Assignment 10

10.1

10.1 Analyisis

The variables I chose were:

- Profitability: A fundamental measure of a firm's efficiency and its ability to generate profits relative to its revenue, assets, equity etc. Low or declining profitability is often a red flag, signaling potential financial distress.
- Tangibility: Refers to the proportion of a firm's assets that are tangible. Firms with higher tangibility might have more collateral to offer for debts, potentially lowering bankruptcy risk.
- Market-to-Book Ratio: This ratio compares a company's market value to its book value. A low ratio can indicate that a company is undervalued and potentially in financial trouble. Conversely, a very high ratio might suggest overvaluation.
- Debt-to-Asset Ratio: Indicates how much of a firm's assets are financed by debt. A high debt-to-asset ratio suggests greater financial leverage, which can increase bankruptcy risk, especially if cash flows are insufficient to meet debt obligations.
- Current Ratio: A liquidity ratio that measures a company's ability to pay short-term obligations with short-term assets. A ratio under 1 may indicate that the company is not in good financial health and might struggle to meet short-term debts.
- Interest_burden: Reflects the firm's ability to cover its interest expenses. Firms with a high interest burden relative to earnings are at increased risk of financial distress, particularly in an environment of rising interest rates.
- Leverage: Indicates the degree to which a firm is using borrowed money. Higher leverage means more debt relative to equity, which can increase the likelihood of default, especially if earnings are volatile.
- Working Capital: The difference between a firm's current assets and current liabilities. Adequate working capital is essential for a firm to meet its operational needs and to maintain solvency.
- Excess Return: Refers to the return on an investment relative to the return of a benchmark or a risk-free rate. This measure can be an indicator of how well a firm is performing compared to market expectations.

- Return: Measures the profitability of an investment. Consistently low or negative returns could indicate operational difficulties, while high returns could either suggest good performance or, if excessively high, potential accounting anomalies or risk-taking.
- Knn had the lowest misclassification rate (compared to LASSO & Ridge), and I'm not surprised because it was the model that took the longest to run.
- Survival random forest had the lowest misclassification rates of all the models (not incluing xgboost).
- XGboost has the lowest misclassifcation rate, highest accuracy & precsiion combined, and I'll be using it in 10.2

10.2 Analysis

-
- Incorporating text-based features into a model that's primarily focused on numeric financial indicators can potentially enhance its performance, especially in the context of bankruptcy prediction. Textual data can provide insights into risks and challenges that are not immediately evident in the financial figures. For instance, management discussion in 10-K filings might reveal strategic shifts, market challenges, or operational risks that numbers alone don't fully capture. Analyzing the sentiment of textual data can offer a gauge of market perceptions and internal management outlook. For example, overly optimistic or pessimistic tones in financial reports or news articles can be quantified and used as predictors.
- Text-based features should complement and enhance the existing quantitative model, rather than overshadow or contradict it without good reason. Text data needs extensive preprocessing (like tokenization, stemming, removal of stopwords) and standardization to be effectively used in a model. Textual data can be particularly useful in industries or periods where regulatory changes or technological disruptions occur, as these might not be immediately reflected in financial data.

In [ ]:

```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import gc

from scipy.stats import norm
from scipy.optimize import fsolve, newton
from sklearn.linear_model import LogisticRegression, LassoCV, Lasso, Ridge
from sklearn import metrics
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, confusion_matrix,
```

```
    roc_curve, roc_auc_score, auc, mean_squared_error, classification_report)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

import statsmodels.api as sm
from sksurv.linear_model import CoxPHSurvivalAnalysis


# Suppress warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
file_path = 'funda_2022.csv'

# Read the CSV file
funda_df = pd.read_csv(file_path, usecols = ['fyear','indfmt', 'datafmt', 'popsrc',
funda_df['CUSIP'] = funda_df['cusip'].str[0:6]
funda_df['DLC'] = funda_df['dlc'] * 1000000
funda_df['DLTT'] = funda_df['dltt'] * 1000000
funda_df = funda_df[(funda_df.indfmt == 'INDL') & (funda_df.datafmt == 'STD') & (fu
funda_df['F'] = funda_df['DLC'] + 0.5*funda_df['DLTT']
funda_df.head()

funda_df['Profitability'] = funda_df['ni']/funda_df['at']
funda_df['Profitability'].replace([np.inf, -np.inf], np.nan, inplace=True)

funda_df['Tangibility'] = funda_df['ppent']/funda_df['at']
funda_df['Tangibility'].replace([np.inf, -np.inf], np.nan, inplace=True)

funda_df['Intangibility'] = funda_df['intan']/funda_df['at']
funda_df['Intangibility'].replace([np.inf, -np.inf], np.nan, inplace=True)

funda_df['M2BRatio'] = funda_df['mkvalt']/funda_df['at']
funda_df['M2BRatio'].replace([np.inf, -np.inf], np.nan, inplace=True)

funda_df['Debt2AssetRatio'] = funda_df['lt']/funda_df['at']
funda_df['Debt2AssetRatio'].replace([np.inf, -np.inf], np.nan, inplace=True)

funda_df['CurrentRatio'] = funda_df['lct']/funda_df['act']
funda_df['CurrentRatio'].replace([np.inf, -np.inf], np.nan, inplace=True)

funda_df['Interest_burden'] = (funda_df['oiadp']-funda_df['xint'])/funda_df['oiadp'
funda_df['Interest_burden'] = funda_df['Interest_burden']/funda_df['at']
funda_df['Interest_burden'].replace([np.inf, -np.inf], np.nan, inplace=True)

funda_df['Leverage'] = funda_df['lct']/funda_df['at']
funda_df['Leverage'].replace([np.inf, -np.inf], np.nan, inplace=True)

funda_df['Working_Cap'] = funda_df['wcapc']/funda_df['at']
funda_df['Working_Cap'].replace([np.inf, -np.inf], np.nan, inplace=True)

#BAAFFM Data
path = 'BAAFFM.csv'
fred_baaffm = pd.read_csv(path, header = 0)
```

```
fred_baaffm['DATE'] = pd.to_datetime(fred_baaffm['DATE'])
fred_baaffm['DATE'] = pd.DatetimeIndex(fred_baaffm['DATE']).year
fred_baaffm = (fred_baaffm.groupby(['DATE'], as_index=False).mean().groupby('DATE')
fred_baaffm = fred_baaffm.reset_index()
fred_baaffm = fred_baaffm.rename(columns= {"DATE":"fyear"})
fred_baaffm.head()

#Daily CRSP data
file_path = 'dsf_new.csv'

chunk_size=10000

percentage = 0.25  # 25% of each chunk to save memory
dataframes = []
cols = ["DATE","CUSIP","PERMNO","RET","PRC","SHROUT","SHRCD"]
for chunk in pd.read_csv(file_path, chunksize=chunk_size, usecols=cols):
    sample = chunk.sample(frac=percentage)  # Sample each chunk
    dataframes.append(sample)

# Concatenate all sampled chunks
dsf_df = pd.concat(dataframes, ignore_index=True)
del dataframes  # Free memory
gc.collect()


dsf_df['RET']= pd.to_numeric(dsf_df.RET, errors='coerce')
dsf_df['PRC'] = abs(dsf_df.PRC)
dsf_df = dsf_df[dsf_df.SHRCD.isin([10, 11])]
dsf_df = dsf_df.drop("SHRCD",axis=1)
dsf_df['CUSIP'] = dsf_df['CUSIP'].str[0:6]
dsf_df['DATE'] =pd.to_datetime(dsf_df.DATE,format="%Y%m%d")
dsf_df['YEAR'] = pd.DatetimeIndex(dsf_df['DATE']).year
dsf_df['Market_Cap'] = dsf_df.PRC*dsf_df.SHROUT


vol = (dsf_df.groupby(by=['CUSIP','PERMNO','YEAR'])['RET'].std()*np.sqrt(250)).to_f
vol.columns = ['CUSIP','PERMNO','YEAR','sigmaE']


annual_ret = dsf_df.groupby(by=['CUSIP','PERMNO','YEAR']).apply(lambda x:np.exp(np.
annual_ret.columns = ['CUSIP','PERMNO','YEAR','RET']

market_cap = dsf_df.groupby(by=['CUSIP','PERMNO','YEAR'])['Market_Cap'].first().to_
market_cap.columns = ['CUSIP','PERMNO','YEAR','Market_Cap']

csrp_annual = vol.merge(annual_ret,how='inner',on=['CUSIP','PERMNO','YEAR']).merge(
del vol
del annual_ret
del market_cap
gc.collect()  # Explicitly clean memory

# Merge BAAFFM with funda/COMPUSTAT
funda_df = funda_df.merge(fred_baaffm, how='inner', on=['fyear'])
funda_data = funda_df[["fyear","CUSIP","BAAFFM","Profitability","Tangibility","Inta
                        "M2BRatio","Debt2AssetRatio","CurrentRatio","Working_Cap",
funda_data['fyear'] = pd.to_numeric(funda_data.fyear)
```

```
#Bankruptcy data
file_path = 'BR1964_2019.xlsx'
bankruptcy_data = pd.read_excel(file_path)
bankruptcy_data['YEAR'] = pd.DatetimeIndex(bankruptcy_data['bankruptcy_dt']).year
bankruptcy_data['Bankruptcy'] = 1
bankruptcy_data = bankruptcy_data.dropna().drop(columns = ['bankruptcy_dt'])
bankruptcy_data['YEAR']= bankruptcy_data['YEAR']-1

funda_data['YEAR'] = funda_data['fyear']
data_to_train = csrp_annual.merge(funda_data,how='inner',on=['CUSIP','YEAR'])
data_to_train['YEAR']= data_to_train.groupby(by=['CUSIP','PERMNO']).YEAR.shift(-1)
data_to_train = data_to_train.merge(bankruptcy_data,how='left',on=['PERMNO','YEAR']
data_to_train['Bankruptcy'] = data_to_train.Bankruptcy.fillna(0)
data_to_train.dropna(how='all', axis=1, inplace=True)

#save for later
data_to_train.to_csv("data_to_train.csv",index=False)
```

In [ ]:
```
def plot_matrix_and_print_metrics(y_test, pred_test):

    con_mat = confusion_matrix(y_test, pred_test)
    con_mat = pd.DataFrame(con_mat, range(2), range(2))

    plt.figure(figsize=(4,4))
    sns.heatmap(con_mat, annot=True, annot_kws={"size": 12}, fmt='g',cmap='Reds')

    print("Accuracy:", metrics.accuracy_score(y_test, pred_test))
    print("Precision:", metrics.precision_score(y_test, pred_test))
    print("ROC score:", metrics.roc_auc_score(y_test, pred_test))
```

In [ ]:
```
data_to_train = pd.read_csv('data_to_train.csv', header = 0)
```

In [ ]:
```
data_to_train.fillna(method='bfill', inplace=True)
data_to_train.replace([np.inf, -np.inf], np.nan, inplace=True)
data_to_train.fillna(0, inplace = True)
data_to_train = data_to_train.dropna()
display(data_to_train)
```

	CUSIP	PERMNO	YEAR	sigmaE	RET	Market_Cap	fyear	BAAFFM	Profital
0	000307	14945	2015.0	0.519389	1.340821	401056.2000	2014	4.765000	0.05
1	000307	14945	2016.0	1.136931	0.870253	855254.4000	2015	4.866667	0.03
2	000307	14945	2017.0	0.646156	0.937478	210227.8700	2016	4.322500	-0.00
3	000307	14945	1992.0	0.595891	0.981847	216941.9200	2017	3.613750	-0.04
4	000360	76868	1992.0	0.678064	0.555399	7268.0795	1991	4.114167	0.02
...	...	...	...	...	...	...	...	...	
114682	U72603	13705	2013.0	0.610966	0.979973	162371.8000	2012	4.795000	-0.16
114683	U72603	13705	2014.0	0.534445	0.493802	133729.0800	2013	4.994167	0.00
114684	U72603	13705	2015.0	0.697032	1.058804	111293.7000	2014	4.765000	0.09
114685	U72603	13705	2016.0	0.492954	0.987910	126383.0000	2015	4.866667	0.23
114686	U72603	13705	0.0	0.521604	0.998508	159267.3000	2016	4.322500	0.42

114687 rows × 17 columns

In [ ]:
```python
# Convert the 'year' column from float to int to remove the decimal part
data_to_train['YEAR'] = data_to_train['YEAR'].astype(int)
data_to_train['YEAR'] = data_to_train['YEAR'].astype(str)

# Remove rows with year as 0 or '0'
data_to_train = data_to_train[(data_to_train['YEAR'] != 0) & (data_to_train['YEAR']

data_to_train['YEAR'] = pd.to_datetime(data_to_train['YEAR'], format='%Y')

# Filter for the period between 1964 and 2019 (inclusive)
data_to_train = data_to_train[(data_to_train['YEAR'] >= pd.to_datetime('1964')) & (

data_to_train['YEAR'] = data_to_train['YEAR'].dt.year
data_to_train['Bankruptcy'] = data_to_train['Bankruptcy'].astype(int)
display(data_to_train)
```

	CUSIP	PERMNO	YEAR	sigmaE	RET	Market_Cap	fyear	BAAFFM	Profitab
0	000307	14945	2015	0.519389	1.340821	401056.2000	2014	4.765000	0.051
1	000307	14945	2016	1.136931	0.870253	855254.4000	2015	4.866667	0.033
2	000307	14945	2017	0.646156	0.937478	210227.8700	2016	4.322500	-0.001
3	000307	14945	1992	0.595891	0.981847	216941.9200	2017	3.613750	-0.048
4	000360	76868	1992	0.678064	0.555399	7268.0795	1991	4.114167	0.026
...	...	...	...	...	...	...	...	...	
114681	989929	83582	2013	0.554610	1.061672	95938.0800	2005	2.850833	-0.313
114682	U72603	13705	2013	0.610966	0.979973	162371.8000	2012	4.795000	-0.162
114683	U72603	13705	2014	0.534445	0.493802	133729.0800	2013	4.994167	0.000
114684	U72603	13705	2015	0.697032	1.058804	111293.7000	2014	4.765000	0.093
114685	U72603	13705	2016	0.492954	0.987910	126383.0000	2015	4.866667	0.232

114521 rows × 17 columns

In [ ]:

```
file_path = 'DTB3.csv'
fed_df = pd.read_csv(file_path)

fed_df['DTB3'] = pd.to_numeric(fed_df['DTB3'], errors='coerce')
fed_df['r'] = np.log(1 + fed_df['DTB3'] / 100)
fed_df['YEAR'] =pd.to_datetime(fed_df['DATE']).dt.year
fed_df = fed_df.dropna()

display(fed_df)

fed_funda_df = pd.merge(data_to_train, fed_df, on = ['YEAR'], how = 'inner')
fed_funda_df['r'] = np.log(1 + fed_funda_df['DTB3'] / 100)
fed_funda_df['Excess_Return'] = fed_funda_df['RET'] - fed_funda_df['r']
fed_funda_df.to_csv("data.csv",index=False)
```

	DATE	DTB3	r	YEAR
0	1970-01-02	7.92	0.076220	1970
1	1970-01-05	7.91	0.076127	1970
2	1970-01-06	7.93	0.076313	1970
3	1970-01-07	7.90	0.076035	1970
4	1970-01-08	7.91	0.076127	1970
...	...	...	...	...
13820	2022-12-23	4.23	0.041430	2022
13822	2022-12-27	4.35	0.042580	2022
13823	2022-12-28	4.35	0.042580	2022
13824	2022-12-29	4.34	0.042485	2022
13825	2022-12-30	4.30	0.042101	2022

13241 rows × 4 columns

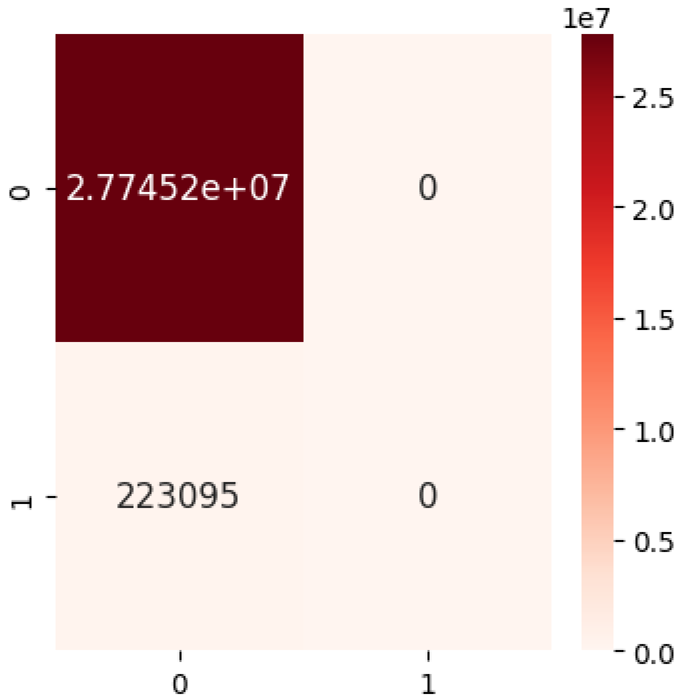
In [ ]:
```
data = pd.read_csv('data.csv', header = 0)
data['YEAR'] = pd.to_datetime(data['YEAR'], format='%Y')
data['YEAR'] = data['YEAR'].dt.year
```

In [ ]:
```
features = ['RET', 'Market_Cap', 'BAAFFM', 'Profitability', 'Tangibility', 'M2BRati
       'Debt2AssetRatio', 'CurrentRatio', 'Working_Cap', 'Leverage', 'Excess_Return
target = ['Bankruptcy']

X_train = data[features]
Y_train = np.ravel(data[target])
```

Training Logistic Model & Insample Testing

In [ ]:
```
Y_train = Y_train.reshape(-1,1)
model = LogisticRegression(solver='lbfgs', class_weight='balanced')
model.fit(X_train, Y_train)

Y_pred = model.predict(X_train)
plot_matrix_and_print_metrics(Y_train,Y_pred)
```

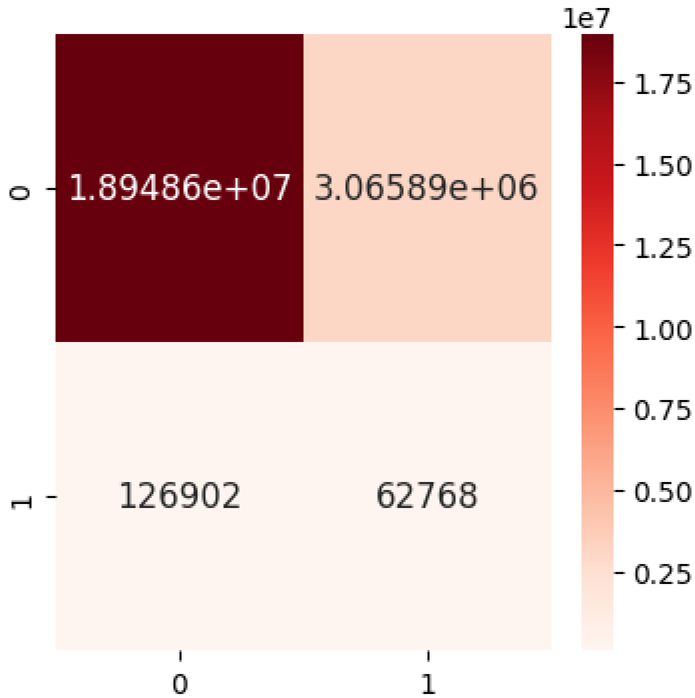
```
Accuracy: 0.992023277249387
Precision: 0.0
ROC score: 0.5
```



Logistic Regression Out of Sample

In [ ]:
```
train = data.copy()
test = data.copy()
out_of_sample_train = train[train.YEAR.between(1964, 1990)]
out_of_sample_test = test[test.YEAR.between(1990, 2020)]

X_train = out_of_sample_train[features]
X_test = out_of_sample_test[features]
Y_train = out_of_sample_train[target]
Y_test = out_of_sample_test[target]

model = LogisticRegression(solver='newton-cg', class_weight='balanced', max_iter =
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)

plot_matrix_and_print_metrics(Y_test,Y_pred)
```

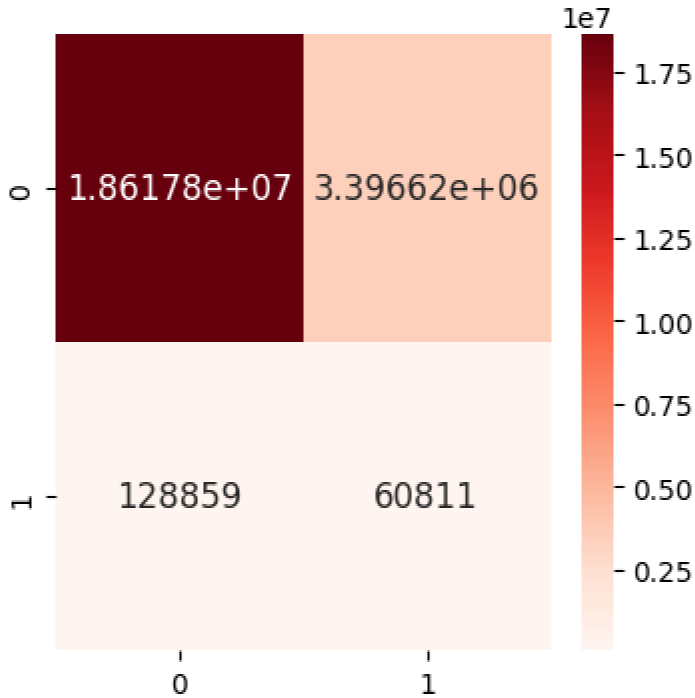
```
Accuracy: 0.856207420095067
Precision: 0.020062288727172306
ROC score: 0.5958328507228872
```



Lasso Regression

In [ ]:
```
train = data.copy()
test = data.copy()
out_of_sample_train = train[train.YEAR.between(1964, 1990)]
out_of_sample_test = test[test.YEAR.between(1990, 2020)]
X_train = out_of_sample_train[features]
X_test = out_of_sample_test[features]
Y_train = out_of_sample_train[target]
Y_test = out_of_sample_test[target]

model = Lasso(alpha = 0.0005).fit(X_train, Y_train)
score = model.score(X_train, Y_train)
y_predicted = model.predict(X_test)

coefficients = model.coef_
importance = np.abs(coefficients)
new_features = ['RET', 'Profitability', 'BAAFFM', 'CurrentRatio', 'M2BRatio', 'Debt
       'Leverage', 'Excess_Return']
X_train = out_of_sample_train[new_features]
X_test = out_of_sample_test[new_features]
model = LogisticRegression(solver='newton-cg', class_weight='balanced', max_iter =
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)
plot_matrix_and_print_metrics(Y_test,Y_pred)
```

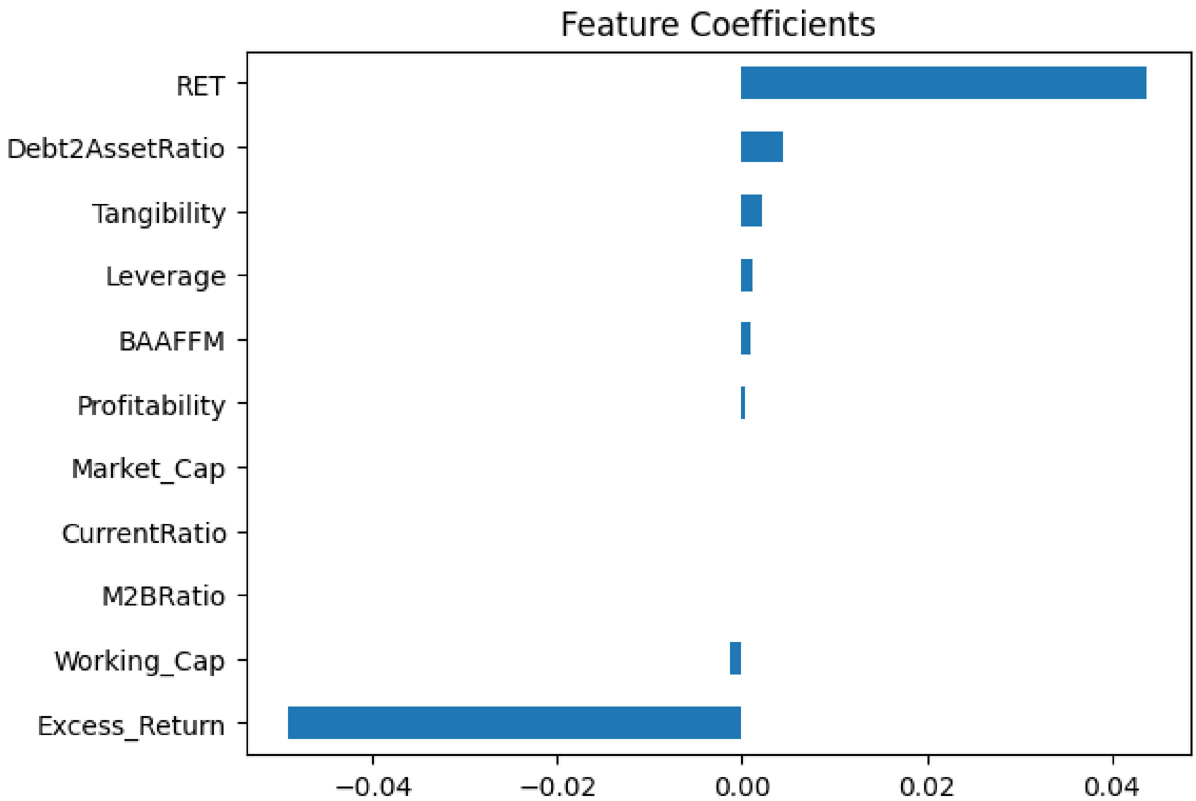
```
Accuracy: 0.8412243106419452
Precision: 0.01758850290201187
ROC score: 0.5831622387667318
```



Ridge Regression

In [ ]:
```
train = data.copy()
test = data.copy()
out_of_sample_train = train[train.YEAR.between(1964, 1990)]
out_of_sample_test = test[test.YEAR.between(1990, 2020)]
X_train = out_of_sample_train[features]
X_test = out_of_sample_test[features]
Y_train = out_of_sample_train[target]
Y_test = out_of_sample_test[target]
model = Ridge(alpha = 50).fit(X_train, Y_train)
score = model.score(X_train, Y_train)
y_predicted = model.predict(X_test)

coef = pd.Series(model.coef_[0], index = X_train.columns)
imp_coef = coef.sort_values()
imp_coef.plot(kind = "barh")
plt.title("Feature Coefficients")
```

Out[ ]:
```
Text(0.5, 1.0, 'Feature Coefficients')
```

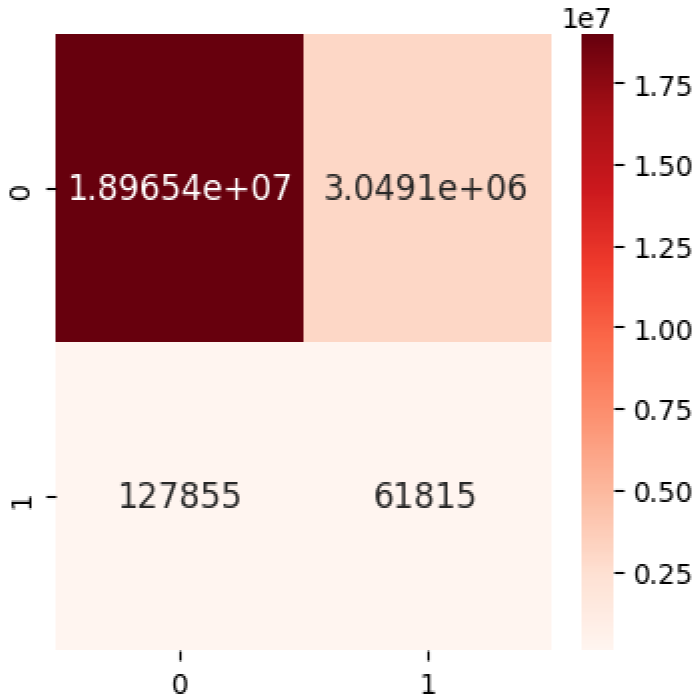


In [ ]:
```
coefficients = model.coef_
coeffs = np.abs(coefficients)
print(coeffs)
```

```
[[4.38211654e-02 1.72953507e-10 8.68230125e-04 2.84826856e-04
  2.25386027e-03 5.11021097e-06 4.53304539e-03 1.30428558e-06
  1.41188317e-03 1.20323219e-03 4.90638117e-02]]
```

In [ ]:
```
new_features = ['RET','Excess_Return', 'BAAFFM' , 'Profitability', 'Tangibility', '
X_train = out_of_sample_train[new_features]
X_test = out_of_sample_test[new_features]
model = LogisticRegression(solver='newton-cg', class_weight='balanced', max_iter =
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)
plot_matrix_and_print_metrics(Y_test,Y_pred)
```

```
Accuracy: 0.8569204854446937
Precision: 0.01987034691057331
ROC score: 0.5937018419015896
```



KNN

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

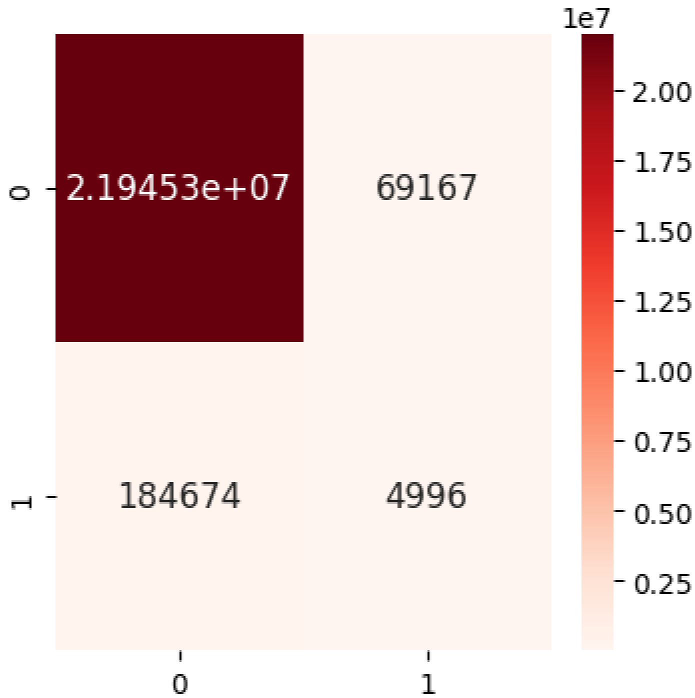
In [ ]:
```
gc.collect()
```

Out[ ]: 2675

In [ ]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(X_train_scaled, Y_train)
Y_pred = knn.predict(X_test_scaled)
```

In [ ]:
```
plot_matrix_and_print_metrics(Y_test,Y_pred)
```

```
Accuracy: 0.9885678506022482
Precision: 0.06736512816363956
ROC score: 0.511599298748285
```



Random Forest

In [ ]:
```
features = ['CUSIP', 'PERMNO', 'YEAR', 'RET', 'Market_Cap', 'BAAFFM', 'Profitabilit
train = data.copy()
test = data.copy()
out_of_sample_train = train[train.YEAR.between(1964, 1990)]
out_of_sample_test = test[test.YEAR.between(1990, 2020)]
X_train = out_of_sample_train[features]
X_test = out_of_sample_test[features]
Y_train = out_of_sample_train[target]
Y_test = out_of_sample_test[target]
```

In [ ]:
```
param_grid = {'n_estimators': [200,400, 600, 1000],
               'max_depth': [5,10,20]}

rf = RandomForestClassifier()
rf_grid = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2)
rf_grid.fit(X_train, Y_train)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

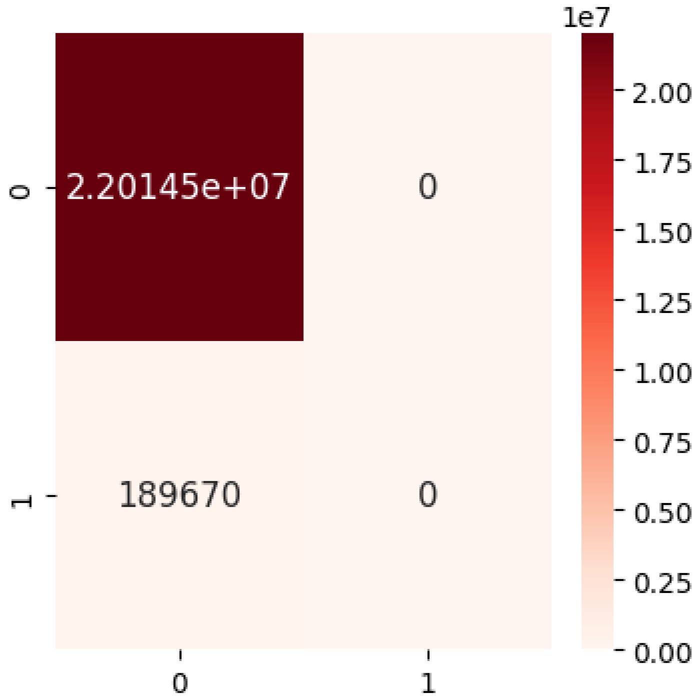
Out[ ]:
```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
  ▸ RandomForestClassifier
```

In [ ]:
```
gc.collect()
```

Out[ ]:
```
1022
```

In [ ]:
```
model = RandomForestClassifier(n_estimators = 400, random_state = 42,max_depth=5)
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)
plot_matrix_and_print_metrics(Y_test,Y_pred)
```

```
Accuracy: 0.9914578977538239
Precision: 0.0
ROC score: 0.5
```



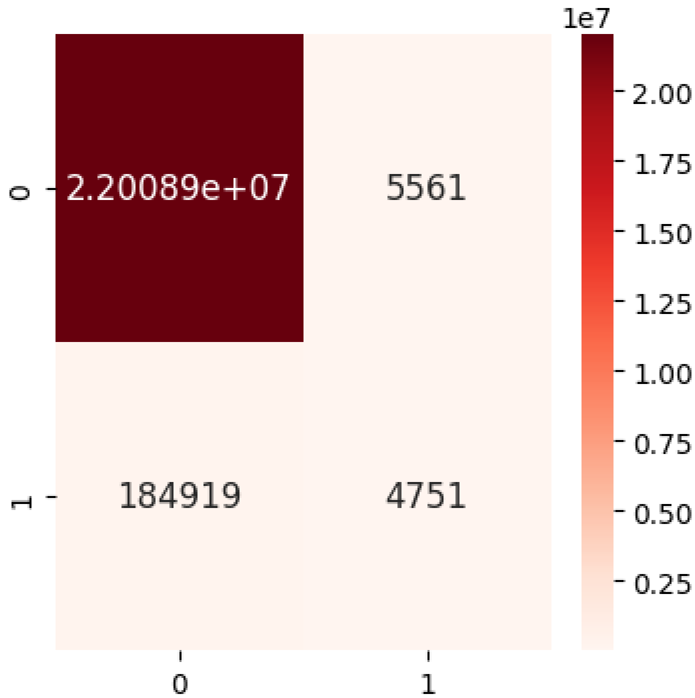
XGBoost

In [ ]:
```
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import StratifiedKFold
```

In [ ]:
```
model = XGBClassifier()

param_grid = dict(n_estimators=[100,200,300])
kfold = StratifiedKFold(n_splits=3, shuffle=True, random_state=7)

grid_search = GridSearchCV(model, param_grid, scoring="neg_log_loss", n_jobs=-1, cv
grid_result = grid_search.fit(X_train, Y_train)
y_pred = grid_search.predict(X_test.values)
```

In [ ]:
```
plot_matrix_and_print_metrics(Y_test, y_pred)
```

```
Accuracy: 0.9914214180637337
Precision: 0.4607253685027153
ROC score: 0.5123980811437485
```



In [ ]:
```
# Define a range of `max_depth` values to test
param_grid = {'max_depth': [10, 20, 30, 40, 50]}

# Initialize the LGBMClassifier
lgbm = LGBMClassifier()

# Setup GridSearchCV
grid_search = GridSearchCV(estimator=lgbm, param_grid=param_grid, scoring='accuracy

# Fit GridSearchCV
grid_search.fit(X_train, Y_train)

# Print best parameter
print("Best max_depth:", grid_search.best_params_)
# Using the best parameter from the grid search
best_max_depth = grid_search.best_params_['max_depth']
lgbm_optimal = LGBMClassifier(max_depth=best_max_depth)

# Fit the model
lgbm_optimal.fit(X_train, Y_train)

# Predict on out-of-sample data
predictions = lgbm_optimal.predict(X_test)
from sklearn.metrics import accuracy_score

# Calculate the accuracy
accuracy = accuracy_score(Y_test, predictions)

# Misclassification rate
misclassification_rate = 1 - accuracy
print("Misclassification rate of LightGBM:", misclassification_rate)
```

10.2

In [ ]:
```
#Best performing model is XGBoost
# Convert y_pred to a pandas Series for easier handling, if it isn't already
y_pred_series = pd.Series(y_pred, index=Y_test.index)

# Identify False Negatives
false_negatives = (Y_test['Bankruptcy'] == 1) & (y_pred_series == 0)

# Extract samples of False Negatives
false_negative_samples = X_test[false_negatives]
```

In [ ]:
```

```

In [ ]:
```
false_negative_samples = false_negative_samples.sort_values(by='YEAR')
false_negative_samples = false_negative_samples[false_negative_samples['YEAR'] >= 2
display(false_negative_samples)
```

	CUSIP	PERMNO	YEAR	RET	Market_Cap	BAAFFM	Profitability	Tangibility
10008028	550260	87259	2000	0.913804	797729.25	2.900833	-0.113362	0.01555
10008029	550260	87259	2000	0.913804	797729.25	2.900833	-0.113362	0.01555
10008030	550260	87259	2000	0.913804	797729.25	2.900833	-0.113362	0.01555
10008035	550260	87259	2000	0.913804	797729.25	2.900833	-0.113362	0.01555
10008032	550260	87259	2000	0.913804	797729.25	2.900833	-0.113362	0.01555
...	...	...	...	...	...	...	...	.
1625069	269279	91109	2017	0.546896	347658.27	4.322500	-0.340570	0.54189
1625070	269279	91109	2017	0.546896	347658.27	4.322500	-0.340570	0.54189
1625071	269279	91109	2017	0.546896	347658.27	4.322500	-0.340570	0.54189
1625063	269279	91109	2017	0.546896	347658.27	4.322500	-0.340570	0.54189
2011899	871639	13167	2017	1.033642	788198.52	4.322500	-2.210379	0.00660

127310 rows × 14 columns

In [ ]:
```
file_path = 'funda_2022.csv'

# Read the CSV file
funda_df = pd.read_csv(file_path, usecols = ['cik', 'cusip', 'datadate', 'dlc', 'dl
funda_df['CUSIP'] = funda_df['cusip'].str[0:6]
```

In [ ]:
```
# List of columns you want to keep
columns_to_keep = ['CUSIP', 'cik']
```

```
# Creating a new DataFrame with only the specified columns
funda_df = funda_df[columns_to_keep]
```

In [ ]:
```
false_negative_samples = pd.merge(funda_df, false_negative_samples, on='CUSIP', how
```

In [ ]:
```
display(false_negative_samples)
```

	CUSIP	cik	PERMNO	YEAR	RET	Market_Cap	BAAFFM	Profitability
0	02376R	6201.0	21020	2010	1.406792	1921595.68	7.135833	-0.057709
1	02376R	6201.0	21020	2010	1.406792	1921595.68	7.135833	-0.057709
2	02376R	6201.0	21020	2010	1.406792	1921595.68	7.135833	-0.057709
3	02376R	6201.0	21020	2010	1.406792	1921595.68	7.135833	-0.057709
4	02376R	6201.0	21020	2010	1.406792	1921595.68	7.135833	-0.057709
...	...	...	...	...	...	...	...	...
2016045	67086E	1355128.0	93355	2012	0.868453	293836.60	5.562500	-0.049252
2016046	67086E	1355128.0	93355	2012	0.868453	293836.60	5.562500	-0.049252
2016047	67086E	1355128.0	93355	2012	0.868453	293836.60	5.562500	-0.049252
2016048	67086E	1355128.0	93355	2012	0.868453	293836.60	5.562500	-0.049252
2016049	67086E	1355128.0	93355	2012	0.868453	293836.60	5.562500	-0.049252

2016050 rows × 15 columns

In [ ]:
```
#save for later
false_negative_samples.to_csv("false_negatives.csv",index=False)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 2
      1 #save for later
----> 2 false_negative_samples.to_csv("false_negatives.csv",index=False)

NameError: name 'false_negative_samples' is not defined
```

In [ ]:
```
false_negatives = pd.read_csv('false_negatives.csv', header = 0)
```

In [ ]:
```
unique_ciks = false_negatives['cik'].unique()
```

In [ ]:
```
import os
import pandas as pd
import json
import nltk
from nltk.tokenize import sent_tokenize
from transformers import BertTokenizer, BertForSequenceClassification
```

```
from torch.nn.functional import softmax
import torch
```

```
C:\Users\mhlad\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfr
a8p0\LocalCache\local-packages\Python310\site-packages\tqdm\auto.py:21: TqdmWarning:
IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.re
adthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [ ]:

```
# Download NLTK models (if not already downloaded)
nltk.download('punkt')

# Load pre-trained FinBERT model
model_name = 'yiyanghkust/finbert-tone'
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name)

def finbert_sentiment(sentence, tokenizer, model):
    inputs = tokenizer(sentence, return_tensors='pt')
    outputs = model(**inputs)
    probs = softmax(outputs.logits, dim=-1)
    sentiment = torch.argmax(probs, dim=-1).numpy()[0]
    sentiments = ['positive', 'neutral', 'negative']
    return sentiments[sentiment]
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\mhlad\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
vocab.txt: 100%|██████████| 226k/226k [00:00<00:00, 2.64MB/s]
config.json: 100%|██████████| 533/533 [00:00<?, ?B/s]
pytorch_model.bin: 100%|██████████| 439M/439M [01:57<00:00, 3.75MB/s]
```

In [ ]:

```
import os
import pandas as pd
import json

# Directory containing the files
folder_path = '10k'

# Iterate through files in the folder
for file_name in os.listdir(folder_path):
    for cik in unique_ciks:
        # Check if the file name starts with the CIK followed by an underscore
        if file_name.startswith(str(cik) + '_'):
            file_path = os.path.join(folder_path, file_name)
            try:
                # Read the JSON file
                with open(file_path, 'r') as file:
                    json_data = json.load(file)

                # Extract relevant text for analysis
                    # (Modify this based on your JSON structure and the sections yo
                    text_content = json_data.get('Text_Section', '')  # Example key

                    # Tokenize into sentences
                    sentences = sent_tokenize(text_content)
```

```
                # Perform sentiment analysis
                sentence_sentiments = [(sentence, finbert_sentiment(sentence, t

                # Process the data (e.g., store in DataFrame, print, or upload)
                # Example: print each sentence with its sentiment
                for sentence, sentiment in sentence_sentiments:
                    print(f"Sentence: {sentence}, Sentiment: {sentiment}")
        except (json.JSONDecodeError, FileNotFoundError):
            print(f"Error processing file: {file_name}")
```

In [ ]:
```
# LABEL_2 is positive, LABEL_1 is neutral, LABEL_0 is negative
from transformers import pipeline
from transformers import AutoTokenizer, AutoModelForSequenceClassification
# Load tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("ipuneetrathore/bert-base-cased-finetuned
model = AutoModelForSequenceClassification.from_pretrained("ipuneetrathore/bert-bas
# Initialize sentiment classifier
classifier = pipeline('sentiment-analysis', model=model, tokenizer=tokenizer, frame
# Analyze sentiments of example sentences
results = classifier([
'The revenue increased above and beyond over last quarter.',
'We did poorly on sales',
'The inflation increased over last year.','It is a normal trading day'
], batch_size=2, truncation="only_first")
# Print results
print(results)
```

```
tokenizer_config.json: 100%|██████████| 40.0/40.0 [00:00<?, ?B/s]
config.json: 100%|██████████| 1.29k/1.29k [00:00<?, ?B/s]
vocab.txt: 100%|██████████| 213k/213k [00:00<00:00, 4.20MB/s]
special_tokens_map.json: 100%|██████████| 112/112 [00:00<?, ?B/s]
pytorch_model.bin: 100%|██████████| 433M/433M [01:54<00:00, 3.79MB/s]
[{'label': 'LABEL_2', 'score': 0.9999865293502808}, {'label': 'LABEL_0', 'score': 0.
9999785423278809}, {'label': 'LABEL_2', 'score': 0.9999114274978638}, {'label': 'LAB
EL_1', 'score': 0.999990701675415}]
```

In [ ]:
```
def document_sentiment_measure(sentences, classifier):
    # Analyze sentiments of the sentences
    results = classifier(sentences, batch_size=2, truncation="only_first")

    # Count positive and negative sentiments
    positive_count = sum(1 for result in results if result['label'] == 'LABEL_2')
    negative_count = sum(1 for result in results if result['label'] == 'LABEL_0')
    total_count = len(sentences)

    # Calculate the document-level sentiment measure
    measure = (positive_count - negative_count) / total_count
    return measure

# Example usage
sentences = [
    'The revenue increased above and beyond over last quarter.',
    'We did poorly on sales',
    'The inflation increased over last year.',
    'It is a normal trading day'
```

```
]

measure = document_sentiment_measure(sentences, classifier)
print(f"Document Sentiment Measure: {measure}")
```

```
Document Sentiment Measure: 0.25
```

In [ ]:
```
display(false_negatives)
```

	CUSIP	cik	PERMNO	YEAR	RET	Market_Cap	BAAFFM	Profitability
0	02376R	6201.0	21020	2010	1.406792	1921595.68	7.135833	-0.057709
1	02376R	6201.0	21020	2010	1.406792	1921595.68	7.135833	-0.057709
2	02376R	6201.0	21020	2010	1.406792	1921595.68	7.135833	-0.057709
3	02376R	6201.0	21020	2010	1.406792	1921595.68	7.135833	-0.057709
4	02376R	6201.0	21020	2010	1.406792	1921595.68	7.135833	-0.057709
...	...	...	...	...	...	...	...	...
2016045	67086E	1355128.0	93355	2012	0.868453	293836.60	5.562500	-0.049252
2016046	67086E	1355128.0	93355	2012	0.868453	293836.60	5.562500	-0.049252
2016047	67086E	1355128.0	93355	2012	0.868453	293836.60	5.562500	-0.049252
2016048	67086E	1355128.0	93355	2012	0.868453	293836.60	5.562500	-0.049252
2016049	67086E	1355128.0	93355	2012	0.868453	293836.60	5.562500	-0.049252

2016050 rows × 15 columns

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming your data is in a DataFrame named false_negatives

# Select only numeric columns for descriptive statistics
numeric_cols = false_negatives.select_dtypes(include=['number'])

# Calculate descriptive statistics for each numeric column
descriptive_stats = numeric_cols.describe()

# Calculate skewness and kurtosis for numeric columns
skewness = numeric_cols.skew()
kurtosis = numeric_cols.kurt()

# Print the results
print(descriptive_stats)
print("\nSkewness:\n", skewness)
print("\nKurtosis:\n", kurtosis)

# Plotting trends over time for numeric columns
try:
```

```
    grouped_data = numeric_cols.groupby(false_negatives['YEAR']).mean()
    grouped_data.plot(subplots=True, layout=(4, 4), figsize=(15, 10), title="Trend
    plt.show()
except Exception as e:
    print("An error occurred while plotting:", e)
```

```
                cik        PERMNO          YEAR           RET    Market_Cap  \
count  2.001284e+06  2.016050e+06  2.016050e+06  2.016050e+06  2.016050e+06
mean   8.241302e+05  7.182886e+04  2.005961e+03  9.543100e-01  9.591706e+05
std    3.981966e+05  2.553209e+04  5.241627e+00  4.854798e-01  3.307531e+06
min    6.201000e+03  1.011400e+04  2.000000e+03  1.003981e-01  1.453582e+03
25%    7.570110e+05  7.603700e+04  2.001000e+03  7.046061e-01  5.048952e+04
50%    9.133640e+05  8.175600e+04  2.005000e+03  8.996058e-01  1.736000e+05
75%    1.065910e+06  8.725300e+04  2.010000e+03  1.073836e+00  6.516973e+05
max    1.724965e+06  9.339800e+04  2.017000e+03  6.411307e+00  4.012746e+07

             BAAFFM  Profitability   Tangibility      M2BRatio  \
count  2.016050e+06   2.016050e+06  2.016050e+06  2.016050e+06
mean   3.768737e+00  -3.023750e-01  3.152593e-01  1.002436e+00
std    1.661211e+00   1.812934e+00  2.700115e-01  3.770232e+00
min    1.463333e+00  -6.119655e+01  0.000000e+00  2.341553e-03
25%    2.128333e+00  -2.261569e-01  6.951733e-02  1.062102e-01
50%    4.060000e+00  -6.723197e-02  2.392651e-01  2.487671e-01
75%    5.044167e+00  -8.975268e-04  5.421082e-01  7.764250e-01
max    7.135833e+00   5.442113e-01  9.651410e-01  8.979184e+01

       Debt2AssetRatio  CurrentRatio   Working_Cap      Leverage  \
count     2.016050e+06  2.016050e+06  2.016050e+06  2.016050e+06
mean      8.030775e-01  1.015700e+00  8.541834e-02  3.232623e-01
std       4.952809e-01  1.581737e+00  2.039625e-01  3.492403e-01
min       4.980762e-02  2.148171e-02 -5.444191e-01  1.034340e-03
25%       5.578288e-01  4.010244e-01 -5.985634e-03  1.427440e-01
50%       7.535551e-01  6.640908e-01  4.036290e-02  2.337802e-01
75%       9.202741e-01  1.074586e+00  1.393553e-01  3.906222e-01
max       4.995110e+00  2.054364e+01  9.736716e-01  4.099939e+00

       Excess_Return
count   2.016050e+06
mean    9.316036e-01
std     4.843042e-01
min     7.333646e-02
25%     6.729443e-01
50%     8.786270e-01
75%     1.053898e+00
max     6.395139e+00

Skewness:
 cik                -0.823927
PERMNO             -1.568816
YEAR                0.535124
RET                 3.638771
Market_Cap          8.539333
BAAFFM              0.162627
Profitability     -28.637944
Tangibility         0.595679
M2BRatio           16.240639
Debt2AssetRatio     3.597523
CurrentRatio        7.314262
Working_Cap         1.431070
Leverage            5.168353
Excess_Return       3.645800
dtype: float64
```

```
Kurtosis:
 cik                -0.194541
PERMNO               0.835298
YEAR                -0.949417
RET                 28.034440
Market_Cap          89.151660
BAAFFM              -1.138671
Profitability      945.932247
Tangibility         -0.816503
M2BRatio           351.042363
Debt2AssetRatio     21.192259
CurrentRatio        71.085531
Working_Cap          4.070290
Leverage            41.513030
Excess_Return       27.945935
dtype: float64
```