```
import sys
!{sys.executable} -m pip install fundamentalanalysis
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting fundamentalanalysis
  Downloading fundamentalanalysis-0.2.14-py3-none-any.whl (12 kB)
Installing collected packages: fundamentalanalysis
Successfully installed fundamentalanalysis-0.2.14
```

```
# Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

import yfinance as yf
import fundamentalanalysis as fa

# installing additional package for desriptive statistics in html format
!pip install ydata_profiling

import ydata_profiling as ypr
from ydata_profiling import ProfileReport
from IPython.display import IFrame

import statsmodels.api as sm
import statsmodels.formula.api as smf
from scipy import stats
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tsa.stattools import adfuller

from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting ydata_profiling
  Downloading ydata_profiling-4.3.0-py2.py3-none-any.whl (352 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 352.9/352.9 kB 6.8 MB/s eta 0:00:00
Requirement already satisfied: scipy<1.11,>=1.4.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pandas!=1.4.0,<2.1,>1.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: matplotlib<4,>=3.2 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pydantic<2,>=1.8.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.10/dist-
Collecting visions[type_image_path]==0.7.5 (from ydata_profiling)
  Downloading visions-0.7.5-py3-none-any.whl (102 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 102.7/102.7 kB 10.1 MB/s eta 0:00:00
Requirement already satisfied: numpy<1.24,>=1.16.0 in /usr/local/lib/python3.10/dist-
Collecting htmlmin==0.1.12 (from ydata_profiling)
```

```
✓ 0s completed at 3:31 PM
Collecting phik<0.13,>=0.11.1 (from ydata_profiling)
  Downloading phik-0.12.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 679.5/679.5 kB 25.6 MB/s eta 0:00:00
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: seaborn<0.13,>=0.10.1 in /usr/local/lib/python3.10/dis
Collecting multimethod<2,>=1.4 (from ydata_profiling)
  Downloading multimethod-1.9.1-py3-none-any.whl (10 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.10/di
Collecting typeguard<3,>=2.13.2 (from ydata_profiling)
  Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Collecting imagehash==4.3.1 (from ydata_profiling)
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 296.5/296.5 kB 30.8 MB/s eta 0:00:00
Collecting wordcloud>=1.9.1 (from ydata_profiling)
  Downloading wordcloud-1.9.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 455.4/455.4 kB 37.8 MB/s eta 0:00:00
Collecting dacite>=1.8 (from ydata_profiling)
  Downloading dacite-1.8.1-py3-none-any.whl (14 kB)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/dist-packag
Collecting tangled-up-in-unicode>=0.0.4 (from visions[type_image_path]==0.7.5->ydata_
  Downloading tangled_up_in_unicode-0.2.0-py3-none-any.whl (4.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.7/4.7 MB 88.0 MB/s eta 0:00:00
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.10/
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
```

Multicollinearity

Demonstration

For our demonstration, we will use the fundamentalanalysis library from by JerBouma. We will

extract financial statements and compute the ratios for a few of the line items. The ratios that we are going to take into account are quick ratio and current ratio

```
# API KEY

api_key = "c86bfa9376e4c574e4228028e6e9ad4f"


# Company Ticker
ticker = "SNEX"

 # Collect general company information
profile_cs = fa.profile(ticker, api_key)


# Collect the Balance Sheet statements
balance_sheet_quarter_cs = fa.balance_sheet_statement(ticker, api_key, period="quarter")

balance_sheet_columns = balance_sheet_quarter_cs.columns

balance_sheet_quarter_cs.index.name = 'lineitem'

df = balance_sheet_quarter_cs.T

# We are only going to do this analysis for the period March 2017(included) to December 202
filtered_df = df.iloc[1:25]
filtered_df =  filtered_df[["totalCurrentAssets", "totalCurrentLiabilities", "cashAndCashEc
filtered_df["current_ratio"] = filtered_df["totalCurrentAssets"] / filtered_df["totalCurrer
filtered_df["quick_ratio"] = filtered_df["cashAndCashEquivalents"] / filtered_df["totalCurr

filtered_df = filtered_df.apply(pd.to_numeric)

vif = pd.DataFrame()
vif["features"] = filtered_df.columns
vif["vif_Factor"] = [variance_inflation_factor(filtered_df.values, i) for i in range(len(fi
print(vif)

                     features  vif_Factor
     0       totalCurrentAssets  216.269876
     1  totalCurrentLiabilities   83.357819
     2   cashAndCashEquivalents  176.150203
     3            current_ratio   52.095059
     4              quick_ratio   69.683160
```

VIF Factor is greater than 5.

```
filtered_df.head()
```

lineitem	totalCurrentAssets	totalCurrentLiabilities	cashAndCashEquivalents	currer
2022-12	11693900000	11041800000	1252100000	
2022-09	12303100000	11452700000	1108500000	
2022-06	14089800000	11448000000	1363700000	
2022-03	14495300000	10852800000	1299700000	
2021-12	12316600000	8697900000	983400000	

```
fundamental_data = filtered_df

profile = ProfileReport(fundamental_data)
profile.to_file('fundamental_data.html')
```

Summarize dataset: 100% 39/39 [00:11<00:00, 1.76it/s, Completed]

Generate report structure: 100% 1/1 [00:06<00:00, 6.74s/it]

Render HTML: 100% 1/1 [00:01<00:00, 1.29s/it]

Export report to file: 100% 1/1 [00:00<00:00, 20.00it/s]

Diagrams

```
profile
```

Overview

Dataset statistics

Number of variables	5
Number of observations	24
Missing cells	0
Missing cells (%)	0.0%

Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	1.1 KiB
Average record size in memory	48.0 B

Variable types

Numeric	5

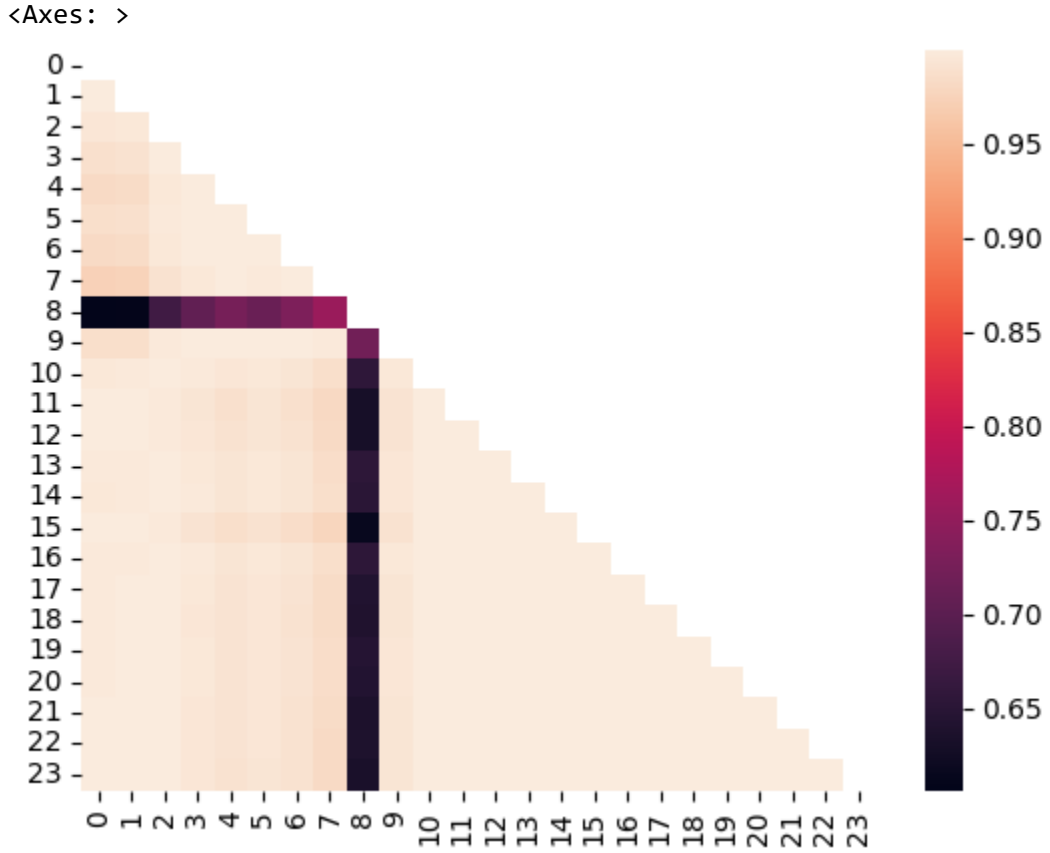
Alerts

`totalCurrentAssets` is highly overall correlated with `totalCurrentLiabilities` and 1 other fields (totalCurrentLiabilities, cashAndCashEquivalents)	**High correlation**
`totalCurrentLiabilities` is highly overall correlated with `totalCurrentAssets` and 1 other fields (totalCurrentAssets, cashAndCashEquivalents)	**High correlation**
`cashAndCashEquivalents` is highly overall correlated with	**High correlation**

```
test_fa_data = fundamental_data
test_fa_data.index = pd.to_datetime(test_fa_data.index)

test_fa_data.head()
```

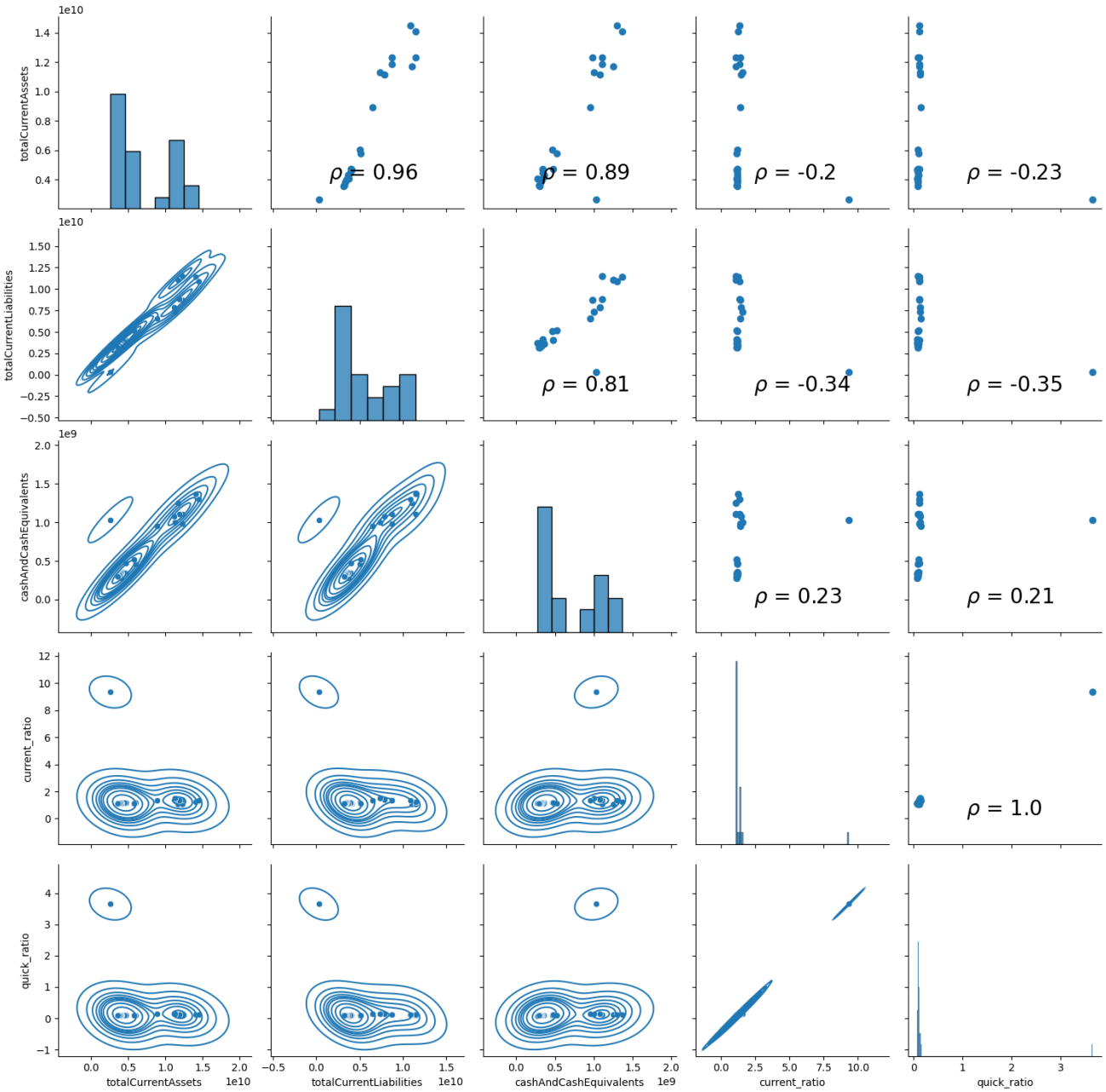
lineitem	totalCurrentAssets	totalCurrentLiabilities	cashAndCashEquivalents	curr
2022-12-01	11693900000	11041800000	1252100000	
2022-09-01	12303100000	11452700000	1108500000	
2022-06-01	14089800000	11448000000	1363700000	
2022-03-01	14495300000	10852800000	1299700000	
2021-12-01	12316600000	8697900000	983400000	

```
# Compute the correlation matrix
corr = np.corrcoef(test_fa_data.reset_index(drop=True))

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Create the heatmap
sns.heatmap(corr, mask=mask)
```




```
# Function to calculate correlation coefficient between two arrays


def corr(x, y, **kwargs):
    # Calculate the value
    coef = np.corrcoef(x, y)[0][1]
    # Make the label
    label = r"$\rho$ = " + str(round(coef, 2))

    # Add the label to the plot
    ax = plt.gca()
    ax.annotate(label, xy=(0.3, 0.15), size=20, xycoords=ax.transAxes)

# Create the default pairplot
grid = sns.pairplot(
    test_fa_data.reset_index(drop=True),
    height=3
)

# Map a scatter plot and Pearson correlation coefficient to the upper triangle
grid = grid.map_upper(plt.scatter)
grid = grid.map_upper(corr)

# Map a histogram to the diagonal
# grid = grid.map_diag(plt.hist)
```

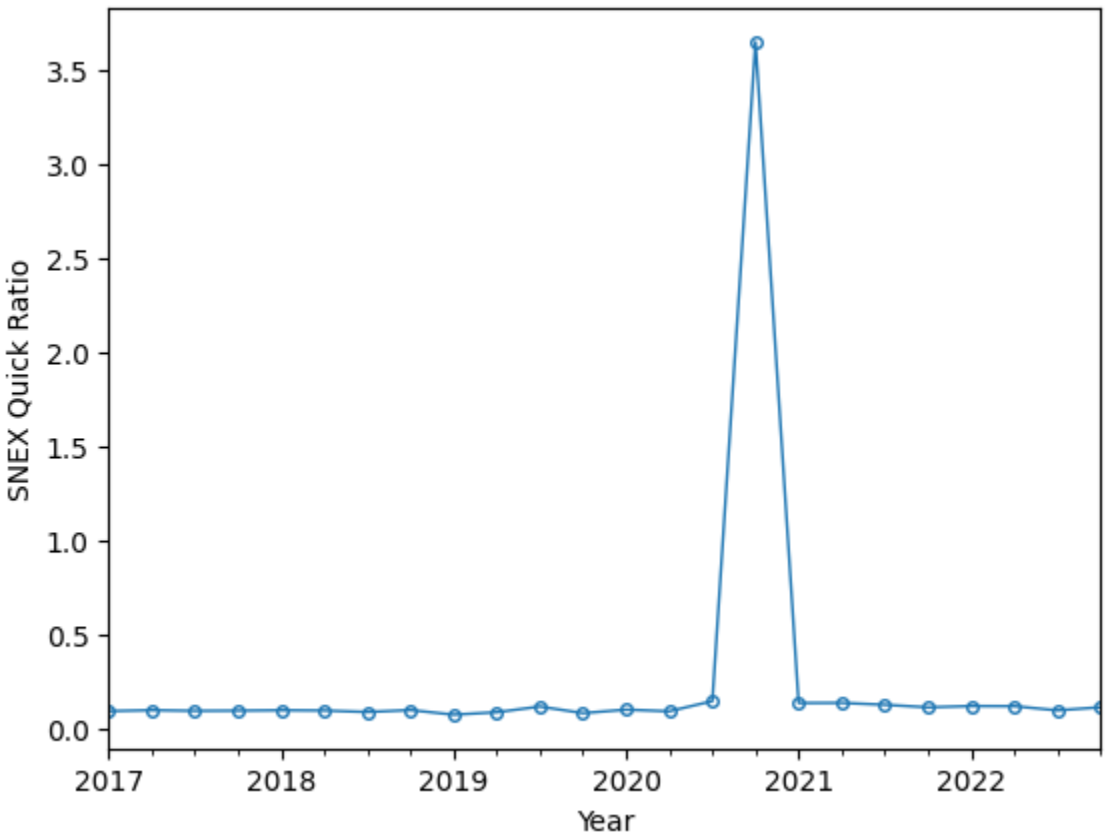
```
# Map a density plot to the lower triangle
grid = grid.map_lower(sns.kdeplot)
```



```
test_fa_data.columns
```

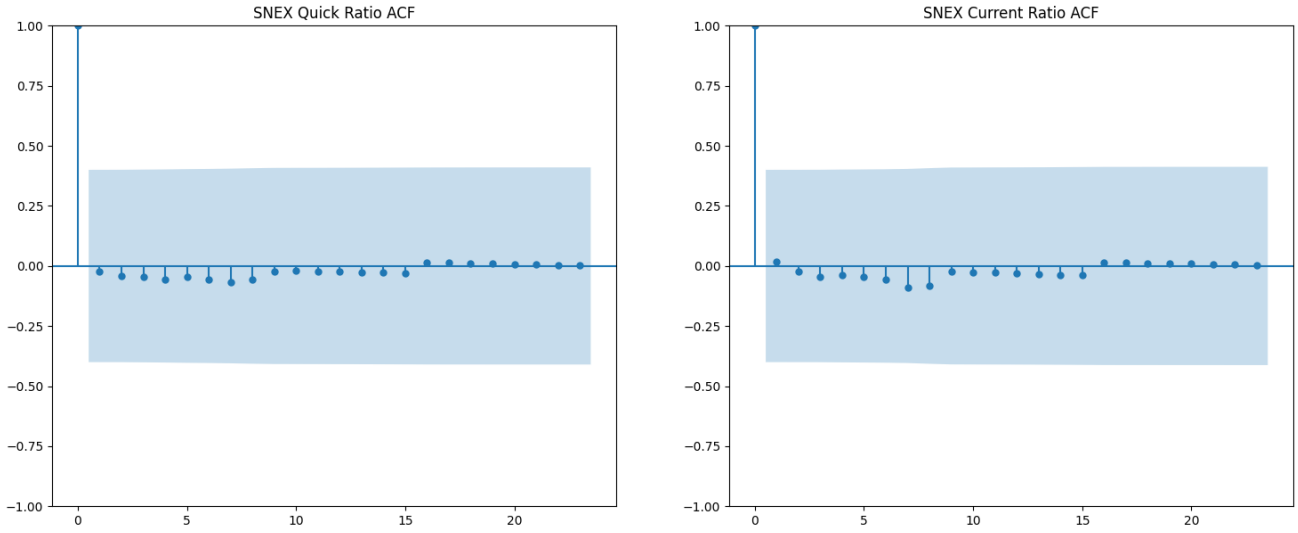
```
Index(['totalCurrentAssets', 'totalCurrentLiabilities',
       'cashAndCashEquivalents', 'current_ratio', 'quick_ratio'],
      dtype='object', name='lineitem')
```

```
# Plot SNEX Quick Ratio time series chart
test_fa_data['quick_ratio'].plot(
    marker="o",
    markersize=4,
    markerfacecolor="none",
    linestyle="-",
    linewidth=1,
    xlabel="Year",
    ylabel="SNEX Quick Ratio",
)
plt.show()
```



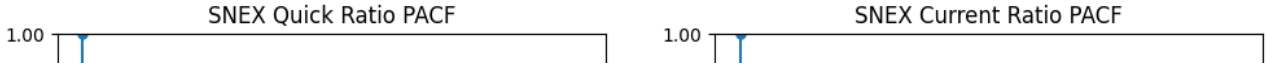

```
# ACF Plots for SNEX's Quick Ratio and Current Ratio
fig, ax = plt.subplots(1, 2, figsize=(18, 7))
sm.graphics.tsa.plot_acf(test_fa_data['quick_ratio'],
                         title="SNEX Quick Ratio ACF",
                         lags=23,
```

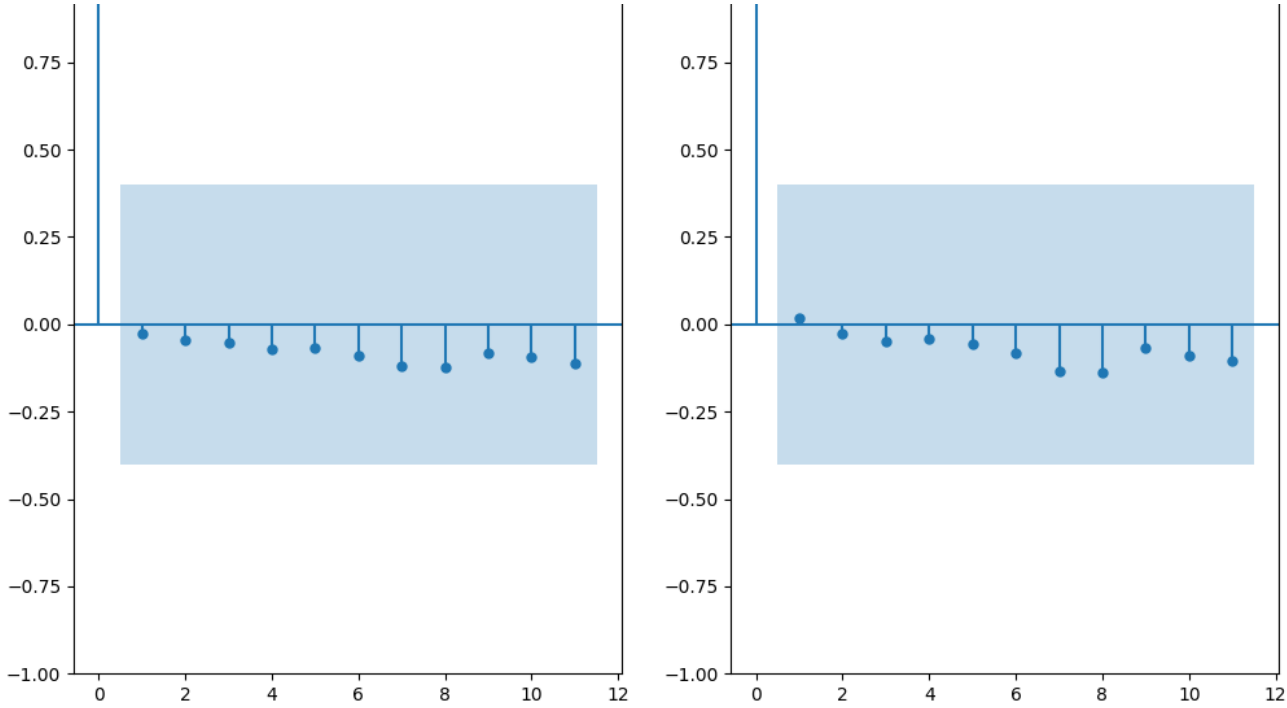
```
                          ax=ax[0])
sm.graphics.tsa.plot_acf(
    test_fa_data['current_ratio'],
    title="SNEX Current Ratio ACF",
    lags=23,
    ax=ax[1]
)

plt.show()
```




```
# PACF plots for SNEX's Quick Ratio and Current Ratio
fig, ax = plt.subplots(1, 2, figsize=(12, 7))
sm.graphics.tsa.plot_pacf(test_fa_data['quick_ratio'],
                          title="SNEX Quick Ratio PACF",
                          lags=11,
                          ax=ax[0])
sm.graphics.tsa.plot_pacf(
    test_fa_data['current_ratio'],
    title="SNEX Current Ratio PACF",
    lags=11,
    ax=ax[1]
)

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/graphics/tsaplots.py:348: FutureW
  warnings.warn(
```





With the plots above we can come to the conclusion that an ARMA(1,1) model is suitable for this data. Since "quick_ratio" variable is derived from the ratio of "cashAndCashEquivalents" and "totalCurrentLiabilities", it contains information from both of the variables, so they can be droped. This is why only the variable: "quic_ratio" can be used for additional data experimentation with time further series analysis and forecasting by ARMA models.

Unit Root Testing

Demonstration

```
# Get data for this ticker
tickerData = yf.Ticker(ticker)

# Get the historical prices for this ticker
tickerDf = tickerData.history(period='1d', start='2017-01-01', end='2022-12-31')
```

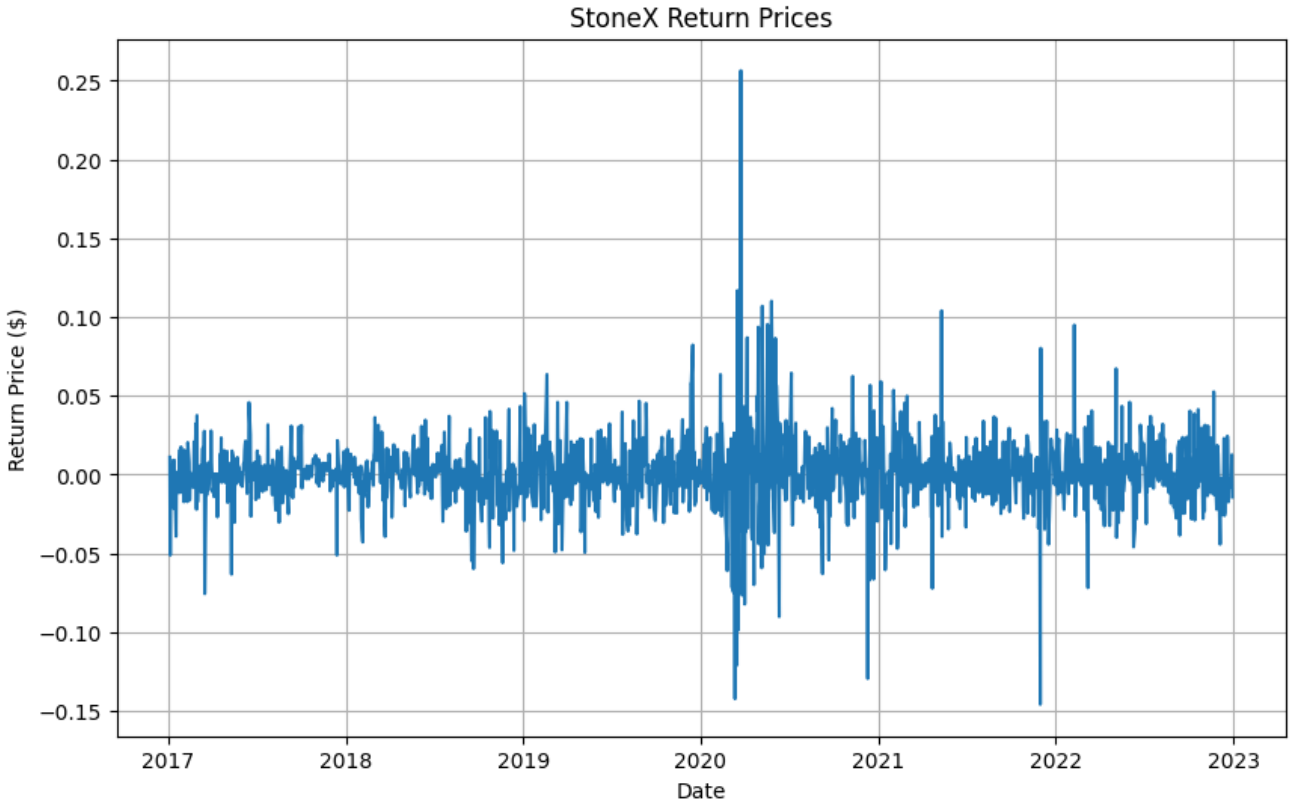
```
tickerDf = tickerData.history(period='1d', start='2017-01-01', end='2022-12-31')

# See your data
tickerDf

# Plot time series
plt.figure(figsize=(10, 6))
plt.plot(tickerDf['Close'])
plt.title('StoneX Closing Prices')
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.grid(True)
plt.show()

# Perform ADF test
result = adfuller(tickerDf['Close'])
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {round(result[1],3)}')
```


```
ADF Statistic: -0.3148789236861879
p-value: 0.923
```

p-value is greater than 0.05, then we conclude the data is non stationary, the process contains unit root

Diagrams

```
# Apply log transformation
# tickerDf['LogClose'] = np.log(tickerDf['Close'])
tickerDf['Return'] = tickerDf['Close'].pct_change()

# Plot the return
plt.figure(figsize=(10, 6))
plt.plot(tickerDf['Return'])
plt.title('StoneX Return Prices')
plt.xlabel('Date')
plt.ylabel('Return Price ($)')
plt.grid(True)
plt.show()

# Perform ADF test on log-transformed series
result = adfuller(tickerDf['Return'].dropna())
print(f'ADF Statistic (Return series): {result[0]}')
print(f'p-value (Return series): {round(result[1],18)}')
```




```
ADF Statistic (Return series): -10.324974800186709
```

```
p-value (Return series): 3e-18
```

Feature Extraction

Demonstration

```
test_fa_data.columns
```

```
Index(['totalCurrentAssets', 'totalCurrentLiabilities',
       'cashAndCashEquivalents', 'current_ratio', 'quick_ratio'],
      dtype='object', name='lineitem')
```

```
#from sklearn.linear_model import Lasso
#from sklearn.preprocessing import StandardScaler

# Load the data
data = test_fa_data
X = data[['totalCurrentAssets',
          'totalCurrentLiabilities',
          'cashAndCashEquivalents',
          'current_ratio']]

y = data['quick_ratio']

# Standardize the data
scaler = StandardScaler()
X_std = scaler.fit_transform(X)

# Create a lasso regression object
lasso = Lasso(alpha=0.1)

# Fit the model
lasso.fit(X_std, y)

# Display the coefficients
print(lasso.coef_)
```

```
[-0.         -0.          0.          0.60748044]
```

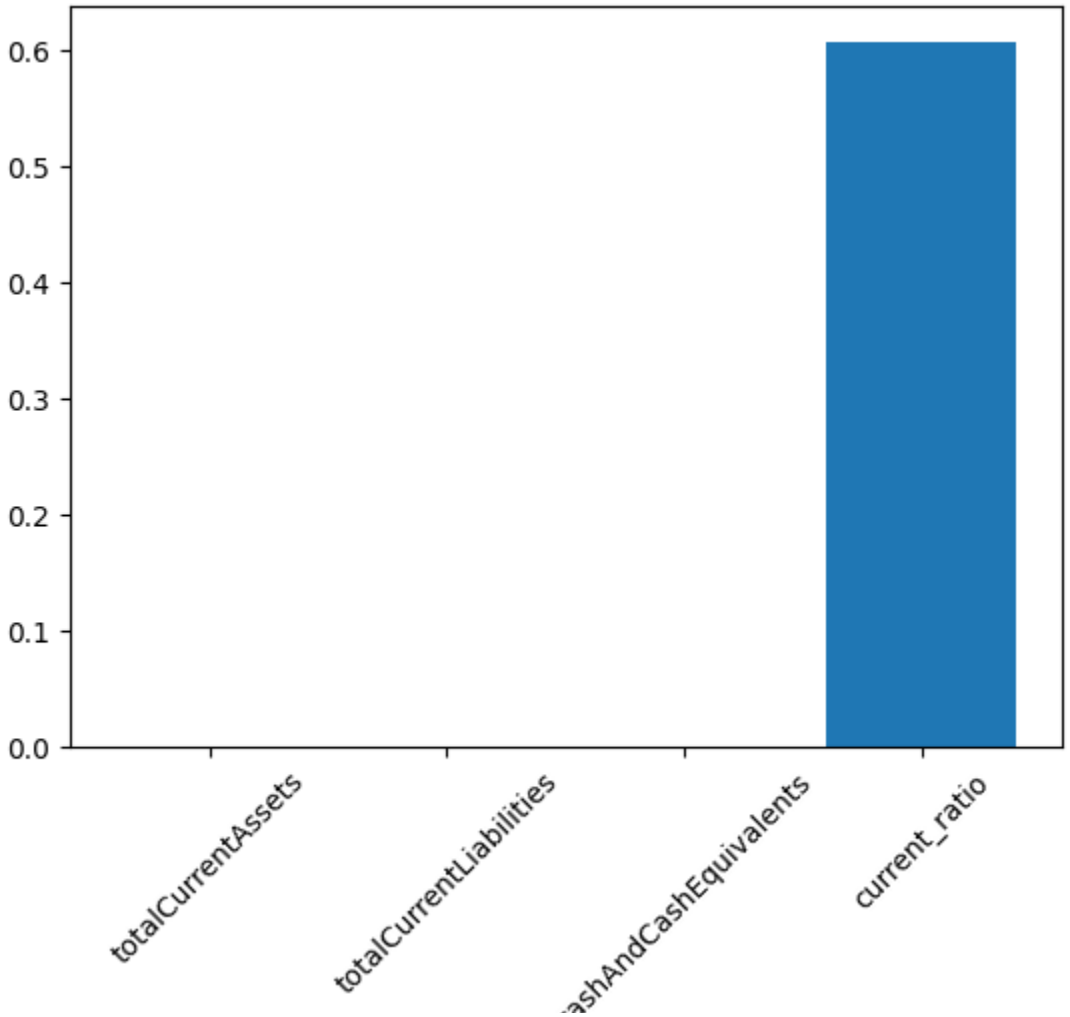
```
# looking for the most important feature

names = X.columns

print(np.sum(lasso.coef_ != 0))

print([names[i] for i in range(len(names)) if lasso.coef_[i] != 0])
```

```
1
['current_ratio']
```

Diagrams

Double-click (or enter) to edit

```
# visualizing the feature importance by Lasso regression and thus feature extraction
plt.bar(test_fa_data.columns.drop('quick_ratio'), lasso.coef_)
plt.xticks(rotation=45)
plt.show()
```



Double-click (or enter) to edit