Quantamental Trading

In this notebook, we have developed a trading Strategy by using the Fundamental data and some Quant Techniques. Notebook is divided in two parts, Strategy Formulation the Backtesting.

Import the relevant Libraries

In [1]:

```
import pandas as pd
import numpy as np
from scipy.stats import norm
import scipy.stats
import matplotlib.pyplot as plt
%reload_ext autoreload
%autoreload 2
%matplotlib inline
import yfinance as yf
from datetime import date
import seaborn as sb
import monthly_returns_heatmap as mrh
import fundamentalanalysis as fa
import seaborn as sb
sb.set_style('darkgrid')
import warnings
warnings.filterwarnings('ignore')
```

Strategy Formulation

Analysis on Group of Companies taking latest Ratios

We will do the inter-company analysis to find the relatively undervalued companies among its peers. We will be using the Z-score method to calculate the scores of the companies and rank them accordingly.

In [2]:

```
key = '15d3a78098390cc68b5bf8132b213ae7'
```

In [3]:

```
"Reference from S&P Value bse Factor indices"

# Z-score testing to compare the company with its peer
def zscore2(data_frame, headline='Z-Score'):
    index = data_frame.index
    z = scipy.stats.zscore(data_frame.astype(float))
    table = pd.DataFrame(z, index=index)
    mean = table.mean(axis=1)
    table2 = pd.DataFrame(mean, index=index)
    condition1 = np.where(table2 == 0, 1, table2)
    condition2 = np.where(condition1 > 0, 1+condition1, 1/(1-condition1))
    final = pd.DataFrame(condition2, index=index)
    final.columns = [headline]
    return final
```

Testes

In [4]:

```
# Using top Nasdaq stocks
tickers = ["GILD", 'MU', 'AMAT', 'MSFT', 'AMD', 'CSCO', 'QCOM',
```

```
                 'GOOG', 'TSLA', 'TXN', 'INTC', 'ATVI', 'PEP', 'PCAR']

# Ratios where the denominator must be high, to derive the value
ratio = ['peRatio', 'pbRatio', 'evToSales',
         'debtToEquity', 'debtToAssets', 'netDebtToEBITDA']

f = pd.DataFrame()
for steps in range(len(tickers)):
    p = str(tickers[steps])
    data = fa.key_metrics(
        ticker=p, api_key=key, period='quarter')
    table = data.T[ratio]
    f_1 = table.head(1)
    f = f.append(f_1)

f.index = tickers
# Derive the value metrics out of the algorithm "Inter Company Analysis"
inverse = 1/f

# Inverse matrix consist of the ratios that must be maximised as we have already taken the
inverse
```

Out[4]:

	peRatio	pbRatio	evToSales	debtToEquity	debtToAssets	netDebtToEBITDA
GILD	0.039016	0.202218	0.050765	0.789674	2.333534	0.093188
MU	-0.100126	0.600719	0.046821	3.278813	4.742923	0.085714
AMAT	0.066165	0.148388	0.068857	2.497172	5.141746	1.906542
MSFT	0.03412	0.090751	0.024255	3.216731	6.280163	0.762691
AMD	-0.003547	0.346397	0.034114	19.204354	23.747893	-0.425793
CSCO	0.066499	0.218912	0.075281	5.038718	11.618894	13.154286
QCOM	0.049414	0.142804	0.061655	1.23228	3.025461	0.204849
GOOG	0.045293	0.196275	0.052479	8.958964	12.688129	37.756993
TSLA	0.015304	0.073162	0.036618	17.957399	32.448804	-0.196441
TXN	0.040495	0.09035	0.025115	1.505333	2.885641	0.406267
INTC	-0.083085	0.735842	0.066828	1.95053	3.685935	0.020385
ATVI	0.044055	0.299383	0.038708	5.570479	7.585987	-0.147022
PEP	0.031315	0.069056	0.062886	0.408025	2.227644	0.091791
PCAR	0.076607	0.362263	0.129997	0.802938	2.014368	0.060345

Function to calculate the Z-score of the above Dataframe. So, that we can have normalized data and can compare the ratios and do the ranking.

In [5]:

```
# Calling the function and calculating the relative score of each company
z = zscore2(inverse)
z
```

Out[5]:

	Z-Score
GILD	0.753015
MU	0.740383
AMAT	0.964086
MSFT	0.687200

	Z-Score
AMD	1.468417
CSCO	1.448593
QCOM	0.801546
GOOG	1.754668
TSLA	1.445974
TXN	0.645975
INTC	0.901813
ATVI	0.930774
PEP	0.712877
PCAR	1.435084

Comparing the past year performance of each company. We will be using some critical ratios and then will be quantifying so as to convert them into a score.

In [6]:

```
f_score_ratio_max = ['roe', 'returnOnTangibleAssets',
                     'researchAndDdevelopementToRevenue', 'currentRatio']

# WRT to assets and liability
f_score_ratio_min = ['averagePayables',
                     'daysOfInventoryOnHand', 'interestDebtPerShare']

# Ratios where change must reduce Q/Q
g_score_ratio_max = ['netIncomePerShare', 'freeCashFlowPerShare',
                     "shareholdersEquityPerShare", 'cashPerShare', 'bookValuePerShare']

# WRT to cash, income and book value
columns = ['F-Score-Max', 'F-Score-Min', 'G-Score']


tickers = ["GILD", 'MU', 'AMAT', 'MSFT', 'AMD', 'CSCO', 'QCOM',
           'GOOG', 'TSLA', 'TXN', 'INTC', 'ATVI', 'PEP', 'PCAR']

# F-Score and G-Score analysis to compare the companies with its Past Quarter Performances

M = pd.DataFrame()
for steps in range(len(tickers)):
    p = str(tickers[steps])
    data = fa.key_metrics(
        ticker=p, api_key=key, period="quarter")
    gh = data.T
    gh = gh.iloc[:, 1:]
    gh = gh.loc[::-1]
    gh = gh.loc['2019-12':]
    change = gh.pct_change()
    test = change[f_score_ratio_max].dropna()
    test1 = change[f_score_ratio_min].dropna()
    test2 = change[g_score_ratio_max].dropna()
    g = np.where(test > 0, 1, -1)

    # Conditions to Score the past performance and create a ranking table for every set
    g1 = np.where(test < 0, 1, -1)
    g2 = np.where(test2 > 0, 1, -1)
    array = np.array([g.sum(), g1.sum(), g2.sum()])
    df = pd.DataFrame(array)
    df = df.T
    M = M.append(df)
```

```python
M.index = tickers
M.columns = columns

# Now, as we already have the scores, we take the z-score of the final DataFrame,
# in order to compare the scores with each other

# Comparing the Score table along with its peers
ZS = scipy.stats.zscore(M)
ZS = pd.DataFrame(ZS)
ZS.index = M.index
ZS.columns = M.columns
v = pd.concat([ZS, z], axis=1)
Score_table = pd.DataFrame({"Score": v.sum(axis=1)})
y = Score_table.sort_values(by='Score', ascending=False)

# Final DataFrame having the Score of each company
y
```

Out[6]:

	Score
GOOG	2.566864
TSLA	2.407785
CSCO	1.961559
ATVI	1.892584
TXN	1.757400
MSFT	1.649011
AMAT	1.477052
PCAR	1.199975
AMD	1.083693
QCOM	0.865666
MU	0.505274
INTC	-0.679831
PEP	-0.868768
GILD	-1.127860

Above Dataframe has final score and it represents the relatively undervalued companies, which are also fundamentally strong.

Backtesting the strategy

Selecting the companies at start of the year and then taking the positions in them for the year long and rebalancing the companies at start of next year.

In [7]:

```python
tickers = ["GILD", 'MU', 'AMAT', 'MSFT', 'AMD', 'CSCO', 'QCOM',
           'GOOG', 'TSLA', 'TXN', 'INTC', 'ATVI', 'PEP', 'PCAR']
```

In [8]:

```python
# WRT to cash, income and book value
columns = ['F-Score-Max', 'F-Score-Min', 'G-Score']

ratio = ['peRatio', 'priceToSalesRatio', 'pbRatio', 'evToSales',
         'debtToEquity', 'evToFreeCashFlow', 'debtToAssets', 'netDebtToEBITDA']
```

```python
# Ratios to derive out the value factor
Full_table = pd.DataFrame()
year = ['2017','2018', '2019', '2020']

for steps1 in range(len(year)):
    for steps2 in range(len(tickers)):
        p = str(tickers[steps2])
        data = fa.key_metrics(
            ticker=p, api_key=key, period='annual')
        table = data.T[ratio]
        table = table.loc[year[steps1]]
        Full_table = Full_table.append(table)

Full_table
```

Out[8]:

	peRatio	priceToSalesRatio	pbRatio	evToSales	debtToEquity	evToFreeCashFlow	debtToAssets	net
2017	20.231953	3.586528	4.580446	4.580667	1.640837	10.575476	0.477242	
2017	6.841291	1.713184	1.869681	2.010694	0.599055	11.951252	0.315684	
2017	17.466715	4.126071	6.415734	4.146296	0.567333	18.466513	0.273135	
2017	25.180710	5.935873	7.375360	6.808925	1.190624	19.518860	0.357524	
2017	227.594419	1.836472	16.017283	1.875879	2.283142	-222.145778	0.394068	
2017	16.434093	3.289557	2.387698	3.748030	0.509805	13.934650	0.259725	
2017	30.917575	3.420337	2.479761	2.831041	0.712060	15.764861	0.334316	
2017	57.274197	6.541932	4.755386	6.481078	0.026026	30.052281	0.020117	
2017	-26.312244	4.388973	12.179818	5.132932	2.859395	-14.571886	0.422816	
2017	28.109734	6.917989	10.012580	7.079810	0.394408	22.690883	0.231096	
2017	22.601621	3.457532	3.144035	3.830056	0.388487	23.265405	0.217551	
2017	174.883810	6.803945	5.045792	6.757914	0.463961	23.041924	0.235162	
2017	35.059996	2.680620	15.638387	3.131954	3.607402	28.321338	0.492218	
2017	14.931393	1.285596	3.107021	1.689252	1.269288	37.812666	0.435935	
2018	14.883575	3.669268	3.796227	4.093275	1.277505	12.115021	0.429085	
2018	4.299931	1.999918	1.882068	1.938486	0.143649	6.913804	0.106949	
2018	9.515412	1.827193	4.609528	1.935522	0.776283	10.550888	0.298712	
2018	45.820832	6.880183	9.179344	7.513220	0.988999	25.708762	0.316046	
2018	51.926528	2.702585	13.822464	2.729149	0.987362	-136.986357	0.274363	
2018	1871.919000	4.174155	4.766019	4.511374	0.591820	17.343056	0.235044	
2018	-21.836719	4.672435	114.454526	4.874485	17.640086	35.617743	0.500826	
2018	23.419484	5.261121	4.052409	5.168378	0.022587	30.971105	0.017234	
2018	-58.140896	2.644336	11.527118	3.116892	2.808570	-301.706064	0.464945	
2018	16.427419	5.807463	10.191795	5.974088	0.563487	15.565368	0.295734	
2018	10.239123	3.042630	2.891035	3.372068	0.353513	16.764034	0.205989	
2018	19.573271	4.731512	3.124623	4.524312	0.235185	20.453490	0.149762	
2018	12.477779	2.415048	10.756261	2.780028	2.226271	29.310191	0.416250	
2018	9.136769	0.853608	2.334034	1.166729	1.256025	26.358772	0.423543	
2019	15.322057	3.676092	3.643470	4.253490	1.085784	11.478134	0.399062	

	peRatio	priceToSalesRatio	pbRatio	evToSales	debtToEquity	evToFreeCashFlow	debtToAssets	net
2019	7.882525	2.126052	1.386873	2.070468	0.163067	14.535507	0.119684	
2019	19.719937	3.652940	6.496488	3.802447	0.646822	19.795492	0.279279	
2019	26.530903	8.272789	10.173680	8.805278	0.765816	28.961909	0.273475	
2019	147.748915	7.485126	17.821854	7.369095	0.257517	179.715145	0.120770	
2019	21.496091	4.812848	7.441127	5.061692	0.734741	17.606358	0.252227	
2019	21.043958	3.802530	18.801956	3.971194	3.245671	15.063729	0.483448	
2019	26.963700	5.721188	4.596928	5.698143	0.079264	29.778004	0.057871	
2019	-95.757201	3.019441	11.213634	3.310393	2.027652	84.052511	0.391122	
2019	23.934511	8.348706	13.481469	8.582732	0.651510	21.276360	0.322067	
2019	12.608008	3.687534	3.423996	4.032243	0.374187	17.137985	0.212424	
2019	30.322781	7.023446	3.559168	6.542786	0.208903	24.755767	0.134795	
2019	26.157799	2.848649	12.939141	3.244102	2.168808	40.220997	0.408265	
2019	11.491173	1.071879	2.827065	1.370120	1.216761	39.431897	0.416415	
2020	822.840674	2.966212	4.019144	3.995213	1.723396	13.120221	0.459047	
2020	19.138928	2.399174	1.318758	2.378274	0.184019	614.196386	0.133686	
2020	15.062492	3.168885	5.153258	3.174524	0.521081	16.146706	0.246589	
2020	34.974619	10.829012	13.090945	11.230522	0.600132	35.507209	0.235630	
2020	43.655839	11.134184	18.623101	11.025201	0.098338	138.531583	0.064048	
2020	17.527234	3.986743	5.183291	4.043009	0.384573	13.600191	0.153743	
2020	25.868305	5.714311	22.126617	6.097593	2.587790	32.557851	0.441816	
2020	29.618501	6.534416	5.359423	6.526817	0.120300	27.806652	0.083763	
2020	954.188430	20.877410	29.623848	20.634989	0.528189	240.927440	0.225109	
2020	27.017646	10.453200	16.454091	10.708439	0.739959	28.206690	0.351300	
2020	9.457243	2.538263	2.438941	2.930419	0.449184	10.901674	0.237774	
2020	32.584137	8.853246	4.760747	8.229699	0.239742	30.609637	0.156000	
2020	28.115562	2.844637	14.879055	3.355707	3.281552	37.054417	0.475150	
2020	23.045213	1.597667	2.872935	2.012601	1.085991	27.945584	0.399882	

In [9]:

```
pd.__version__
```

Out[9]:

```
'1.3.5'
```

In [10]:

```
#!pip uninstall pandas --yes
#!pip install pandas==1.3.5
```

In [11]:

```
rank = pd.DataFrame()

for steps in range(len(year)):
    y = Full_table.loc[year[steps]]
    t_inv = 1/y
    bv = zscore2(t_inv)
    bv = bv.T
    bv.index = [year[steps]]
```

```
    bv.columns = tickers

    rank = rank.append(bv)
```

In [12]:

```
name = pd.DataFrame(tickers)
concated = pd.concat([name]*len(year), axis=0)
concated.index = Full_table.index
concated.columns = ['Company']

lista_z_score = []
for i in year:
    lista_z_score += list(rank.loc[i])

lista_z_score = pd.DataFrame(lista_z_score,index=Full_table.index,columns=['Z-Score'])
rank = pd.concat([concated, lista_z_score], axis=1)

rank.head()
```

Out[12]:

	Company	Z-Score
2017	GILD	1.009932
2017	MU	2.056281
2017	AMAT	1.244755
2017	MSFT	0.732541
2017	AMD	0.903451

Now doing the Intra-Company analysis, comparing the past performances of the companies to derive out fundamentally strong companies.

In [13]:

```
f_score_ratio_max = ['roe', 'returnOnTangibleAssets',
                     'researchAndDdevelopementToRevenue', 'currentRatio']

# WRT to assets and liability
f_score_ratio_min = ['averagePayables',
                     'daysOfInventoryOnHand', 'interestDebtPerShare']

# Ratios where change must reduce Q/Q
g_score_ratio_max = ['netIncomePerShare', 'freeCashFlowPerShare',
                     "shareholdersEquityPerShare", 'cashPerShare', 'bookValuePerShare', ':


percent_fscore_max = pd.DataFrame()
percent_fscore_min = pd.DataFrame()
percent_gscore_max = pd.DataFrame()

for steps in range(len(tickers)):
    # Fscore-Max analysis
    p = str(tickers[steps])
    data3 = fa.key_metrics(
        ticker=p, api_key=key, period='annual')
    tranpose = data3.T[f_score_ratio_max].fillna(0)
    reverse = tranpose.loc[::-1]
    change = reverse.pct_change()
    change = change.assign(Company=p)
    percent_fscore_max = percent_fscore_max.append(change)

    # Fscore-Min analysis
    data4 = fa.key_metrics(
        ticker=p, api_key=key, period='annual')
    tranpose1 = data4.T[f_score_ratio_min].fillna(0)
```

```
    reverse1 = tranpose1.loc[::-1]
    change1 = reverse1.pct_change()
    change1 = change1.assign(Company=p)
    percent_fscore_min = percent_fscore_min.append(change1)

    # Gscore-Max analysis
    data5 = fa.key_metrics(
        ticker=p, api_key=key, period='annual')
    tranpose2 = data5.T[g_score_ratio_max].fillna(0)
    reverse2 = tranpose2.loc[::-1]
    change2 = reverse2.pct_change()
    change2 = change2.assign(Company=p)
    percent_gscore_max = percent_gscore_max.append(change2)

# Condition to quantify the performance
condition = np.where(percent_fscore_max.iloc[:, :-1] > 0, 1, -1)
condition = pd.DataFrame(condition)
condition.index = percent_fscore_max.index
condition.columns = percent_fscore_max.columns[:-1]
condition
sum_score = condition.sum(axis=1)
sum_score = pd.DataFrame(sum_score)
sum_score = pd.concat([sum_score, percent_fscore_max['Company']], axis=1)
sum_score.columns = ['F-Score-Max','Company']
sum_score

# Condition to quantify the performance
condition2 = np.where(percent_fscore_min.iloc[:, :-1] < 0, 1, -1)
condition2 = pd.DataFrame(condition2)
condition2.index = percent_fscore_min.index
condition2.columns = percent_fscore_min.columns[:-1]
condition2
sum_score2 = condition2.sum(axis=1)
sum_score2 = pd.DataFrame(sum_score2)
sum_score2 = pd.concat([sum_score2, percent_fscore_min['Company']], axis=1)
sum_score2.columns = ['F-Score-Min','Company']
sum_score2

# Condition to quantify the performance
condition3 = np.where(percent_gscore_max.iloc[:, :-1] > 0, 1, -1)
condition3 = pd.DataFrame(condition3)
condition3.index = percent_gscore_max.index
condition3.columns = percent_gscore_max.columns[:-1]
sum_score3 = condition3.sum(axis=1)
sum_score3 = pd.DataFrame(sum_score3)
sum_score3 = pd.concat([sum_score3, percent_gscore_max['Company']], axis=1)
sum_score3.columns = ['G-Score-Max','Company']
```

In [14]:

```
Final_yr_wise = pd.concat([sum_score['F-Score-Max'], sum_score2['F-Score-Min'], sum_score3
Final_yr_wise.head(4)
```

Out[14]:

	F-Score-Max	F-Score-Min	G-Score-Max	Company
1991	-4	-3	-6	GILD
1992	-2	-1	4	GILD
1993	0	-3	-2	GILD
1994	2	-1	2	GILD

In [15]:

```
rank.loc[year[0:]].head()
Final_yr_wise.loc[year[0:]].head()
```

Out[15]:

	F-Score-Max	F-Score-Min	G-Score-Max	Company
2017	-2	1	2	GILD
2017	-2	-1	0	MU
2017	2	-3	4	AMAT
2017	4	-1	4	MSFT
2017	-4	1	-2	AMD

In [16]:

```
company = pd.DataFrame(Final_yr_wise['Company'])

vc = zscore2(Final_yr_wise[Final_yr_wise.columns[:-1]], headline='Cumulative-F&G-Score')
vc['Company'] = company

rank = rank.loc[year[0]:]

# Now Joing the two Dataframes and getting a cumulative score of each company for each yea
vc_new = vc.reset_index()
vc_new['key'] = vc_new['index']+str('_')+vc_new['Company']
vc_new.head(3)
rank_new = rank.reset_index()
rank_new['key'] = rank_new['index']+str('_')+rank_new['Company']
df = vc_new.merge(rank_new, on='key')
df = df.drop(labels=['Company_x', 'key', 'index_y'], axis=1)
df.index = df['index_x']
dataframe = df.drop(labels='index_x', axis=1)
dataframe['Cumulative-Score'] = dataframe['Cumulative-F&G-Score'] + \
    dataframe['Z-Score'].values
dataframe_final = dataframe.drop(
    labels=['Cumulative-F&G-Score', 'Z-Score'], axis=1)
dataframe_final.head(4)
```

Out[16]:

	Company_y	Cumulative-Score
index_x		
2017	GILD	2.266047
2018	GILD	3.538244
2019	GILD	2.655932
2020	GILD	1.327655

In [17]:

```
stock = yf.download(tickers=tickers, start='2017-01-01',
                    end='2020-12-31', interval='1mo')[['Close']]
```

```
[*********************100%***********************]  14 of 14 completed
```

In [18]:

```
stock.index = pd.to_datetime(stock.index)
```

In [19]:

```
stock = (stock.T.drop_duplicates().T).dropna()
stock.columns = tickers
returns = stock.pct_change().dropna()
```

Taking the positions in the top 6 Companies, with equally weighted rule.

In [20]:

```
df = pd.DataFrame()
df2 = pd.DataFrame()
```

```
for steps in range(len(year)):
    sorted_table = (dataframe_final.loc[year[steps]]).sort_values(
        by='Cumulative-Score', ascending=False)
    selection = sorted_table[['Company_y']].head(6)
    array = np.array(selection['Company_y'])
    frame1 = pd.DataFrame(array)
    returns_selection = returns.loc[year[steps]]
    selected_returns = returns_selection[array]

    # Making the portfolio with equal weights among its components
    weight = np.repeat(1/6, 6)
    series = (selected_returns*weight).sum(axis=1)
    frame = pd.DataFrame(series)
    df = df.append(frame)
    df2 = df2.append(frame1)
```

In [21]:

```
comp_index = np.repeat(year, 6)
df2.index = comp_index
df2.columns = ['Companies-Invested']
```

In [22]:

```
list_comp = pd.DataFrame()
for steps in range(len(year)):
    f = df2.loc[year[steps]]['Companies-Invested'].T
    mk = np.array(f)
    l = pd.DataFrame(mk).T
    list_comp = list_comp.append(l)

list_comp.index = year
list_comp
```

Out[22]:

	0	1	2	3	4	5
2017	PCAR	MU	MSFT	AMAT	GILD	GOOG
2018	GILD	MU	INTC	AMD	PCAR	GOOG
2019	ATVI	MU	MSFT	PCAR	GILD	GOOG
2020	AMAT	INTC	AMD	CSCO	MU	PCAR

Following DataFrame shows the list of the companies in which we Invested Year-Wise.

Weights- Equally Weighted.

In [23]:

```
df.columns = ['Returns']
df.index = pd.to_datetime(df.index).to_period('M')
```

Downloading the Benchmark 'NASDAQ' and Comparing the performance with it.

In [24]:

```
benchmark = yf.download(tickers='NDAQ', start='2017-01-01',
                        end='2020-12-31', interval='1mo')[['Close']]
```

```
[*********************100%***********************]  1 of 1 completed
```

In [25]:

```
benchmark = (benchmark.T.drop_duplicates().T).dropna()
```

In [26]:

```
benchmark.index = pd.to_datetime(benchmark.index).to_period('M')
```

In [27]:

```
benchmark_returns = benchmark.pct_change().dropna()
```

In [28]:

```
frame_final = pd.concat([df, benchmark_returns], axis=1)
frame_final.columns = ['strategy', 'nasdaq']
frame_final
```

Out[28]:

Date	strategy	nasdaq
2017-02	0.115141	0.008081
2017-03	0.042452	-0.023344
2017-04	0.006053	-0.008351
2017-05	0.013280	-0.017715
2017-06	-0.014882	0.056763
2017-07	0.058938	0.040285
2017-08	0.017496	0.013581
2017-09	0.040320	0.029053
2017-10	0.028540	-0.063427
2017-11	-0.001799	0.089608
2017-12	0.007340	-0.029434
2018-01	0.119895	0.053104
2018-02	-0.036374	-0.001977
2018-03	-0.053337	0.067740
2018-04	-0.023398	0.024356
2018-05	0.061800	0.040082
2018-06	0.020207	-0.006423
2018-07	0.086013	0.001424
2018-08	0.045573	0.044201
2018-09	0.019340	-0.101006
2018-10	-0.145403	0.010606
2018-11	0.092924	0.053166
2018-12	-0.102875	-0.106767
2019-01	0.095176	0.079318
2019-02	0.040222	0.040095
2019-03	0.047593	-0.044556
2019-04	0.159775	0.053835
2019-05	-0.098671	-0.016920
2019-06	0.107063	0.061011
2019-07	0.030415	0.002080
2019-08	-0.014159	0.036007
2019-09	0.008361	-0.004908
2019-10	0.036338	0.004227

Date	strategy	nasdaq
2019-11	0.047995	0.050416
2019-12	0.074848	0.021947
2020-01	0.005658	0.087395
2020-02	-0.021887	-0.119440
2020-03	-0.040929	-0.074110
2020-04	0.128304	0.155029
2020-05	0.026639	0.080150
2020-06	0.014150	0.008526
2020-07	0.093098	0.099104
2020-08	0.062932	0.023684
2020-09	-0.047994	-0.087115
2020-10	-0.024773	-0.014017
2020-11	0.101177	0.057856
2020-12	0.026548	0.037112

Comparing the Returns of the Strategy with the NASDAQ

Import `teersheet` module to get full performance metric.

In [29]:
```
import tearsheet as ts
```

In [30]:
```
tear = ts.tear_sheet(frame_final, headline='Quantemantal Strategy')
```

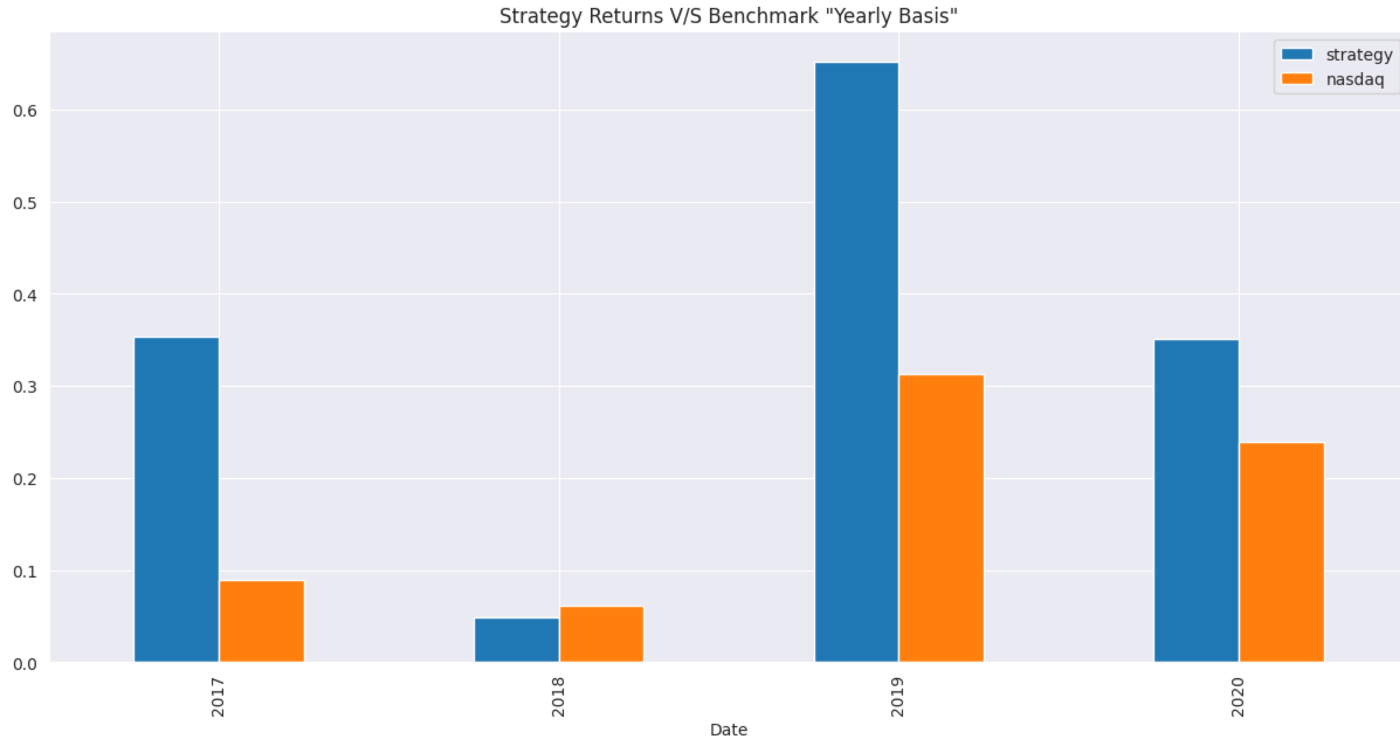
```
Performance Metrics: Quantemantal Strategy
```

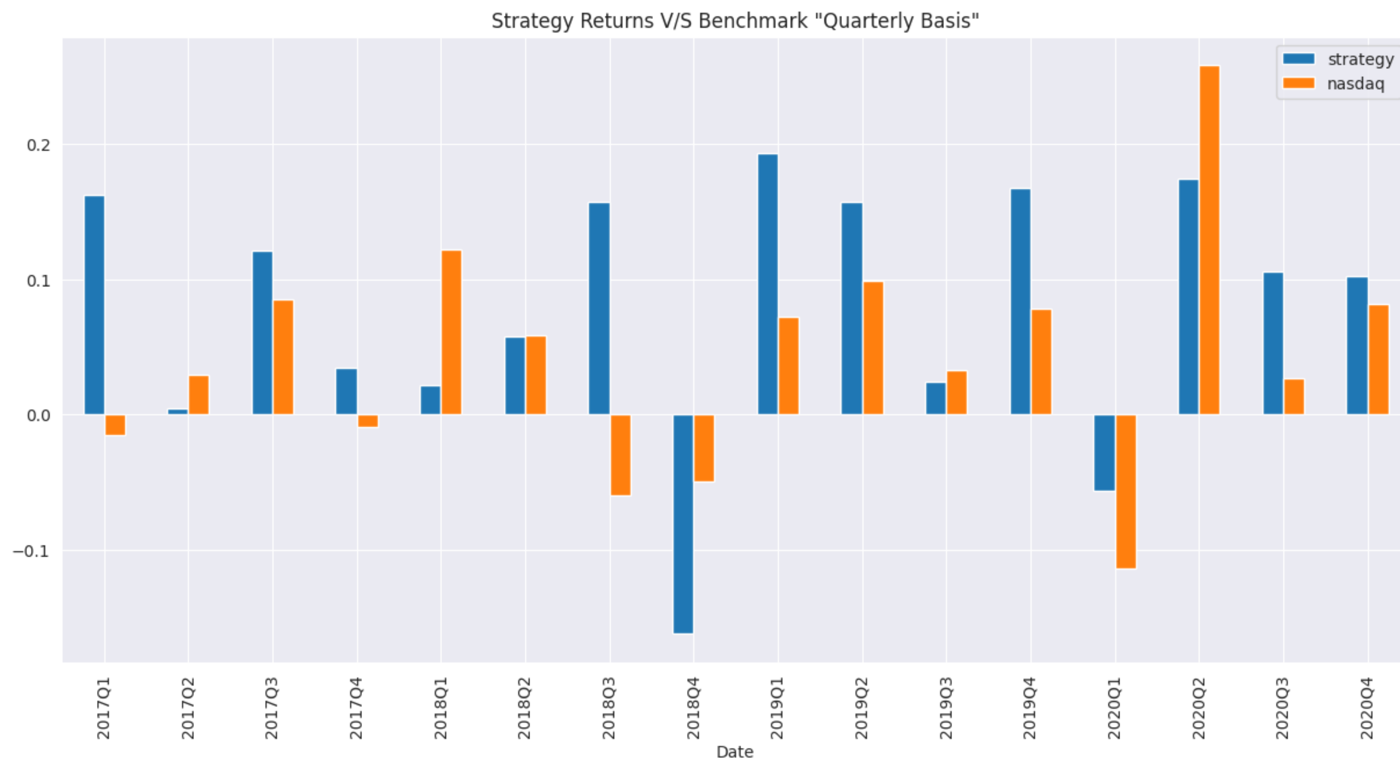
	strategy	nasdaq
Start-Date	2017-02	2017-02
End-Date	2020-12	2020-12
Total-Months-Tested	47	47
Monthly-Returns	2.48	1.35
CAGR	34.21	17.52
Absolute-Return	183.91	86.67
Monthly-Volatility	6.22	5.66
Annual-Volatility	21.54	19.59
Sharpe-Ratio	1.53	0.83
Sortino-Ratio	2.25	1.15
Calmar-Ratio	2.11	0.95

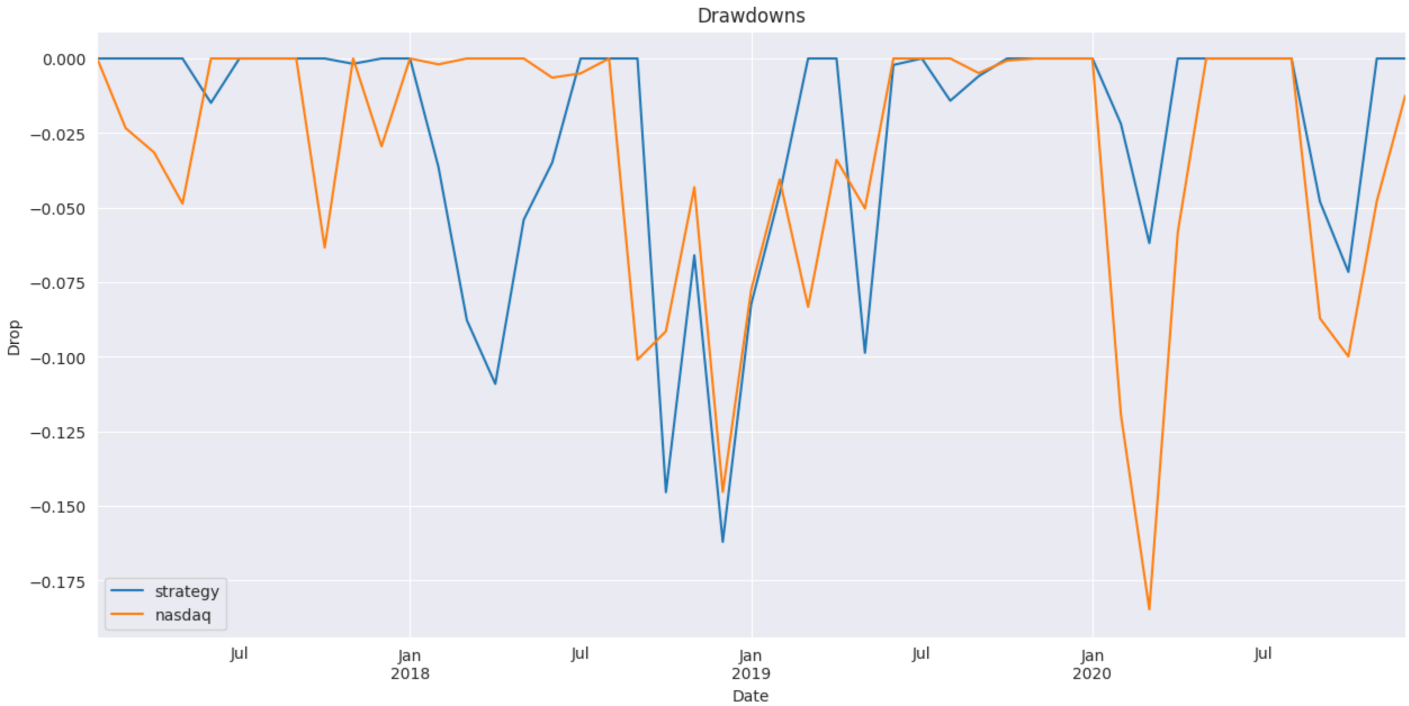
Skewness	-0.39	-0.4
Excess of Kurtosis	0.53	0.47
Cornish-Fischer-Var	8.41	8.6

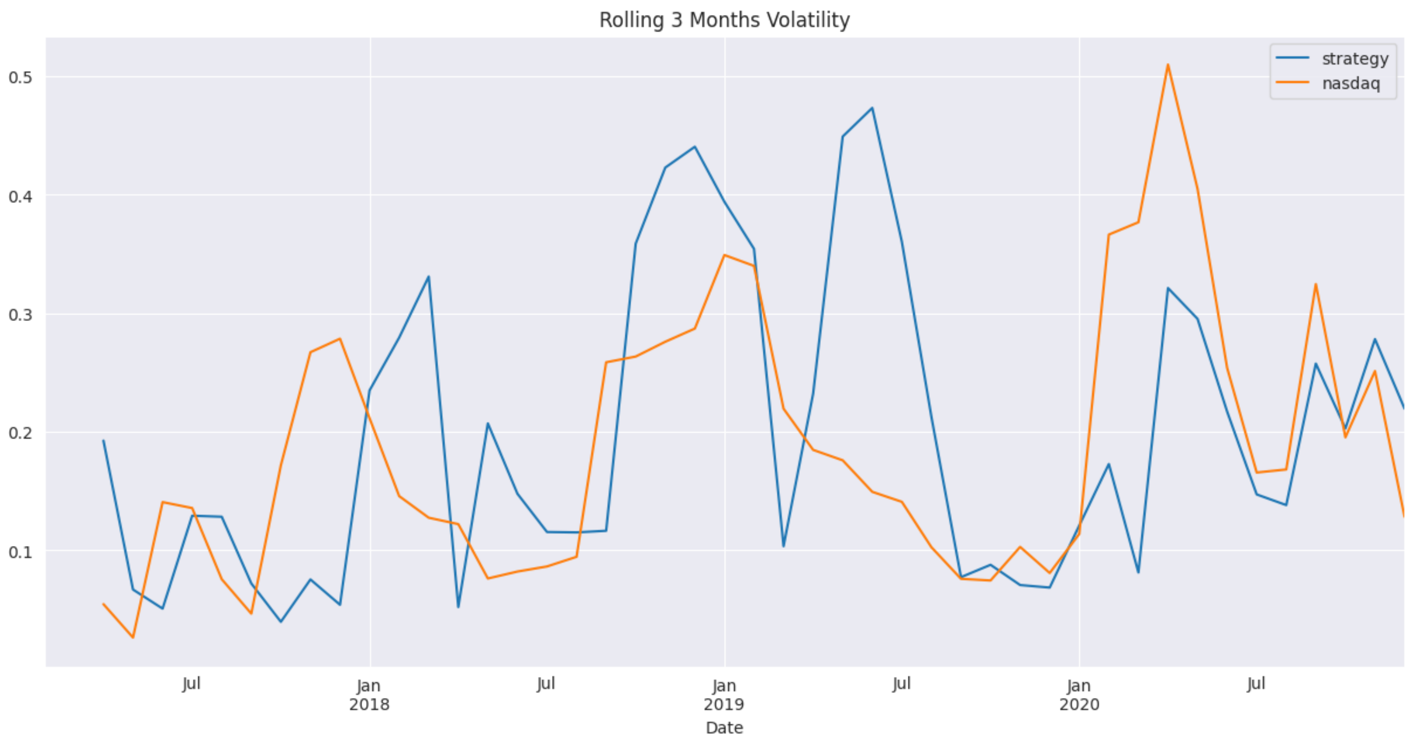
	strategy Yearly Returns	nasdaq Yearly Returns
2017	35.3349	8.91693
2018	4.81956	6.16946
2019	65.1486	31.2983
2020	35.1394	23.9402

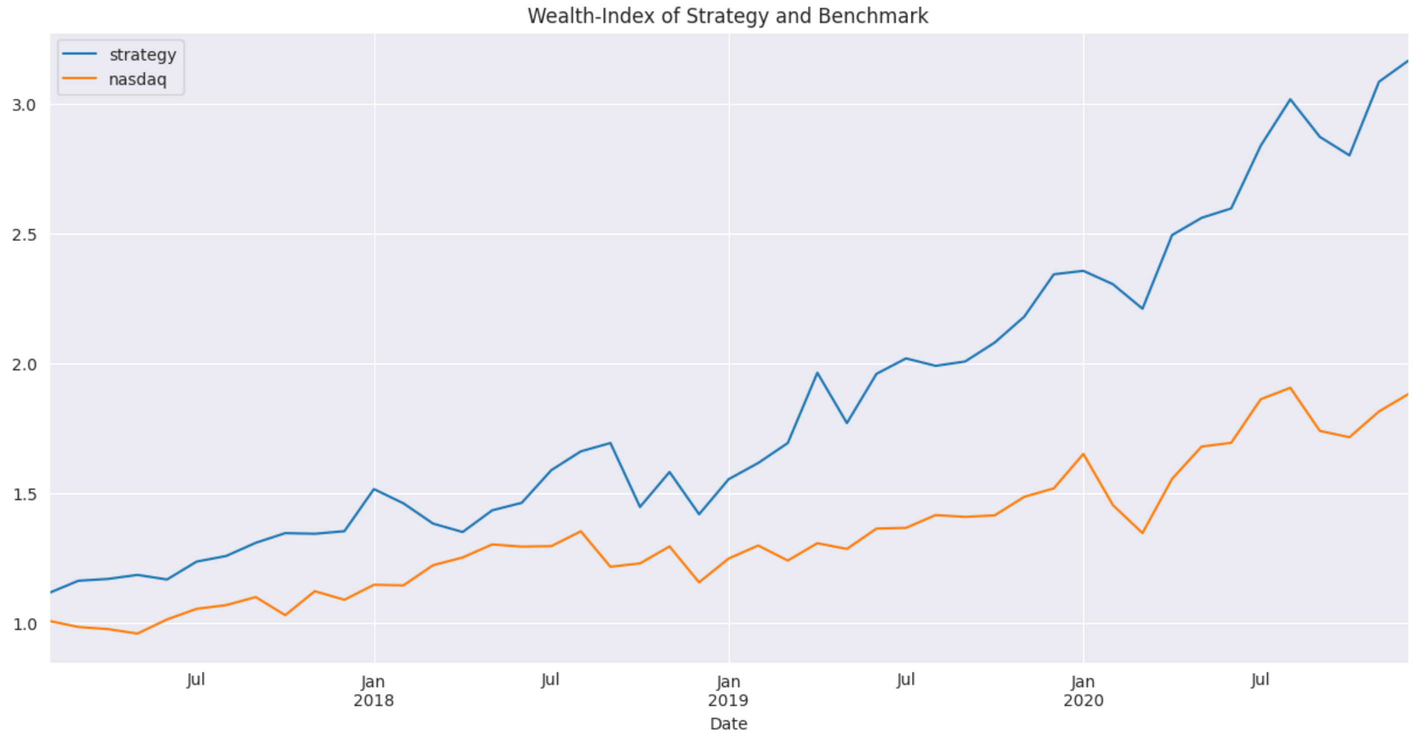
	strategy Quarterly Returns	nasdaq Quarterly Returns
2017Q1	16.2481	-1.54522
2017Q2	0.42424	2.93737
2017Q3	12.0908	8.50469
2017Q4	3.42254	-0.953974
2018Q1	2.16004	12.2218
2018Q2	5.79099	5.85711
2018Q3	15.7466	-5.9932
2018Q4	-16.2077	-4.93007
2019Q1	19.3445	7.25757
2019Q2	15.7256	9.92113
2019Q3	2.43186	3.30665
2019Q4	16.7368	7.8007
2020Q1	-5.66133	-11.3445
2020Q2	17.4751	25.8241
2020Q3	10.6125	2.71197
2020Q4	10.2407	8.17375

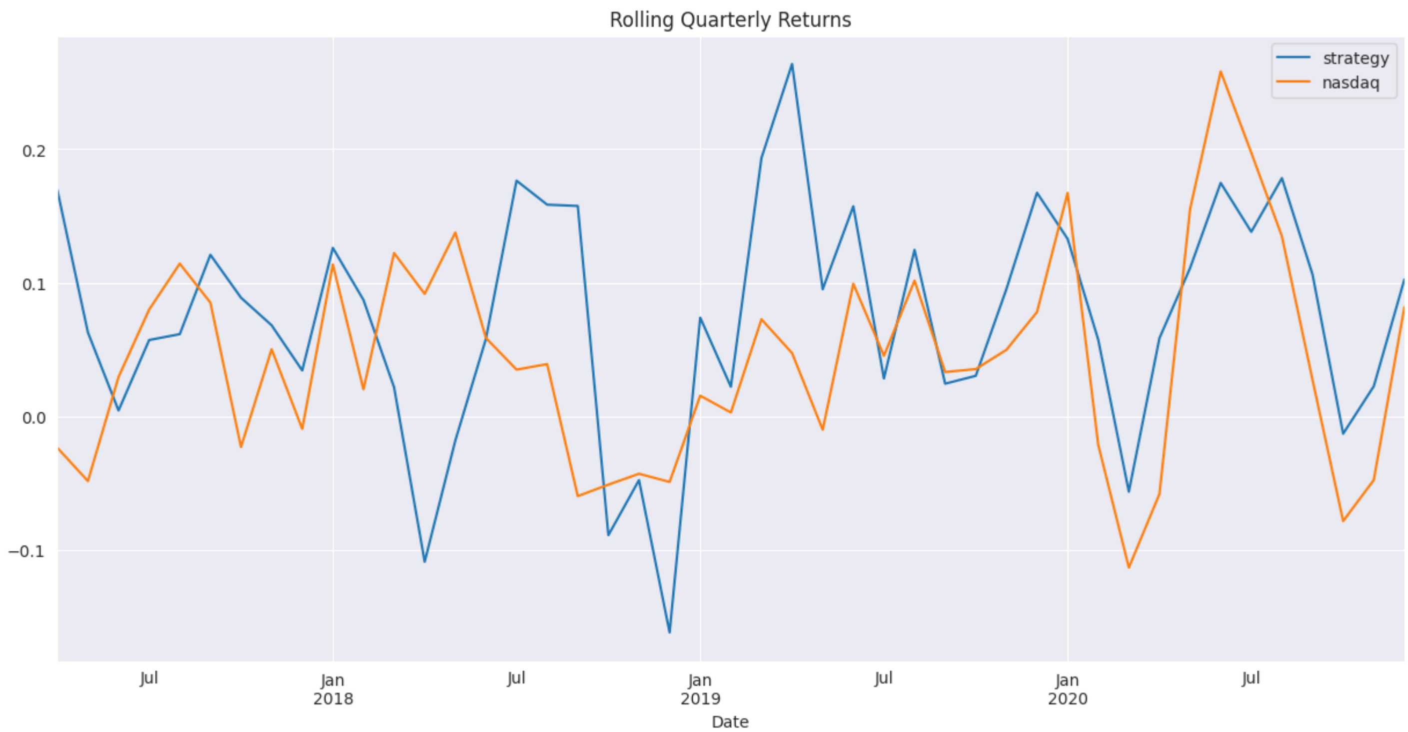

Strategy Returns V/S Benchmark "Yearly Basis"
strategy
nasdaq
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2017
2018
2019
2020
Date


Strategy Returns V/S Benchmark "Quarterly Basis"
strategy
nasdaq
0.2
0.1
0.0
−0.1
2017Q1
2017Q2
2017Q3
2017Q4
2018Q1
2018Q2
2018Q3
2018Q4
2019Q1
2019Q2
2019Q3
2019Q4
2020Q1
2020Q2
2020Q3
2020Q4
Date

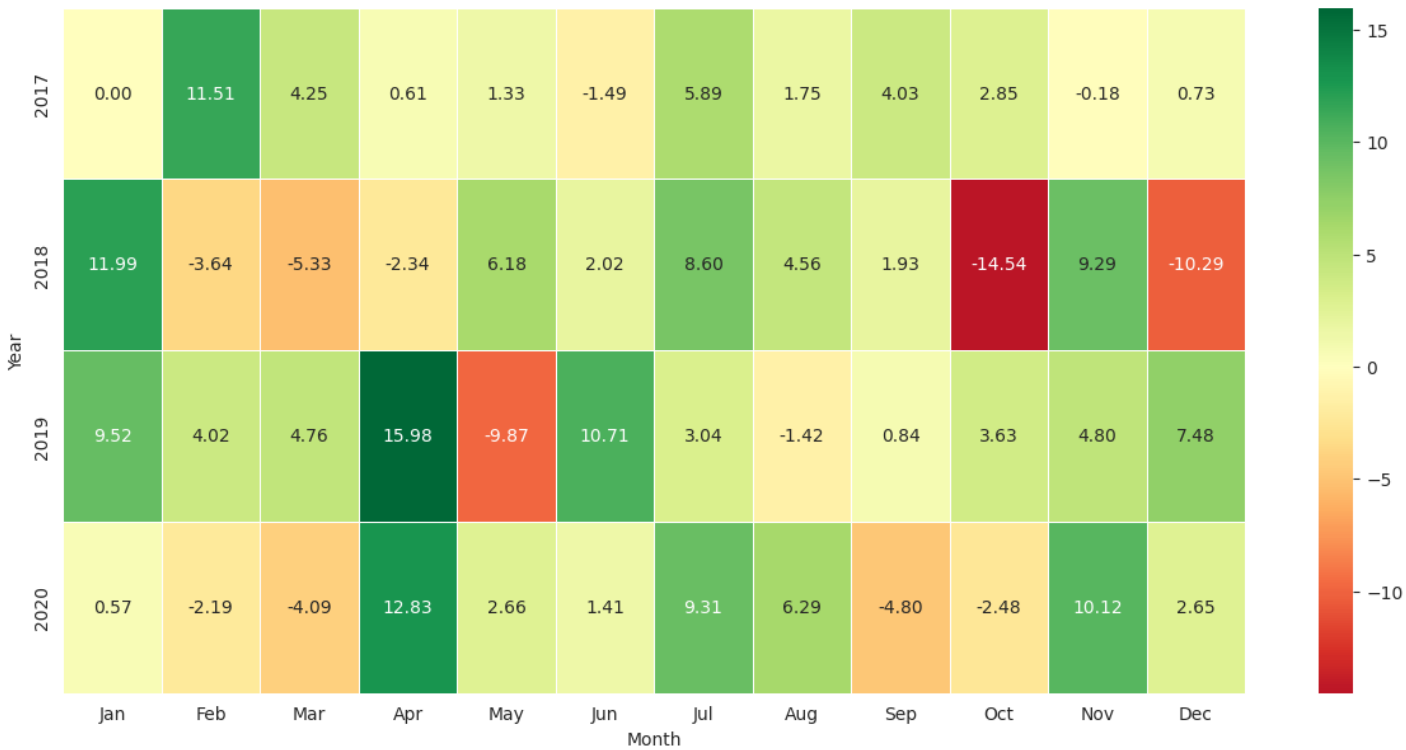

Drawdowns
Drop
0.000
−0.025
−0.050
−0.075
−0.100
−0.125
−0.150
−0.175
strategy
nasdaq
Jul
Jan
2018
Jul
Jan
2019
Jul
Jan
2020
Jul
Date


Rolling 3 Months Volatility
0.5
0.4
0.3
0.2
0.1
strategy
nasdaq
Jul
Jan
2018
Jul
Jan
2019
Jul
Jan
2020
Jul
Date

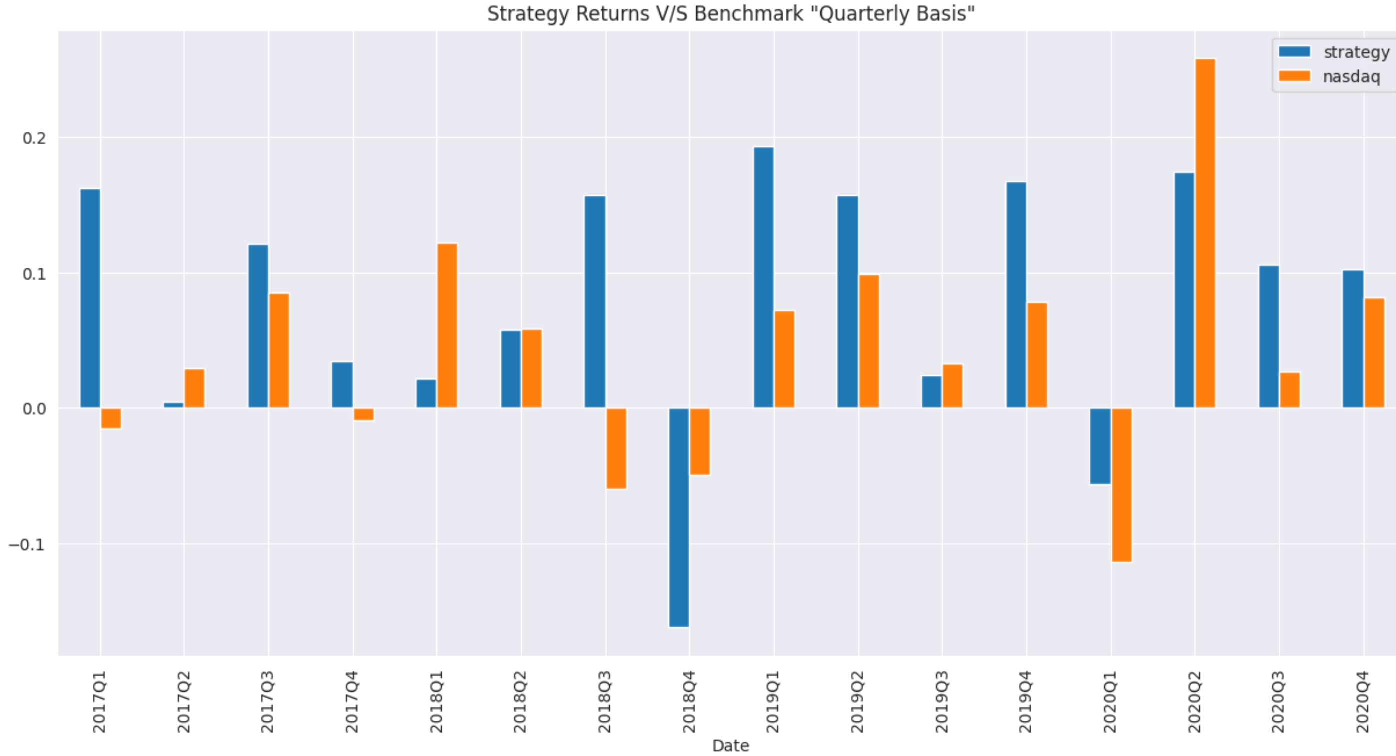

Wealth-Index of Strategy and Benchmark
strategy
nasdaq
3.0
2.5
2.0
1.5
1.0
Jul
Jan
2018
Jul
Jan
2019
Jul
Jan
2020
Jul
Date

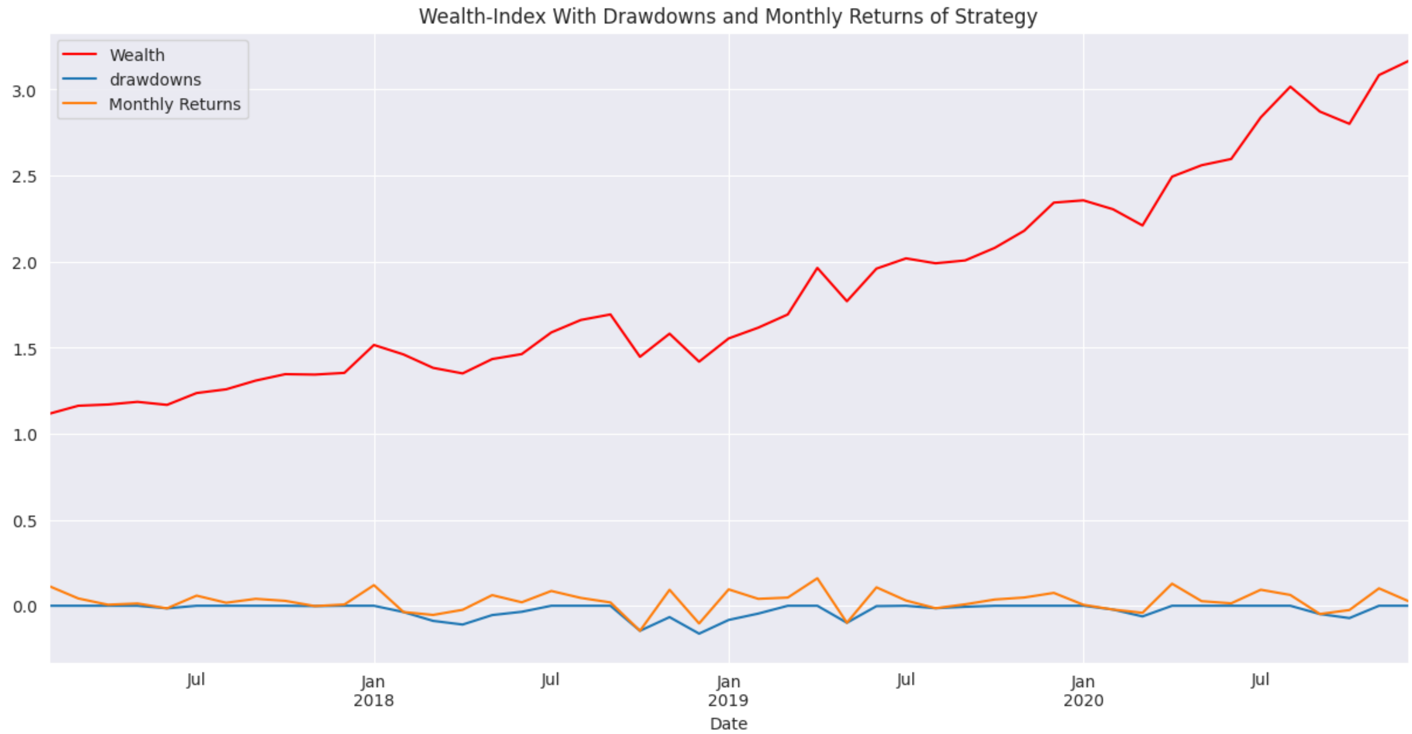

Rolling Quarterly Returns
strategy
nasdaq
0.2
0.1
0.0
−0.1
Jul
Jan
2018
Jul
Jan
2019
Jul
Jan
2020
Jul
Date

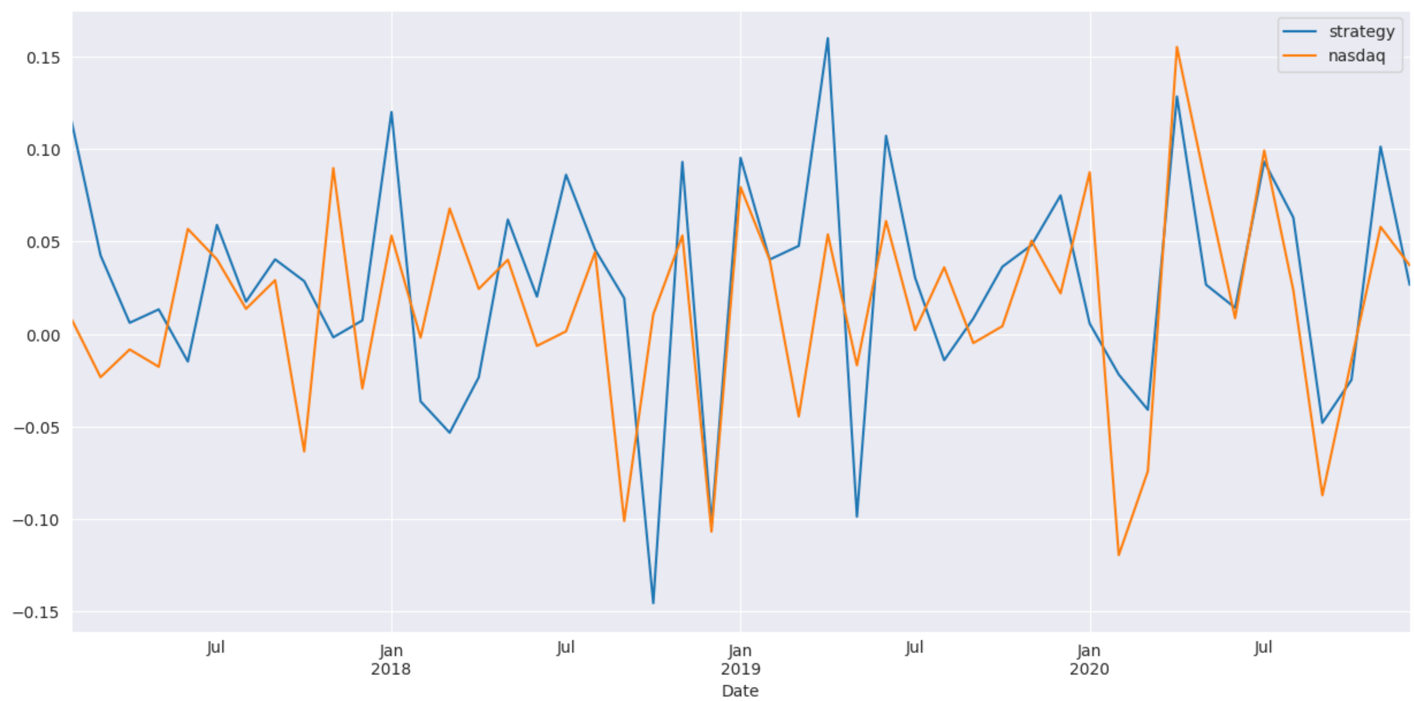
Monthly Returns (%)



Strategy Returns V/S Benchmark "Quarterly Basis"







Replicando estudo sem nenhuma alteração para as 20 principais ações do Ibovespa

In [31]:

```
tickers = ["VALE3.SA","PETR4.SA","ITUB4.SA","BBDC4.SA","B3SA3.SA"
           ,"ELET3.SA","BBAS3.SA","ABEV3.SA","RENT3.SA","ITSA4.SA",
           "WEGE3.SA","CSAN3.SA","LREN3.SA","EQTL3.SA","RADL3.SA",
           "SUZB3.SA","PRIO3.SA","GGBR4.SA","JBSS3.SA","BBDC3.SA"]

len(tickers)
```

Out[31]:

```
20
```

In [32]:

```
# WRT to cash, income and book value
columns = ['F-Score-Max', 'F-Score-Min', 'G-Score']
```

```python
ratio = ['peRatio', 'priceToSalesRatio', 'pbRatio', 'evToSales',
         'debtToEquity', 'evToFreeCashFlow', 'debtToAssets', 'netDebtToEBITDA']

# Ratios to derive out the value factor
Full_table = pd.DataFrame()
year = ['2011','2012','2013','2014','2015','2016','2017','2018', '2019', '2020']

for steps1 in range(len(year)):
    for steps2 in range(len(tickers)):
        p = str(tickers[steps2])
        data = fa.key_metrics(
            ticker=p, api_key=key, period='annual')
        table = data.T[ratio]
        table = table.loc[year[steps1]]
        Full_table = Full_table.append(table)

Full_table
```

Out[32]:

	peRatio	priceToSalesRatio	pbRatio	evToSales	debtToEquity	evToFreeCashFlow	debtToAssets	netDeb
2011	4.849135	1.881208	1.427941	2.236777	0.315332	15.668858	0.190370	
2011	7.480568	1.031536	0.855995	1.469256	0.471610	-64.457760	0.259626	
2011	11.126058	2.072692	2.082083	2.190671	1.965148	-14.380404	0.177605	
2011	5.349839	1.129452	0.991288	0.643243	1.158981	-0.795088	0.094953	
2011	18.232143	10.036799	0.993552	10.036799	0.000000	13.007715	0.000000	
...	...	...	...	...	...	...	...	
2020	-7.371343	2.592999	10.921654	4.932327	10.798336	18.262323	0.767103	
2020	20.979705	4.988432	3.015127	5.895143	0.804916	14.771019	0.373308	
2020	17.417276	0.949303	1.347797	1.263752	0.596063	12.029619	0.291410	
2020	13.661434	0.232489	1.565817	0.426163	1.794925	6.488485	0.439624	
2020	13.502600	2.596121	1.468469	3.710879	1.496448	2.212232	0.135801	

200 rows × 8 columns

In [33]:

```python
rank = pd.DataFrame()

for steps in range(len(year)):
    y = Full_table.loc[year[steps]]
    t_inv = 1/y
    bv = zscore2(t_inv)
    bv = bv.T
    bv.index = [year[steps]]
    bv.columns = tickers

    rank = rank.append(bv)
```

In [34]:

```python
name = pd.DataFrame(tickers)
concated = pd.concat([name]*len(year), axis=0)
concated.index = Full_table.index
concated.columns = ['Company']

lista_z_score = []
for i in year:
    lista_z_score += list(rank.loc[i])
```

```
lista_z_score = pd.DataFrame(lista_z_score,index=Full_table.index,columns=['Z-Score'])
rank = pd.concat([concated, lista_z_score], axis=1)

rank
```

Out[34]:

	Company	Z-Score
2011	VALE3.SA	1.198905
2011	PETR4.SA	1.282752
2011	ITUB4.SA	0.797955
2011	BBDC4.SA	1.099543
2011	B3SA3.SA	0.751432
...	...	...
2020	SUZB3.SA	0.525550
2020	PRIO3.SA	0.767735
2020	GGBR4.SA	1.164545
2020	JBSS3.SA	1.979066
2020	BBDC3.SA	1.181850

200 rows × 2 columns

Now doing the Intra-Company analysis, comparing the past performances of the companies to derive out fundamentally strong companies.

In [35]:

```
f_score_ratio_max = ['roe', 'returnOnTangibleAssets',
                     'researchAndDdevelopementToRevenue', 'currentRatio']

# WRT to assets and liability
f_score_ratio_min = ['averagePayables',
                     'daysOfInventoryOnHand', 'interestDebtPerShare']

# Ratios where change must reduce Q/Q
g_score_ratio_max = ['netIncomePerShare', 'freeCashFlowPerShare',
                     "shareholdersEquityPerShare", 'cashPerShare', 'bookValuePerShare', ':


percent_fscore_max = pd.DataFrame()
percent_fscore_min = pd.DataFrame()
percent_gscore_max = pd.DataFrame()

for steps in range(len(tickers)):
    # Fscore-Max analysis
    p = str(tickers[steps])
    data3 = fa.key_metrics(
        ticker=p, api_key=key, period='annual')
    tranpose = data3.T[f_score_ratio_max].fillna(0)
    reverse = tranpose.loc[::-1]
    change = reverse.pct_change()
    change = change.assign(Company=p)
    percent_fscore_max = percent_fscore_max.append(change)

    # Fscore-Min analysis
    data4 = fa.key_metrics(
        ticker=p, api_key=key, period='annual')
    tranpose1 = data4.T[f_score_ratio_min].fillna(0)
    reverse1 = tranpose1.loc[::-1]
    change1 = reverse1.pct_change()
```

```python
    change1 = change1.assign(Company=p)
    percent_fscore_min = percent_fscore_min.append(change1)

    # Gscore-Max analysis
    data5 = fa.key_metrics(
        ticker=p, api_key=key, period='annual')
    tranpose2 = data5.T[g_score_ratio_max].fillna(0)
    reverse2 = tranpose2.loc[::-1]
    change2 = reverse2.pct_change()
    change2 = change2.assign(Company=p)
    percent_gscore_max = percent_gscore_max.append(change2)

# Condition to quantify the performance
condition = np.where(percent_fscore_max.iloc[:, :-1] > 0, 1, -1)
condition = pd.DataFrame(condition)
condition.index = percent_fscore_max.index
condition.columns = percent_fscore_max.columns[:-1]
condition
sum_score = condition.sum(axis=1)
sum_score = pd.DataFrame(sum_score)
sum_score = pd.concat([sum_score, percent_fscore_max['Company']], axis=1)
sum_score.columns = ['F-Score-Max','Company']
sum_score

# Condition to quantify the performance
condition2 = np.where(percent_fscore_min.iloc[:, :-1] < 0, 1, -1)
condition2 = pd.DataFrame(condition2)
condition2.index = percent_fscore_min.index
condition2.columns = percent_fscore_min.columns[:-1]
condition2
sum_score2 = condition2.sum(axis=1)
sum_score2 = pd.DataFrame(sum_score2)
sum_score2 = pd.concat([sum_score2, percent_fscore_min['Company']], axis=1)
sum_score2.columns = ['F-Score-Min','Company']
sum_score2

# Condition to quantify the performance
condition3 = np.where(percent_gscore_max.iloc[:, :-1] > 0, 1, -1)
condition3 = pd.DataFrame(condition3)
condition3.index = percent_gscore_max.index
condition3.columns = percent_gscore_max.columns[:-1]
sum_score3 = condition3.sum(axis=1)
sum_score3 = pd.DataFrame(sum_score3)
sum_score3 = pd.concat([sum_score3, percent_gscore_max['Company']], axis=1)
sum_score3.columns = ['G-Score-Max','Company']
```

In [36]:

```python
Final_yr_wise = pd.concat([sum_score['F-Score-Max'], sum_score2['F-Score-Min'], sum_score3

print(rank.loc[year[0:]].head())
Final_yr_wise.loc[year[0:]].head()
```

```
        Company   Z-Score
2011  VALE3.SA  1.198905
2011  PETR4.SA  1.282752
2011  ITUB4.SA  0.797955
2011  BBDC4.SA  1.099543
2011  B3SA3.SA  0.751432
```

Out[36]:

	F-Score-Max	F-Score-Min	G-Score-Max	Company
2011	4	1	2	VALE3.SA
2011	2	1	-4	PETR4.SA
2011	2	1	-4	ITUB4.SA

	F-Score-Max	F-Score-Min	G-Score-Max	Company
2011	-2	-1	2	BBDC4.SA
2011	-4	-3	2	B3SA3.SA

In [37]:

```
company = pd.DataFrame(Final_yr_wise['Company'])

vc = zscore2(Final_yr_wise[Final_yr_wise.columns[:-1]], headline='Cumulative-F&G-Score')
vc['Company'] = company

rank = rank.loc[year[0]:]

# Now Joing the two Dataframes and getting a cumulative score of each company for each yea
vc_new = vc.reset_index()
vc_new['key'] = vc_new['index']+str('_')+vc_new['Company']
vc_new.head(3)
rank_new = rank.reset_index()
rank_new['key'] = rank_new['index']+str('_')+rank_new['Company']
df = vc_new.merge(rank_new, on='key')
df = df.drop(labels=['Company_x', 'key', 'index_y'], axis=1)
df.index = df['index_x']
dataframe = df.drop(labels='index_x', axis=1)
dataframe['Cumulative-Score'] = dataframe['Cumulative-F&G-Score'] + \
    dataframe['Z-Score'].values
dataframe_final = dataframe.drop(
    labels=['Cumulative-F&G-Score', 'Z-Score'], axis=1)
dataframe_final.head(4)
```

Out[37]:

	Company_y	Cumulative-Score
index_x		
2011	VALE3.SA	3.627003
2012	VALE3.SA	1.590086
2013	VALE3.SA	2.420884
2014	VALE3.SA	2.408017

In [38]:

```
stock = yf.download(tickers=tickers, start='2011-01-01',
                    end='2020-12-31', interval='1mo')[['Close']]

stock.index = pd.to_datetime(stock.index)
stock = (stock.T.drop_duplicates().T).dropna()
stock.columns = tickers
returns = stock.pct_change().dropna()
```

```
[*********************100%***********************]  20 of 20 completed
```

Assumindo as posições das 6 maiores Empresas, com peso igual

In [39]:

```
df = pd.DataFrame()
df2 = pd.DataFrame()

for steps in range(len(year)):
    sorted_table = (dataframe_final.loc[year[steps]]).sort_values(
        by='Cumulative-Score', ascending=False)
    selection = sorted_table[['Company_y']].head(6)
    array = np.array(selection['Company_y'])
    frame1 = pd.DataFrame(array)
    returns_selection = returns.loc[year[steps]]
    selected_returns = returns_selection[array]
```

```
    # Making the portfolio with equal weights among its components
    weight = np.repeat(1/6, 6)
    series = (selected_returns*weight).sum(axis=1)
    frame = pd.DataFrame(series)
    df = df.append(frame)
    df2 = df2.append(frame1)
```

In [40]:

```
comp_index = np.repeat(year, 6)
df2.index = comp_index
df2.columns = ['Companies-Invested']

list_comp = pd.DataFrame()
for steps in range(len(year)):
    f = df2.loc[year[steps]]['Companies-Invested'].T
    mk = np.array(f)
    l = pd.DataFrame(mk).T
    list_comp = list_comp.append(l)

list_comp.index = year

list_comp
```

Out[40]:

	0	1	2	3	4	5
2011	VALE3.SA	ELET3.SA	JBSS3.SA	EQTL3.SA	PETR4.SA	BBAS3.SA
2012	CSAN3.SA	PRIO3.SA	ABEV3.SA	RENT3.SA	WEGE3.SA	ITSA4.SA
2013	ABEV3.SA	ELET3.SA	GGBR4.SA	LREN3.SA	JBSS3.SA	RENT3.SA
2014	JBSS3.SA	ITSA4.SA	RADL3.SA	ELET3.SA	ITUB4.SA	EQTL3.SA
2015	EQTL3.SA	PRIO3.SA	LREN3.SA	RADL3.SA	CSAN3.SA	JBSS3.SA
2016	PRIO3.SA	CSAN3.SA	SUZB3.SA	PETR4.SA	WEGE3.SA	JBSS3.SA
2017	JBSS3.SA	PETR4.SA	VALE3.SA	ABEV3.SA	SUZB3.SA	EQTL3.SA
2018	VALE3.SA	ABEV3.SA	CSAN3.SA	ITSA4.SA	PRIO3.SA	ELET3.SA
2019	BBAS3.SA	JBSS3.SA	WEGE3.SA	ELET3.SA	EQTL3.SA	B3SA3.SA
2020	JBSS3.SA	BBAS3.SA	WEGE3.SA	GGBR4.SA	RENT3.SA	VALE3.SA

Following DataFrame shows the list of the companies in which we Invested Year-Wise.

Weights- Equally Weighted.

In [41]:

```
df.columns = ['Returns']
df.index = pd.to_datetime(df.index).to_period('M')
```

Downloading the Benchmark 'IBOVESPA' and Comparing the performance with it.

In [42]:

```
benchmark = yf.download(tickers='^BVSP', start='2011-01-01',
                        end='2020-12-31', interval='1mo')[['Close']]
```

```
[*********************100%***********************]  1 of 1 completed
```

In [43]:

```
benchmark = (benchmark.T.drop_duplicates().T).dropna()
benchmark.index = pd.to_datetime(benchmark.index).to_period('M')
benchmark_returns = benchmark.pct_change().dropna()

frame_final = pd.concat([df, benchmark_returns], axis=1)
```

```
frame_final.columns = ['strategy', 'ibovespa']
frame_final
```

Out[43]:

Date	strategy	ibovespa
2011-02	0.007671	0.012137
2011-03	0.008670	0.017868
2011-04	-0.026741	-0.035779
2011-05	-0.136440	-0.022878
2011-06	-0.020166	-0.034293
...	...	...
2020-08	-0.000842	-0.034427
2020-09	-0.030777	-0.047963
2020-10	0.035161	-0.006881
2020-11	0.136237	0.158975
2020-12	0.107781	0.095676

119 rows × 2 columns

Comparing the Returns of the Strategy with the IBOVESPA

Import `teersheet` module to get full performance metric.

In [44]:

```
import tearsheet as ts
tear = ts.tear_sheet(frame_final, headline='Quantemantal Strategy para ações brasileiras')
```

```
Performance Metrics: Quantemantal Strategy para ações brasileiras
```

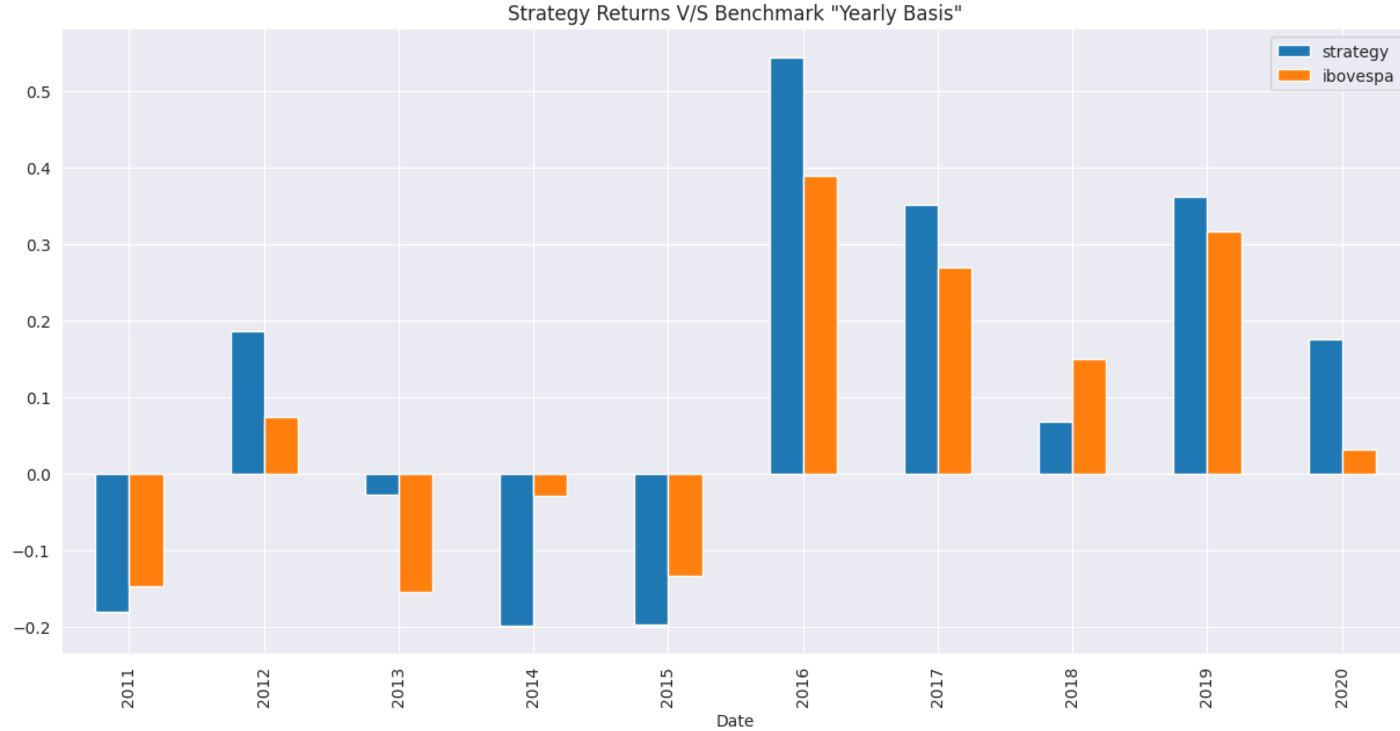
	strategy	ibovespa
Start-Date	2011-02	2011-02
End-Date	2020-12	2020-12
Total-Months-Tested	119	119
Monthly-Returns	0.65	0.49
CAGR	8.13	6.06
Absolute-Return	115.44	77.06
Monthly-Volatility	7.01	6.6
Annual-Volatility	24.27	22.86
Sharpe-Ratio	0.28	0.21
Sortino-Ratio	0.41	0.29
Calmar-Ratio	0.15	0.15
Skewness	-0.14	-0.6

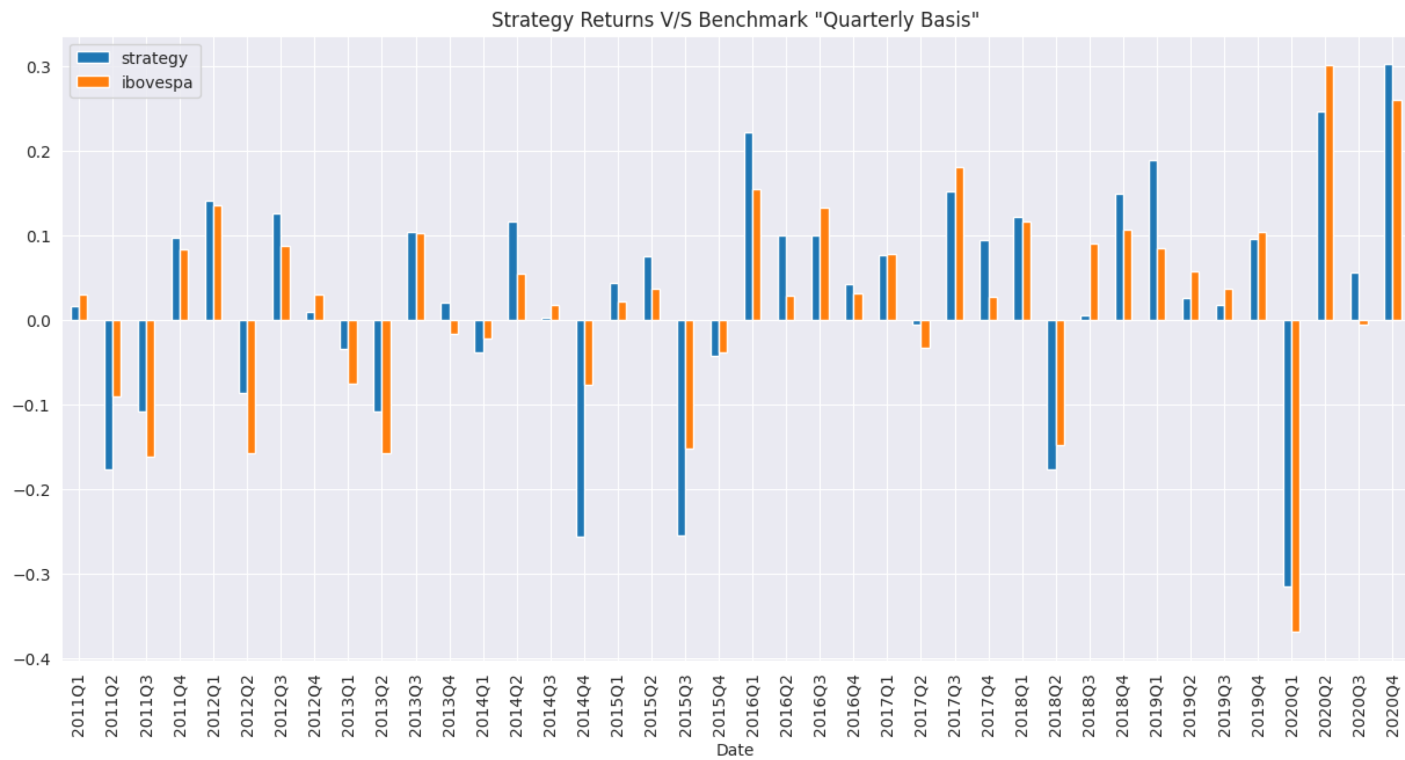
Excess of Kurtosis	1.11	3.03
Cornish-Fischer-Var	11.12	11.17

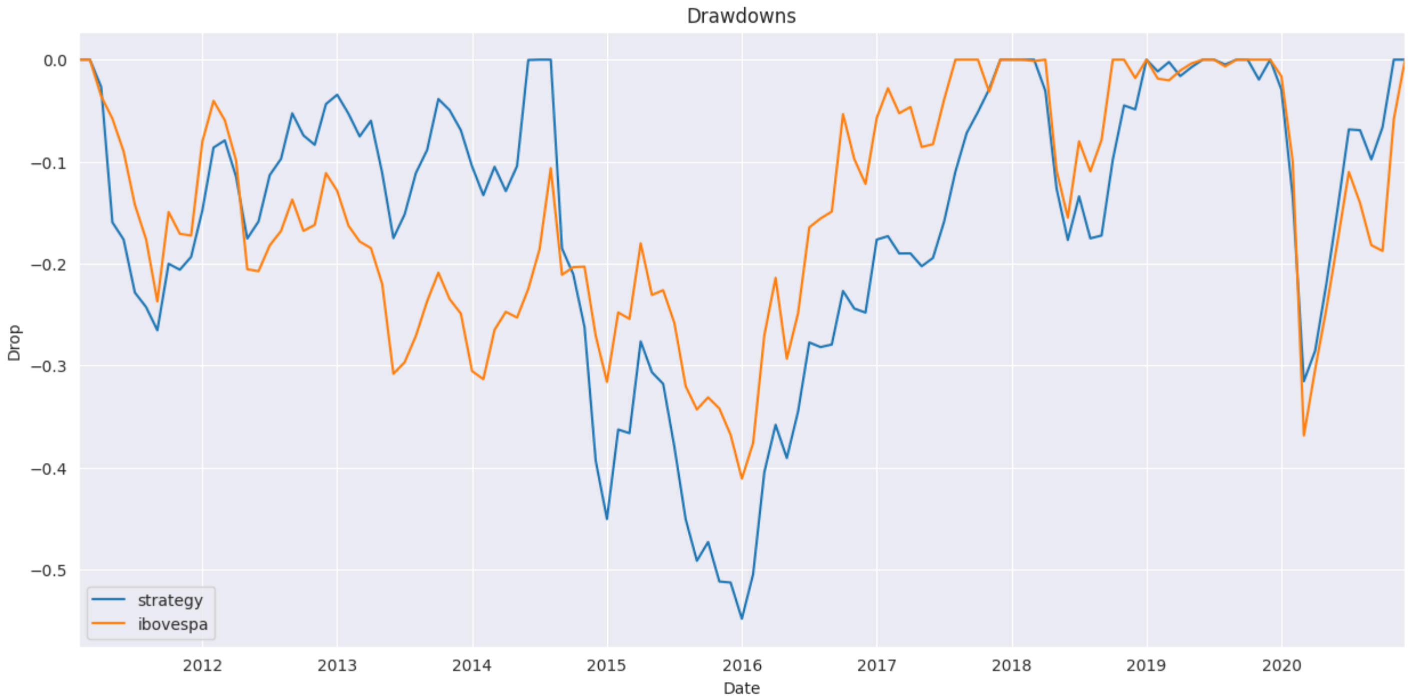
	strategy Yearly Returns	ibovespa Yearly Returns
2011	-18.026	-14.7518
2012	18.5922	7.39684
2013	-2.68781	-15.4958
2014	-19.8532	-2.91223
2015	-19.6756	-13.3121
2016	54.3439	38.9319
2017	35.1062	26.8567
2018	6.76237	15.0323
2019	36.2133	31.5837
2020	17.5455	3.16572

	strategy Quarterly Returns	ibovespa Quarterly Returns
2011Q1	1.64075	3.02216
2011Q2	-17.6481	-9.01483
2011Q3	-10.7998	-16.1528
2011Q4	9.79153	8.46648
2012Q1	14.1738	13.6678
2012Q2	-8.63957	-15.7431
2012Q3	12.6095	8.86947
2012Q4	0.961671	3.00122
2013Q1	-3.32052	-7.54692
2013Q2	-10.7836	-15.7847
2013Q3	10.4778	10.2851
2013Q4	2.12054	-1.58776
2014Q1	-3.84944	-2.1201
2014Q2	11.7031	5.46068
2014Q3	0.253136	1.78303
2014Q4	-25.566	-7.59295
2015Q1	4.4771	2.28568
2015Q2	7.62486	3.77517

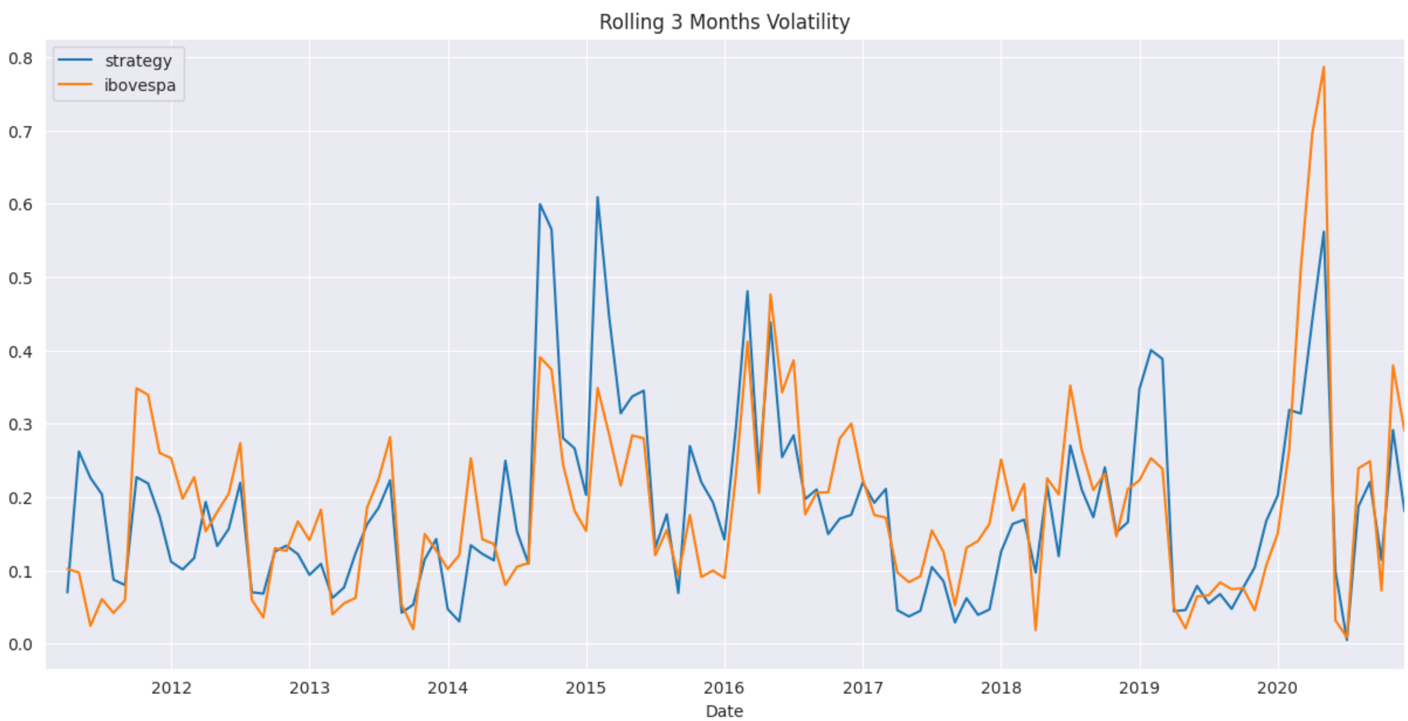
2015Q3	-25.4315	-15.1128
2015Q4	-4.20162	-3.7928
2016Q1	22.2219	15.4671
2016Q2	9.98742	2.94077
2016Q3	10.0306	13.2746
2016Q4	4.34789	3.18673
2017Q1	7.71347	7.89845
2017Q2	-0.54804	-3.20694
2017Q3	15.211	18.1145
2017Q4	9.47077	2.83738
2018Q1	12.2406	11.7327
2018Q2	-17.6883	-14.7635
2018Q3	0.532919	9.04168
2018Q4	14.9472	10.7698
2019Q1	18.9523	8.56554
2019Q2	2.63394	5.81879
2019Q3	1.76268	3.74182
2019Q4	9.63953	10.4062
2020Q1	-31.5389	-36.8585
2020Q2	24.6773	30.178
2020Q3	5.69248	-0.476561
2020Q4	30.2959	26.1123


Strategy Returns V/S Benchmark "Yearly Basis"
strategy
ibovespa
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
Date

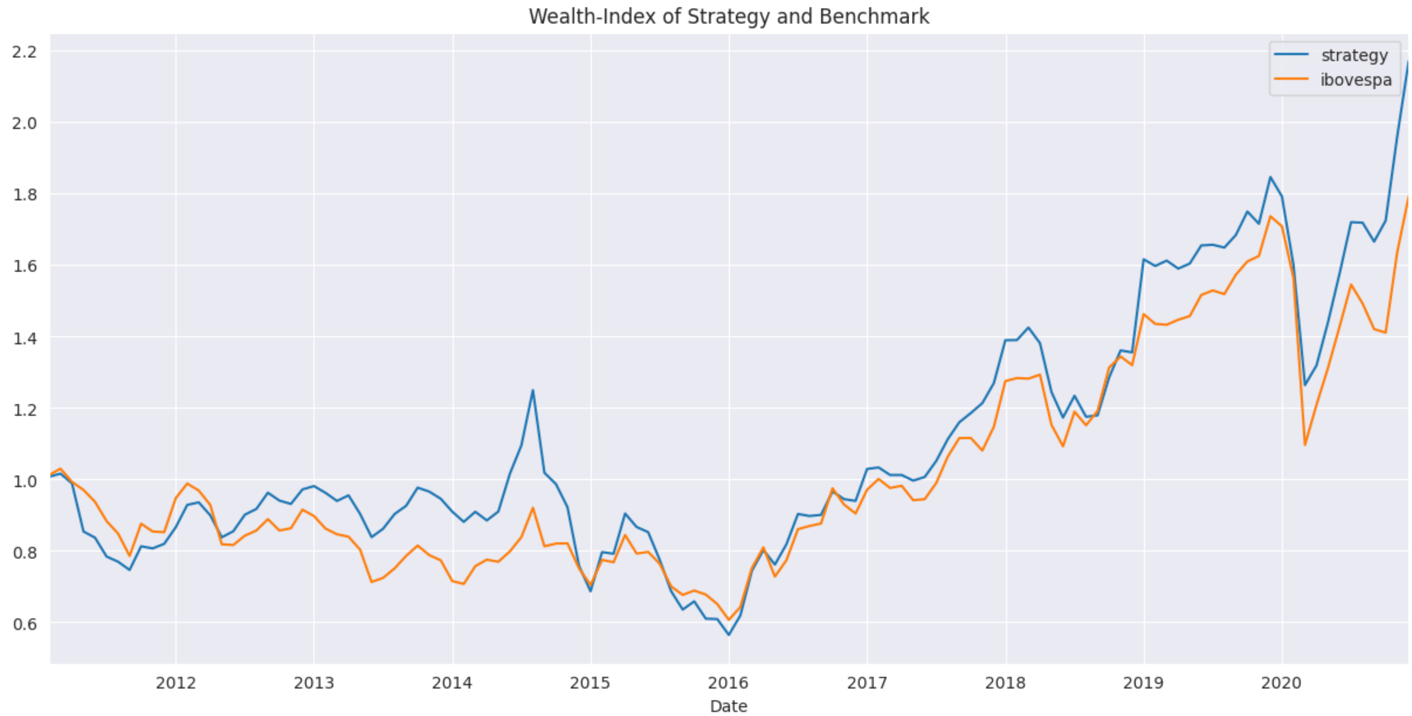

Strategy Returns V/S Benchmark "Quarterly Basis"
strategy
ibovespa
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
2011Q1
2011Q2
2011Q3
2011Q4
2012Q1
2012Q2
2012Q3
2012Q4
2013Q1
2013Q2
2013Q3
2013Q4
2014Q1
2014Q2
2014Q3
2014Q4
2015Q1
2015Q2
2015Q3
2015Q4
2016Q1
2016Q2
2016Q3
2016Q4
2017Q1
2017Q2
2017Q3
2017Q4
2018Q1
2018Q2
2018Q3
2018Q4
2019Q1
2019Q2
2019Q3
2019Q4
2020Q1
2020Q2
2020Q3
2020Q4
Date

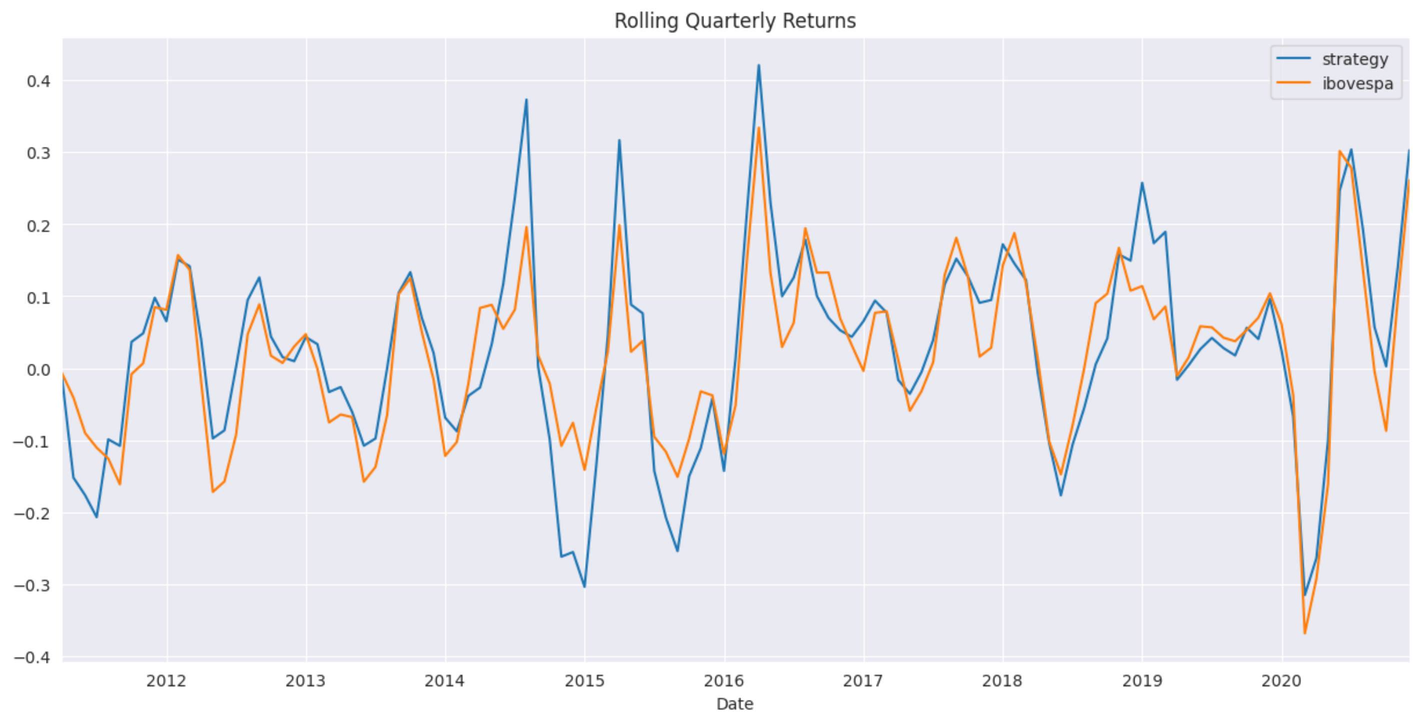


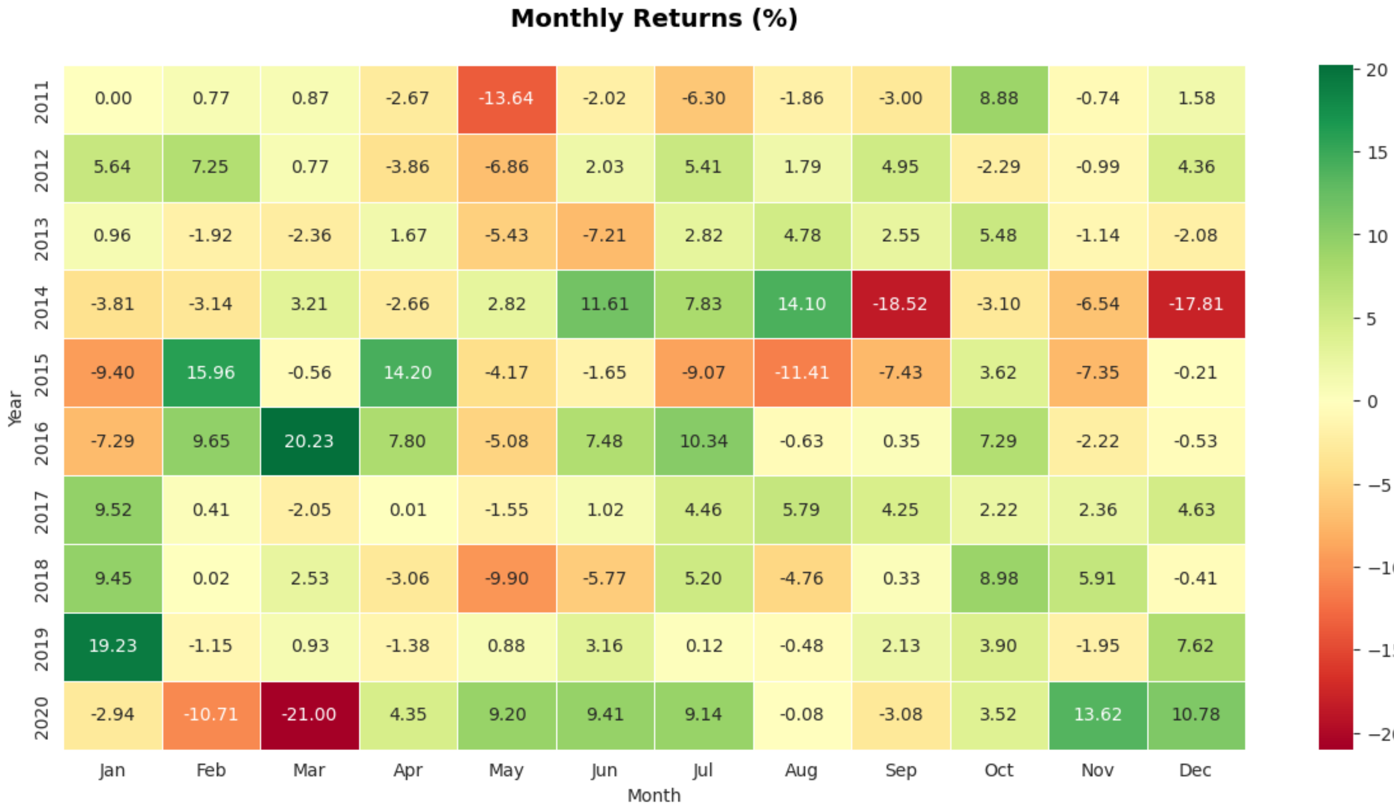
Drawdowns
Drop
Date
strategy
ibovespa
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
2012
2013
2014
2015
2016
2017
2018
2019
2020

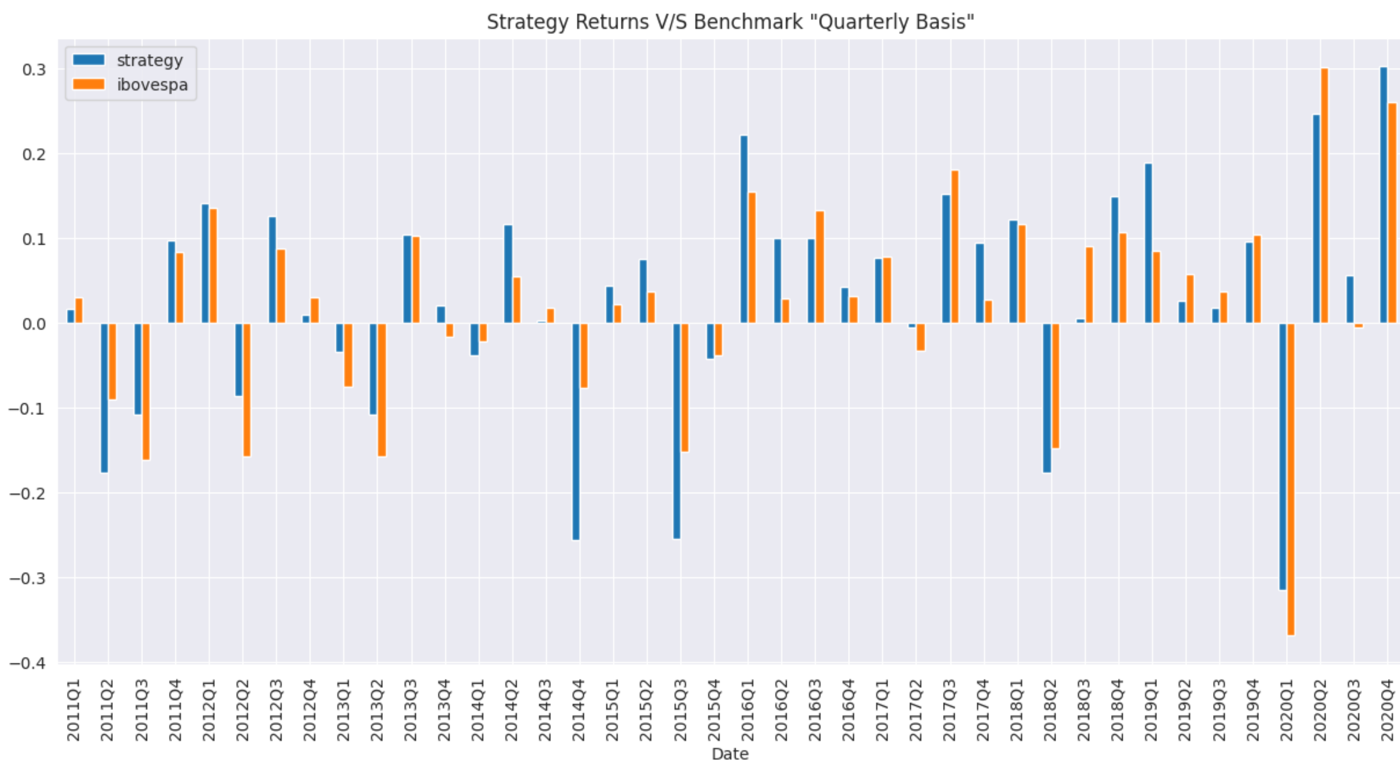


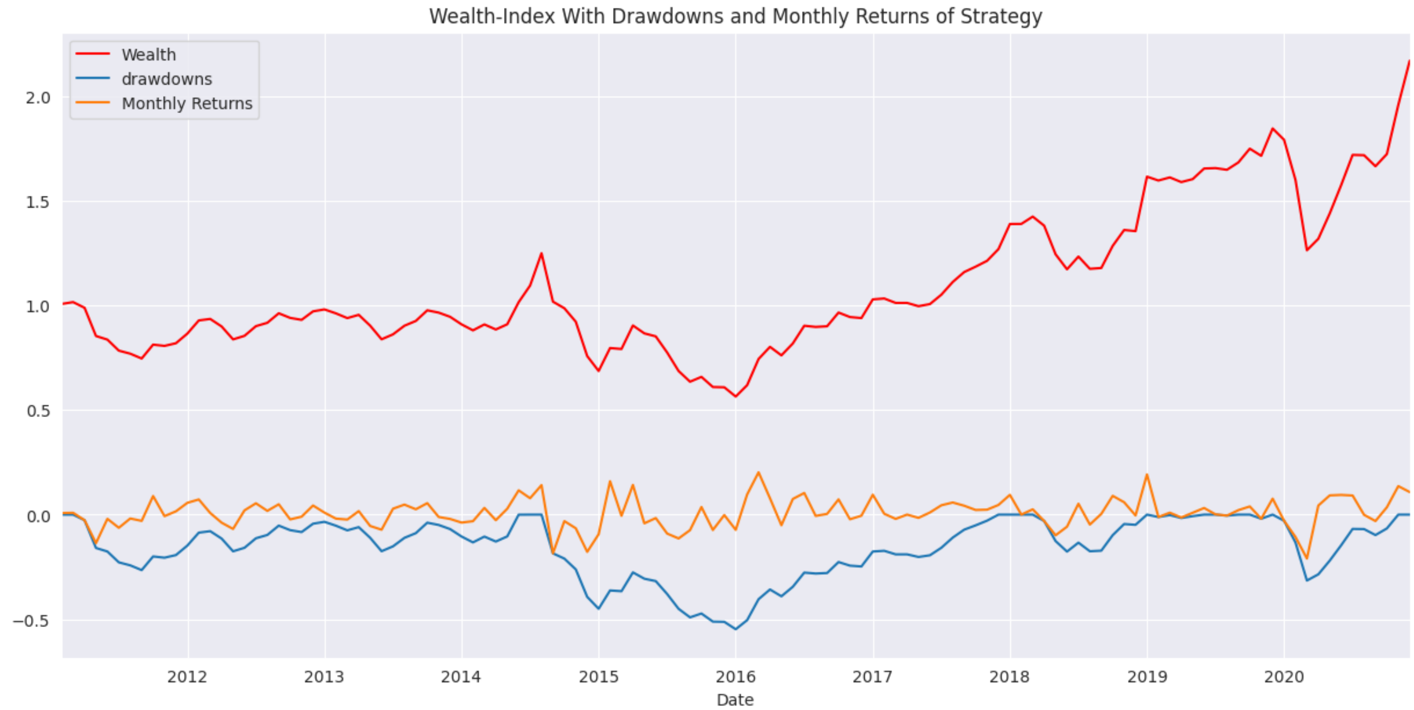
Rolling 3 Months Volatility
Date
strategy
ibovespa
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2012
2013
2014
2015
2016
2017
2018
2019
2020

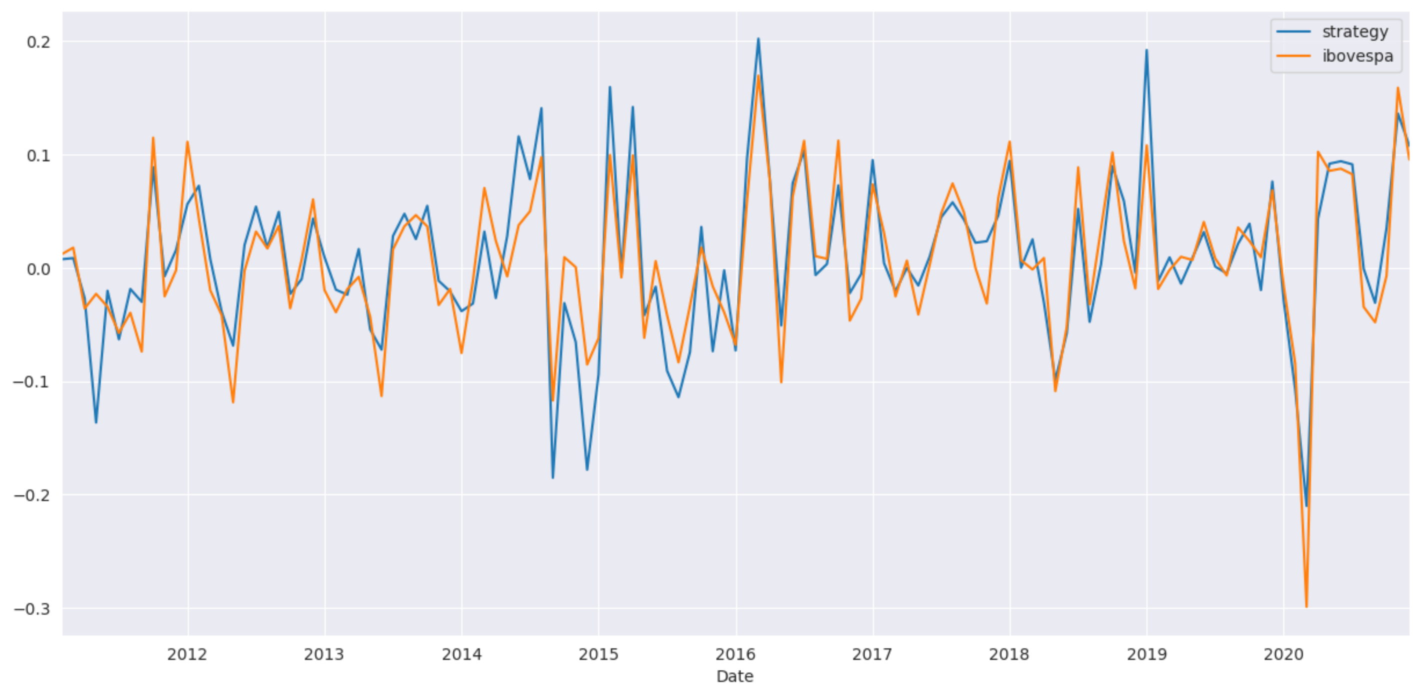

Wealth-Index of Strategy and Benchmark
strategy
ibovespa
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
2012
2013
2014
2015
2016
2017
2018
2019
2020
Date


Rolling Quarterly Returns
strategy
ibovespa
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
2012
2013
2014
2015
2016
2017
2018
2019
2020
Date


Monthly Returns (%)
Year
Month
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2011 0.00 0.77 0.87 -2.67 -13.64 -2.02 -6.30 -1.86 -3.00 8.88 -0.74 1.58
2012 5.64 7.25 0.77 -3.86 -6.86 2.03 5.41 1.79 4.95 -2.29 -0.99 4.36
2013 0.96 -1.92 -2.36 1.67 -5.43 -7.21 2.82 4.78 2.55 5.48 -1.14 -2.08
2014 -3.81 -3.14 3.21 -2.66 2.82 11.61 7.83 14.10 -18.52 -3.10 -6.54 -17.81
2015 -9.40 15.96 -0.56 14.20 -4.17 -1.65 -9.07 -11.41 -7.43 3.62 -7.35 -0.21
2016 -7.29 9.65 20.23 7.80 -5.08 7.48 10.34 -0.63 0.35 7.29 -2.22 -0.53
2017 9.52 0.41 -2.05 0.01 -1.55 1.02 4.46 5.79 4.25 2.22 2.36 4.63
2018 9.45 0.02 2.53 -3.06 -9.90 -5.77 5.20 -4.76 0.33 8.98 5.91 -0.41
2019 19.23 -1.15 0.93 -1.38 0.88 3.16 0.12 -0.48 2.13 3.90 -1.95 7.62
2020 -2.94 -10.71 -21.00 4.35 9.20 9.41 9.14 -0.08 -3.08 3.52 13.62 10.78
20
15
10
5
0
−5
−10
−15
−20


Strategy Returns V/S Benchmark "Quarterly Basis"
strategy
ibovespa
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
Date
2011Q1 2011Q2 2011Q3 2011Q4 2012Q1 2012Q2 2012Q3 2012Q4 2013Q1 2013Q2 2013Q3 2013Q4 2014Q1 2014Q2 2014Q3 2014Q4 2015Q1 2015Q2 2015Q3 2015Q4 2016Q1 2016Q2 2016Q3 2016Q4 2017Q1 2017Q2 2017Q3 2017Q4 2018Q1 2018Q2 2018Q3 2018Q4 2019Q1 2019Q2 2019Q3 2019Q4 2020Q1 2020Q2 2020Q3 2020Q4


Wealth-Index With Drawdowns and Monthly Returns of Strategy
Wealth
drawdowns
Monthly Returns
2.0
1.5
1.0
0.5
0.0
−0.5
2012
2013
2014
2015
2016
2017
2018
2019
2020
Date


strategy
ibovespa
0.2
0.1
0.0
−0.1
−0.2
−0.3
2012
2013
2014
2015
2016
2017
2018
2019
2020
Date