22123.986	9958.494	22476.322
2	AMD	3963.000	13147.000	401.000	2852.000	7072.000	464.000	5649.000
3	APD	2612.600	11180.700	1013.500	2323.400	6078.700	5521.800	8850.400
4	HON	12304.000	30941.000	3544.000	10135.000	21221.000	11256.000	31367.000

Adding ratios columns

Re-organize the columns

calculations

In [178]:

```
# log TOTAL ASSETS & TOTAL SALES
df['TOTAL ASSETS'] = np.log(df['TOTAL ASSETS'])
df['TOTAL SALES'] = np.log(df['TOTAL SALES'])
```

In [179]:

```
df['ROA'] = df['EBIT']/df['TOTAL ASSETS']
df['Current Ratio'] = df['CURRENT ASSETS']/df['CURRENT LIABIL']
df['NET PROFIT MARGIN'] = df['EBIT'] / df['TOTAL SALES']
```

In [180]:

```
# market value:
df['market value'] = df['prcc_c'] * df['csho']
```

In [181]:

```
# price-to-sales ratio:
df['p/s_ratio'] = df['market value'] / df['TOTAL SALES']
```

In [182]:

```
# p/e ratio
df['p/e_ratio'] = df['market value'] / df['ni']
```

In [183]:

```
# m/b ratio
df['m/b_ratio'] = df['market value'] / df['ceq']
```

In [184]:

```
df.head()
```

Out[184]:

	TICKER	CURRENT ASSETS	TOTAL ASSETS	EBIT	CURRENT LIABIL	TOTAL LIABILITIES	RETAINED EARNINGS	TOTAL SALES
0	ARXX	328.354	6.458373	47.473	119.215	150.352	95.273	6.313269
1	ABT	11281.883	10.496211	4860.219	11951.195	22123.986	9958.494	10.020218
2	AMD	3963.000	9.483949	401.000	2852.000	7072.000	464.000	8.639234
3	APD	2612.600	9.321944	1013.500	2323.400	6078.700	5521.800	9.088218
4	HON	12304.000	10.339837	3544.000	10135.000	21221.000	11256.000	10.353512

In [185]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 333 entries, 0 to 353
Data columns (total 20 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   TICKER              333 non-null    object 
 1   CURRENT ASSETS      333 non-null    float64
 2   TOTAL ASSETS        333 non-null    float64
 3   EBIT                333 non-null    float64
 4   CURRENT LIABIL      333 non-null    float64
 5   TOTAL LIABILITIES   333 non-null    float64
 6   RETAINED EARNINGS   333 non-null    float64
 7   TOTAL SALES         333 non-null    float64
 8   CREDIT_RATING       333 non-null    int64  
 9   ceq                 333 non-null    float64
 10  csho                333 non-null    float64
 11  ni                  333 non-null    float64
 12  prcc_c              333 non-null    float64
 13  ROA                 333 non-null    float64
 14  Current Ratio       333 non-null    float64
 15  NET PROFIT MARGIN   333 non-null    float64
 16  market value        333 non-null    float64
 17  p/s_ratio           333 non-null    float64
 18  p/e_ratio           333 non-null    float64
 19  m/b_ratio           333 non-null    float64
dtypes: float64(18), int64(1), object(1)
memory usage: 54.6+ KB
```

Check Outliers

outliers for ROA:

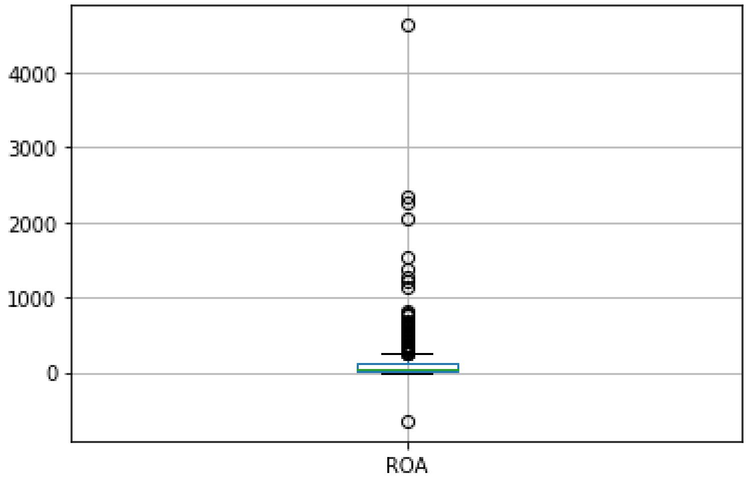
In [186]:

```
print(df['ROA'].describe())
df.boxplot(column='ROA')
```

```
count     333.000000
mean      152.035415
std       384.095796
min      -651.412682
25%        17.879442
50%        43.707471
75%       110.853527
max      4630.355686
Name: ROA, dtype: float64
```

Out[186]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f304f91d150>
```



In [187]:

```
# winsorize
df['ROA_win'] = winsorize(df['ROA'], (0.01,0.01))
```

In [188]:

```
df['ROA_win'].describe()
```

Out[188]:

```
count     333.000000
mean      144.762129
std       300.031980
min        -4.287101
25%        17.879442
50%        43.707471
75%       110.853527
max      2047.805277
Name: ROA_win, dtype: float64
```

outliers for Current Ratio:

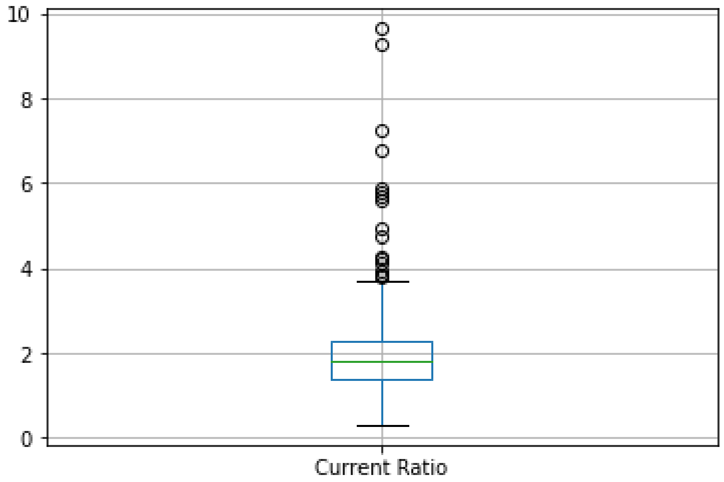
In [189]:

```
print(df['Current Ratio'].describe())
df.boxplot(column='Current Ratio')
```

```
count    333.000000
mean       2.016033
std        1.114997
min        0.304163
25%        1.363975
50%        1.798826
75%        2.306078
max        9.643438
Name: Current Ratio, dtype: float64
```

Out[189]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f304f916990>
```



In [190]:

```
# winsorize
df['Current_Ratio_win'] = winsorize(df['Current Ratio'], (0.01,0.01))
```

In [191]:

```
df['Current_Ratio_win'].describe()
```

Out[191]:

```
count    333.000000
mean       2.000952
std        1.012856
min        0.684313
25%        1.363975
50%        1.798826
75%        2.306078
max        6.795579
Name: Current_Ratio_win, dtype: float64
```

outliers for NET PROFIT MARGIN:

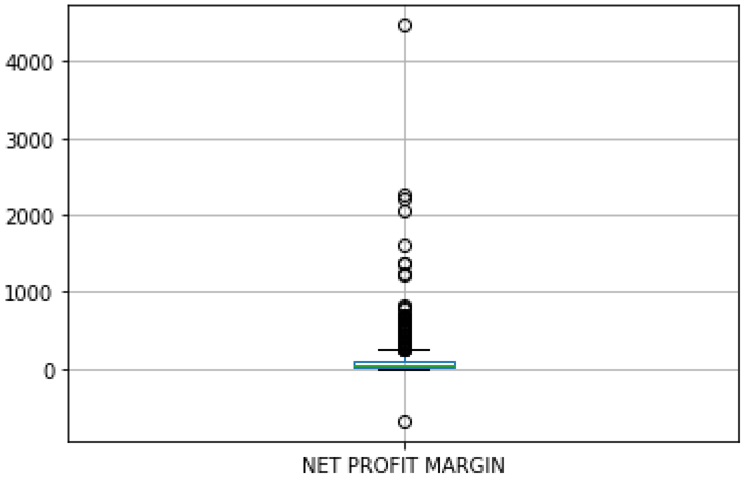
In [192]:

```
print(df['NET PROFIT MARGIN'].describe())
df.boxplot(column='NET PROFIT MARGIN')
```

```
count     333.000000
mean      152.861002
std       380.133358
min      -681.509189
25%        18.200166
50%        44.888661
75%       109.722297
max      4475.582633
Name: NET PROFIT MARGIN, dtype: float64
```

Out[192]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f304f86a190>
```



In [193]:

```
# winsorize
df['net_profit_margin_win'] = winsorize(df['NET PROFIT MARGIN'], (0.01,0.01))
```

In [194]:

```
df['net_profit_margin_win'].describe()
```

Out[194]:

```
count     333.000000
mean      146.462374
std       303.776360
min        -3.958541
25%        18.200166
50%        44.888661
75%       109.722297
max      2044.939932
Name: net_profit_margin_win, dtype: float64
```

outliers for ps_ratio :

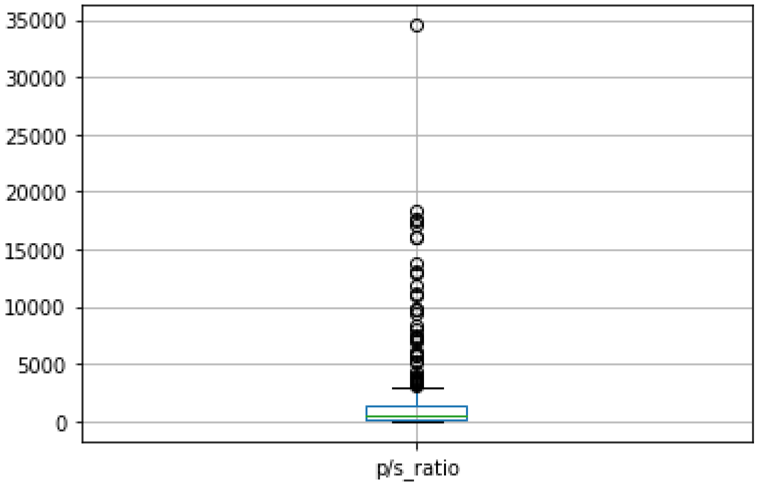
In [195]:

```
print(df['p/s_ratio'].describe())
df.boxplot(column='p/s_ratio')
```

```
count      333.000000
mean      1738.591246
std       3586.268544
min          0.042723
25%        222.628583
50%        502.108112
75%       1337.129328
max      34507.809530
Name: p/s_ratio, dtype: float64
```

Out[195]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f304f854ad0>
```



In [196]:

```
# winsorize
df['p/s_ratio_win'] = winsorize(df['p/s_ratio'], (0.01,0.01))
df['p/s_ratio_win'].describe()
```

Out[196]:

```
count      333.000000
mean      1684.581741
std       3205.375158
min         11.809683
25%        222.628583
50%        502.108112
75%       1337.129328
max      17468.113383
Name: p/s_ratio_win, dtype: float64
```

outliers for p/e_ratio :

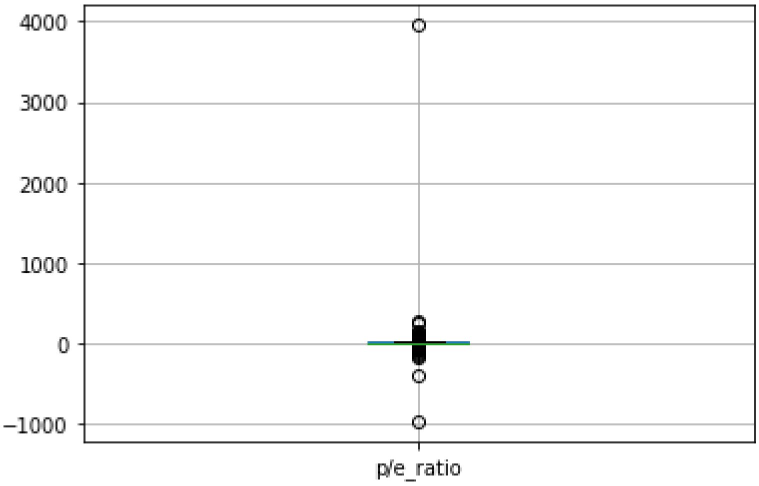
In [197]:

```
print(df['p/e_ratio'].describe())
df.boxplot(column='p/e_ratio')
```

```
count     333.000000
mean       24.596344
std       226.128783
min      -964.287735
25%        11.061894
50%        17.281376
75%        22.270221
max      3953.541714
Name: p/e_ratio, dtype: float64
```

Out[197]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f304f7d3d90>
```



In [198]:

```
# winsorize
df['p/e_ratio_win'] = winsorize(df['p/e_ratio'], (0.01,0.01))
df['p/e_ratio_win'].describe()
```

Out[198]:

```
count     333.000000
mean       15.888862
std        32.199886
min      -137.967037
25%        11.061894
50%        17.281376
75%        22.270221
max       158.593060
Name: p/e_ratio_win, dtype: float64
```

outliers for m/b_ratio :

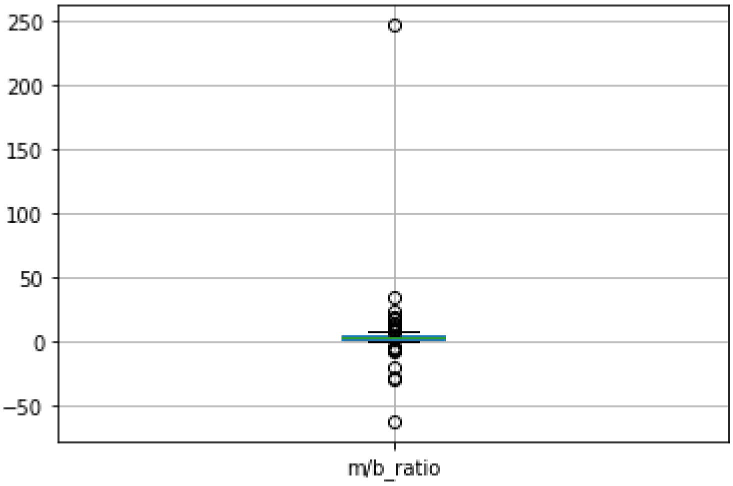
In [199]:

```
print(df['m/b_ratio'].describe())
df.boxplot(column='m/b_ratio')
```

```
count    333.000000
mean       3.715009
std       14.483983
min      -62.216453
25%        1.861362
50%        2.861326
75%        4.241607
max      246.084854
Name: m/b_ratio, dtype: float64
```

Out[199]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f304f73ccd0>
```



In [200]:

```
# winsorize
df['m/b_ratio_win'] = winsorize(df['m/b_ratio'], (0.01,0.01))
df['m/b_ratio_win'].describe()
```

Out[200]:

```
count    333.000000
mean       3.157035
std        4.003221
min      -19.768837
25%        1.861362
50%        2.861326
75%        4.241607
max       18.941696
Name: m/b_ratio_win, dtype: float64
```

outliers for logged TOTAL ASSETS:

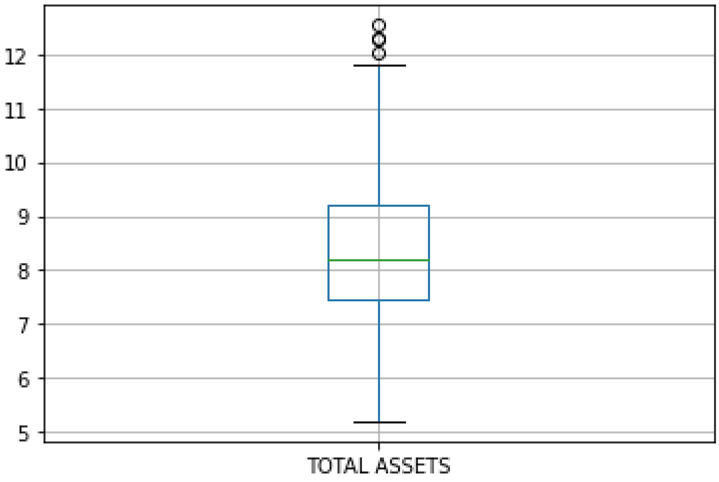
In [201]:

```
print(df['TOTAL ASSETS'].describe())
df.boxplot(column='TOTAL ASSETS')
```

```
count    333.000000
mean       8.369707
std        1.409345
min        5.200484
25%        7.445476
50%        8.193796
75%        9.212438
max       12.537367
Name: TOTAL ASSETS, dtype: float64
```

Out[201]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f304f6c5750>
```



In [202]:

```
# winsorize
df['TOTAL_ASSETS_win'] = winsorize(df['TOTAL ASSETS'], (0.01,0.01))
df['TOTAL_ASSETS_win'].describe()
```

Out[202]:

```
count    333.000000
mean       8.367891
std        1.397265
min        5.478366
25%        7.445476
50%        8.193796
75%        9.212438
max       12.012373
Name: TOTAL_ASSETS_win, dtype: float64
```

outliers for TOTAL SALES :

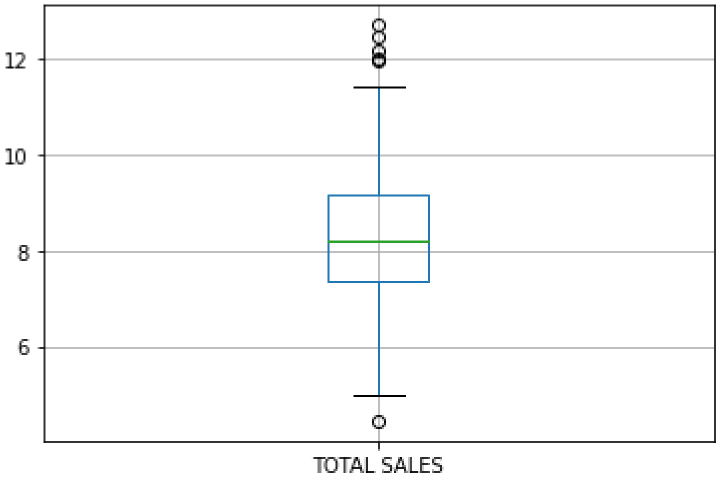
In [203]:

```
print(df['TOTAL SALES'].describe())
df.boxplot(column='TOTAL SALES')
```

```
count    333.000000
mean       8.316679
std        1.405777
min        4.433302
25%        7.370402
50%        8.208247
75%        9.169144
max       12.722142
Name: TOTAL SALES, dtype: float64
```

Out[203]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f304f6375d0>
```



In [204]:

```
# winsorize
df['TOTAL_SALES_win'] = winsorize(df['TOTAL SALES'], (0.01,0.01))
df['TOTAL_SALES_win'].describe()
```

Out[204]:

```
count    333.000000
mean       8.315864
std        1.386860
min        5.205676
25%        7.370402
50%        8.208247
75%        9.169144
max       12.029204
Name: TOTAL_SALES_win, dtype: float64
```

Linear regression

In [205]:

```
df.head(5)
```

Out[205]:

	TICKER	CURRENT ASSETS	TOTAL ASSETS	EBIT	CURRENT LIABIL	TOTAL LIABILITIES	RETAINED EARNINGS	TOTAL SALES
0	ARXX	328.354	6.458373	47.473	119.215	150.352	95.273	6.313269
1	ABT	11281.883	10.496211	4860.219	11951.195	22123.986	9958.494	10.020218
2	AMD	3963.000	9.483949	401.000	2852.000	7072.000	464.000	8.639234
3	APD	2612.600	9.321944	1013.500	2323.400	6078.700	5521.800	9.088218
4	HON	12304.000	10.339837	3544.000	10135.000	21221.000	11256.000	10.353512

5 rows × 28 columns

In [206]:

```
df['constant'] = 1
cols = ['CREDIT_RATING','ROA_win','Current_Ratio_win','net_profit_margin_win',
'p/s_ratio_win','p/e_ratio_win','m/b_ratio_win','TOTAL_ASSETS_win','TOTAL_SALES_
win','constant']
df = df[cols]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 333 entries, 0 to 353
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   CREDIT_RATING          333 non-null    int64  
 1   ROA_win                333 non-null    float64
 2   Current_Ratio_win      333 non-null    float64
 3   net_profit_margin_win  333 non-null    float64
 4   p/s_ratio_win          333 non-null    float64
 5   p/e_ratio_win          333 non-null    float64
 6   m/b_ratio_win          333 non-null    float64
 7   TOTAL_ASSETS_win       333 non-null    float64
 8   TOTAL_SALES_win        333 non-null    float64
 9   constant               333 non-null    int64  
dtypes: float64(8), int64(2)
memory usage: 28.6 KB
```

In [207]:

```
X = df.drop(columns = 'CREDIT_RATING')
y = df['CREDIT_RATING']
```

In [208]:

```
model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
====================================================================
==========
Dep. Variable:          CREDIT_RATING   R-squared:
0.555
Model:                            OLS   Adj. R-squared:
0.544
Method:                 Least Squares   F-statistic:
50.51
Date:                Fri, 08 Apr 2022   Prob (F-statistic):
1.39e-52
Time:                        01:59:30   Log-Likelihood:
-753.54
No. Observations:                 333   AIC:
1525.
Df Residuals:                     324   BIC:
1559.
Df Model:                           8
Covariance Type:            nonrobust
====================================================================
=====================
                            coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------
---------------------
ROA_win                   0.0173      0.017      1.046      0.296
-0.015       0.050
Current_Ratio_win        -0.1670      0.148     -1.129      0.260
-0.458       0.124
net_profit_margin_win    -0.0191      0.017     -1.129      0.260
-0.052       0.014
p/s_ratio_win             0.0005      0.000      3.760      0.000
0.000       0.001
p/e_ratio_win             0.0102      0.004      2.496      0.013
0.002       0.018
m/b_ratio_win             0.1121      0.033      3.358      0.001
0.046       0.178
TOTAL_ASSETS_win          0.8377      0.337      2.485      0.013
0.175       1.501
TOTAL_SALES_win           0.4011      0.344      1.165      0.245
-0.276       1.078
constant                  1.1742      1.331      0.882      0.378
-1.445       3.793
====================================================================
==========
Omnibus:                        6.520   Durbin-Watson:
1.885
Prob(Omnibus):                  0.038   Jarque-Bera (JB):
6.312
Skew:                          -0.303   Prob(JB):
0.0426
Kurtosis:                       3.295   Cond. No.
3.77e+04
====================================================================
==========

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors
is correctly specified.
[2] The condition number is large, 3.77e+04. This might indicate tha
```

```
t there are
strong multicollinearity or other numerical problems.
```

The results shows that only current ratio and net profit margin have negative coefficient with credit rating. All of other variables have positive coefficient.

Besides, both of coefficient and t-value of current ratio and net profit margin are negative, which means bad.

Next, only p/s ratio, p/e ratio, m/b ratio and total assets have p-value less than 0.05, which means statistically significant.

Finally, R-square equals to 0.555 and adj R-square equals to 0.544 means the results is good enough because the values are more than 0.5.