Project: Multifactors Risk Research of China Stock Listed Banks

---

1. Problem Description & Goal

Weighted Average Cost of Capital (WACC) is probably the most important financial indicator to assess the load of a company to raise capital. It means the average rate of return a company is expected to pay its investors (both equity holders and debt holders) for using their capital. It reflects the company's cost of financing its assets, weighted by the proportion of equity and debt in its capital structure. The formula is:

$$r_{waac} = \frac{E}{E+D} \times r_e + \frac{D}{E+D} \times r_d \times (1-t)$$

Where:

- E: Company's market value of equity
- D: Company's market value of debt
- r_e: Company's cost rate of equity, i.e. rate of return a company is expected to pay its equity holders
- r_d: Company's cost rate of debt, i.e. rate of return a company is expected to pay its debt holders
- t: Corporate tax rate

Furthermore, the risky part (***risk premium***) of WAAC means the risky load a company afford in order to raise capital with an expected average rate of return that exceed market interest rates, as defined:

$$r^*_{waac} = r_{waac} - r_f$$

Where r_f: Market risk-free interest rate, often replaced by a proxy: Shanghai Interbank Offered Rate (SHIBOR)

This project aims to explore how the risks hidden in different financial factors impact the risk of Chinese bank industry to raise capital, namely different financial factors' causal effects on the r^*_{waac}.

2. Dataset Description

1. Define Modeler Class for Data Cleaning

2. Instantiate the Modeler

The modeler imports data as DataFrames from pickles and align them into `Modeler.factors_datas`.

In [2]:

```
#from Modeler import Modeler

mode = input('Dataset is for train or test?')
mol = Modeler(mode)
```

2.1. Explore Original Data Structure

The `Modeler.factors_datas` includes 7 categories of financial factors (factors) about all listed banks in China, as the names shown:

- `'factors_data'` : Factors of banks' performance in stocks market in technical analyisis perspective.
- `'fundamentals_data'` : Factors from financial statement of banks.
- `'macros_data'` : Factors of macroeconomic enviroment.
- `'securities_margins_data'` : Factors of banks' margin trading.
- `'industries_data'` : Factors of bank's industry type: The bank is is a national or regional?
- `'indexes_data'` : Factor of total number of enterprises appearing in various composite indices.

In [3]:

```
mol.factors_datas.keys()
```

Out[3]:

```
dict_keys(['factors_data', 'fundamentals_data', 'macros_data', 'money_flows_data', 'sec
urities_margins_data', 'industries_data', 'indexes_data'])
```

Each category dict include many factors data table, take the category 'factors_data' for example:

In [4]:

```
mol.factors_datas['factors_data'].keys()
```

Out[4]:

```
dict_keys(['administration_expense_ttm', 'asset_impairment_loss_ttm', 'cash_flow_to_price_ratio', 'EBIT', 'EBITDA', 'financial_assets', 'interest_free_current_liability', 'market_cap', 'net_debt', 'net_finance_cash_flow_ttm', 'net_interest_expense', 'net_invest_cash_flow_ttm', 'net_operate_cash_flow_ttm', 'net_profit_ttm', 'non_operating_net_profit_ttm', 'non_recurring_gain_loss', 'np_parent_company_owners_ttm', 'OperateNetIncome', 'operating_assets', 'operating_profit_ttm', 'operating_revenue_ttm', 'retained_earnings', 'sales_to_price_ratio', 'total_operating_cost_ttm', 'total_operating_revenue_ttm', 'total_profit_ttm', 'value_change_profit_ttm', 'AR', 'ARBR', 'ATR14', 'ATR6', 'BR', 'DAVOL10', 'DAVOL20', 'DAVOL5', 'MAWVAD', 'money_flow_20', 'PSY', 'turnover_volatility', 'TVMA20', 'TVMA6', 'TVSTD20', 'TVSTD6', 'VDEA', 'VDIFF', 'VEMA10', 'VEMA12', 'VEMA26', 'VEMA5', 'VMACD', 'VOL10', 'VOL120', 'VOL20', 'VOL240', 'VOL5', 'VOL60', 'VOSC', 'VR', 'VROC12', 'VROC6', 'VSTD10', 'VSTD20', 'WVAD', 'financing_cash_growth_rate', 'net_asset_growth_rate', 'net_operate_cashflow_growth_rate', 'net_profit_growth_rate', 'np_parent_company_owners_growth_rate', 'operating_revenue_growth_rate', 'PEG', 'total_asset_growth_rate', 'total_profit_growth_rate', 'arron_down_25', 'arron_up_25', 'BBIC', 'bear_power', 'BIAS10', 'BIAS20', 'BIAS5', 'BIAS60', 'bull_power', 'CCI10', 'CCI15', 'CCI20', 'CCI88', 'CR20', 'fifty_two_week_close_rank', 'MASS', 'PLRC12', 'PLRC24', 'PLRC6', 'Price1M', 'Price1Y', 'Price3M', 'Rank1M', 'ROC12', 'ROC120', 'ROC20', 'ROC6', 'ROC60', 'single_day_VPT', 'single_day_VPT_12', 'single_day_VPT_6', 'TRIX10', 'TRIX5', 'Volume1M', 'capital_reserve_fund_per_share', 'cashflow_per_share_ttm', 'cash_and_equivalents_per_share', 'eps_ttm', 'net_asset_per_share', 'net_operate_cash_flow_per_share', 'operating_profit_per_share', 'operating_profit_per_share_ttm', 'operating_revenue_per_share', 'operating_revenue_per_share_ttm', 'retained_earnings_per_share', 'retained_profit_per_share', 'surplus_reserve_fund_per_share', 'total_operating_revenue_per_share', 'total_operating_revenue_per_share_ttm', 'ACCA', 'adjusted_profit_to_total_profit', 'admin_expense_rate', 'cash_rate_of_sales', 'cfo_to_ev', 'debt_to_asset_ratio', 'debt_to_equity_ratio', 'debt_to_tangible_equity_ratio', 'equity_to_asset_ratio', 'equity_to_fixed_asset_ratio', 'equity_turnover_rate', 'fixed_assets_turnover_rate', 'fixed_asset_ratio', 'intangible_asset_ratio', 'invest_income_associates_to_total_profit', 'LVGI', 'net_non_operating_income_to_total_profit', 'net_operate_cash_flow_to_asset', 'net_operate_cash_flow_to_net_debt', 'net_operate_cash_flow_to_operate_income', 'net_operate_cash_flow_to_total_liability', 'net_operating_cash_flow_coverage', 'net_profit_ratio', 'net_profit_to_total_operate_revenue_ttm', 'operating_profit_growth_rate', 'operating_profit_ratio', 'operating_profit_to_operating_revenue', 'operating_profit_to_total_profit', 'operating_tax_to_operating_revenue_ratio_ttm', 'profit_margin_ttm', 'ROAEBITTTM', 'roa_ttm', 'roa_ttm_8y', 'roe_ttm', 'roe_ttm_8y', 'SGAI', 'SGI', 'total_asset_turnover_rate', 'Kurtosis120', 'Kurtosis20', 'Kurtosis60', 'sharpe_ratio_120', 'sharpe_ratio_20', 'sharpe_ratio_60', 'Skewness120', 'Skewness20', 'Skewness60', 'Variance120', 'Variance20', 'Variance60', 'average_share_turnover_annual', 'average_share_turnover_quarterly', 'beta', 'book_to_price_ratio', 'cash_earnings_to_price_ratio', 'cube_of_size', 'cumulative_range', 'daily_standard_deviation', 'debt_to_assets', 'earnings_growth', 'earnings_to_price_ratio', 'earnings_yield', 'growth', 'historical_sigma', 'leverage', 'liquidity', 'long_term_predicted_earnings_growth', 'momentum', 'natural_log_of_market_cap', 'non_linear_size', 'predicted_earnings_to_price_ratio', 'raw_beta', 'relative_strength', 'residual_volatility', 'sales_growth', 'share_turnover_monthly', 'short_term_predicted_earnings_growth', 'size', 'boll_down', 'boll_up', 'EMA5', 'EMAC10', 'EMAC12', 'EMAC120', 'EMAC20', 'EMAC26', 'MAC10', 'MAC120', 'MAC20', 'MAC5', 'MAC60', 'MACDC', 'MFI14', 'price_no_fq'])
```

Each indicator data table display the

> - ***not imputed*** *(missing values contained)*
> - ***tail-shrinked*** *(outlies processed, see[1])*
> - ***standardized***
> - ***industry-neutralized*** *and* ***circulating market capitalization-neutralized*** *(residualized, see[2]) *

indicator's values for all **(42)** Chinese listed banks (displayed as their codes) in **978** dates (from **2019-01-02** to **2022-12-30**).
[1]: Treat the values ouside $[\text{median} - 5\,\text{IQR}, \text{median} + 5\,\text{IQR}]$ as outliers and replace them with the boundary value they hit. The oulier processing is conservative because the outliers are collected in a formal statistical process which preserves its reality.
[2]: $\text{factor}_i = \beta_0 + \beta_1\,\text{industry}_i + \beta_2 \text{market capitalization}_i + \epsilon_i$ where i denotes the ordinal number of the bank and $\text{industry}_i := \mathbb{I}(\text{bank}_i \text{ is in the industry})$. The ϵ_i, as the residualized factor_i is called neturalized factor_i which is orthogonalized to industry_i and $\text{market capitalization}_i$.

Take a indicator $'\texttt{administration\_expense\_ttm}'$ form category $'\texttt{factors\_data}'$ for example:

In [5]:

```
mol.factors_datas['factors_data']['administration_expense_ttm']
```

Out[5]:

code	600928.XSHG	600016.XSHG	002807.XSHE	600926.XSHG	601658.XSHG	002142.XSHI
date						
2019-01-02	NaN	-0.192939	-0.153628	-0.003386	NaN	0.184528
2019-01-03	NaN	-0.019341	-0.163877	0.007466	NaN	0.221772
2019-01-04	NaN	-0.019341	-0.163877	0.007466	NaN	0.221772
2019-01-07	NaN	-0.019341	-0.163877	0.007466	NaN	0.221772
2019-01-08	NaN	-0.019341	-0.163877	0.007466	NaN	0.221772
...	...	...	...	...	...	...
2022-12-26	-0.2165	0.183639	-0.250547	0.159265	0.183639	0.807242
2022-12-27	-0.2165	0.183639	-0.250547	0.159265	0.183639	0.807242
2022-12-28	-0.2165	0.183639	-0.250547	0.159265	0.183639	0.807242
2022-12-29	-0.2165	0.183639	-0.250547	0.159265	0.183639	0.807242
2022-12-30	-0.2165	0.183639	-0.250547	0.159265	0.183639	0.807242

972 rows × 42 columns

More details about indicators see the following dictionary data table or **[Data Dictionary from JoinQuant]**:

In [6]:

```
pd.read_csv('data/dict/factors_datas_dicts_english.csv')
```

Out[6]:

	Factor Code	Factor Name	Details	Supplimental Details	
0	size	Market capitalization	Capturing the difference in earnings between l...	NaN	
1	beta	Beta	Characterize the volatility sensitivity of sto...	NaN	
2	momentum	Conventional momentum	Describes the difference between relatively st...	NaN	
3	residual_volatility	Residual volatility	Explain the difference in yield caused by the ...	NaN	
4	non_linear_size	Nonlinear Market Cap	Describes differences in returns that cannot b...	NaN	
...	...	...	...	...	
417	sec_sell_volume	int	Securities lending sales volume (shares)	NaN	sec
418	fin_sec_value	decimal (20, 2)	Balance of margin financing and securities len...	NaN	sec
419	HY07101	dummy	Whether the bank is	NaN	

	Factor Code	Factor Name	Details	Supplimental Details
			a national bank	
420	HY07102	dummy	Whether the bank is a regional bank	NaN
421	included_indexes_number	Numerical value	Total number of enterprises appearing in vario...	NaN

422 rows × 5 columns

3 Data Cleaning

3.1 Check whether each factor is an empty table

if yes, delete it.

In [7]:

```
mol.check_factor_data_nan()
```

```
handling factors datas missing values progress: 100%|██████████| 7/7 [00:00
<00:00, 57.40it/s]
```

Out[7]:

```
{'factors_data': [],
 'fundamentals_data': [],
 'macros_data': [],
 'money_flows_data': [],
 'securities_margins_data': [],
 'industries_data': [],
 'indexes_data': []}
```

No factors data table are empty.

3.2 Define standardized factor risk classification data table

Import the manually divided factor risk classification data table `factors_risks_data_standardized`, which has four columns, including the the factor code and the risk that the factor belongs to:

- Default Risk
- Liquidity Risk
- Market Risk

In [8]:

```
try:
    mol.factors_risks_data_standardized = pd.read_csv('data/dict/factors_risks_data_standardized.csv')
except:
    factors_codes_missing, factors_codes_excessive, factors_codes_duplicated, mol.factors_risks_data_sta
    mol.factors_risks_data_standardized.to_csv('data/dict/factors_risks_data_standardized.csv', encoding
```

The risk classification, i.e. the risks contained in the factors are as follow:

In [9]:

```
mol.factors_risks_data_standardized
```

Out[9]:

	factor_code	default_risk	liquidity_risk	market_risk
0	size	0	0	1
1	beta	0	0	1
2	momentum	0	0	1
3	residual_volatility	0	1	1
4	non_linear_size	0	0	1
...	...	...	...	...
429	fin_refund_ratio	1	0	1
430	sec_refund_ratio	1	0	1
431	HY07101	1	0	0
432	HY07102	1	0	0
433	included_indexes_number	0	0	1

434 rows × 4 columns

3.3 Missing values processing and Enterprise value weighting

- **Missing Values processing**:

> *Some factor values in factor_data are missing. These missing values are either due to stocks not having factor values before listing or after delisting, or because factor values were not disclosed or recorded during the trading period. When handling missing values, the first type of missing values should be ignored, while the second type should be filled. There are generally three methods for filling missing values: SimpleImputation, KNNImputation, and IterativeImputation[1].*
>
> - *For this panel data, SimpleImputation (such as mean or median filling) may not be suitable because it does not consider the time series characteristics and the correlation between stocks. Simply filling missing values with a constant may introduce bias, especially when the proportion of missing values is high.*
> - *KNNImputation can consider the correlation between stocks, but it does not take into account the time series characteristics either. In addition, KNNImputation may be relatively slow when dealing with large-scale panel data because it requires calculating the distance matrix between all stocks.*
> - *Therefore, IterativeImputer should be chosen. Its advantages and disadvantages include considering the time series characteristics and the correlation between stocks.*
> - *However, using IterativeImputer to fill all missing values and then removing the first type of missing values may not be the best*

approach. This is because IterativeImputer considers all features, including those that should not exist and are to be filled (the first type of missing values), when estimating missing values. This may affect the quality of the estimates.

- *Therefore, the following approach is a better choice. First, identify the sample dates (index) that do not contain the first type of missing values (all stocks are listed) and use only these samples to train the IterativeImputer. Then, use the trained IterativeImputer to estimate the second type of missing values in all samples. It is worth noting that some columns of these samples only contain missing values (the second type of missing values), and such samples cannot be used for training. We first obtain the average rank of each row in the entire data factor_data, then take the quantile corresponding to this rank in factor_data_without_type1_missing to fill the missing value column.*

- **Enterprise Value Weighting**:

> *Form a enterprise value weighted sectoral factors data table:*
> - *Formula for indicator j in row t:* $\textbf{Indicator}_{j,t}$*:*
>
> *$$*

\operatorname{Bank \space Industry \space Indicator}*{j, t} :=* *frac{**mathbf{v}*{j, t}^T \cdot \textbf{Indicator}*{j, t}}{**sum*{i = 1}^{42} v_{i, j, t}}

> *$$*
> *Where:*
>
> > - *i denotes the ordinal of bank, namely the i-th column,*

> $i \in \{1, 2, \ldots, 42\}$.
> - j *denotes the ordinal of indicators, namely the* j*-th datafame in* `mol.factors_datas`, $j \in \{1, 2, \ldots, 421\}$.
> - t *denotes the trading days' timestamp, namely the* t*-th row,* $t \in \{2019\text{-}01\text{-}01, 2019\text{-}01\text{-}02, \ldots, 2024\text{-}03\text{-}01\}$.
> - V *denotes the enterprise value.*
>
> - *Reasonableness: If your research goal is to investigate the predictive power of factors on the overall market return, and you assume that the enterprise value of individual stocks reflects their importance in the market, then weighting the sectoral factors by enterprise value is appropriate.*
> - *Advantages: This method takes into account the impact of enterprise value and reflects the overall change in market return, which is closer to the actual return of an investment portfolio.*
> - *Disadvantages: This method may be dominated by large-cap stocks, and the impact of small-cap stocks may be obscured.*

For the research subjectives, the bank sector in this paper, the weighted sectoral factors by enterprise value is chosen.
For example:
[1]: IterativeImputer iterates the following steps:

1. Be trained on the complete part of data
2. Impute all missing values
3. Be retrained on this imputed data
4. Repeat step 2 and 3 until convergence of imputation.

In [10]:

```
mol.factors_datas['factors_data']['administration_expense_ttm']
```

Out[10]:

code	600928.XSHG	600016.XSHG	002807.XSHE	600926.XSHG	601658.XSHG	002142.XSHI
date						
2019-01-02	NaN	-0.192939	-0.153628	-0.003386	NaN	0.184528
2019-01-03	NaN	-0.019341	-0.163877	0.007466	NaN	0.221772
2019-01-04	NaN	-0.019341	-0.163877	0.007466	NaN	0.221772
2019-01-07	NaN	-0.019341	-0.163877	0.007466	NaN	0.221772
2019-01-08	NaN	-0.019341	-0.163877	0.007466	NaN	0.221772
...	...	...	...	...	...	...
2022-12-26	-0.2165	0.183639	-0.250547	0.159265	0.183639	0.807242
2022-12-27	-0.2165	0.183639	-0.250547	0.159265	0.183639	0.807242
2022-12-28	-0.2165	0.183639	-0.250547	0.159265	0.183639	0.807242
2022-12-29	-0.2165	0.183639	-0.250547	0.159265	0.183639	0.807242
2022-12-30	-0.2165	0.183639	-0.250547	0.159265	0.183639	0.807242

972 rows × 42 columns

In [11]:

```
(mol.FCF_discounted_model_params_data['panal_enterprise_value_weights'] * mol.factors_datas['factors_dat
```

Out[11]:

```
date
2019-01-02    0.280478
2019-01-03    0.201603
2019-01-04    0.201571
2019-01-07    0.201541
2019-01-08    0.201483
                ...
2022-12-26    0.126160
2022-12-27    0.126219
2022-12-28    0.126257
2022-12-29    0.126236
2022-12-30    0.126251
Length: 972, dtype: float64
```

Through **Enterprise Value weighting**, we transform a sectoral row in a given date into a value, and sectoral rows in any dates (factors data table) into values in any dates (factor series).

For `factors_datas`, do the above:

In [12]:

```
try:
    mol.industry_factors_df = pd.read_csv('data/modeled_data/bank_factors_df_train.csv', index_col='date
except:
    mol.industry_factors_datas = mol.clean_and_average_factors_datas()
    mol.industry_factors_df.to_csv('data/modeled_data/bank_factors_df_train.csv', encoding='utf-8')
```

In [13]:

```
mol.industry_factors_df
```

Out[13]:

	administration_expense_ttm	asset_impairment_loss_ttm	cash_flow_to_price_ratio	
date				
2019-01-02	0.280478	0.164990	0.252082	0.42
2019-01-03	0.201603	0.158617	0.258845	0.41
2019-01-04	0.201571	0.158539	0.260929	0.41
2019-01-07	0.201541	0.158494	0.261784	0.41
2019-01-08	0.201483	0.158507	0.252722	0.41
...	...	...	...	
2022-12-26	0.126203	0.033570	0.350878	0.22
2022-12-27	0.126262	0.033777	0.353522	0.22
2022-12-28	0.126300	0.033934	0.355941	0.22
2022-12-29	0.126280	0.033854	0.354558	0.22
2022-12-30	0.126294	0.033967	0.356119	0.22

972 rows × 375 columns

For `r_waac_data`, do the same:

In [14]:

```
try:
    mol.industry_r_waac_df = pd.read_csv('data/modeled_data/bank_r_waac_df_train.csv', index_col='date',
except:
    mol.industry_r_waac_data = mol.clean_and_average_r_waac_data()
    mol.industry_r_waac_df.to_csv('data/modeled_data/bank_r_waac_df_train.csv', encoding='utf-8')
```

In [15]:

```
mol.FCF_discounted_model_params_data.keys()
```

Out[15]:

```
dict_keys(['interest_rate_1m', 'panal_enterprise_value_weights', 'r_wacc'])
```

4. Data Displaying

Randomly sample 4 factors from mol.industry_factors_df and r_waac factors to form display dataset:

In [16]:

```
sampled_columns = random.sample(mol.industry_factors_df.columns.tolist(), k=3)
displayed_df = pd.concat([mol.industry_r_waac_df, mol.industry_factors_df[sampled_columns]], axis=1)

displayed_df
```

Out[16]:

	r_waac	mid_term_loan_annualized_average_balance	average_share_turnover_quart
date			
2019-01-02	0.005285	0.413501	0.0247
2019-01-03	0.004781	0.405591	0.0068
2019-01-04	0.004891	0.405480	0.0087
2019-01-07	0.005047	0.405402	0.0023
2019-01-08	0.005290	0.405329	0.0138
...	...	...	
2022-12-26	0.003746	0.226907	-0.0230
2022-12-27	0.002748	0.226756	-0.0225
2022-12-28	0.002240	0.226637	-0.0237
2022-12-29	0.002291	0.226671	-0.0238
2022-12-30	0.001677	0.226700	-0.0228

972 rows × 4 columns

In [17]:

```
displayed_df
```

Out[17]:

	r_waac	mid_term_loan_annualized_average_balance	average_share_turnover_quarte
date			
2019-01-02	0.005285	0.413501	0.0247
2019-01-03	0.004781	0.405591	0.0068
2019-01-04	0.004891	0.405480	0.0087
2019-01-07	0.005047	0.405402	0.0023
2019-01-08	0.005290	0.405329	0.0138
...	...	...	
2022-12-26	0.003746	0.226907	-0.0230
2022-12-27	0.002748	0.226756	-0.0225
2022-12-28	0.002240	0.226637	-0.0237
2022-12-29	0.002291	0.226671	-0.0238
2022-12-30	0.001677	0.226700	-0.0228

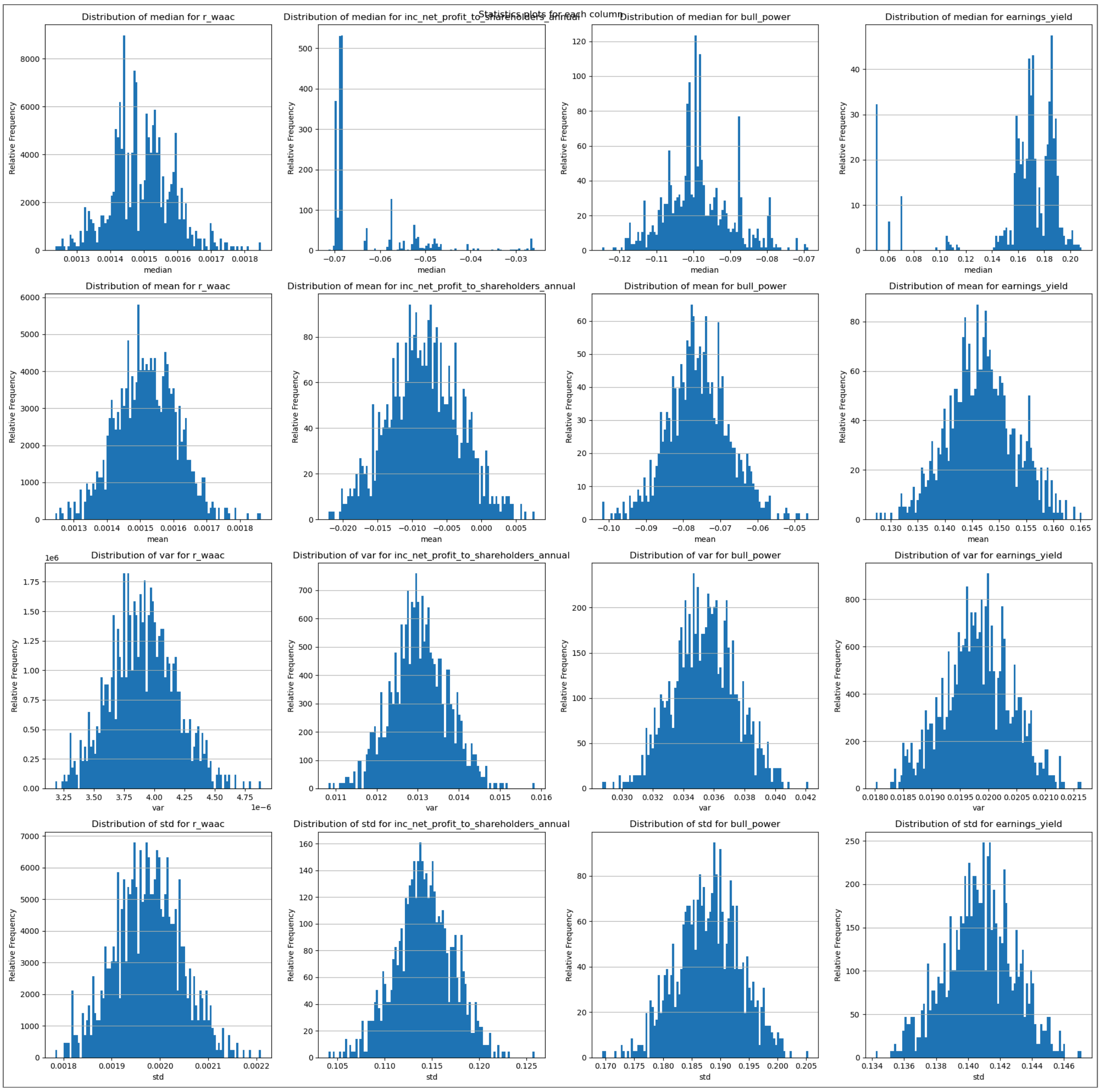
972 rows × 4 columns

4.1 Bootstraping to show statistics distribution

In [18]:

```
random.seed(mol.random_state)

sampled_columns = random.sample(mol.industry_factors_df.columns.tolist(), k=3)
displayed_df = pd.concat([mol.industry_r_waac_df, mol.industry_factors_df[sampled_columns]], axis=1)

dcf.bootstrapping(
    sample=displayed_df,
    sample_frac=0.5,
    samples_count=1000,
    stats=['median', 'mean', 'var', 'std'],
    plot=True
)
```

```
d:\Documents\GitHub\Multifactors-Risk-Research-of-China-Stock-Listed-Banks\script\dat
acamp_common_function.py:472: FutureWarning: The behavior of DataFrame concatenation
with empty or all-NA entries is deprecated. In a future version, this will no longer
exclude empty or all-NA columns when determining the result dtypes. To retain the old
behavior, exclude the relevant entries before the concat operation.
  samples_stats[col] = pd.concat([samples_stats[col], sample_stats_col])
d:\Documents\GitHub\Multifactors-Risk-Research-of-China-Stock-Listed-Banks\script\dat
acamp_common_function.py:472: FutureWarning: The behavior of DataFrame concatenation
with empty or all-NA entries is deprecated. In a future version, this will no longer
exclude empty or all-NA columns when determining the result dtypes. To retain the old
behavior, exclude the relevant entries before the concat operation.
  samples_stats[col] = pd.concat([samples_stats[col], sample_stats_col])
d:\Documents\GitHub\Multifactors-Risk-Research-of-China-Stock-Listed-Banks\script\dat
acamp_common_function.py:472: FutureWarning: The behavior of DataFrame concatenation
with empty or all-NA entries is deprecated. In a future version, this will no longer
exclude empty or all-NA columns when determining the result dtypes. To retain the old
behavior, exclude the relevant entries before the concat operation.
  samples_stats[col] = pd.concat([samples_stats[col], sample_stats_col])
d:\Documents\GitHub\Multifactors-Risk-Research-of-China-Stock-Listed-Banks\script\dat
acamp_common_function.py:472: FutureWarning: The behavior of DataFrame concatenation
with empty or all-NA entries is deprecated. In a future version, this will no longer
exclude empty or all-NA columns when determining the result dtypes. To retain the old
behavior, exclude the relevant entries before the concat operation.
  samples_stats[col] = pd.concat([samples_stats[col], sample_stats_col])
```


Statistics plots for each column
Distribution of median for r_waac
Distribution of median for inc_net_profit_to_shareholders_annual
Distribution of median for bull_power
Distribution of median for earnings_yield
Distribution of mean for r_waac
Distribution of mean for inc_net_profit_to_shareholders_annual
Distribution of mean for bull_power
Distribution of mean for earnings_yield
Distribution of var for r_waac
Distribution of var for inc_net_profit_to_shareholders_annual
Distribution of var for bull_power
Distribution of var for earnings_yield
Distribution of std for r_waac
Distribution of std for inc_net_profit_to_shareholders_annual
Distribution of std for bull_power
Distribution of std for earnings_yield
Relative Frequency
median
mean
var
std

Out[18]:

```
{'r_waac':       median      mean       var       std
 0   0.001505  0.001547  0.000004  0.001914
 0   0.001477  0.001411  0.000004  0.001912
 0   0.001451  0.001530  0.000004  0.002005
 0   0.001524  0.001430  0.000004  0.002023
 0   0.001698  0.001626  0.000004  0.001987
 ..       ...       ...       ...       ...
 0   0.001531  0.001508  0.000003  0.001866
 0   0.001497  0.001523  0.000004  0.002010
 0   0.001285  0.001283  0.000004  0.001939
 0   0.001424  0.001505  0.000003  0.001862
 0   0.001441  0.001593  0.000004  0.002002

 [1000 rows x 4 columns],
 'inc_net_profit_to_shareholders_annual':       median      mean       var       std
 0  -0.040539  0.000615  0.013043  0.114207
 0  -0.068564 -0.009871  0.012797  0.113123
 0  -0.068791 -0.006897  0.013393  0.115727
 0  -0.069868 -0.011776  0.014200  0.119165
 0  -0.069758 -0.016851  0.012102  0.110009
 ..       ...       ...       ...       ...
 0  -0.069915 -0.008790  0.013206  0.114916
 0  -0.047865 -0.001820  0.013224  0.114995
 0  -0.069147 -0.009561  0.013365  0.115606
 0  -0.069147 -0.010469  0.012575  0.112140
 0  -0.068750 -0.014130  0.012690  0.112650

 [1000 rows x 4 columns],
 'bull_power':       median      mean       var       std
 0  -0.093724 -0.071463  0.036893  0.192076
 0  -0.083200 -0.069067  0.035687  0.188910
 0  -0.087266 -0.071119  0.033180  0.182155
 0  -0.087531 -0.067183  0.035710  0.188972
 0  -0.112025 -0.088709  0.034803  0.186556
 ..       ...       ...       ...       ...
 0  -0.108066 -0.078372  0.032143  0.179286
 0  -0.106458 -0.076221  0.031974  0.178814
 0  -0.094012 -0.079579  0.034798  0.186542
 0  -0.096922 -0.069728  0.036658  0.191462
 0  -0.101166 -0.071565  0.035002  0.187089

 [1000 rows x 4 columns],
 'earnings_yield':       median      mean       var       std
 0   0.170292  0.147004  0.019750  0.140536
 0   0.163995  0.144113  0.020201  0.142128
 0   0.165148  0.141819  0.019653  0.140190
 0   0.171499  0.148115  0.019594  0.139979
 0   0.051818  0.135477  0.020029  0.141522
 ..       ...       ...       ...       ...
 0   0.181174  0.149550  0.020243  0.142277
 0   0.160518  0.143729  0.020203  0.142137
 0   0.160487  0.142672  0.019682  0.140294
 0   0.172110  0.150077  0.019860  0.140924
 0   0.186612  0.154552  0.020030  0.141526

 [1000 rows x 4 columns]}
```

As can be seen, r_waac and bull_power is highly normalized while others resemble noise.

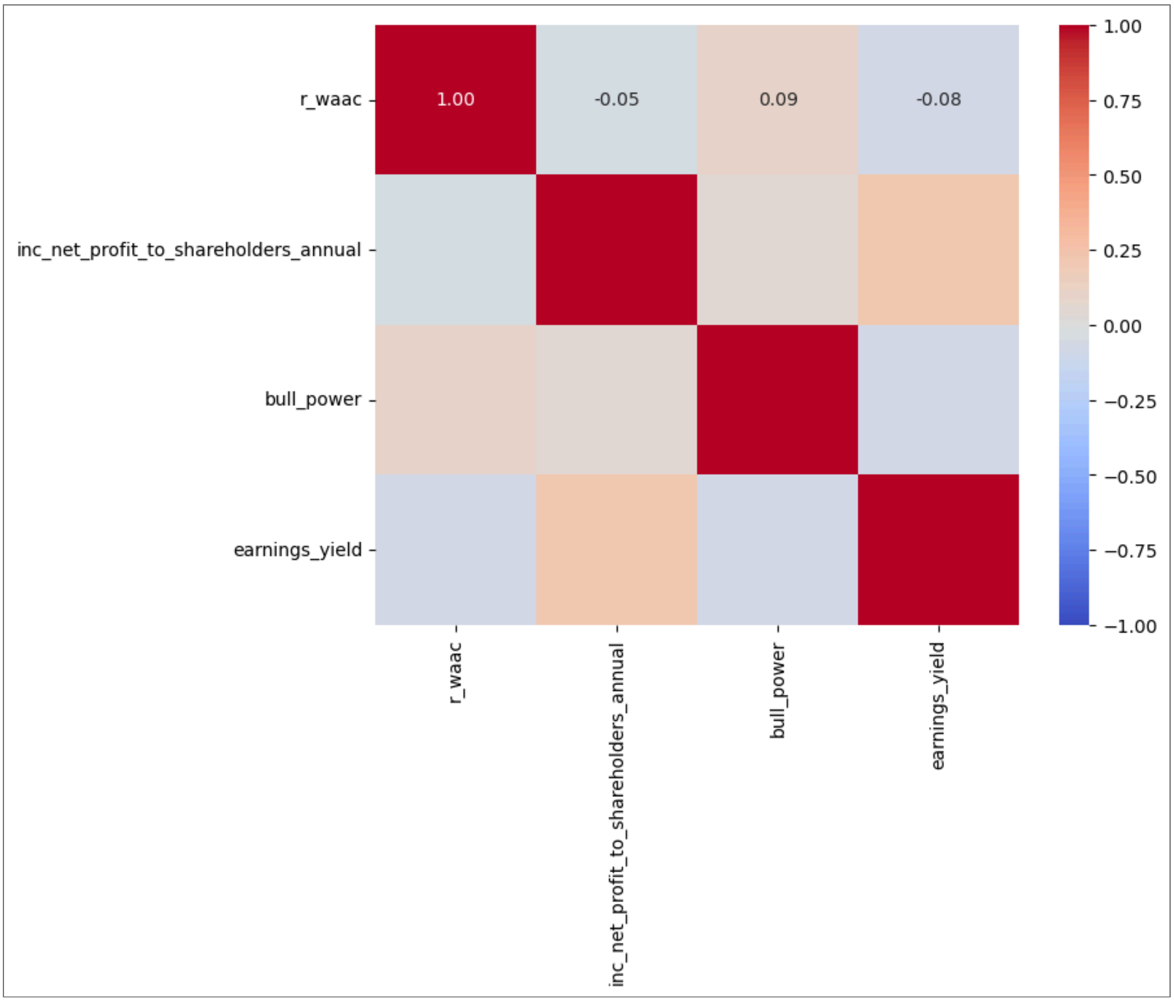
4.2 HEATMAP FOR CORRELATION

In [19]:

```
correlation_matrix = displayed_df.corr()

plt.figure(figsize=(8, 6))

sns.heatmap(
    correlation_matrix,
    annot=True,
    cmap='coolwarm',
    fmt='.2f',
    vmin=-1,
    vmax=1,
    center=0
)
```

Out[19]:

<Axes: >



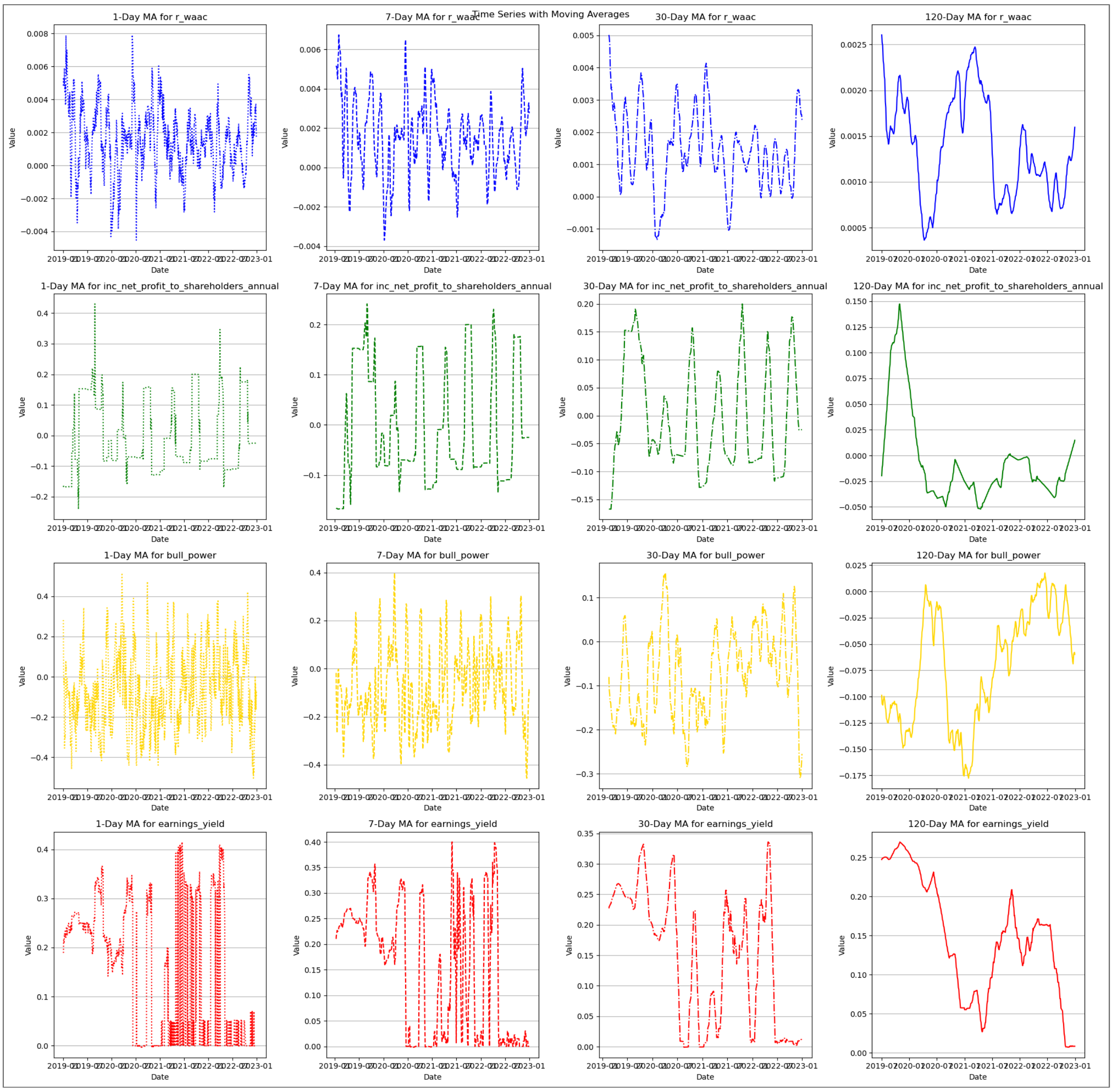
Note that 3 factors are weakly corrrelated:

- 'r_waac' and 'bull_power' : Positively correlated.
- 'r_waac' and 'earnings_yield' : Negatively correlated.
- 'bull_power' and 'earnings_yield' : Negatively correlated. \
while
- 'inc_net_profit_to_shareholders_annual' is uncorrelatd factor.

4.3 Time series and Moving average

In [20]:

```
windows = (1, 7, 30, 120)
num_row = len(windows)
num_cols = len(displayed_df.columns)

fig, axes = plt.subplots(
    nrows=num_row,
    ncols=num_cols,
    figsize=(5 * num_cols, 5 * num_row),
    squeeze=False
)

fig.suptitle('Time Series with Moving Averages')
for j, window, linestyle in zip(
        range(num_row),
        (1, 7, 30, 120),
        (':', '--', '-.', '-')
):
    ma_df = pd.DataFrame()
    for i, col, color in zip(
        range(num_cols),
        displayed_df.columns,
        ('blue', 'green', 'gold', 'red')
    ):
        ma_df['MA_' + str(window)] = displayed_df[col].rolling(window=window).mean()

        axes[i , j].plot(mol.index, ma_df['MA_' + str(window)], color=color, linestyle=linestyle)
        axes[i, j].set_title(f'{window}-Day MA for {col}')
        axes[i, j].set_xlabel('Date')
        axes[i, j].set_ylabel('Value')
        axes[i, j].grid(axis='y')

    ma_df.index = mol.index

plt.tight_layout()
plt.show()
```



The observations about 120-Day MA are that:

- ’r_waac’ and ’bull_power’ : Coincided pattern.
- ’r_waac’ and ’earnings_yield’ : Contrary pattern.
- ’bull_power’ and ’earnings_yield’ : Contrary pattern. \

while

- ’inc_net_profit_to_shareholders_annual’ has unique pattern.

3. Propoesed Method

Based on the data of **421** financial indicator's values for all **(42)** Chinese listed banks (displayed as their codes) in **978** dates (from **2019-01-02** to **2022-12-30**), this project aim to: 0. Distribute all indicators into 3 risk categories (Done in 3.2):

- Default Risk
- Liquidity Risk
- Market Risk

Steps of the Method

1. **Data Preparation**: Prepare the feature matrix `X` (a `DataFrame`) and the target variable `y` (a `Series`). Here, `X` contains all feature columns, and `y` is the target column, `r_waac`.
2. **Quantile Binning**: For each feature column `X[col]`, divide it into 4 groups (Q1, Q2, Q3, Q4) based on quantiles:
    - Q1: Feature values in the 0%-25% range.
    - Q2: Feature values in the 25%-50% range.
    - Q3: Feature values in the 50%-75% range.
    - Q4: Feature values in the 75%-100% range.

   Note: If the feature has too few unique values, it may not be possible to divide it into 4 groups. Such features will be skipped.
3. **Grouping the Target Variable**: Based on the binning labels of `X[col]`, split the target variable `y` into 4 groups corresponding to the quantile bins.
4. **One-Way ANOVA**: Perform one-way ANOVA on the 4 groups of `y` to test whether the mean values of `y` differ significantly across the bins of `X[col]`.
    - **Null Hypothesis (H0)**: The means of `y` across the bins are not significantly different.
    - **Alternative Hypothesis (H1)**: The means of `y` across the bins are significantly different.

   Use `scipy.stats.f_oneway` to calculate the p-value. If the p-value is below the significance level (e.g., 0.05), reject the null hypothesis and conclude that `X[col]` significantly affects `y`.
   By that means, the indicator `X[col]` has significant causal effect on r^*_{waac}, namely the risk contained by the indicator significantly impacts the risk of raising capital afforded by Chinese bank industry.

5. **Output Results**: Output the names of all significant feature columns.
6. **Summarize**: Summarize the percentage of significant indicators under each risk category, which means **how commonly these types of risk impact the risk of raising capital afforded by Chinese bank industry**.

In [21]:

```
from scipy.stats import f_oneway

# Function to process the DataFrame and perform ANOVA
def process_and_analyze_anova(X, y, significance_threshold=0.05):
    """
    Perform quantile binning and ANOVA testing.

    Parameters:
    - X: DataFrame containing independent variables (features).
    - y: Series containing the dependent variable ('r_waac').

    Returns:
    - List of significant columns based on ANOVA test.
    """
    significant_columns = set()  # Store column names with significant ANOVA results

    for col in X.columns:
        # Perform Q1, Q2, Q3, Q4 quantile binning
        # Dynamically adjust the number of labels to match the number of bins
        binned = pd.qcut(X[col], q=4, duplicates='drop')

        # Ensure that the binning resulted in at least 2 distinct bins
        if len(binned.cat.categories) < 2:
            print(f"Skip column {col} because it can't be binned into at least 2 groups")
            continue  # Skip this column if it can't be binned into at least 2 groups

        # Split 'y' into groups based on the binning labels
        groups = [y[binned == label] for label in binned.cat.categories]

        # Ensure all groups have at least two samples before performing ANOVA
        if all(len(group) > 1 for group in groups):
            # Perform ANOVA test
            _, p_value = f_oneway(*groups)

            # If p-value < 0.05, consider it significant and save the column name
            if p_value < 0.001:
                significant_columns.add(col)

    return significant_columns

mol.FCF_discounted_model_params_data['interest_rate_1m'] = mol.FCF_discounted_model_params_data['interes
mol.industry_r_waac_star_df = mol.industry_r_waac_df - mol.FCF_discounted_model_params_data['interest_ra
mol.industry_r_waac_star_df = mol.industry_r_waac_star_df['r_waac']

significant_columns = process_and_analyze_anova(
    mol.industry_factors_df,
    mol.industry_r_waac_star_df
)

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.expand_frame_repr', False)

factors_risks_data_standardized_sig = mol.factors_risks_data_standardized[
    mol.factors_risks_data_standardized['factor_code'].isin(significant_columns)
].reset_index(drop=True)

factors_risks_data_standardized_sig
```

```
Skip column money_gold_central_bank because it can't be binned into at least 2 groups
Skip column government_claim_central_bank because it can't be binned into at least 2
groups
```

Out[21]:

	factor_code	default_risk	liquidity_risk
0	size	0	0
1	beta	0	0
2	momentum	0	0
3	growth	1	0
4	leverage	1	0
5	cfo_to_ev	1	1
6	fixed_asset_ratio	0	1
7	operating_tax_to_operating_revenue_ratio_ttm	1	0
8	net_operate_cash_flow_to_operate_income	1	1
9	net_operating_cash_flow_coverage	1	1
10	intangible_asset_ratio	0	1
11	debt_to_equity_ratio	1	0
12	operating_profit_growth_rate	1	0
13	net_operate_cash_flow_to_net_debt	1	1
14	net_operate_cash_flow_to_asset	1	1
15	total_asset_turnover_rate	0	1
16	debt_to_tangible_equity_ratio	1	0
17	ROAEBITTTM	1	0
18	operating_profit_ratio	1	0
19	debt_to_asset_ratio	1	0
20	net_operate_cash_flow_to_total_liability	1	1
21	cash_rate_of_sales	0	1
22	operating_profit_to_operating_revenue	1	0
23	roa_ttm	1	0
24	admin_expense_rate	1	0
25	fixed_assets_turnover_rate	0	1
26	invest_income_associates_to_total_profit	1	0
27	ACCA	0	1
28	roe_ttm	1	0
29	adjusted_profit_to_total_profit	1	0
30	equity_turnover_rate	0	1
31	SGI	1	0

	factor_code	default_risk	liquidity_risk
32	roe_ttm_8y	1	0
33	roa_ttm_8y	1	0
34	SGAI	1	0
35	total_operating_revenue_ttm	1	0
36	operating_profit_ttm	1	0
37	net_operate_cash_flow_ttm	1	1
38	operating_revenue_ttm	1	0
39	total_operating_cost_ttm	1	0
40	non_operating_net_profit_ttm	1	0
41	net_invest_cash_flow_ttm	0	1
42	administration_expense_ttm	1	0
43	value_change_profit_ttm	1	0
44	total_profit_ttm	1	0
45	net_finance_cash_flow_ttm	0	1
46	interest_free_current_liability	0	1
47	net_profit_ttm	1	0
48	OperateNetIncome	1	0
49	EBITDA	1	0
50	asset_impairment_loss_ttm	1	0
51	np_parent_company_owners_ttm	1	0
52	non_recurring_gain_loss	1	0
53	market_cap	0	0
54	cash_flow_to_price_ratio	0	1
55	sales_to_price_ratio	1	0
56	operating_assets	0	1
57	total_asset_growth_rate	1	0
58	net_operate_cashflow_growth_rate	1	1
59	np_parent_company_owners_growth_rate	1	0
60	financing_cash_growth_rate	0	1
61	net_profit_growth_rate	1	0
62	net_asset_growth_rate	1	0
63	PEG	1	0
64	total_operating_revenue_per_share_ttm	1	0

	factor_code	default_risk	liquidity_risk
65	cash_and_equivalents_per_share	0	1
66	surplus_reserve_fund_per_share	1	0
67	retained_profit_per_share	1	0
68	operating_revenue_per_share_ttm	1	0
69	retained_earnings_per_share	1	0
70	net_operate_cash_flow_per_share	1	1
71	operating_profit_per_share_ttm	1	0
72	eps_ttm	1	0
73	cashflow_per_share_ttm	1	1
74	VEMA10	0	0
75	VR	0	0
76	VEMA12	0	0
77	TVMA20	0	0
78	VDIFF	0	0
79	WVAD	0	0
80	MAWVAD	0	0
81	DAVOL10	0	0
82	VDEA	0	0
83	VSTD20	0	0
84	VOL20	0	0
85	DAVOL20	0	0
86	turnover_volatility	0	0
87	TVSTD20	0	0
88	money_flow_20	0	1
89	VEMA5	0	0
90	VOL240	0	0
91	VEMA26	0	0
92	VOSC	0	0
93	TVSTD6	0	0
94	Variance20	0	0
95	Kurtosis20	0	0
96	Variance60	0	0
97	Kurtosis60	0	0

	factor_code	default_risk	liquidity_risk
98	Variance120	0	0
99	Skewness120	0	0
100	Kurtosis120	0	0
101	sharpe_ratio_120	0	0
102	boll_up	0	0
103	EMAC120	0	0
104	MAC60	0	0
105	MAC120	0	0
106	BIAS60	0	0
107	Price3M	0	0
108	Price1Y	0	0
109	ROC120	0	0
110	ROC60	0	0
111	TRIX10	0	0
112	cumulative_range	0	0
113	daily_standard_deviation	0	0
114	historical_sigma	0	0
115	raw_beta	0	0
116	relative_strength	0	0
117	debt_to_assets	1	0
118	earnings_to_price_ratio	1	0
119	long_term_predicted_earnings_growth	1	0
120	predicted_earnings_to_price_ratio	1	0
121	sales_growth	1	0
122	pe_ratio	0	0
123	pe_ratio_lyr	0	0
124	pb_ratio	0	0
125	ps_ratio	0	0
126	pcf_ratio	0	1
127	eps	1	0
128	operating_profit	1	0
129	roe	1	0
130	inc_return	1	0

	factor_code	default_risk	liquidity_risk
131	net_profit_margin	0	0
132	expense_to_total_revenue	1	0
133	operation_profit_to_total_revenue	1	0
134	net_profit_to_total_revenue	1	0
135	ga_expense_to_total_revenue	1	0
136	operating_profit_to_profit	1	0
137	invesment_profit_to_profit	1	0
138	adjusted_profit_to_profit	1	0
139	ocf_to_revenue	1	1
140	ocf_to_operating_profit	1	1
141	inc_total_revenue_year_on_year	1	0
142	inc_revenue_year_on_year	1	0
143	inc_operation_profit_annual	1	0
144	inc_net_profit_annual	1	0
145	inc_net_profit_to_shareholders_year_on_year	1	0
146	inc_net_profit_to_shareholders_annual	1	0
147	total_loan	1	0
148	interest_earning_assets_yield	1	0
149	non_interest_income	0	1
150	net_profit_margin	0	0
151	core_level_capital_adequacy_ratio	1	0
152	level_1_capital_adequacy_ratio	1	0
153	net_capital	1	0
154	capital_adequacy_ratio	1	0
155	deposit_loan_ratio	0	1
156	short_term_asset_liquidity_ratio_CNY	0	1
157	short_term_asset_liquidity_ratio_FC	0	1
158	cost_to_income_ratio	1	0
159	normal_amount	1	0
160	secondary_amount	1	0
161	secondary_amount_ratio	1	0
162	loss_amount	1	0
163	short_term_loan_average_balance	1	0

	factor_code	default_risk	liquidity_risk
164	short_term_loan_annualized_average_interest_rate	1	0
165	mid_term_loan_annualized_average_balance	1	0
166	spot_sell	0	0
167	cash_offer_prc	0	0
168	safe_prc	0	0
169	bank_reduced_prc	0	0
170	interest_rate_3m	0	0
171	interest_rate_1y	0	0
172	m1	0	1
173	m2_yoy	0	1
174	m1_yoy	0	1
175	m0_yoy	0	1
176	gold	0	0
177	foreign	0	0
178	HY07101	1	0
179	HY07102	1	0

In [24]:

```
for factors_risk_name in mol.factors_risks_data_standardized.columns[1:]:
    m_sig = factors_risks_data_standardized_sig[
        factors_risks_data_standardized_sig[factors_risk_name] == 1
    ].size
    m = mol.factors_risks_data_standardized[
        mol.factors_risks_data_standardized[factors_risk_name] == 1
    ].size
    print(f"Percentage of significant factors in {factors_risk_name} category: {m_sig / m:.3f}")
```

```
Percentage of significant factors in default_risk category: 0.453
Percentage of significant factors in liquidity_risk category: 0.300
Percentage of significant factors in market_risk category: 0.373
```

As can be seen, the **default risk factors** most commonly impacts the capital raising risk r^*_{waac}.
This is probably because investors are less willing to provide funding to entities with higher probabilities of default. A higher default risk increases the cost of capital, as lenders demand higher premiums to offset potential losses. This makes it more difficult for firms to raise capital efficiently. Thus, the link between default risk and capital raising risk is both intuitive and financially logical.