Análise Estatística no Mercado Financeiro - Prova Final - AEDI

August 30, 2022

Análise Estatistica no Mercado Financeiro

Mestrado Profissional em Computação Aplicada - PPCA - Universidade de Brasília

Disciplina: Análise Estatística de Dados e Informações

Orientador: João Gabriel de Moraes Souza

Aluno: Bruno Gomes Resende brunogomesresende@gmail.com

1 Introdução

O presente trabalho tem como objetivo realizar uma análise estatística sob uma carteira de ações do mercado financeiro brasileiro. Serão utilizadas nesse estudo as seguintes técnicas:

- Coleta de dados históricos do mercado financeiro a partir de fonte pública *yahoo finance*
- Montagem de portfólio e maximização de índice sharpe
- Análise descritiva dos dados
- Análise de variância e testes de Hipóteses
- Regressão linear em uma ação específica da carteira e o índice de benchmark
- Modelo de previsibilidade utilizando Machine Learning

2 Coleta de Dados

Configurações Iniciais de Ambiente

[1]:
```
#Instalação de Bibliotecas
#Remova os comentários para configuração do ambiente

#%%capture
#!pip install --upgrade ipykernel
#!pip install --upgrade pandas
#!pip install --upgrade pandas-datareader
#!pip install --upgrade matplotlib
#!pip install --upgrade numpy
#!pip install --upgrade plotly
#!pip install --upgrade seaborn
#!pip install --upgrade scipy
#!pip install --upgrade pywinpty
#!pip install --upgrade nbformat
```

```
#!pip install --upgrade scikit-learn
#!pip install --upgrade statsmodels
#!pip install --upgrade nbconvert
#!pip install -U kaleido
#!pip install -U pandas_ta
#!pip install -U tensorflow
#!pip install -U keras
#!pip install -U pmdarima
#!pip install -U pydot
#!pip install -U pyppeteer
#!pip install -U graphviz
```

[2]:
```
%%capture
#Importação de Bibliotecas
import pandas as pd
import pandas_ta as pta
import pandas_datareader.data as web
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.io as pio
import seaborn as sns
from scipy import optimize
from scipy import stats
import warnings
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
import statsmodels.api as sm
import pylab as py
import statsmodels.stats.api as sms
from sklearn import svm
import sklearn
import math
import sklearn.preprocessing
import datetime
import os
import tensorflow as tf
from tensorflow.python.framework import ops
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Dropout
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA
from pmdarima.arima import auto_arima
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
from sklearn.tree import export_graphviz
import pydot
import graphviz
```

[3]:
```
%%capture
#Exibição de Plotly ambiente Visual Studio
#pio.renderers.default = "notebook_connected+pdf"
#Exibição de Plotly PDF
pio.renderers.default = "notebook_connected+pdf"
#Omitindo warnings
warnings.filterwarnings('ignore')
#Estilo de gráfico
sns.set_style("darkgrid")
# PATH graphviz
os.environ["PATH"] += os.pathsep + 'C:/Users/bruno/anaconda3/Library/bin/
 ↪graphviz/'
```

2.1 Construindo uma Base de Dados Financeiros

As ações escolhidas arbitrariamente para compor a base de dados foram:

- BBDC4 - Bradesco Preferenciais
- PETR4 - Petrobras Preferenciais
- VALE3 - Vale Ordinárias
- GGBR4 - Gerdau Preferenciais
- CYRE3 - Cyrela Ordinárias
- GOLL4 - Gol Preferenciais
- KLBN4 - Klabin Preferenciais

O índice de referência escolhido como benchmark da carteira, será o ETF *BOVA11*, que espelha o índice *BOVESPA*.

O período de dados históricos coletados partirá de 01/01/2010 até o dia atual.

[4]:
```
# Definição dos papeis que irão compor a base de dados
acoes = ['BBDC4.SA', 'PETR4.SA', 'VALE3.SA','GGBR4.SA', 'CYRE3.SA', 'GOLL4.
 ↪SA','KLBN4.SA', 'BOVA11.SA']
# Coleta de dados no yahoo finance
acoes_df = pd.DataFrame()
for acao in acoes:
    acoes_df[acao] = web.DataReader(acao, data_source='yahoo',␣
 ↪start='2010-01-01')['Close']
#Transforma index Date em coluna Date para fins de plotagem gráfica dos dados.
acoes_df.reset_index(inplace=True)
# Visualização dos dados coletados
acoes_df
```

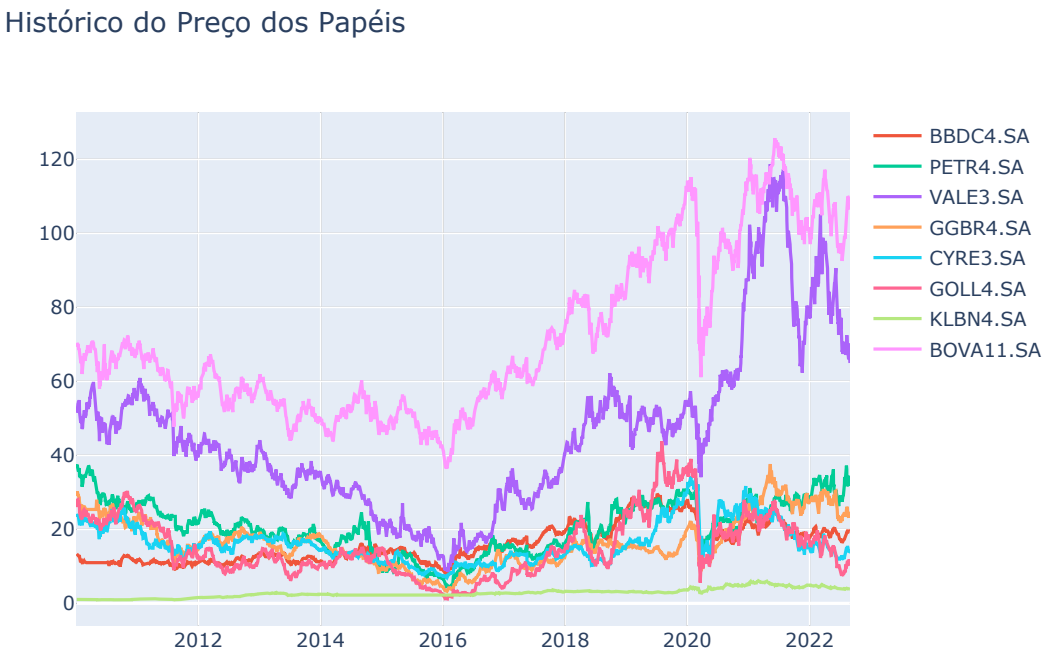
[4]:

	Date	BBDC4.SA	PETR4.SA	VALE3.SA	GGBR4.SA	CYRE3.SA	GOLL4.SA	KLBN4.SA	BOVA11.SA
0	2010-01-04	13.379374	37.320000	51.490002	29.600000	24.200001	26.299999	1.062	69.370003
1	2010-01-05	13.308395	37.000000	51.970001	30.100000	23.969999	26.080000	1.082	69.900002
2	2010-01-06	13.201928	37.500000	53.070000	30.299999	24.000000	26.379999	1.068	70.300003
3	2010-01-07	13.166439	37.150002	53.290001	29.620001	23.540001	26.660000	1.078	70.000000
4	2010-01-08	13.152244	36.950001	53.810001	29.540001	23.500000	27.950001	1.054	69.480003
…	…	…	…	…	…	…	…	…	…
3137	2022-08-24	19.370001	33.639999	67.949997	24.320000	14.600000	11.300000	4.090	109.180000
3138	2022-08-25	19.510000	33.279999	69.269997	24.670000	14.520000	11.920000	4.000	109.599998
3139	2022-08-26	19.440001	33.639999	68.230003	24.059999	14.250000	11.480000	3.930	108.360001
3140	2022-08-29	19.510000	34.480000	66.910004	23.959999	14.550000	11.090000	3.930	108.400002
3141	2022-08-30	19.450001	32.430000	64.970001	23.540001	14.450000	10.400000	3.850	106.400002

[3142 rows x 9 columns]

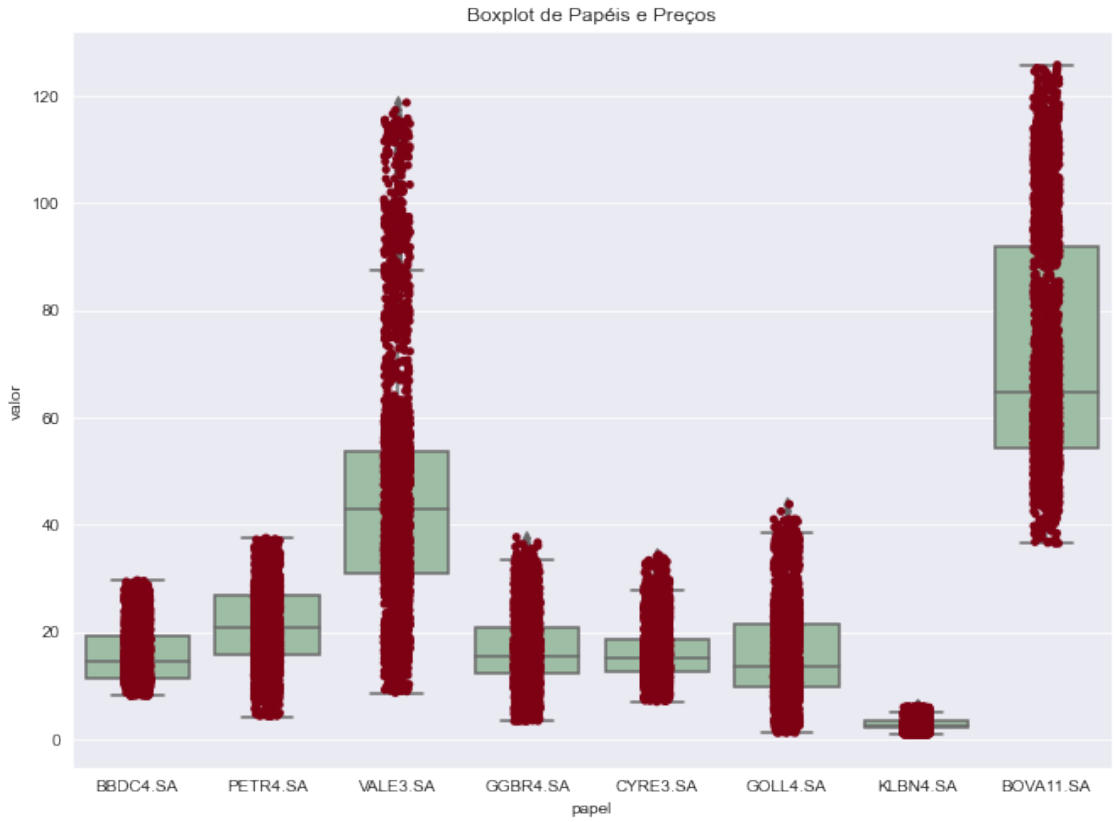
2.1.1 Visualização e Análise Descritiva dos Dados

[5]:
```
figura = px.line(title = 'Histórico do Preço dos Papéis')
for i in acoes_df.columns[1:]:
  figura.add_scatter(x = acoes_df["Date"] ,y = acoes_df[i], name = i)
figura.show()
```



Loading [MathJax]/extensions/MathMenu.js

[6]:
```
acoes_df_nodate = acoes_df.copy()
acoes_df_nodate.drop(labels = ['Date'], axis=1, inplace=True)

plt.figure(figsize =(11, 8))
df_melt = pd.melt(acoes_df_nodate.reset_index(), id_vars=['index'],␣
 ↪value_vars=['BBDC4.SA', 'PETR4.SA', 'VALE3.SA','GGBR4.SA', 'CYRE3.SA',␣
 ↪'GOLL4.SA','KLBN4.SA', 'BOVA11.SA'])
df_melt.columns = ['index', 'papel', 'valor']

ax = sns.boxplot(x='papel', y='valor', data=df_melt, color='#99c2a2')
ax = sns.stripplot(x="papel", y="valor", data=df_melt, color='#7d0013')
plt.title('Boxplot de Papéis e Preços')
plt.show()
```



2.2 Cálculo das Taxas de Retorno

O cálculo de retorno de uma ação é uma medida da variação contínua dos preços ao longo de um recorte temporal, dado por:

$$\mathbb{E}[R_i] = log\left(\frac{P_t}{P_{t-1}}\right)$$

[7]:
```
dataset = acoes_df.copy() # copiando o dataset para manipulações
dataset.drop(labels = ['Date'], axis=1, inplace=True) # removendo coluna Date␣
 ↪do dataset cópia

# Calculando as taxas de retorno diárias de cada papel.
# De forma a facilitar o cálculo entre tuplas do dataset, faz-se um shift,␣
 ↪deslocando os dados do dataset.
taxas_retorno = np.log(dataset / dataset.shift(1))

# Removendo primeira linha vazia
taxas_retorno = taxas_retorno.iloc[1:, :]
taxas_retorno
```

[7]:
```
      BBDC4.SA  PETR4.SA  VALE3.SA  GGBR4.SA  CYRE3.SA  GOLL4.SA  KLBN4.SA  \
1    -0.005319 -0.008611  0.009279  0.016751 -0.009550 -0.008400  0.018657
2    -0.008032  0.013423  0.020945  0.006623  0.001251  0.011437 -0.013023
3    -0.002692 -0.009377  0.004137 -0.022698 -0.019353  0.010558  0.009320
4    -0.001079 -0.005398  0.009711 -0.002705 -0.001701  0.047253 -0.022515
5     0.000809 -0.003253 -0.002978  0.000677  0.009740  0.002858  0.024369
…          …         …         …         …         …         …         …
3137 -0.005663  0.005963 -0.032719 -0.008190  0.017272  0.032377 -0.007308
3138  0.007202 -0.010759  0.019240  0.014289 -0.005495  0.053415 -0.022251
3139 -0.003594  0.010759 -0.015127 -0.025037 -0.018770 -0.037611 -0.017655
3140  0.003594  0.024664 -0.019536 -0.004165  0.020834 -0.034563  0.000000
3141 -0.003080 -0.061296 -0.029423 -0.017685 -0.006897 -0.064238 -0.020566

      BOVA11.SA
1      0.007611
2      0.005706
3     -0.004277
4     -0.007456
5      0.007313
…          …
3137   0.000733
3138   0.003839
3139  -0.011378
3140   0.000369
3141  -0.018623

[3141 rows x 8 columns]
```

2.3 Análise Descritiva das Taxas de Retorno

[8]:
```
taxas_retorno.describe()
```

[8]:
```
          BBDC4.SA     PETR4.SA     VALE3.SA     GGBR4.SA     CYRE3.SA  \
count  3141.000000  3141.000000  3141.000000  3141.000000  3141.000000
mean      0.000119    -0.000045     0.000074    -0.000073    -0.000164
std       0.021370     0.029537     0.026339     0.027678     0.027418
min      -0.154019    -0.352367    -0.281822    -0.197922    -0.283029
25%      -0.010986    -0.014317    -0.013989    -0.015259    -0.014933
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       0.011299     0.014222     0.013459     0.014771     0.015006
max       0.155866     0.200671     0.193574     0.160867     0.165985

          GOLL4.SA     KLBN4.SA    BOVA11.SA
count  3141.000000  3141.000000  3141.000000
mean     -0.000295     0.000410     0.000136
std       0.042572     0.018191     0.015854
min      -0.450890    -0.143310    -0.157528
```

```
25%      -0.022318    -0.008016    -0.008010
50%      -0.001554     0.000000     0.000172
75%       0.019710     0.007921     0.008858
max       0.407641     0.121743     0.125708
```

2.3.1 Cálculo de Médias

Em estatística, média é definida como o valor que demonstra a concentração dos dados de uma distribuição, como o ponto de equilíbrio das frequências em um histograma.

[9]:
```
taxas_retorno.mean()*100
```

[9]:
```
BBDC4.SA     0.011911
PETR4.SA    -0.004471
VALE3.SA     0.007403
GGBR4.SA    -0.007293
CYRE3.SA    -0.016417
GOLL4.SA    -0.029537
KLBN4.SA     0.041003
BOVA11.SA    0.013618
dtype: float64
```

2.3.2 Cálculo de Variâncias

Na teoria da probabilidade e na estatística, a variância de uma variável aleatória ou processo estocástico é uma medida da sua dispersão estatística, indicando "o quão longe" em geral os seus valores se encontram do valor esperado.

[10]:
```
taxas_retorno.var()
```

[10]:
```
BBDC4.SA     0.000457
PETR4.SA     0.000872
VALE3.SA     0.000694
GGBR4.SA     0.000766
CYRE3.SA     0.000752
GOLL4.SA     0.001812
KLBN4.SA     0.000331
BOVA11.SA    0.000251
dtype: float64
```

2.3.3 Cálculo dos Desvios Padrão

Em probabilidade, o desvio padrão ou desvio padrão populacional é uma medida de dispersão em torno da média populacional de uma variável aleatória.

[11]:
```
taxas_retorno.std()*100
```

[11]:
```
BBDC4.SA     2.137018
PETR4.SA     2.953746
```

```
VALE3.SA     2.633853
GGBR4.SA     2.767797
CYRE3.SA     2.741762
GOLL4.SA     4.257172
KLBN4.SA     1.819051
BOVA11.SA    1.585445
dtype: float64
```

2.3.4 Cálculo das Covariâncias

Em teoria da probabilidade e na estatística, a covariância, ou variância conjunta, é uma medida do grau de interdependência numérica entre duas variáveis aleatórias.

[12]:
```
taxas_retorno.cov()
```

[12]:
```
           BBDC4.SA  PETR4.SA  VALE3.SA  GGBR4.SA  CYRE3.SA  GOLL4.SA  \
BBDC4.SA   0.000457  0.000356  0.000210  0.000274  0.000318  0.000434
PETR4.SA   0.000356  0.000872  0.000353  0.000392  0.000362  0.000535
VALE3.SA   0.000210  0.000353  0.000694  0.000442  0.000204  0.000319
GGBR4.SA   0.000274  0.000392  0.000442  0.000766  0.000305  0.000439
CYRE3.SA   0.000318  0.000362  0.000204  0.000305  0.000752  0.000575
GOLL4.SA   0.000434  0.000535  0.000319  0.000439  0.000575  0.001812
KLBN4.SA   0.000083  0.000123  0.000114  0.000147  0.000119  0.000154
BOVA11.SA  0.000254  0.000337  0.000244  0.000271  0.000286  0.000388

           KLBN4.SA  BOVA11.SA
BBDC4.SA   0.000083   0.000254
PETR4.SA   0.000123   0.000337
VALE3.SA   0.000114   0.000244
GGBR4.SA   0.000147   0.000271
CYRE3.SA   0.000119   0.000286
GOLL4.SA   0.000154   0.000388
KLBN4.SA   0.000331   0.000106
BOVA11.SA  0.000106   0.000251
```

2.3.5 Cálculo das Correlações

Em probabilidade e estatística, correlação, dependência ou associação é qualquer relação estatística entre duas variáveis.

[13]:
```
taxas_retorno.corr()
```

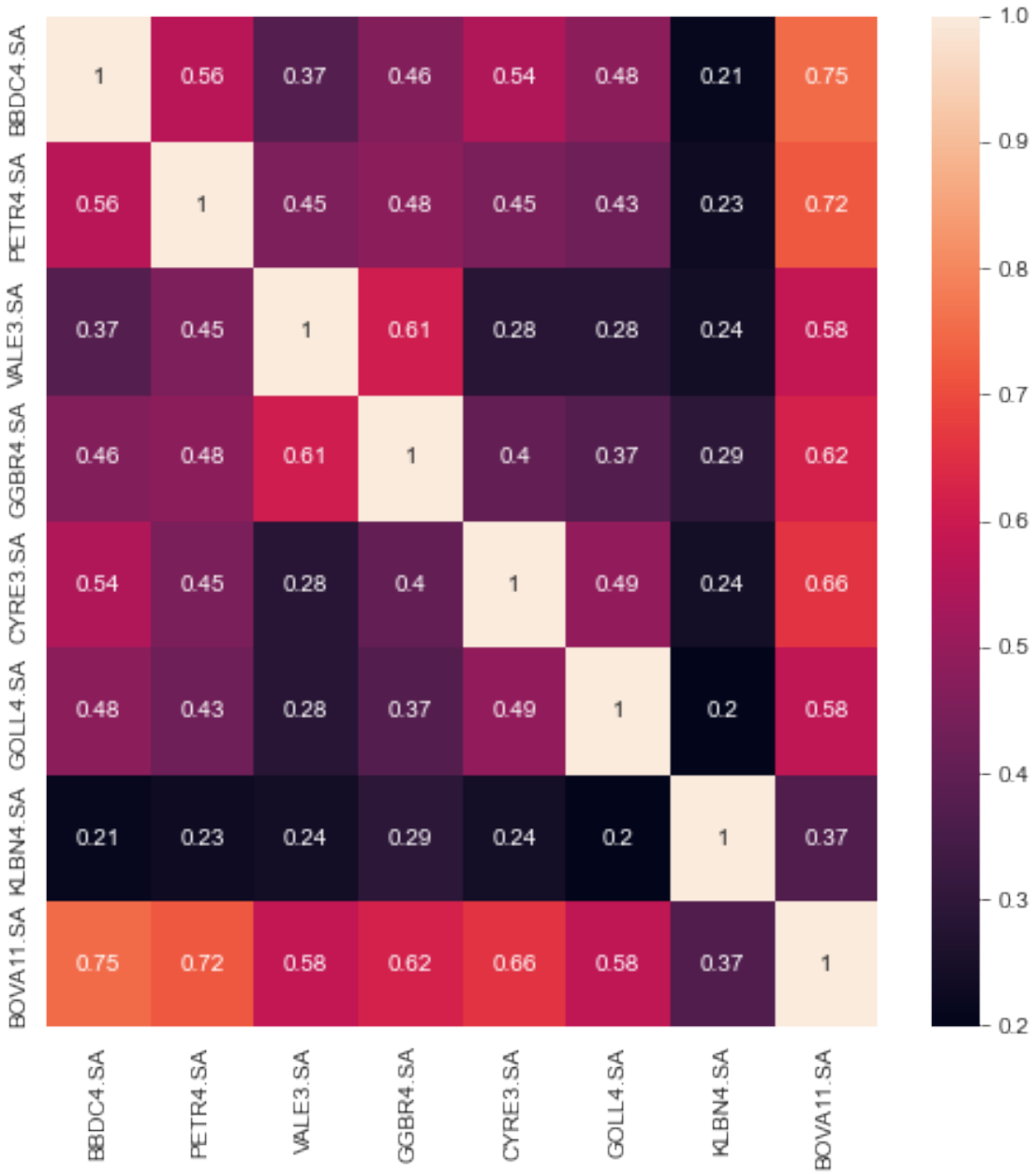
[13]:
```
           BBDC4.SA  PETR4.SA  VALE3.SA  GGBR4.SA  CYRE3.SA  GOLL4.SA  \
BBDC4.SA   1.000000  0.564511  0.373955  0.463683  0.542833  0.477291
PETR4.SA   0.564511  1.000000  0.453256  0.479129  0.447279  0.425426
VALE3.SA   0.373955  0.453256  1.000000  0.606024  0.282941  0.284754
GGBR4.SA   0.463683  0.479129  0.606024  1.000000  0.401857  0.372326
CYRE3.SA   0.542833  0.447279  0.282941  0.401857  1.000000  0.492624
```

```
GOLL4.SA   0.477291  0.425426  0.284754  0.372326  0.492624  1.000000
KLBN4.SA   0.213191  0.228211  0.237766  0.292673  0.239447  0.198260
BOVA11.SA  0.750565  0.720655  0.583294  0.618367  0.658028  0.575447
```

```
           KLBN4.SA  BOVA11.SA
BBDC4.SA   0.213191   0.750565
PETR4.SA   0.228211   0.720655
VALE3.SA   0.237766   0.583294
GGBR4.SA   0.292673   0.618367
CYRE3.SA   0.239447   0.658028
GOLL4.SA   0.198260   0.575447
KLBN4.SA   1.000000   0.366684
BOVA11.SA  0.366684   1.000000
```

2.3.6 Mapa de Correlações

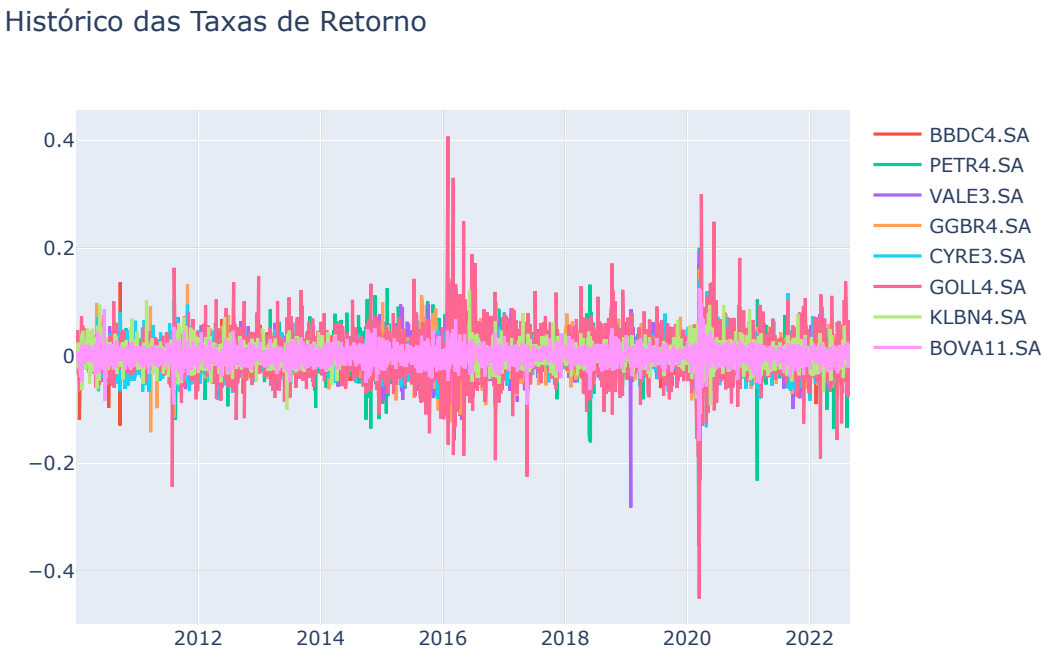
[14]:
```
plt.figure(figsize=(8,8))
sns.heatmap(taxas_retorno.corr(), annot=True);
```



2.3.7 Gráfico do Histórico das Taxas de Retorno

[15]:
```
# recuperando coluna Date do dataset original
dataset_date = acoes_df.copy()
date = dataset_date.filter(["Date"])

# montando novo dataset com taxas de retorno e data
taxas_retorno_date = pd.concat([date, taxas_retorno], axis=1)
```

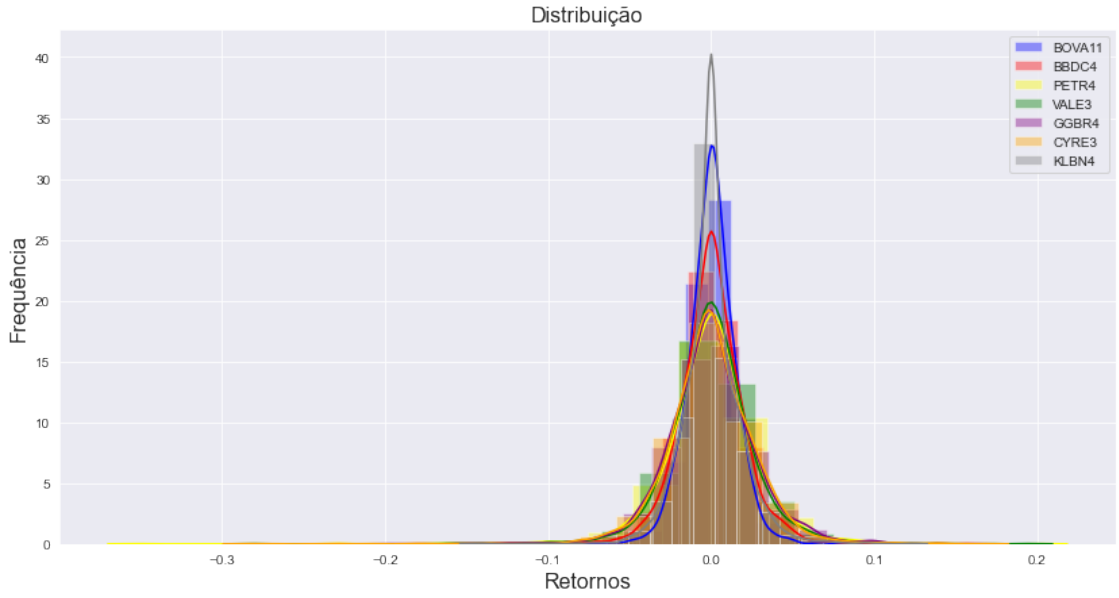
```
# montando do gráfico
figura = px.line(title = 'Histórico das Taxas de Retorno')
for i in taxas_retorno_date.columns[1:]:
  figura.add_scatter(x = taxas_retorno_date["Date"] ,y = taxas_retorno_date[i],␣
 ↪name = i)
figura.show()
```



2.3.8 Histograma da Distribuição das Taxas de Retorno

[16]:
```
fig, ax = plt.subplots(figsize=(14,7))
ax = sns.distplot(taxas_retorno['BOVA11.SA'], bins=20, label='BOVA11', color =␣
 ↪'blue')
ax = sns.distplot(taxas_retorno['BBDC4.SA'], bins=20, label='BBDC4', color =␣
 ↪'red')
ax = sns.distplot(taxas_retorno['PETR4.SA'], bins=20, label='PETR4', color =␣
 ↪'yellow')
ax = sns.distplot(taxas_retorno['VALE3.SA'], bins=20, label='VALE3', color =␣
 ↪'green')
ax = sns.distplot(taxas_retorno['GGBR4.SA'], bins=20, label='GGBR4', color =␣
 ↪'purple')
```

```
ax = sns.distplot(taxas_retorno['CYRE3.SA'], bins=20, label='CYRE3', color =␣
 ↪'orange')
ax = sns.distplot(taxas_retorno['KLBN4.SA'], bins=20, label='KLBN4', color =␣
 ↪'gray')
ax.set_xlabel("Retornos",fontsize=16)
ax.set_ylabel("Frequência",fontsize=16)
plt.title('Distribuição',fontsize=16)
plt.legend();
```



3 Montagem de Carteiras de Ativos

Para desenvolvimento da análise proposta, iremos alocar os ativos coletados em três diferentes carteiras: - Carteira **IGUAIS**, cuja alocação dos ativos será arbitrária com pesos iguais - Carteira **ALEATORIA**, com alocação de ativos com pesos calculados aleatoriamente - Carteira **EFICIENTE**, onde utilizaremos o Modelo de Markovitz para maximização da relação Risco x Retorno na alocação dos ativos

3.1 Função de Alocação

De forma a facilitar a alocação de ativos em carteiras nos próximos passos, iremos definir uma função de alocação de ativos.

[17]:
```
#função para alocação de ativos
def alocacao_ativos(dataset, dinheiro_total, seed = 0, melhores_pesos = []):
  dataset = dataset.copy()

  if seed != 0:
```

```
    np.random.seed(seed)

  if len(melhores_pesos) > 0:
    pesos = melhores_pesos
  else:
    pesos = np.random.random(len(dataset.columns) - 1)
    pesos = pesos / pesos.sum()

  colunas = dataset.columns[1:]
  for i in colunas:
    dataset[i] = (dataset[i] / dataset[i][0])

  for i, acao in enumerate(dataset.columns[1:]):
    dataset[acao] = dataset[acao] * pesos[i] * dinheiro_total

  dataset['SOMA VALOR'] = dataset.sum(axis = 1)

  datas = dataset['Date']

  dataset.drop(labels = ['Date'], axis = 1, inplace = True)
  dataset['TR CARTEIRA'] = 0.0

  for i in range(1, len(dataset)):
    dataset['TR CARTEIRA'][i] = np.log(dataset['SOMA VALOR'][i] / dataset['SOMA␣
↪VALOR'][i - 1]) * 100

  acoes_pesos = pd.DataFrame(data = {'Ações': colunas, 'Pesos': pesos})

  return dataset, datas, acoes_pesos, dataset.loc[len(dataset) - 1]['SOMA␣
↪VALOR']
```

[18]:
```
# Matriz de Sumarização das Carteiras
arr_carteiras = ["IGUAIS","ALEATORIA","EFICIENTE","BOVA11.SA"]
arr_valor_inicial = [10000,10000,10000,10000]
arr_tr_media = []
arr_var = []
arr_dp = []
arr_cov = []
arr_corr = []

sumario_carteiras = pd.DataFrame()
sumario_carteiras['CARTEIRA'] = arr_carteiras
sumario_carteiras['VALOR INICIAL'] = arr_valor_inicial

#sumario_carteiras
```

3.2 Carteira IGUAIS

Esta carteira possui igual alocação entre os 7 ativos coletados.

[19]:
```
# Retirada do ETF BOVA11 para criação da carteira
acoes_port = acoes_df.copy()
acoes_port.drop(labels = ['BOVA11.SA'], axis=1, inplace=True)

# Alocação da carteira
sim_carteira_iguais, sim_datas_iguais, sim_pesos_carteira_iguais,␣
 ↪sim_soma_valor_carteira_iguais = alocacao_ativos(acoes_port, 10000, 10, [1/
 ↪7,1/7,1/7,1/7,1/7,1/7,1/7])

# Inclusão da Taxa de Retorno de BOVA11
sim_carteira_iguais['TR BOVA11.SA'] = taxas_retorno_date['BOVA11.SA']*100

# Inclusão de Date
sim_carteira_iguais['Date'] = sim_datas_iguais

# Remoção da primeira linha vazia
sim_carteira_iguais = sim_carteira_iguais.iloc[1:,:]

sim_carteira_iguais
```

[19]:

	BBDC4.SA	PETR4.SA	VALE3.SA	GGBR4.SA	CYRE3.SA	\
1	1420.992794	1416.322167	1441.888842	1452.702702	1414.994012	
2	1409.624843	1435.461656	1472.407901	1462.355157	1416.765008	
3	1405.835526	1422.064072	1478.511755	1429.536702	1389.610400	
4	1404.319819	1414.408248	1492.938979	1425.675701	1387.249071	
5	1405.456625	1409.814811	1488.499841	1426.640882	1400.826375	
…	…	…	…	…	…	
3137	2068.215650	1287.704780	1885.248030	1173.745144	861.865403	
3138	2083.163968	1273.924324	1921.870943	1190.637054	857.142857	
3139	2075.689809	1287.704780	1893.016706	1161.196870	841.204224	
3140	2083.163968	1319.859127	1856.393793	1156.370597	858.913798	
3141	2076.757575	1241.387252	1802.569129	1136.100416	853.010588	

	GOLL4.SA	KLBN4.SA	SOMA VALOR	TR CARTEIRA	TR BOVA11.SA	\
1	1416.621438	1455.474819	10018.996775	0.189788	0.761113	
2	1432.916889	1436.642349	10066.173803	0.469771	0.570617	
3	1448.126052	1450.094044	10023.778551	-0.422055	-0.427660	
4	1518.196718	1417.810040	10060.598577	0.366654	-0.745625	
5	1522.542178	1452.784512	10106.565225	0.455857	0.731335	
…	…	…	…	…	…	
3137	613.796878	5501.748740	13392.324624	-0.614037	0.073302	
3138	647.474222	5380.683163	13354.896531	-0.279865	0.383946	
3139	623.574138	5286.521298	13168.907825	-1.402452	-1.137834	
3140	602.390031	5286.521298	13163.612612	-0.040218	0.036908	

```
3141   564.910370  5178.907416  12853.642746    -2.382915     -1.862251

           Date
1    2010-01-05
2    2010-01-06
3    2010-01-07
4    2010-01-08
5    2010-01-11
…           …
3137 2022-08-24
3138 2022-08-25
3139 2022-08-26
3140 2022-08-29
3141 2022-08-30

[3141 rows x 11 columns]
```

3.2.1 Análise Descritiva

Pesos de Alocação

[20]:
```
sim_pesos_carteira_iguais
```

[20]:
```
       Ações     Pesos
0  BBDC4.SA  0.142857
1  PETR4.SA  0.142857
2  VALE3.SA  0.142857
3  GGBR4.SA  0.142857
4  CYRE3.SA  0.142857
5  GOLL4.SA  0.142857
6  KLBN4.SA  0.142857
```

Sumário

[21]:
```
sim_carteira_iguais.filter(["TR CARTEIRA", "TR BOVA11.SA"]).describe()
```

[21]:
```
        TR CARTEIRA  TR BOVA11.SA
count  3141.000000   3141.000000
mean      0.007992      0.013618
std       1.555935      1.585445
min     -18.601521    -15.752788
25%      -0.842554     -0.800978
50%       0.008038      0.017151
75%       0.883418      0.885836
max      10.085339     12.570843
```

Cálculo de Médias

[22]:
```

```

```
arr_tr_media.append(sim_carteira_iguais.filter(["TR CARTEIRA"]).values.
  ↪mean()*100)
sim_carteira_iguais.filter(["TR CARTEIRA", "TR BOVA11.SA"]).mean()*100
```

[22]:
```
TR CARTEIRA     0.799243
TR BOVA11.SA    1.361831
dtype: float64
```

Cálculo de Variâncias

[23]:
```
arr_var.append(sim_carteira_iguais.filter(["TR CARTEIRA"]).values.var())
sim_carteira_iguais.filter(["TR CARTEIRA", "TR BOVA11.SA"]).var()
```

[23]:
```
TR CARTEIRA     2.420934
TR BOVA11.SA    2.513635
dtype: float64
```

Cálculo dos Desvios Padrão

[24]:
```
arr_dp.append(sim_carteira_iguais.filter(["TR CARTEIRA"]).values.std())
sim_carteira_iguais.filter(["TR CARTEIRA", "TR BOVA11.SA"]).std()*100
```

[24]:
```
TR CARTEIRA     155.593519
TR BOVA11.SA    158.544463
dtype: float64
```

Cálculo das Covariâncias

[25]:
```
arr_cov.append(sim_carteira_iguais.filter(["TR CARTEIRA", "TR BOVA11.SA"]).
  ↪cov().loc['TR CARTEIRA','TR BOVA11.SA'])
sim_carteira_iguais.filter(["TR CARTEIRA", "TR BOVA11.SA"]).cov()
```

[25]:

	TR CARTEIRA	TR BOVA11.SA
TR CARTEIRA	2.420934	2.074274
TR BOVA11.SA	2.074274	2.513635

Cálculo das Correlações

[26]:
```
arr_corr.append(sim_carteira_iguais.filter(["TR CARTEIRA", "TR BOVA11.SA"]).
  ↪corr().loc['TR CARTEIRA','TR BOVA11.SA'])
sim_carteira_iguais.filter(["TR CARTEIRA", "TR BOVA11.SA"]).corr()
```

[26]:

	TR CARTEIRA	TR BOVA11.SA
TR CARTEIRA	1.00000	0.84086
TR BOVA11.SA	0.84086	1.00000

3.2.2 Análise Gráfica

Mapa de Correlações

[27]:
```
```

```
plt.figure(figsize=(6,6))
sns.heatmap(sim_carteira_iguais.filter(["TR CARTEIRA", "TR BOVA11.SA"]).corr(),
 ↪annot=True);
```

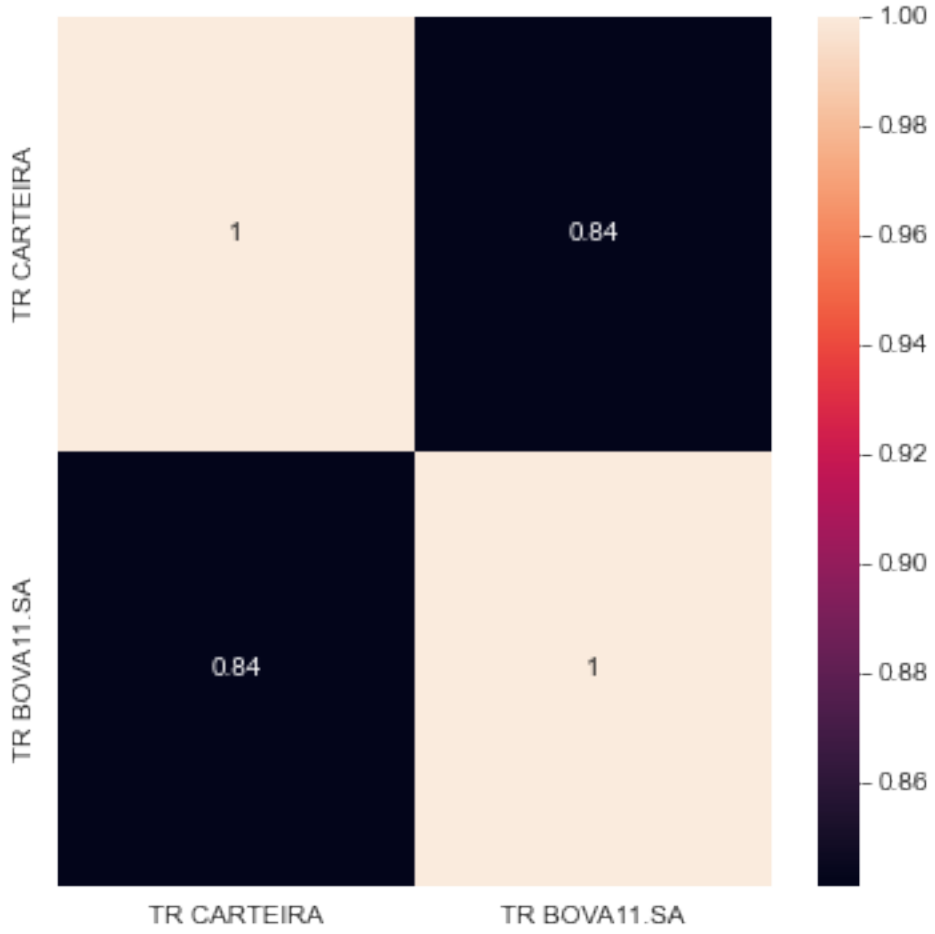
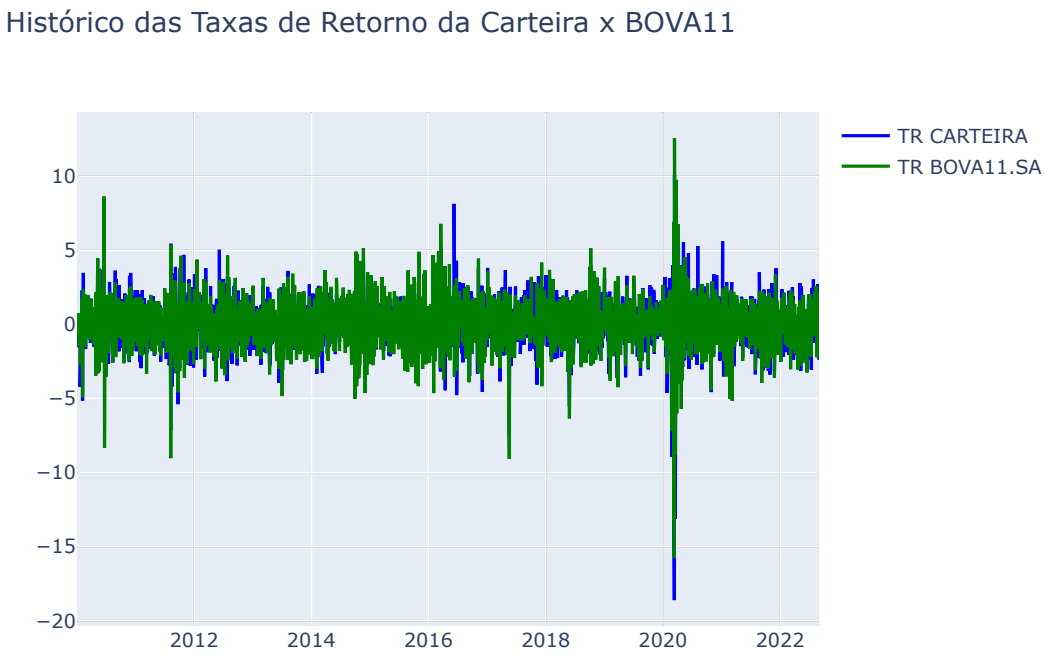


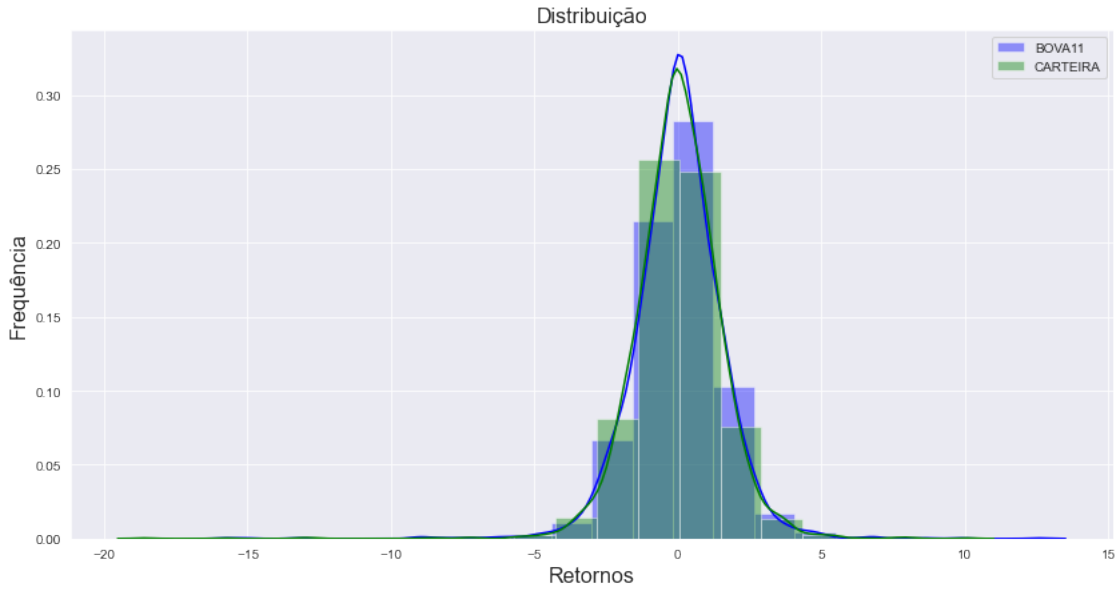
Gráfico do Histórico das Taxas de Retorno

[28]:
```
figura = px.line(title = 'Histórico das Taxas de Retorno da Carteira x BOVA11')
figura.add_scatter(x = sim_carteira_iguais["Date"] ,y = sim_carteira_iguais["TR
 ↪CARTEIRA"], name = "TR CARTEIRA", line_color = 'blue')
figura.add_scatter(x = sim_carteira_iguais["Date"] ,y = sim_carteira_iguais["TR
 ↪BOVA11.SA"], name = "TR BOVA11.SA", line_color = 'green')
figura.show()
```



Histograma da Distribuição das Taxas de Retorno

[29]:
```
fig, ax = plt.subplots(figsize=(14,7))
ax = sns.distplot(sim_carteira_iguais['TR BOVA11.SA'], bins=20, label='BOVA11',␣
 ↪color = 'blue')
ax = sns.distplot(sim_carteira_iguais['TR CARTEIRA'], bins=20,␣
 ↪label='CARTEIRA', color = 'green')
ax.set_xlabel("Retornos",fontsize=16)
ax.set_ylabel("Frequência",fontsize=16)
plt.title('Distribuição',fontsize=16)
plt.legend();
```



3.3 Carteira ALEATORIA

Esta carteira possui percentual aleteatório de alocação entre os 7 ativos coletados.

[30]:
```
# Alocação da carteira
sim_carteira_aleat, sim_datas_aleat, sim_pesos_carteira_aleat,␣
 ↪sim_soma_valor_carteira_aleat = alocacao_ativos(acoes_port, 10000, 10)

# Inclusão da Taxa de Retorno de BOVA11
sim_carteira_aleat['TR BOVA11.SA'] = taxas_retorno_date['BOVA11.SA']*100

# Inclusão de Date
sim_carteira_aleat['Date'] = sim_datas_aleat

# Remoção da primeira linha vazia
sim_carteira_aleat = sim_carteira_aleat.iloc[1:,:]

sim_carteira_aleat
```

[30]:

	BBDC4.SA	PETR4.SA	VALE3.SA	GGBR4.SA	CYRE3.SA	\
1	2478.216112	66.455862	2065.819422	2459.558619	1594.917486	
2	2458.390367	67.353914	2109.544613	2475.901108	1596.913673	
3	2451.781786	66.725281	2118.289712	2420.336460	1566.306222	
4	2449.138389	66.366058	2138.959849	2413.799433	1563.644638	
5	2451.120981	66.150528	2132.599818	2415.433573	1578.948364	
…	…	…	…	…	…	
3137	3606.974904	60.420950	2701.027903	1987.257944	971.455843	
3138	3633.044819	59.774352	2753.498193	2015.857494	966.132803	

```
3139  3620.009862  60.420950  2712.158222  1966.012569   948.167494
3140  3633.044819  61.929678  2659.687932  1957.841247   968.128927
3141  3621.872049  58.247666  2582.572392  1923.521974   961.475095

        GOLL4.SA     KLBN4.SA    SOMA VALOR  TR CARTEIRA  TR BOVA11.SA  \
1     720.038937   651.808618  10036.815056     0.367475      0.761113
2     728.321572   643.374830  10079.800078     0.427359      0.570617
3     736.052071   649.398933  10008.890464    -0.705968     -0.427660
4     771.667519   634.941114  10038.517000     0.295565     -0.745625
5     773.876225   650.603812  10068.733301     0.300552      0.731335
…          …            …           …           …            …
3137  311.980067  2463.860725  12102.978336    -0.968615      0.073302
3138  329.097554  2409.643651  12167.048865     0.527982      0.383946
3139  316.949643  2367.474927  11991.193666    -1.455887     -1.137834
3140  306.182206  2367.474927  11954.289737    -0.308233      0.036908
3141  287.132082  2319.281956  11754.103214    -1.688780     -1.862251

           Date
1    2010-01-05
2    2010-01-06
3    2010-01-07
4    2010-01-08
5    2010-01-11
…         …
3137 2022-08-24
3138 2022-08-25
3139 2022-08-26
3140 2022-08-29
3141 2022-08-30

[3141 rows x 11 columns]
```

3.3.1 Análise Descritiva

Pesos de Alocação

[31]:
```
sim_pesos_carteira_aleat
```

[31]:
```
      Ações     Pesos
0  BBDC4.SA  0.249143
1  PETR4.SA  0.006703
2  VALE3.SA  0.204674
3  GGBR4.SA  0.241870
4  CYRE3.SA  0.161022
5  GOLL4.SA  0.072611
6  KLBN4.SA  0.063976
```

Sumário

[32]:
```
sim_carteira_aleat.filter(["TR CARTEIRA", "TR BOVA11.SA"]).describe()
```

[32]:

	TR CARTEIRA	TR BOVA11.SA
count	3141.000000	3141.000000
mean	0.005145	0.013618
std	1.669498	1.585445
min	-17.887631	-15.752788
25%	-0.892568	-0.800978
50%	0.044064	0.017151
75%	0.962336	0.885836
max	13.125691	12.570843

Cálculo das Médias

[33]:
```
arr_tr_media.append(sim_carteira_aleat.filter(["TR CARTEIRA"]).values.
↪mean()*100)
sim_carteira_aleat.filter(["TR CARTEIRA", "TR BOVA11.SA"]).mean()*100
```

[33]:
```
TR CARTEIRA     0.514541
TR BOVA11.SA    1.361831
dtype: float64
```

Cálculo de Variâncias

[34]:
```
arr_var.append(sim_carteira_aleat.filter(["TR CARTEIRA"]).values.var())
sim_carteira_aleat.filter(["TR CARTEIRA", "TR BOVA11.SA"]).var()
```

[34]:
```
TR CARTEIRA     2.787225
TR BOVA11.SA    2.513635
dtype: float64
```

Cálculo de Desvios Padrão

[35]:
```
arr_dp.append(sim_carteira_aleat.filter(["TR CARTEIRA"]).values.std())
sim_carteira_aleat.filter(["TR CARTEIRA", "TR BOVA11.SA"]).std()*100
```

[35]:
```
TR CARTEIRA     166.949829
TR BOVA11.SA    158.544463
dtype: float64
```

Cálculo de Covariâncias

[36]:
```
arr_cov.append(sim_carteira_aleat.filter(["TR CARTEIRA", "TR BOVA11.SA"]).cov().
↪loc['TR CARTEIRA','TR BOVA11.SA'])
sim_carteira_aleat.filter(["TR CARTEIRA", "TR BOVA11.SA"]).cov()
```

[36]:

	TR CARTEIRA	TR BOVA11.SA
TR CARTEIRA	2.787225	2.319145
TR BOVA11.SA	2.319145	2.513635

Cálculo de Correlações

[37]:
```
arr_corr.append(sim_carteira_aleat.filter(["TR CARTEIRA", "TR BOVA11.SA"]).
  ↪corr().loc['TR CARTEIRA','TR BOVA11.SA'])
sim_carteira_aleat.filter(["TR CARTEIRA", "TR BOVA11.SA"]).corr()
```

[37]:

	TR CARTEIRA	TR BOVA11.SA
TR CARTEIRA	1.000000	0.876175
TR BOVA11.SA	0.876175	1.000000

3.4 Análise Gráfica

Mapa de Correlações

[38]:
```
plt.figure(figsize=(6,6))
sns.heatmap(sim_carteira_aleat.filter(["TR CARTEIRA", "TR BOVA11.SA"]).corr(),␣
  ↪annot=True);
```

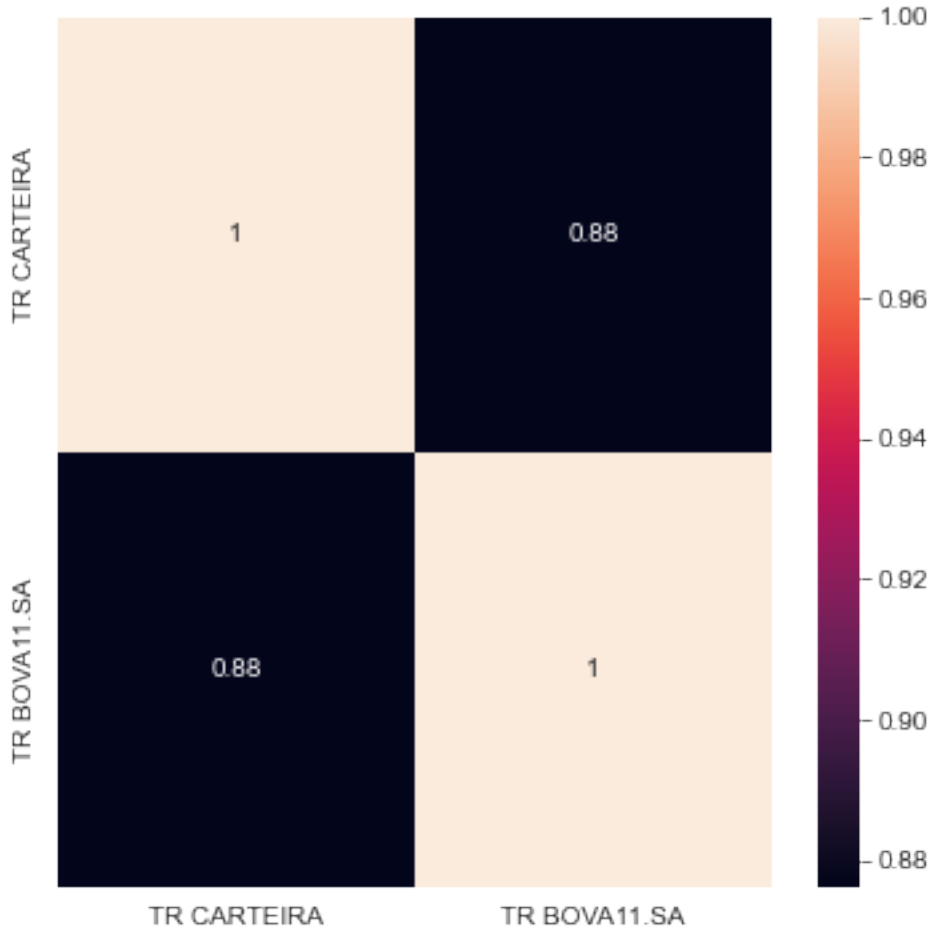
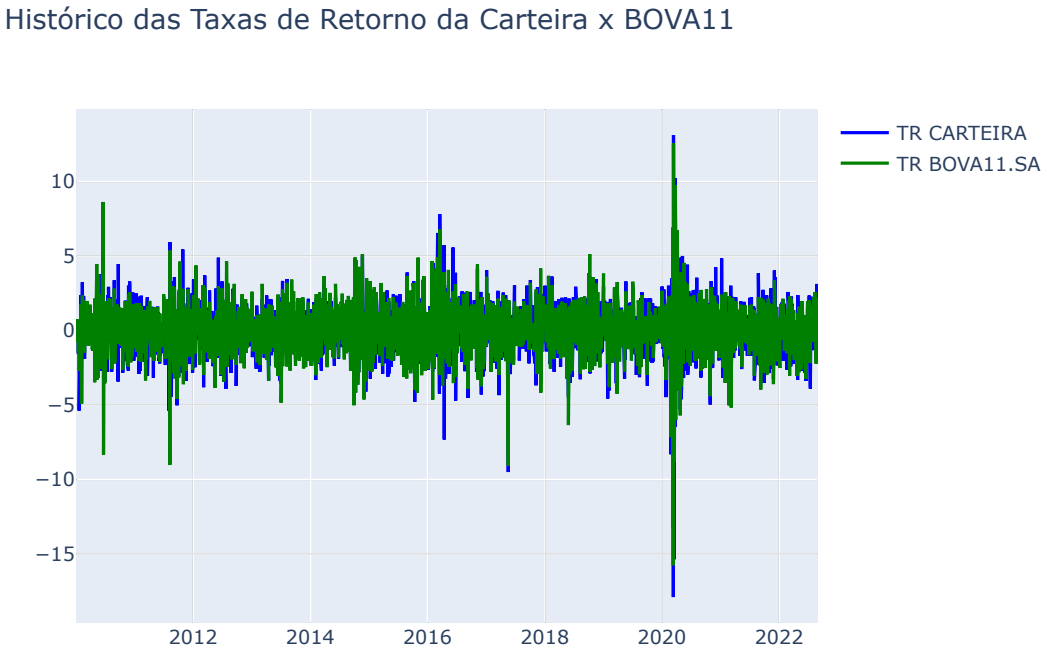


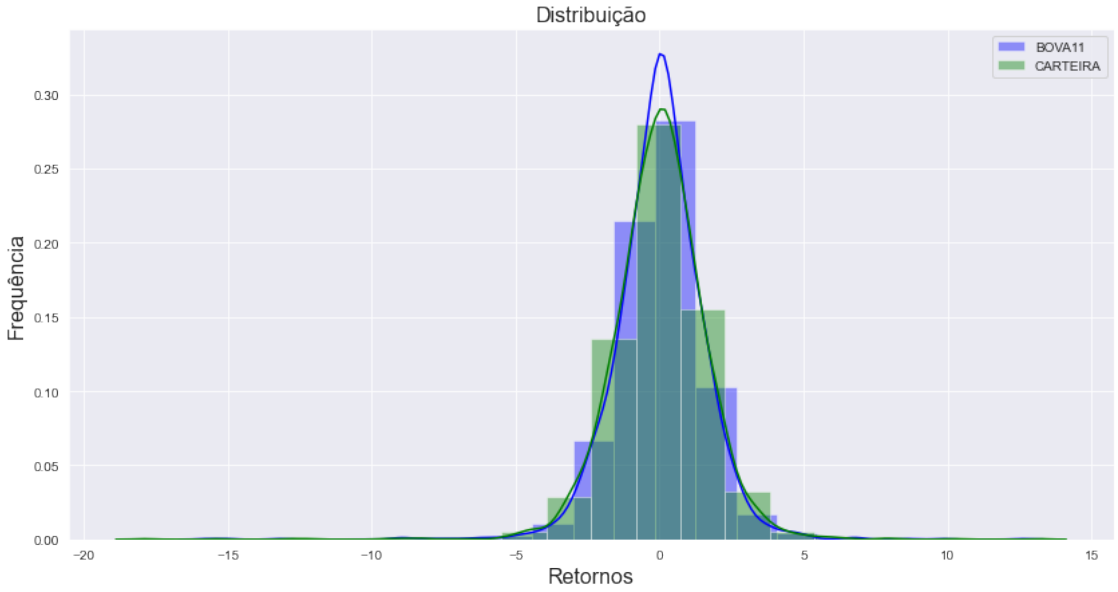
Gráfico do Histórico das Taxas de Retorno

[39]:
```
figura = px.line(title = 'Histórico das Taxas de Retorno da Carteira x BOVA11')
figura.add_scatter(x = sim_carteira_aleat["Date"] ,y = sim_carteira_aleat["TR␣
 ↪CARTEIRA"], name = "TR CARTEIRA", line_color = 'blue')
figura.add_scatter(x = sim_carteira_aleat["Date"] ,y = sim_carteira_aleat["TR␣
 ↪BOVA11.SA"], name = "TR BOVA11.SA", line_color = 'green')
figura.show()
```



Histograma da Distribuição das Taxas de Retorno

[40]:
```
fig, ax = plt.subplots(figsize=(14,7))
ax = sns.distplot(sim_carteira_aleat['TR BOVA11.SA'], bins=20, label='BOVA11',␣
 ↪color = 'blue')
ax = sns.distplot(sim_carteira_aleat['TR CARTEIRA'], bins=20, label='CARTEIRA',␣
 ↪color = 'green')
ax.set_xlabel("Retornos",fontsize=16)
ax.set_ylabel("Frequência",fontsize=16)
plt.title('Distribuição',fontsize=16)
plt.legend();
```



3.5 Carteira EFICIENTE

Neste ponto iremos trabalhar com o Modelo de Markovitz, visando a maximização do Sharpe Ratio, ou seja, uma relação ótima entre Risco e Retorno. Desta forma iremos gerar 1000 carteiras com alocação aleatória e utilizaremos aquela com maior indice Sharpe como carteira EFICIENTE.

3.5.1 Fronteira Eficiente

[41]:
```
log_ret = acoes_port.copy()
log_ret.drop(labels = ["Date"], axis = 1, inplace = True)
log_ret = np.log(log_ret/log_ret.shift(1))

np.random.seed(0)
num_ports = 1000
all_weights = np.zeros((num_ports, len(acoes_port.columns[1:])))
ret_arr = np.zeros(num_ports)
vol_arr = np.zeros(num_ports)
sharpe_arr = np.zeros(num_ports)

for x in range(num_ports):
    # Weights
    weights = np.array(np.random.random(7))
    weights = weights/np.sum(weights)

    # Save weights
    all_weights[x,:] = weights

    # Expected return
```

```
    ret_arr[x] = np.sum((log_ret.mean() * weights))

    # Expected volatility
    vol_arr[x] = np.sqrt(np.dot(weights.T, np.dot(log_ret.cov(), weights)))

    # Sharpe Ratio
    sharpe_arr[x] = ret_arr[x]/vol_arr[x]
```

Sharpe Ratio

[42]:
```
print("Max Sharpe Ratio : {}". format(sharpe_arr.max()))
print("Local do Max SR  : {}". format(sharpe_arr.argmax()))

# salvando os dados do Max Sharpe Ratio
max_sr_ret = ret_arr[sharpe_arr.argmax()]
max_sr_vol = vol_arr[sharpe_arr.argmax()]
print("Retorno          : {}".format(max_sr_ret))
print("Volatilidade     : {}".format(max_sr_vol))

max_sr_weights = all_weights[sharpe_arr.argmax(),:]
print("Pesos            : {}".format(max_sr_weights))
```

```
Max Sharpe Ratio : 0.010426570091626043
Local do Max SR  : 105
Retorno          : 0.00016450456327700584
Volatilidade     : 0.015777438009947818
Pesos            : [0.24538701 0.09982648 0.18874645 0.04225013 0.05753507
0.01684758
 0.34940727]
```

Visualização Gráfica das Carteiras Nós podemos ver no gráfico o conjunto de portfólios simulados, onde o peso w_i de cada ativo foi simulado, criamos um conjunto de $n = 1000$ carteiras e escolhemos no ponto preto a que tem maior **Sharpe Ratio**, que é a razão retorno sobre a volatilidade. Esse dado nos dá uma noção do portfólio ponderado pelo risco.

[43]:
```
# anotações de referência no gráfico
fig, ax = plt.subplots(figsize=(12,8))
ax.text(max_sr_vol*1.01,max_sr_ret*1.01,' SR = {}'.format(sharpe_arr.max().
 ↪round(6))+' Ret = {}'.format(max_sr_ret.round(6))+ ' Vol = {}'.
 ↪format(max_sr_vol.round(6)))

plt.scatter(vol_arr, ret_arr, c=sharpe_arr, cmap='viridis')
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Volatilidade')
plt.ylabel('Retorno')
plt.scatter(max_sr_vol, max_sr_ret,c='black', s=200) # black dot
plt.show()
```

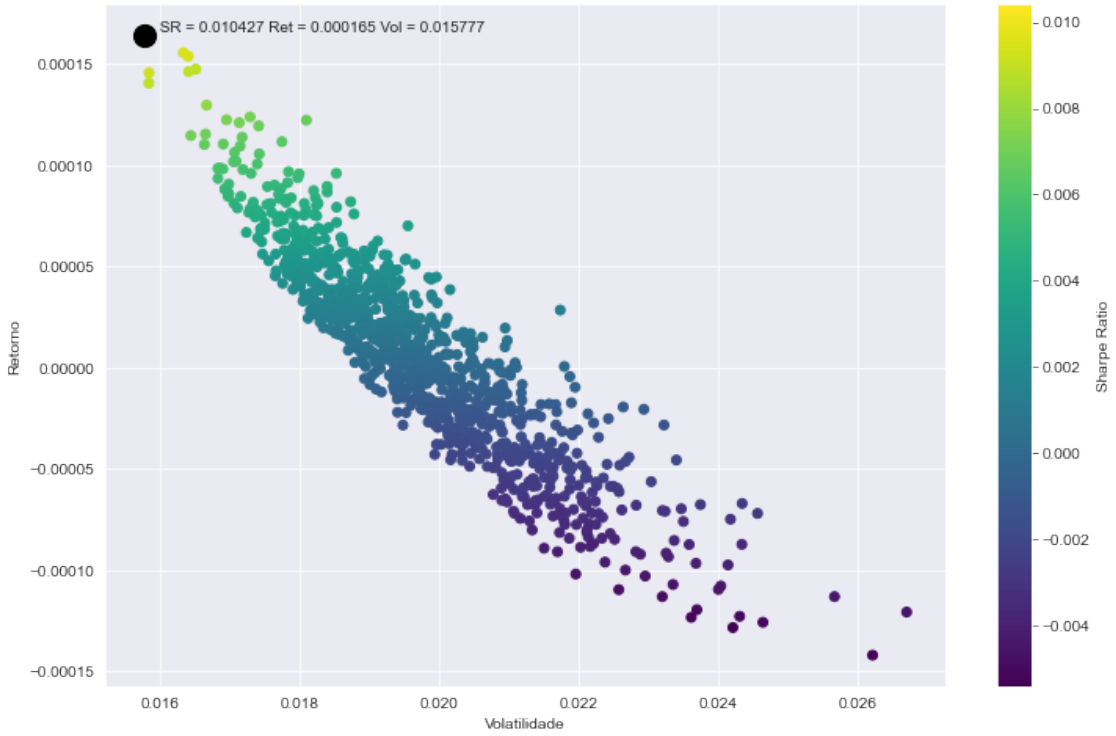


Gráfico da Fronteira Eficiente

[44]:
```
# Definição de algumas funções auxiliares
def get_ret_vol_sr(weights):
    weights = np.array(weights)
    ret = np.sum(log_ret.mean() * weights)
    vol = np.sqrt(np.dot(weights.T, np.dot(log_ret.cov(), weights)))
    sr = ret/vol
    return np.array([ret, vol, sr])

def neg_sharpe(weights):
# the number 2 is the sharpe ratio index from the get_ret_vol_sr
    return get_ret_vol_sr(weights)[2] * -1

def check_sum(weights):
    #return 0 if sum of the weights is 1
    return np.sum(weights)-1
```

[45]:
```
cons = ({'type': 'eq', 'fun': check_sum})
bounds = ((0,1), (0,1), (0,1), (0,1), (0,1), (0,1), (0,1))
init_guess = ((0.2),(0.2),(0.2),(0.2),(0.2),(0.2),(0.2))
```

[46]:
```
op_results = optimize.minimize(neg_sharpe, init_guess, method="SLSQP", bounds=␣
 ↪bounds, constraints=cons)
```

```
print(op_results)
```

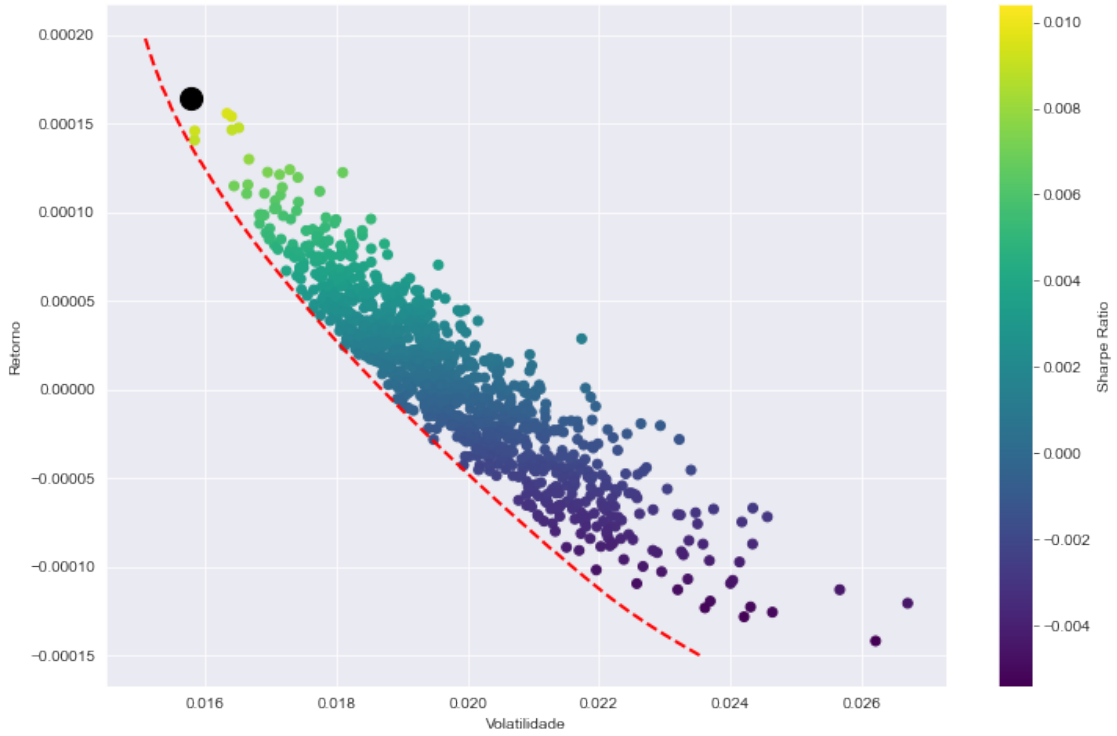
```
     fun: -0.022554718951166418
     jac: array([ 8.99131410e-05,  1.18304810e-02,  4.20250674e-03,
1.49394758e-02,
        1.82465413e-02,  2.82646166e-02, -3.02586704e-06])
 message: 'Optimization terminated successfully'
    nfev: 112
     nit: 14
    njev: 14
  status: 0
 success: True
       x: array([3.25617545e-02, 1.90819582e-17, 2.30718222e-16, 4.33680869e-18,
       1.73472348e-17, 0.00000000e+00, 9.67438246e-01])
```

[47]:
```
frontier_y = np.linspace(-0.00015, 0.00020, 200)
```

[48]:
```
def minimize_volatility(weights):
    return get_ret_vol_sr(weights)[1]
```

[49]:
```
frontier_x = []

for possible_return in frontier_y:
    cons = ({'type':'eq', 'fun':check_sum},
            {'type':'eq', 'fun': lambda w: get_ret_vol_sr(w)[0] -␣
 ↪possible_return})

    result = optimize.minimize(minimize_volatility,init_guess,method='SLSQP',␣
 ↪bounds=bounds, constraints=cons)
    frontier_x.append(result['fun'])
```

[50]:
```
plt.figure(figsize=(12,8))
plt.scatter(vol_arr, ret_arr, c=sharpe_arr, cmap='viridis')
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Volatilidade')
plt.ylabel('Retorno')
plt.plot(frontier_x,frontier_y, 'r--', linewidth=2)
plt.scatter(max_sr_vol, max_sr_ret,c='black', s=200)
plt.show()
```



3.5.2 Alocação

[51]:
```
# Alocação da carteira
sim_carteira_efic, sim_datas_efic, sim_pesos_carteira_efic,␣
 ↪sim_soma_valor_carteira_efic = alocacao_ativos(acoes_port, 10000, 10,␣
 ↪max_sr_weights)

# Inclusão da Taxa de Retorno de BOVA11
sim_carteira_efic['TR BOVA11.SA'] = taxas_retorno_date['BOVA11.SA']*100

# Inclusão de Date
sim_carteira_efic['Date'] = sim_datas_efic

# Remoção da primeira linha vazia
sim_carteira_efic = sim_carteira_efic.iloc[1:,:]

sim_carteira_efic
```

[51]:

	BBDC4.SA	PETR4.SA	VALE3.SA	GGBR4.SA	CYRE3.SA	\
1	2440.852248	989.705224	1905.059823	429.638125	569.882488	
2	2421.325415	1003.079619	1945.382371	432.492847	570.595749	
3	2414.816471	993.717584	1953.446937	422.786759	559.659352	
4	2412.212928	988.367805	1972.508549	421.644865	558.708337	

```
5     2414.165629   985.157979  1966.643448  421.930318  564.176536
…           …          …           …          …          …
3137  3552.592836   899.829274  2490.837139  347.136177  347.112423
3138  3578.269697   890.199693  2539.224253  352.131974  345.210439
3139  3565.431266   899.829274  2501.101309  343.425014  338.791226
3140  3578.269697   922.298262  2452.714195  341.997639  345.923677
3141  3567.265378   867.463263  2381.599694  336.002715  343.546186

        GOLL4.SA      KLBN4.SA    SOMA VALOR  TR CARTEIRA  TR BOVA11.SA  \
1     167.066550   3559.874333  10062.078791     0.618869      0.761113
2     168.988323   3513.812922  10055.677246    -0.063641      0.570617
3     170.781986   3546.713762  10061.922850     0.062091     -0.427660
4     179.045637   3467.751903  10000.240023    -0.614919     -0.745625
5     179.558110   3553.294244  10084.926263     0.843276      0.731335
…          …           …           …          …            …
3137   72.386965  13456.456872  21166.351685    -0.898217      0.073302
3138   76.358638  13160.348528  20941.743223    -1.066829      0.383946
3139   73.540028  12930.042648  20652.160765    -1.392450     -1.137834
3140   71.041720  12930.042648  20642.287839    -0.047817      0.036908
3141   66.621628  12666.835144  20229.334008    -2.020805     -1.862251

           Date
1    2010-01-05
2    2010-01-06
3    2010-01-07
4    2010-01-08
5    2010-01-11
…         …
3137 2022-08-24
3138 2022-08-25
3139 2022-08-26
3140 2022-08-29
3141 2022-08-30

[3141 rows x 11 columns]
```

3.5.3 Análise Descritiva

Pesos de Alocação

[52]:
```
sim_pesos_carteira_efic
```

[52]:
```
      Ações     Pesos
0  BBDC4.SA  0.245387
1  PETR4.SA  0.099826
2  VALE3.SA  0.188746
3  GGBR4.SA  0.042250
4  CYRE3.SA  0.057535
```

```
5  GOLL4.SA  0.016848
6  KLBN4.SA  0.349407
```

Sumário

[53]:
```
sim_carteira_efic.filter(["TR CARTEIRA", "TR BOVA11.SA"]).describe()
```

[53]:
```
       TR CARTEIRA  TR BOVA11.SA
count  3141.000000   3141.000000
mean      0.022431      0.013618
std       1.475113      1.585445
min     -15.118237    -15.752788
25%      -0.759566     -0.800978
50%       0.006060      0.017151
75%       0.831162      0.885836
max       9.478022     12.570843
```

Cálculo das Médias

[54]:
```
arr_tr_media.append(sim_carteira_efic.filter(["TR CARTEIRA"]).values.mean()*100)
sim_carteira_efic.filter(["TR CARTEIRA", "TR BOVA11.SA"]).mean()*100
```

[54]:
```
TR CARTEIRA     2.243071
TR BOVA11.SA    1.361831
dtype: float64
```

Cálculo de Variâncias

[55]:
```
arr_var.append(sim_carteira_efic.filter(["TR CARTEIRA"]).values.var())
sim_carteira_efic.filter(["TR CARTEIRA", "TR BOVA11.SA"]).var()
```

[55]:
```
TR CARTEIRA     2.175958
TR BOVA11.SA    2.513635
dtype: float64
```

Cálculo de Desvios Padrão

[56]:
```
arr_dp.append(sim_carteira_efic.filter(["TR CARTEIRA"]).values.std())
sim_carteira_efic.filter(["TR CARTEIRA", "TR BOVA11.SA"]).std()*100
```

[56]:
```
TR CARTEIRA     147.511291
TR BOVA11.SA    158.544463
dtype: float64
```

Cálculo de Covariâncias

[57]:
```
arr_cov.append(sim_carteira_efic.filter(["TR CARTEIRA", "TR BOVA11.SA"]).cov().
  ↪loc['TR CARTEIRA','TR BOVA11.SA'])
sim_carteira_efic.filter(["TR CARTEIRA", "TR BOVA11.SA"]).cov()
```

[57]:
```
               TR CARTEIRA  TR BOVA11.SA
TR CARTEIRA       2.175958      1.609883
```

```
TR BOVA11.SA     1.609883      2.513635
```

Cálculo de Correlações

[58]:
```
arr_corr.append(sim_carteira_efic.filter(["TR CARTEIRA", "TR BOVA11.SA"]).
  ↪corr().loc['TR CARTEIRA','TR BOVA11.SA'])
sim_carteira_efic.filter(["TR CARTEIRA", "TR BOVA11.SA"]).corr()
```

[58]:

	TR CARTEIRA	TR BOVA11.SA
TR CARTEIRA	1.000000	0.688364
TR BOVA11.SA	0.688364	1.000000

3.5.4 Análise Gráfica

Mapa de Correlações

[59]:
```
plt.figure(figsize=(6,6))
sns.heatmap(sim_carteira_efic.filter(["TR CARTEIRA", "TR BOVA11.SA"]).corr(),␣
  ↪annot=True);
```

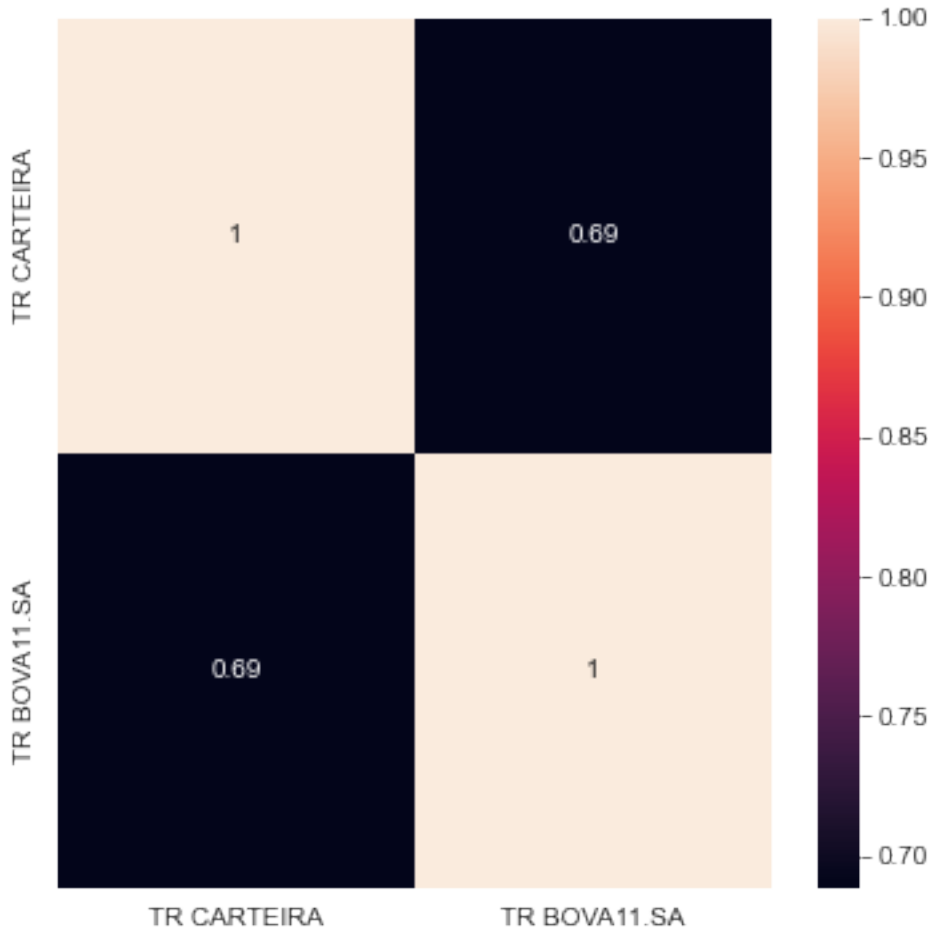
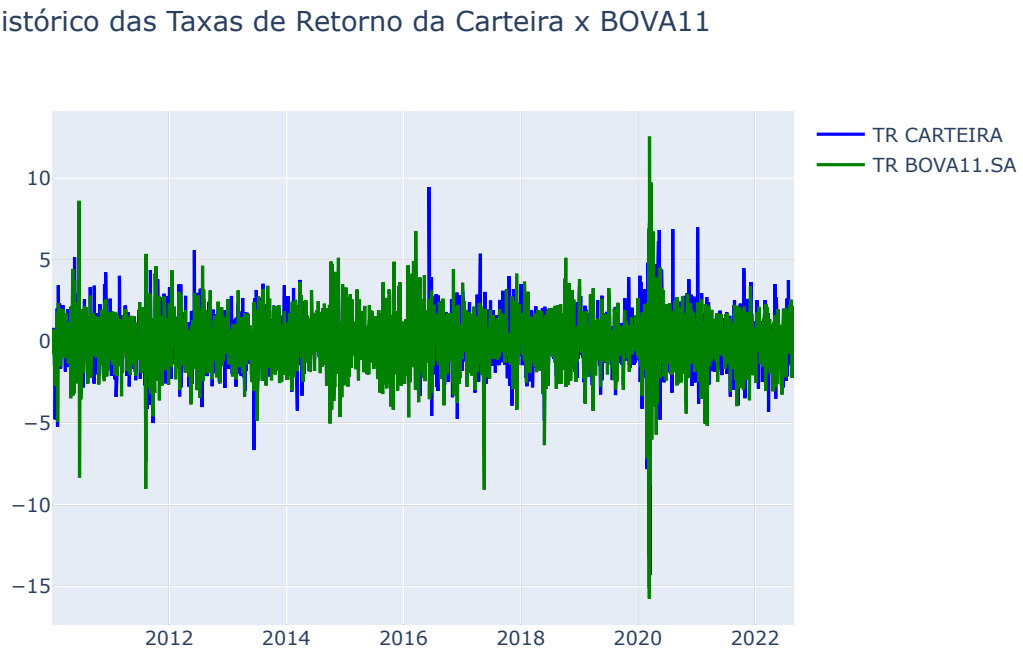


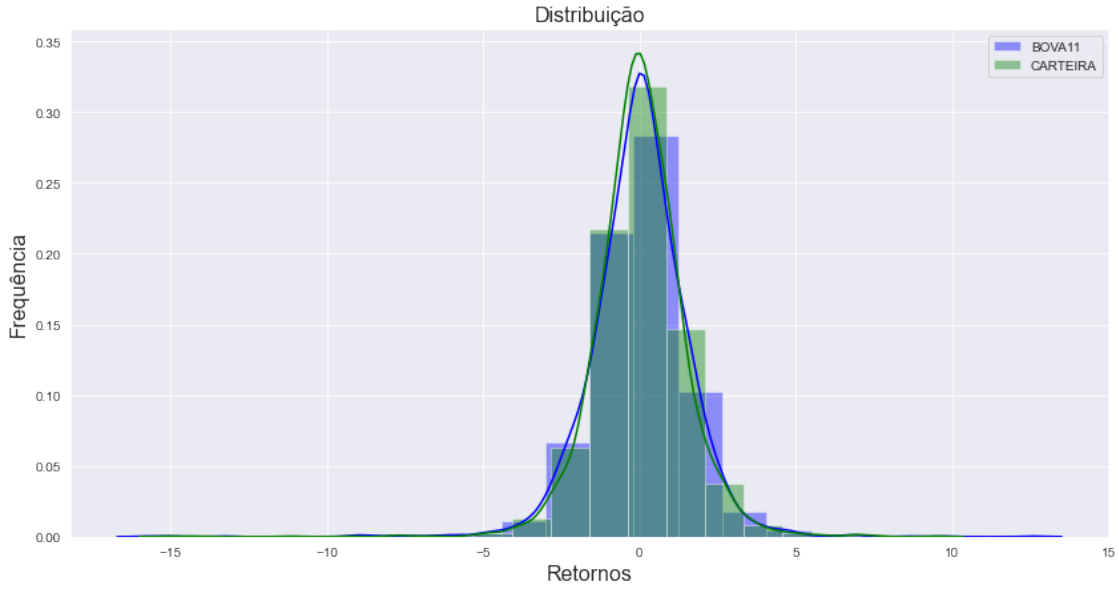
Gráfico do Histórico de Taxas de Retorno

[60]:
```
figura = px.line(title = 'Histórico das Taxas de Retorno da Carteira x BOVA11')
figura.add_scatter(x = sim_carteira_efic["Date"] ,y = sim_carteira_efic["TR␣
 ↪CARTEIRA"], name = "TR CARTEIRA", line_color = 'blue')
figura.add_scatter(x = sim_carteira_efic["Date"] ,y = sim_carteira_efic["TR␣
 ↪BOVA11.SA"], name = "TR BOVA11.SA", line_color = 'green')
figura.show()
```



Histograma da Distribuição das Taxas de Retorno

[61]:
```
fig, ax = plt.subplots(figsize=(14,7))
ax = sns.distplot(sim_carteira_efic['TR BOVA11.SA'], bins=20, label='BOVA11',␣
 ↪color = 'blue')
ax = sns.distplot(sim_carteira_efic['TR CARTEIRA'], bins=20, label='CARTEIRA',␣
 ↪color = 'green')
ax.set_xlabel("Retornos",fontsize=16)
ax.set_ylabel("Frequência",fontsize=16)
plt.title('Distribuição',fontsize=16)
plt.legend();
```



3.6 Comparação entre Carteiras e BOVA11

[62]:
```
# Cria carteira com BOVA11 para comparação
acoes_index = acoes_df.copy()
acoes_index.drop(labels=['BBDC4.SA', 'PETR4.SA', 'VALE3.SA','GGBR4.SA', 'CYRE3.
 ↪SA', 'GOLL4.SA','KLBN4.SA'], axis=1, inplace=True)

# Alocação da carteira
sim_carteira_bova, sim_datas_bova, sim_pesos_carteira_bova,␣
 ↪sim_soma_valor_carteira_bova = alocacao_ativos(acoes_index, 10000, 10)

# Inclusão da Taxa de Retorno de BOVA11
sim_carteira_bova['TR BOVA11.SA'] = taxas_retorno_date['BOVA11.SA']*100

# Inclusão de Date
sim_carteira_bova['Date'] = sim_datas_bova

# Remoção da primeira linha vazia
sim_carteira_bova = sim_carteira_bova.iloc[1:,:]
```

3.6.1 Análise Descritiva

[63]:
```
arr_tr_media.append(sim_carteira_bova.filter(["TR CARTEIRA"]).values.mean()*100)
arr_var.append(sim_carteira_bova.filter(["TR CARTEIRA"]).values.var())
arr_dp.append(sim_carteira_bova.filter(["TR CARTEIRA"]).values.std())
arr_cov.append(sim_carteira_bova.filter(["TR CARTEIRA", "TR BOVA11.SA"]).cov().
 ↪loc['TR CARTEIRA','TR BOVA11.SA'])
```

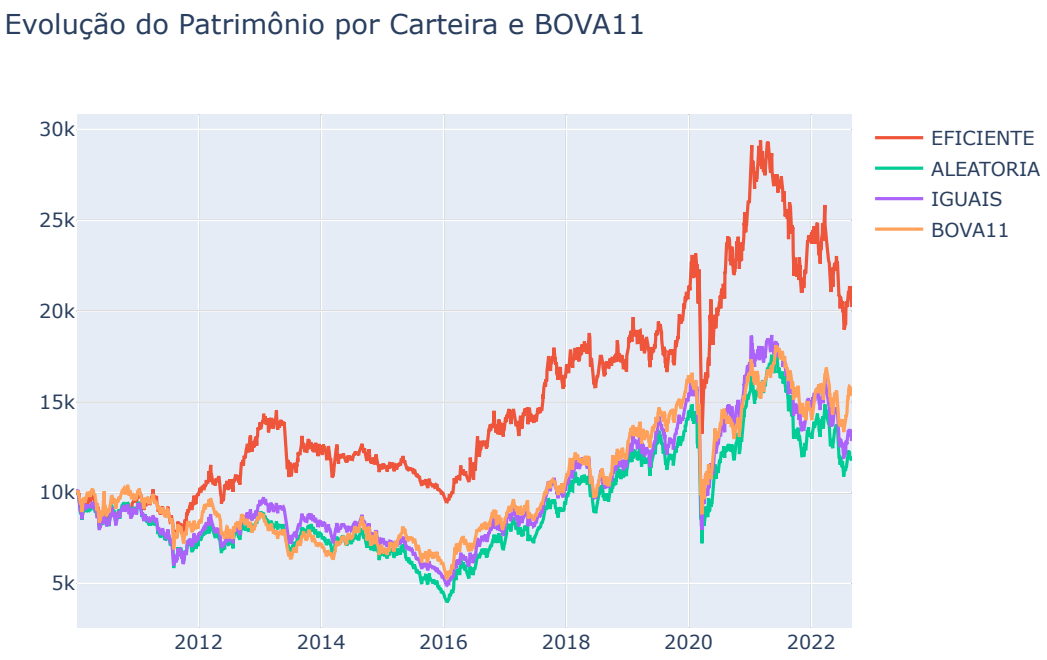
```
arr_corr.append(sim_carteira_bova.filter(["TR CARTEIRA", "TR BOVA11.SA"]).
 ↪corr().loc['TR CARTEIRA','TR BOVA11.SA'])


sumario_carteiras['VALOR FINAL'] = sim_soma_valor_carteira_efic.
 ↪round(2),sim_soma_valor_carteira_aleat.
 ↪round(2),sim_soma_valor_carteira_iguais.
 ↪round(2),sim_soma_valor_carteira_bova.round(2)
sumario_carteiras['LUCRO'] = sumario_carteiras['VALOR FINAL'] -␣
 ↪sumario_carteiras['VALOR INICIAL']
sumario_carteiras['TR MÉDIA'] = arr_tr_media
sumario_carteiras['VAR'] = arr_var
sumario_carteiras['STD'] = arr_dp
sumario_carteiras['COV'] = arr_cov
sumario_carteiras['COR'] = arr_corr

sumario_carteiras
```

[63]:

	CARTEIRA	VALOR INICIAL	VALOR FINAL	LUCRO	TR MÉDIA	VAR	STD	COV	COR
0	IGUAIS	10000	20229.33	10229.33	0.799243	2.420164	1.555687	2.074274	0.840860
1	ALEATORIA	10000	11754.10	1754.10	0.514541	2.786337	1.669233	2.319145	0.876175
2	EFICIENTE	10000	12853.64	2853.64	2.243071	2.175265	1.474878	1.609883	0.688364
3	BOVA11.SA	10000	15338.04	5338.04	1.361831	2.512834	1.585192	2.513635	1.000000

3.6.2 Análise Gráfica

[64]:
```
figura = px.line(title = 'Evolução do Patrimônio por Carteira e BOVA11')
figura.add_scatter(x = sim_datas_efic, y = sim_carteira_efic['SOMA VALOR'],␣
 ↪name = 'EFICIENTE')
figura.add_scatter(x = sim_datas_aleat, y = sim_carteira_aleat['SOMA VALOR'],␣
 ↪name = 'ALEATORIA')
figura.add_scatter(x = sim_datas_iguais, y = sim_carteira_iguais['SOMA VALOR'],␣
 ↪name = 'IGUAIS')
figura.add_scatter(x = sim_datas_bova, y = sim_carteira_bova['SOMA VALOR'],␣
 ↪name = 'BOVA11')
figura.show()
```



4 Análise de Variância

Suposições: Para que os resultados de uma ANOVA unidirecional sejam válidos, as seguintes suposições devem ser atendidas:

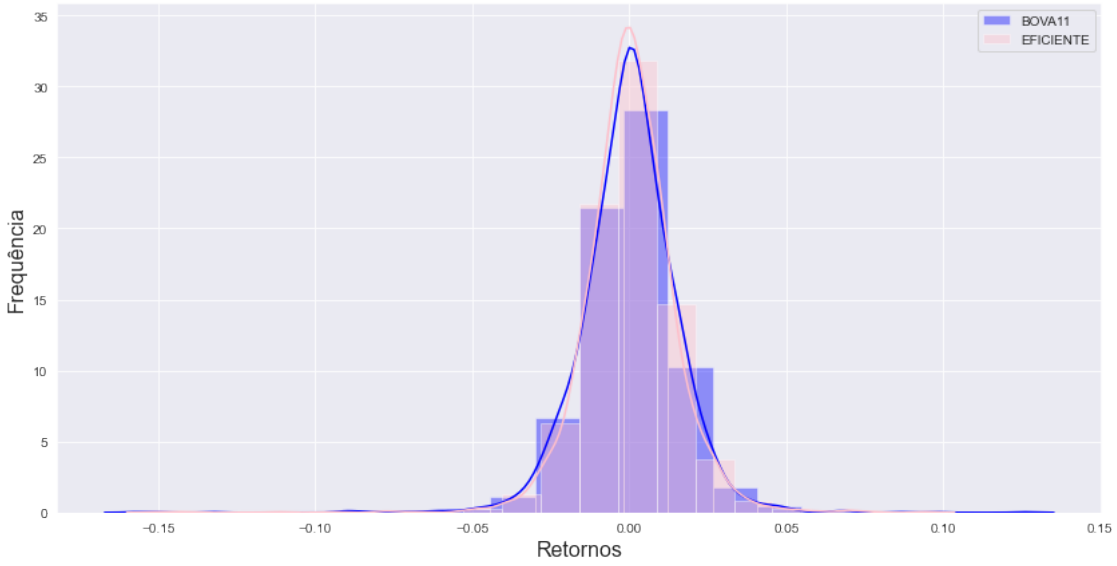
1. Normalidade – Cada amostra foi colhida de uma população normalmente distribuída.
2. Variâncias Iguais – As variâncias das populações de onde as amostras vêm são iguais. Vamos usar o teste de Bartlett para verificar esta suposição.
3. Independência – As observações em cada grupo são independentes entre si e as observações dentro dos grupos foram obtidas por uma amostra aleatória.

4.1 Normalidade

4.1.1 Histograma das Taxas de Retorno

[65]:
```
taxas_retorno = taxas_retorno.assign(EFICIENTE = sim_carteira_efic['TR␣
↪CARTEIRA']/100)
taxas_retorno = taxas_retorno.assign(ALEATORIA = sim_carteira_aleat['TR␣
↪CARTEIRA']/100)
taxas_retorno = taxas_retorno.assign(IGUAIS = sim_carteira_iguais['TR␣
↪CARTEIRA']/100)
```

```
fig, ax = plt.subplots(figsize=(14,7))
ax = sns.distplot(taxas_retorno['BOVA11.SA'], bins=20, label='BOVA11', color =␣
 ↪'blue')
ax = sns.distplot(taxas_retorno['EFICIENTE'], bins=20, label='EFICIENTE', color␣
 ↪= 'pink')
ax.set_xlabel("Retornos",fontsize=16)
ax.set_ylabel("Frequência",fontsize=16)
plt.legend();
```



4.1.2 Teste Shapiro-Wilk nas Taxas de Retorno

O teste de Shapiro-Wilk testa a hipótese nula de que os dados foram extraídos de uma distribuição normal.

Carteira Eficiente

[66]:
```
ShapiroStat, ShapiroPvalue = stats.shapiro(taxas_retorno['EFICIENTE'])
print("p = {:g}".format(ShapiroPvalue))
if ShapiroPvalue < 0.05:  # null hypothesis: x comes from a normal distribution
    print("Hipótese nula pode ser rejeitada")
else:
    print("Hipótese nula não pode ser rejeitada")
```

```
p = 8.17851e-36
Hipótese nula pode ser rejeitada
```

Benchmark (BOVA11)

[67]:
```
ShapiroStat, ShapiroPvalue = stats.shapiro(taxas_retorno['BOVA11.SA'])
print("p = {:g}".format(ShapiroPvalue))
if ShapiroPvalue < 0.05:  # null hypothesis: x comes from a normal distribution
    print("Hipótese nula pode ser rejeitada")
else:
    print("Hipótese nula não pode ser rejeitada")
```

```
p = 7.42382e-37
Hipótese nula pode ser rejeitada
```

4.1.3 Teste Normal nas Taxas de Retorno

Essa função testa a hipótese nula de que uma amostra vem de uma distribuição normal. É baseado no teste de D'Agostino e Pearson que combina skew e curtose para produzir um teste omnibus de normalidade.

Carteira Eficiente

[68]:
```
Normalk2, NormalPvalue = stats.normaltest(taxas_retorno['EFICIENTE'])
print("p = {:g}".format(NormalPvalue))
if NormalPvalue < 0.05:  # null hypothesis: x comes from a normal distribution
    print("Hipótese nula pode ser rejeitada")
else:
    print("Hipótese nula não pode ser rejeitada")
```

```
p = 1.97742e-165
Hipótese nula pode ser rejeitada
```

Benchmark (BOVA11)

[69]:
```
Normalk2, NormalPvalue = stats.normaltest(taxas_retorno['BOVA11.SA'])
print("p = {:g}".format(NormalPvalue))
if NormalPvalue < 0.05:  # null hypothesis: x comes from a normal distribution
    print("Hipótese nula pode ser rejeitada")
else:
    print("Hipótese nula não pode ser rejeitada")
```

```
p = 6.83319e-179
Hipótese nula pode ser rejeitada
```

4.1.4 Conclusão

Não obtivemos resultados com relevância estatística para afirmar que as amostras possuem distribuição normal.

4.2 Variâncias iguais

4.2.1 Teste de Bartlett nas Taxas de Retorno

O teste de Bartlett testa a hipótese nula de que todas as amostras de entrada são de populações com variâncias iguais. Para amostras de populações significativamente não normais, o teste de

Levene é mais robusto.

[70]:
```
Bstats, Bpvalue = stats.bartlett(   taxas_retorno['EFICIENTE'],
                                    taxas_retorno['BOVA11.SA'])
print('Bartlett Test')
print(f'Statistics = {Bstats}')
print(f'Pvalue     = {Bpvalue}')
if (Bpvalue < 0.05):
    print('Hipótese nula pode ser rejeitada')
else:
    ('Hipótese nula não pode ser rejeitada')
```

```
Bartlett Test
Statistics = 16.32001001711044
Pvalue     = 5.349606708895127e-05
Hipótese nula pode ser rejeitada
```

4.2.2 Teste de Levene nas Taxas de Retorno

O teste de Levene testa a hipótese nula de que todas as amostras de entrada são de populações com variâncias iguais. O teste de Levene é uma alternativa ao teste de Bartlett de Bartlett no caso de existirem desvios significativos da normalidade.

[71]:
```
Lstats, Lpvalue = stats.levene(   taxas_retorno['EFICIENTE'],
                                  taxas_retorno['BOVA11.SA'])
print('Levene Test')
print(f'Statistics = {Lstats}')
print(f'Pvalue     = {Lpvalue}')
if (Lpvalue < 0.05):
    print('Hipótese nula pode ser rejeitada')
else:
    ('Hipótese nula não pode ser rejeitada')
```

```
Levene Test
Statistics = 7.064223507183266
Pvalue     = 0.007883758377545758
Hipótese nula pode ser rejeitada
```

4.2.3 Conclusão

Não obtivemos resultados com relevância estatística para afirmar que as amostras possuem variâncias iguais.

4.3 Independência

Cálculo das covariâncias

[72]:
```
taxas_retorno.cov()
```

[72]:

	BBDC4.SA	PETR4.SA	VALE3.SA	GGBR4.SA	CYRE3.SA	GOLL4.SA	\
BBDC4.SA	0.000457	0.000356	0.000210	0.000274	0.000318	0.000434	
PETR4.SA	0.000356	0.000872	0.000353	0.000392	0.000362	0.000535	
VALE3.SA	0.000210	0.000353	0.000694	0.000442	0.000204	0.000319	
GGBR4.SA	0.000274	0.000392	0.000442	0.000766	0.000305	0.000439	
CYRE3.SA	0.000318	0.000362	0.000204	0.000305	0.000752	0.000575	
GOLL4.SA	0.000434	0.000535	0.000319	0.000439	0.000575	0.001812	
KLBN4.SA	0.000083	0.000123	0.000114	0.000147	0.000119	0.000154	
BOVA11.SA	0.000254	0.000337	0.000244	0.000271	0.000286	0.000388	
EFICIENTE	0.000183	0.000220	0.000198	0.000218	0.000191	0.000251	
ALEATORIA	0.000290	0.000315	0.000302	0.000348	0.000312	0.000415	
IGUAIS	0.000233	0.000301	0.000249	0.000291	0.000279	0.000404	

	KLBN4.SA	BOVA11.SA	EFICIENTE	ALEATORIA	IGUAIS
BBDC4.SA	0.000083	0.000254	0.000183	0.000290	0.000233
PETR4.SA	0.000123	0.000337	0.000220	0.000315	0.000301
VALE3.SA	0.000114	0.000244	0.000198	0.000302	0.000249
GGBR4.SA	0.000147	0.000271	0.000218	0.000348	0.000291
CYRE3.SA	0.000119	0.000286	0.000191	0.000312	0.000279
GOLL4.SA	0.000154	0.000388	0.000251	0.000415	0.000404
KLBN4.SA	0.000331	0.000106	0.000236	0.000151	0.000194
BOVA11.SA	0.000106	0.000251	0.000161	0.000232	0.000207
EFICIENTE	0.000236	0.000161	0.000218	0.000204	0.000213
ALEATORIA	0.000151	0.000232	0.000204	0.000279	0.000248
IGUAIS	0.000194	0.000207	0.000213	0.000248	0.000242

Cálculo das correlações

[73]:
```
taxas_retorno.corr()
```

[73]:

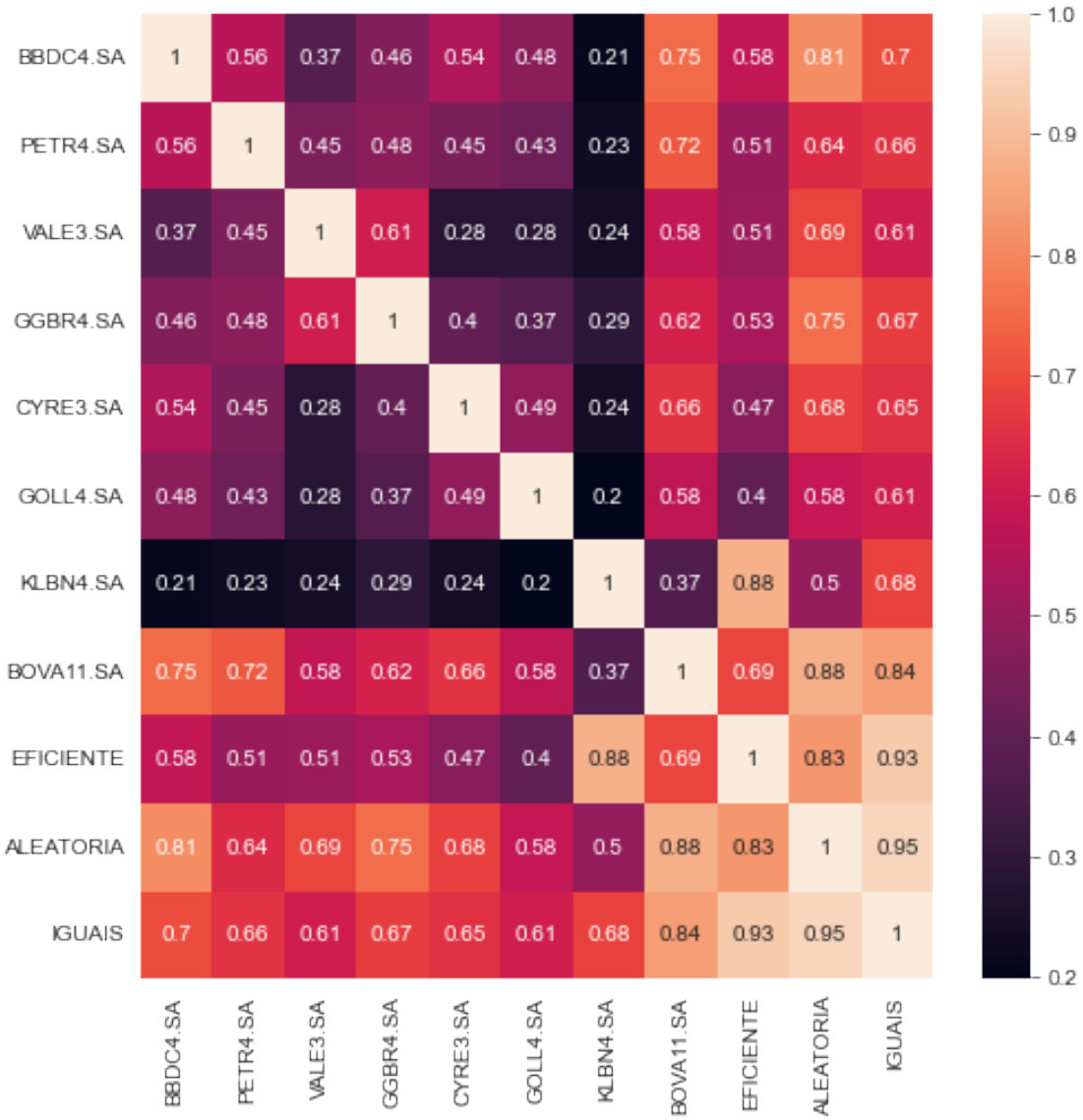
	BBDC4.SA	PETR4.SA	VALE3.SA	GGBR4.SA	CYRE3.SA	GOLL4.SA	\
BBDC4.SA	1.000000	0.564511	0.373955	0.463683	0.542833	0.477291	
PETR4.SA	0.564511	1.000000	0.453256	0.479129	0.447279	0.425426	
VALE3.SA	0.373955	0.453256	1.000000	0.606024	0.282941	0.284754	
GGBR4.SA	0.463683	0.479129	0.606024	1.000000	0.401857	0.372326	
CYRE3.SA	0.542833	0.447279	0.282941	0.401857	1.000000	0.492624	
GOLL4.SA	0.477291	0.425426	0.284754	0.372326	0.492624	1.000000	
KLBN4.SA	0.213191	0.228211	0.237766	0.292673	0.239447	0.198260	
BOVA11.SA	0.750565	0.720655	0.583294	0.618367	0.658028	0.575447	
EFICIENTE	0.581217	0.505240	0.510724	0.533143	0.472854	0.400447	
ALEATORIA	0.811998	0.638177	0.687083	0.753810	0.682018	0.584606	
IGUAIS	0.701083	0.655922	0.608762	0.674810	0.654879	0.609621	

	KLBN4.SA	BOVA11.SA	EFICIENTE	ALEATORIA	IGUAIS
BBDC4.SA	0.213191	0.750565	0.581217	0.811998	0.701083
PETR4.SA	0.228211	0.720655	0.505240	0.638177	0.655922
VALE3.SA	0.237766	0.583294	0.510724	0.687083	0.608762
GGBR4.SA	0.292673	0.618367	0.533143	0.753810	0.674810

CYRE3.SA	0.239447	0.658028	0.472854	0.682018	0.654879
GOLL4.SA	0.198260	0.575447	0.400447	0.584606	0.609621
KLBN4.SA	1.000000	0.366684	0.880623	0.496118	0.683985
BOVA11.SA	0.366684	1.000000	0.688364	0.876175	0.840860
EFICIENTE	0.880623	0.688364	1.000000	0.828872	0.929101
ALEATORIA	0.496118	0.876175	0.828872	1.000000	0.954660
IGUAIS	0.683985	0.840860	0.929101	0.954660	1.000000

Mapa de calor das correlações entre ações da carteira, carteiras simuladas e benchmark

[74]:
```
plt.figure(figsize=(8,8))
sns.heatmap(taxas_retorno.corr(), annot=True);
```



4.3.1 Conclusão

Dada a alta correlação entre as carteiras simuladas, os papéis e o indice de benchmark, não iremos considerar o atributo de independência amostral para a análise de variância, uma vez que não existirá independência entre uma composição de ativos (carteira) e um indice cujos mesmos ativos são parte (BOVA11).

4.4 ANOVA one-way

A ANOVA one-way testa a hipótese nula de que dois ou mais grupos têm a mesma média populacional. O teste é aplicado a amostras de dois ou mais grupos, possivelmente com tamanhos diferentes.

Formulação da Hipótese:

H : = = = … = : as médias das taxas de retorno são iguais

H : nem todas as médias de taxas de retorno são iguais

= 0.05

[75]:
```
anova_hipotese_nula = 'as médias das taxas de retorno são iguais'
```

[76]:
```
taxas_retorno_date = pd.concat([date, taxas_retorno], axis=1)
d_melt = pd.melt(taxas_retorno_date, id_vars=['Date'], value_vars=['BOVA11.SA',␣
 ↪'EFICIENTE'])
d_melt.columns = ['Date', 'Stocks', 'Value']
```

[77]:
```
sample_df = d_melt

# Create ANOVA backbone table
data = [['Entre grupos', '', '', '', '', '', ''], ['Dentro dos grupos', '', '',␣
 ↪'', '', '', ''], ['Total', '', '', '', '', '', '']]
anova_table = pd.DataFrame(data, columns = ['Fonte da Variação', 'SS', 'df',␣
 ↪'MS', 'F', 'P-value', 'F crit'])
anova_table.set_index('Fonte da Variação', inplace = True)

# calculate SSTR and update anova table
x_bar = sample_df['Value'].mean()
SSTR = sample_df.groupby('Stocks').count() * (sample_df.groupby('Stocks').
 ↪mean() - x_bar)**2
anova_table['SS']['Entre grupos'] = SSTR['Value'].sum()

# calculate SSE and update anova table
SSE = (sample_df.groupby('Stocks').count() - 1) * sample_df.groupby('Stocks').
 ↪std()**2
anova_table['SS']['Dentro dos grupos'] = SSE['Value'].sum()

# calculate SSTR and update anova table
SSTR = SSTR['Value'].sum() + SSE['Value'].sum()
```

```
anova_table['SS']['Total'] = SSTR

# update degree of freedom
anova_table['df']['Entre grupos'] = sample_df['Stocks'].nunique() - 1
anova_table['df']['Dentro dos grupos'] = sample_df.shape[0] -␣
 ↪sample_df['Stocks'].nunique()
anova_table['df']['Total'] = sample_df.shape[0] - 1

# calculate MS
anova_table['MS'] = anova_table['SS'] / anova_table['df']

# calculate F
F = anova_table['MS']['Entre grupos'] / anova_table['MS']['Dentro dos grupos']
anova_table['F']['Entre grupos'] = F

# p-value
anova_table['P-value']['Entre grupos'] = 1 - stats.f.cdf(F,␣
 ↪anova_table['df']['Entre grupos'], anova_table['df']['Dentro dos grupos'])

# F critical
alpha = 0.05
# possible types "right-tailed, left-tailed, two-tailed"
tail_hypothesis_type = "two-tailed"
if tail_hypothesis_type == "two-tailed":
    alpha /= 2
anova_table['F crit']['Entre grupos'] = stats.f.ppf(1-alpha,␣
 ↪anova_table['df']['Entre grupos'], anova_table['df']['Dentro dos grupos'])

# Final ANOVA Table
anova_table
```

[77]:

Fonte da Variação	SS	df	MS	F	P-value	F crit
Entre grupos	0.000012	1	0.000012	0.052031	0.819574	5.026296
Dentro dos grupos	1.472532	6282	0.000234			
Total	1.472544	6283	0.000234			

[78]:
```
print('Abordagem 1: P-VALUE')
p = anova_table['P-value']['Entre grupos']
f = anova_table['F']['Entre grupos']
print(f"F score é {f} e P-value é {p}")
if(p < 0.05):
    print(f'Pode-se rejeitar que {anova_hipotese_nula}.')
else:
    print(f'Não se pode rejeitar que {anova_hipotese_nula}.')
```

```
Abordagem 1: P-VALUE
F score é 0.05203071618899679 e P-value é 0.8195740390029633
```

```
Não se pode rejeitar que as médias das taxas de retorno são iguais.
```

[79]:
```
print('Abordagem 1: Valor Crítico')
fc = anova_table['F crit']['Entre grupos']
print(f"F score é {f} e F crit é {fc}")
if(fc==1):
    print(f'Pode-se rejeitar que {anova_hipotese_nula}.')
else:
    print(f'Não se pode rejeitar que {anova_hipotese_nula}.')
```

```
Abordagem 1: Valor Crítico
F score é 0.05203071618899679 e F crit é 5.026295764875974
Não se pode rejeitar que as médias das taxas de retorno são iguais.
```

[80]:
```
Astats, Apvalue = stats.f_oneway(   taxas_retorno['EFICIENTE'],
                                    taxas_retorno['BOVA11.SA'])
print('ANOVA Test')
print(f'Statistics = {Astats}')
print(f'Pvalue     = {Apvalue}')

if (Apvalue < 0.05):
    print(f'Pode-se rejeitar que {anova_hipotese_nula}.')
else:
    print(f'Não se pode rejeitar que {anova_hipotese_nula}.')
```

```
ANOVA Test
Statistics = 0.05201415117269988
Pvalue     = 0.8196022700773641
Não se pode rejeitar que as médias das taxas de retorno são iguais.
```

4.4.1 Conclusão

Não temos evidências suficientes para rejeitar a hipótese nula de igualdade de médias das taxas de retorno, em nível de significância de 5%.

Haja visto que importantes premissas para a realização do Teste ANOVA não foram respeitadas, faremos a seguir um teste H não paramétrico de Kruskal-Wallis, para podemos angariar maiores informações sobre a igualdade das taxas de retorno entre carteira eficiente e benchmark.

4.5 Kruskal-Wallis H-test

O teste H de Kruskal-Wallis testa a hipótese nula de que as medianas populacionais de todos os grupos são iguais. É uma versão não paramétrica da ANOVA. O teste funciona em 2 ou mais amostras independentes, que podem ter tamanhos diferentes. Observe que rejeitar a hipótese nula não indica qual dos grupos difere. Comparações post hoc entre grupos são necessárias para determinar quais grupos são diferentes.

[81]:
```
Kstats, Kpvalue = stats.kruskal(   taxas_retorno['EFICIENTE'],
                                    taxas_retorno['BOVA11.SA'])
print('Kruskal-Wallis H-test')
```

```
print(f'Statistics = {Kstats}')
print(f'Pvalue     = {Kpvalue}')

if (Kpvalue < 0.05):
    print('Hipótese nula pode ser rejeitada')
else:
    print('Hipótese nula não pode ser rejeitada')
```

```
Kruskal-Wallis H-test
Statistics = 0.12985651326712502
Pvalue     = 0.7185808572527445
Hipótese nula não pode ser rejeitada
```

4.5.1 Conclusão

A partir dos resultados do teste H não paramétrico de Kruskal-Wallis, não obtivemos resultados com relevância estatística para rejeitar a hipótese de igualdade das medianas entre as taxas de retorno da carteira eficiente e do parâmetro de benchmark.

5 Regressão Linear

5.1 Visualização dos Dados e Preparação dos Dados

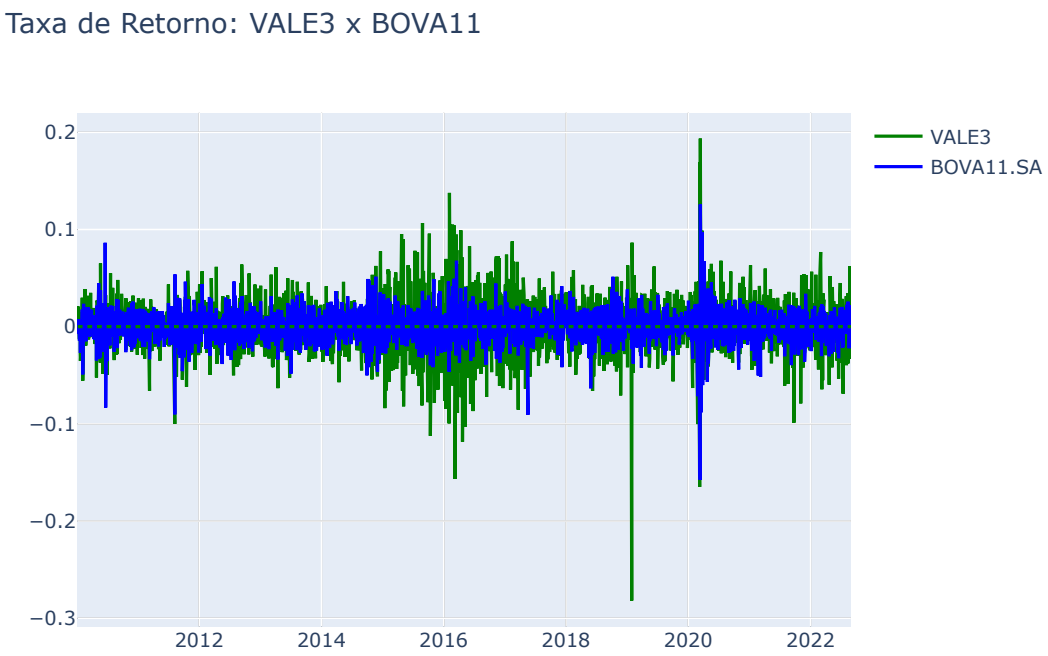
[82]:
```
retornos = taxas_retorno_date.drop(0, axis=0)
retornos
```

[82]:

	Date	BBDC4.SA	PETR4.SA	VALE3.SA	GGBR4.SA	CYRE3.SA	GOLL4.SA	KLBN4.SA	BOVA11.SA	EFICIENTE	ALEATORIA	IGUAIS
1	2010-01-05	-0.005319	-0.008611	0.009279	0.016751	-0.009550	-0.008400	0.018657	0.007611	0.006189	0.003675	0.001898
2	2010-01-06	-0.008032	0.013423	0.020945	0.006623	0.001251	0.011437	-0.013023	0.005706	-0.000636	0.004274	0.004698
3	2010-01-07	-0.002692	-0.009377	0.004137	-0.022698	-0.019353	0.010558	0.009320	-0.004277	0.000621	-0.007060	-0.004221
4	2010-01-08	-0.001079	-0.005398	0.009711	-0.002705	-0.001701	0.047253	-0.022515	-0.007456	-0.006149	0.002956	0.003667
5	2010-01-11	0.000809	-0.003253	-0.002978	0.000677	0.009740	0.002858	0.024369	0.007313	0.008433	0.003006	0.004559
…	…	…	…	…	…	…	…	…	…	…	…	…
3137	2022-08-24	-0.005663	0.005963	-0.032719	-0.008190	0.017272	0.032377	-0.007308	0.000733	-0.008982	-0.009686	-0.006140
3138	2022-08-25	0.007202	-0.010759	0.019240	0.014289	-0.005495	0.053415	-0.022251	0.003839	-0.010668	0.005280	-0.002799
3139	2022-08-26	-0.003594	0.010759	-0.015127	-0.025037	-0.018770	-0.037611					
3140	2022-08-29	0.003594	0.024664	-0.019536	-0.004165	0.020834	-0.034563					
3141	2022-08-30	-0.003080	-0.061296	-0.029423	-0.017685	-0.006897	-0.064238					

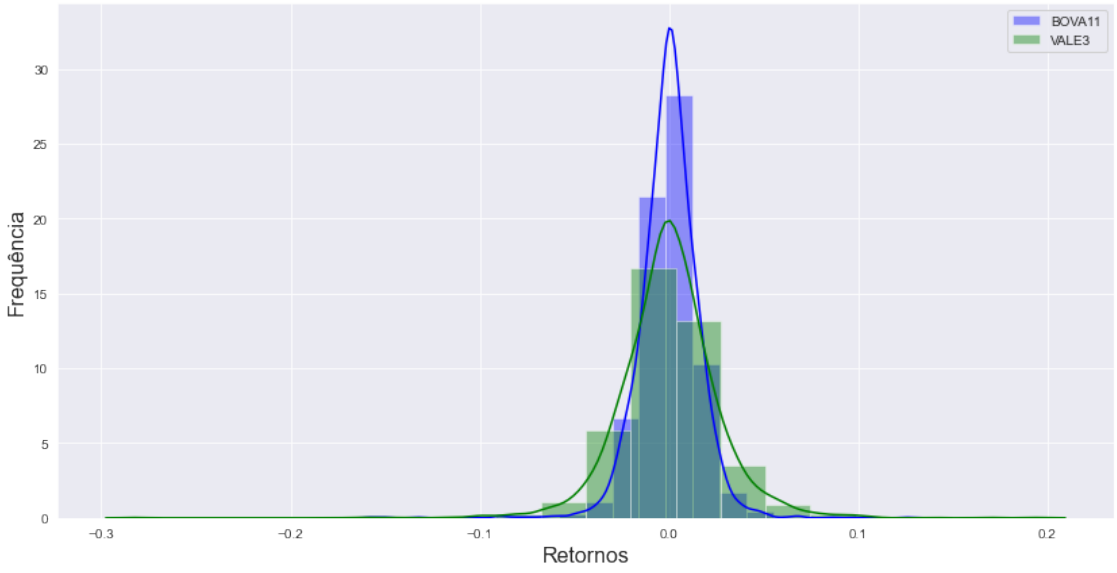
```
3139 -0.017655  -0.011378  -0.013924  -0.014559 -0.014025
3140  0.000000   0.000369  -0.000478  -0.003082 -0.000402
3141 -0.020566  -0.018623  -0.020208  -0.016888 -0.023829

[3141 rows x 12 columns]
```

[83]:
```
figura = px.line(title = 'Taxa de Retorno: VALE3 x BOVA11')
figura.add_scatter(x = retornos["Date"] ,y = retornos['VALE3.SA'], name =␣
 ↪'VALE3',  line_color = 'green')
figura.add_scatter(x = retornos["Date"] ,y = retornos['BOVA11.SA'], name =␣
 ↪'BOVA11.SA', line_color = 'blue')
figura.add_hline(y = retornos['VALE3.SA'].mean(), line_color="green",␣
 ↪line_dash="dot", )
figura.show()
```



[84]:
```
fig, ax = plt.subplots(figsize=(14,7))
ax = sns.distplot(retornos['BOVA11.SA'], bins=20, label='BOVA11', color =␣
 ↪'blue')
ax = sns.distplot(retornos['VALE3.SA'], bins=20, label='VALE3', color = 'green')
ax.set_xlabel("Retornos",fontsize=16)
ax.set_ylabel("Frequência",fontsize=16)
```

```
plt.legend();
```



5.1.1 Base de treino e teste

De forma a poder avaliar nosso modelo de predição, iremos dividir nosso dataset de taxas de retorno entre base de treino e base de teste, de forma aleatória.

[85]:
```
#preparando dados para treino, teste e regressão
retornos_lm     = retornos
retornos_lm     = retornos_lm.drop(labels=['BBDC4.SA','PETR4.SA','GGBR4.
 ↪SA','CYRE3.SA','GOLL4.SA','KLBN4.SA','EFICIENTE','ALEATORIA','IGUAIS'], axis␣
 ↪= 1)
retornos_treino = retornos_lm.sample(frac=0.5)
retornos_teste  =   retornos_lm.drop(retornos_treino.index)
```

5.2 Estimando o Modelo de Regressão

Iremos estimar o modelo por MQO, definindo o retorno do ativo *VALE3* como a variável dependente e o ETF *BOVA11* como a variável independente, a partir de uma regressão linear simples. Este modelo é conhecido como *Market Model*.

$$\mathbb{E}(R_i) = \beta_1 + \beta_2 \cdot \mathbb{E}(R_m)$$

5.3 Modelo

[86]:
```
x = retornos_treino['BOVA11.SA']
y = retornos_treino['VALE3.SA']

x = sm.add_constant(x)
```

```
model = sm.OLS(y, x).fit()
predictions = model.predict(x)

print_model = model.summary()
print(print_model)
```

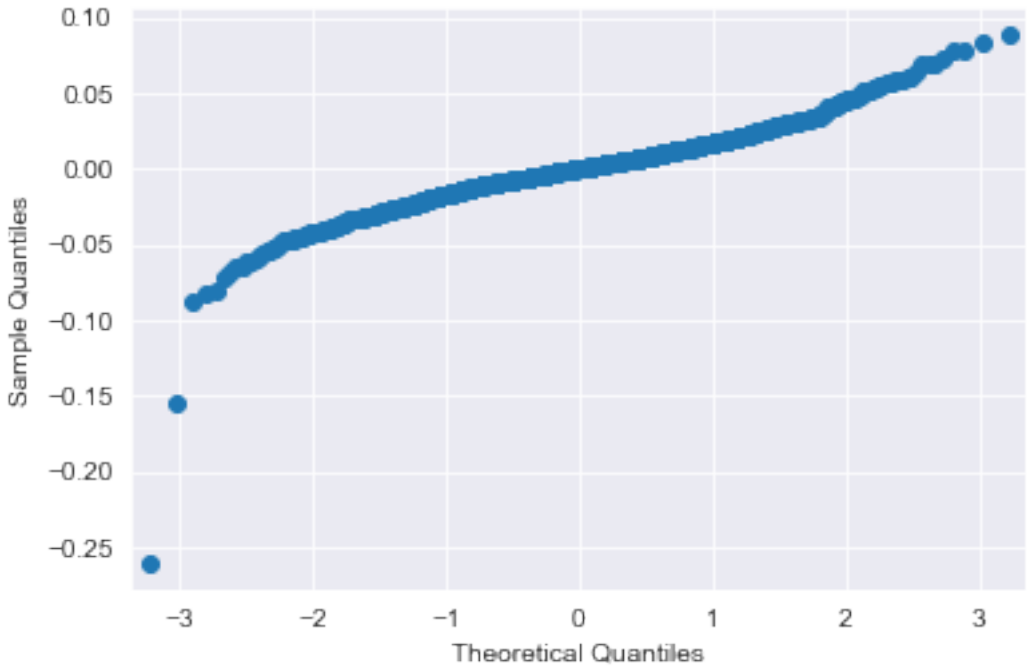
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                VALE3.SA   R-squared:                       0.303
Model:                            OLS   Adj. R-squared:                  0.303
Method:                 Least Squares   F-statistic:                     682.5
Date:                Tue, 30 Aug 2022   Prob (F-statistic):          3.32e-125
Time:                        22:14:22   Log-Likelihood:                 3816.8
No. Observations:                1570   AIC:                            -7630.
Df Residuals:                    1568   BIC:                            -7619.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0004      0.001     -0.749      0.454      -0.001       0.001
BOVA11.SA      0.9520      0.036     26.126      0.000       0.881       1.023
==============================================================================
Omnibus:                      648.171   Durbin-Watson:                   1.976
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            18858.911
Skew:                          -1.314   Prob(JB):                         0.00
Kurtosis:                      19.775   Cond. No.                         67.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

5.4 Diagnóstico do Modelo

5.4.1 Normalidade

QQ Plot

[87]:
```
residuals = y-predictions
sm.qqplot(residuals)
py.show()
```



Shapiro-Wilk O teste de Shapiro-Wilk testa a hipótese nula de que os dados foram extraídos de uma distribuição normal.

[88]:
```
ShapiroStat, ShapiroPvalue = stats.shapiro(residuals)
print("Shapiro-Wilk")
print("Stats    = {:g}".format(ShapiroStat))
print("pvalue   = {:g}\n".format(ShapiroPvalue))
```

```
Shapiro-Wilk
Stats    = 0.915328
pvalue   = 9.37325e-29
```

Kolmogorov-Smirnov Executa o teste de Kolmogorov-Smirnov (uma amostra ou duas amostras) para a qualidade do ajuste.

O teste de uma amostra compara a distribuição subjacente F(x) de uma amostra com uma dada distribuição G(x). O teste de duas amostras compara as distribuições subjacentes de duas amostras independentes. Ambos os testes são válidos apenas para distribuições contínuas.

[89]:
```
KSStat, KSPvalue = stats.kstest(residuals, 'norm')
print("Kolmogorov-Smirnov")
print("Stats    = {:g}".format(KSStat))
print("pvalue   = {:g}\n".format(KSPvalue))
```

```
Kolmogorov-Smirnov
Stats    = 0.470224
pvalue   = 6.10477e-319
```

Cramer-von Mises Realize o teste de Cramér-von Mises de uma amostra para verificar a qualidade do ajuste.

Isso executa um teste da qualidade do ajuste de uma função de distribuição cumulativa (cdf) F em comparação com a função de distribuição empírica F_n de variáveis aleatórias observadas X_1, ..., X_n que são assumidas como independentes e identicamente distribuídas ([1]_). A hipótese nula é que X_i tem distribuição cumulativa F.

[90]:
```
CrMresult = stats.cramervonmises(residuals, 'norm')
print("Cramer-von Mises")
print("Stats    = {:g}".format(CrMresult.statistic))
print("pvalue   = {:g}\n".format(CrMresult.pvalue))
```

```
Cramer-von Mises
Stats    = 124.087
pvalue   = 2.45032e-08
```

Anderson-Darling Teste de Anderson-Darling para dados provenientes de uma distribuição específica.

O teste de Anderson-Darling testa a hipótese nula de que uma amostra é extraída de uma população que segue uma distribuição particular. Para o teste de Anderson-Darling, os valores críticos dependem de qual distribuição está sendo testada. Esta função funciona para distribuições normal, exponencial, logística ou Gumbel (Extreme Value Type I).

Os valores críticos fornecidos são para os seguintes níveis de significância:

- normal/exponencial: 15%, 10%, 5%, 2,5%, 1%
- logística: 25%, 10%, 5%, 2,5%, 1%, 0,5%
- Gumbel: 25%, 10%, 5%, 2,5%, 1%

Se a estatística retornada for maior que esses valores críticos então para o nível de significância correspondente, a hipótese nula de que os dados vêm da distribuição escolhida pode ser rejeitada. A estatística retornada é referida como 'A2' nas referências.

[91]:
```
AnStat, AnCrit, AnSig = stats.anderson(residuals)
print("Anderson-Darling")
print("Stats    = {:g}\n".format(AnStat))
```

```
Anderson-Darling
Stats    = 15.5924
```

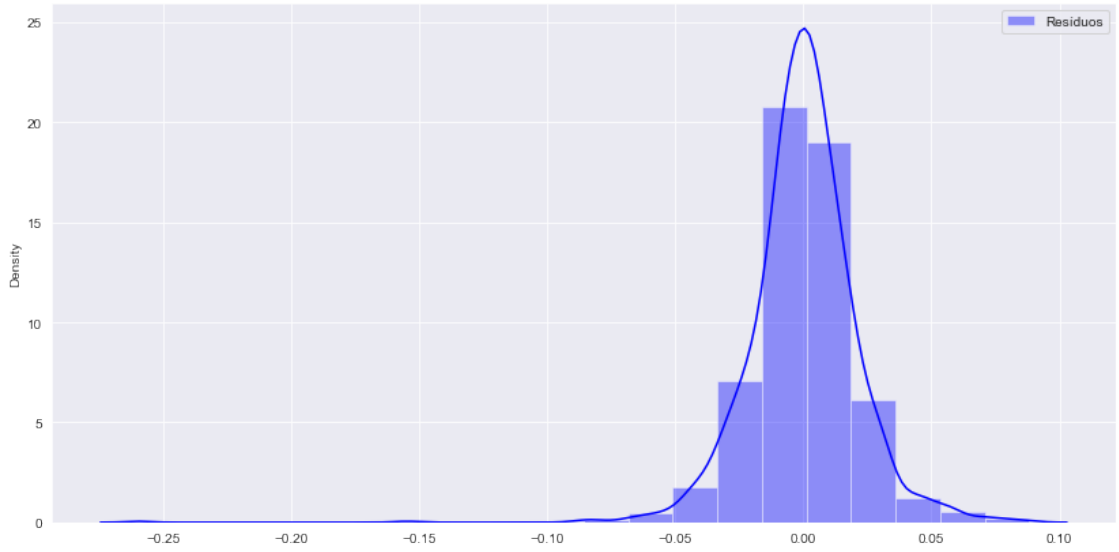
Gráficos de Regressão

[92]:
```
fig = plt.figure(figsize=(14, 8))
fig = sm.graphics.plot_regress_exog(model,
                                    'BOVA11.SA',
                                    fig=fig);
```

```
eval_env: 1
```

Histograma dos Resíduos

[93]:
```
fig, ax = plt.subplots(figsize=(14,7))
ax = sns.distplot(residuals, bins=20, label='Resíduos', color = 'blue')
#ax.set_xlabel("Retornos",fontsize=16)
#ax.set_ylabel("Frequência",fontsize=16)
plt.legend();
```



5.4.2 Testes de Homogeneidade da Variância (Homecedasticidade) e Autocorrelação Durbin-Watson

- Se Durbin–Watson for inferior a 1,0, pode haver motivo para preocupação.
- Pequenos valores de d indicam que termos de erro sucessivos estão positivamente correlacionados.
- Se d > 2, os termos de erro sucessivos são negativamente correlacionados.

[94]:
```
sm.stats.durbin_watson(model.resid)
```

[94]:
```
1.976054510387288
```

Breusch-Pagan Teste multiplicador de Breusch-Pagan Lagrange para heterocedasticidade. O teste testa a hipótese de que a variância residual não depende das variáveis em x na forma. A homocedasticidade implica que alpha=0.

[95]:
```
lm, lmpvalue,fvalue,fpvalue  = sms.het_breuschpagan(model.resid, model.model.
  ↪exog)
print("Breusch-Pagan")
print("Stats    = {:g}".format(lm))
print("pvalue   = {:g}".format(lmpvalue))
print("f        = {:g}".format(fvalue))
print("f pvalue = {:g}\n".format(fpvalue))
```

```
Breusch-Pagan
Stats    = 1.00742
pvalue   = 0.315522
f        = 1.00678
```

```
f pvalue = 0.31583
```

5.4.3 Corrigindo as Estimações para Heterocedasticidade

[96]:
```
model = sm.OLS(y, x).fit(cov_type='HC1')
```

5.5 Modelo Corrigido

[97]:
```
print_model = model.summary()
print(print_model)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                VALE3.SA   R-squared:                       0.303
Model:                            OLS   Adj. R-squared:                  0.303
Method:                 Least Squares   F-statistic:                     491.4
Date:                Tue, 30 Aug 2022   Prob (F-statistic):           6.25e-95
Time:                        22:14:23   Log-Likelihood:                 3816.8
No. Observations:                1570   AIC:                            -7630.
Df Residuals:                    1568   BIC:                            -7619.
Df Model:                           1
Covariance Type:                  HC1
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0004      0.001     -0.749      0.454      -0.001       0.001
BOVA11.SA      0.9520      0.043     22.167      0.000       0.868       1.036
==============================================================================
Omnibus:                      648.171   Durbin-Watson:                   1.976
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            18858.911
Skew:                          -1.314   Prob(JB):                         0.00
Kurtosis:                      19.775   Cond. No.                         67.8
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
```

[98]:
```
fig = plt.figure(figsize=(14, 8))
fig = sm.graphics.plot_regress_exog(model,
                                    'BOVA11.SA',
                                    fig=fig);
```

```
eval_env: 1
```

5.6 Forecast - Predição das Taxas de Retorno de VALE3

[99]:
```
x_test = retornos_teste['BOVA11.SA']

x_test = sm.add_constant(x_test)
retornos_teste['VALE3.SA P'] = model.predict(x_test)
retornos_teste
```

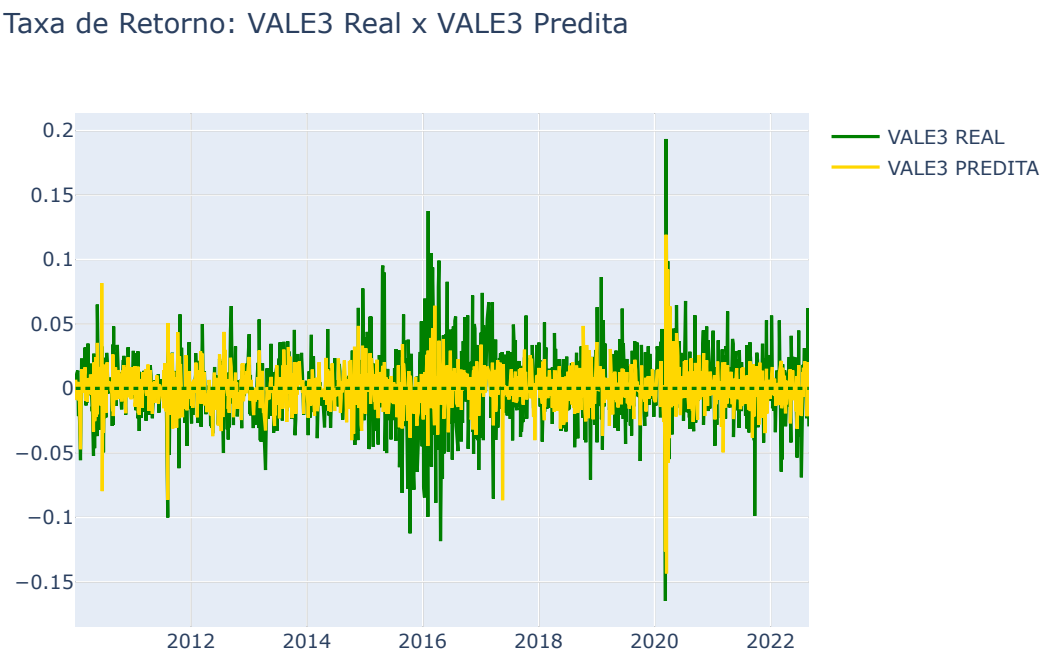
[99]:

	Date	VALE3.SA	BOVA11.SA	VALE3.SA P
1	2010-01-05	0.009279	0.007611	0.006843
3	2010-01-07	0.004137	-0.004277	-0.004474
4	2010-01-08	0.009711	-0.007456	-0.007501
7	2010-01-13	0.012261	0.004295	0.003686
8	2010-01-14	-0.000185	-0.007168	-0.007227
…	…	…	…	…
3136	2022-08-23	0.062139	0.021963	0.020506
3138	2022-08-25	0.019240	0.003839	0.003253
3139	2022-08-26	-0.015127	-0.011378	-0.011234
3140	2022-08-29	-0.019536	0.000369	-0.000051
3141	2022-08-30	-0.029423	-0.018623	-0.018131

[1571 rows x 4 columns]

[100]:
```
figura = px.line(title = 'Taxa de Retorno: VALE3 Real x VALE3 Predita')
figura.add_scatter(x = retornos_teste["Date"] ,y = retornos_teste['VALE3.SA'],␣
 ↪name = 'VALE3 REAL',  line_color = 'green')
```

```
figura.add_scatter(x = retornos_teste["Date"] ,y = retornos_teste['VALE3.SA␣
 ↪P'], name = 'VALE3 PREDITA', line_color = 'gold')
figura.add_hline(y = taxas_retorno['VALE3.SA'].mean(), line_color="green",␣
 ↪line_dash="dot", )
figura.show()
```



6 Modelo de Previsibilidade com Machine Learning

Neste ponto do trabalho, iremos elaborar um novo modelo preditivo para as taxas de retorno de VALE3 com base no algoritmo Random Forest de Machine Learning, utilizando os indicadores de análise técnica SMA9, SMA20, EMA14, RSI14 e OBV. Nosso objetivo é conseguir prever variações da taxa de retorno de VALE3 com base nestes indicadores.

6.1 Aquisição dos Dados

[101]:
```
# Coleta de dados no yahoo finance
vale_df = pd.DataFrame()
vale_df = web.DataReader('VALE3.SA', data_source='yahoo', start='2010-01-01')
# Transforma index Date em coluna Date para fins de plotagem gráfica dos dados.
vale_df.reset_index(inplace=True)
# Cálculo de alguns indicadores de análise técnica
```

```
# média móvel 9, 20
vale_df['SMA9'] = vale_df['Close'].rolling(7).mean()
vale_df['SMA20'] = vale_df['Close'].rolling(21).mean()
vale_df['EMA14']= pta.ema(vale_df['Close'], length=14, offset=None)
# RSI(14)
vale_df['RSI14'] = pta.rsi(vale_df['Close'], length = 14)
# OBV
vale_df['OBV'] = pta.obv(vale_df.Close, vale_df.Volume)
# Cálculo da taxa de retorno
vale_df['RET'] = np.log(vale_df['Close']/vale_df['Close'].shift(1))
# Remove 1a linha sem retorno e colunas desnecessárias
vale_df.dropna(inplace=True)
vale_df.reset_index(drop=True, inplace=True)
vale_df = vale_df.iloc[1:,:]
# Dataset pro modelo
vale_df_mod = vale_df.copy()
vale_df_mod = vale_df_mod.iloc[:,[0,7,8,9,10,11,12]]
# Visualização dos dados coletados
vale_df_mod
```

[101]:

	Date	SMA9	SMA20	EMA14	RSI14	OBV	RET
1	2010-02-03	49.578571	52.038572	50.916277	41.399013	-7493800.0	-0.002185
2	2010-02-04	49.302857	51.828572	50.468774	29.153384	-11582400.0	-0.055615
3	2010-02-05	49.067142	51.547143	50.027604	27.848790	-15357500.0	-0.008446
4	2010-02-08	48.884286	51.271429	49.690590	30.687996	-10938100.0	0.007184
5	2010-02-09	48.825714	51.018572	49.531845	38.369631	-6757200.0	0.020834
…	…	…	…	…	…	…	…
3117	2022-08-24	68.124286	68.953810	68.655396	45.511993	351727532.0	-0.032719
3118	2022-08-25	68.045714	68.894286	68.737343	48.958070	376684232.0	0.019240
3119	2022-08-26	68.045714	68.777143	68.669698	46.464690	348174132.0	-0.015127
3120	2022-08-29	67.930000	68.641905	68.435072	43.440644	326046032.0	-0.019536
3121	2022-08-30	67.645715	68.493810	67.973062	39.383734	326046032.0	-0.029423

[3121 rows x 7 columns]

6.2 Normalização dos Dados

[102]:
```
vale_df_norm = vale_df_mod.copy()

# Normalização dos dados
min_max_scaler = sklearn.preprocessing.MinMaxScaler()
vale_df_norm['SMA9'] = min_max_scaler.fit_transform(vale_df_norm.SMA9.values.
 ↪reshape(-1,1))
vale_df_norm['SMA20'] = min_max_scaler.fit_transform(vale_df_norm.SMA20.values.
 ↪reshape(-1,1))
vale_df_norm['EMA14'] = min_max_scaler.fit_transform(vale_df_norm.EMA14.values.
 ↪reshape(-1,1))
vale_df_norm['RSI14'] = min_max_scaler.fit_transform(vale_df_norm.RSI14.values.
 ↪reshape(-1,1))
vale_df_norm['OBV'] = min_max_scaler.fit_transform(vale_df_norm.OBV.values.
 ↪reshape(-1,1))
vale_df_norm['RET'] = min_max_scaler.fit_transform(vale_df_norm.RET.values.
 ↪reshape(-1,1))

vale_df_norm
```

[102]:

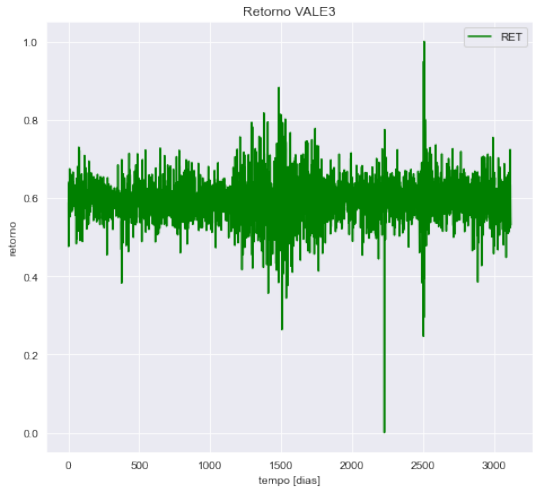
	Date	SMA9	SMA20	EMA14	RSI14	OBV	RET
1	2010-02-03	0.381214	0.406331	0.393848	0.373046	0.359215	0.588219
2	2010-02-04	0.378617	0.404328	0.389597	0.215186	0.356603	0.475829
3	2010-02-05	0.376396	0.401643	0.385406	0.198368	0.354191	0.575050
4	2010-02-08	0.374674	0.399013	0.382205	0.234969	0.357014	0.607927
5	2010-02-09	0.374122	0.396601	0.380697	0.333994	0.359686	0.636641
…	…	…	…	…	…	…	
3117	2022-08-24	0.555914	0.567686	0.562358	0.426067	0.588728	0.523992
3118	2022-08-25	0.555174	0.567118	0.563137	0.470491	0.604673	0.633287
3119	2022-08-26	0.555174	0.566001	0.562494	0.438349	0.586458	0.560995
3120	2022-08-29	0.554084	0.564711	0.560266	0.399365	0.572320	0.551722
3121	2022-08-30	0.551406	0.563298	0.555877	0.347067	0.572320	0.530925

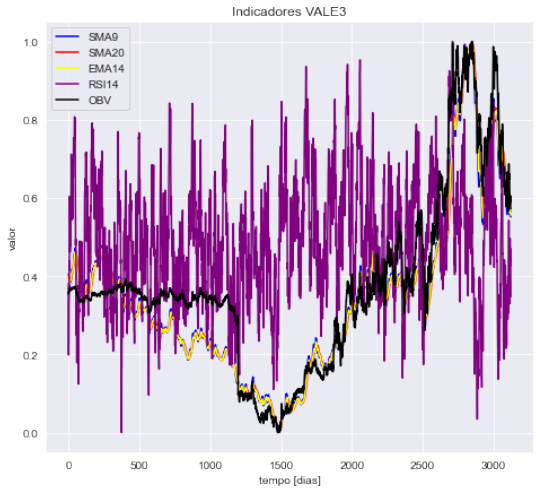
[3121 rows x 7 columns]

6.3 Inspeção dos Dados

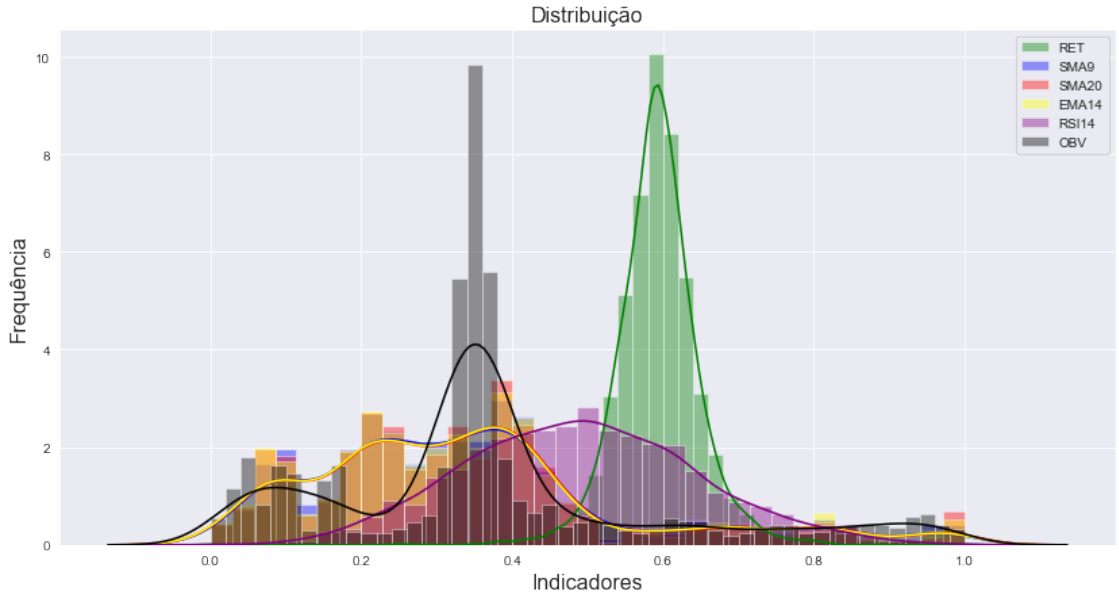
[103]:
```
plt.figure(figsize=(17, 7));
plt.subplot(1,2,1);
plt.plot(vale_df_norm.RET.values, color='green', label='RET')
#plt.plot(df[df.symbol == 'EQIX'].close.values, color='green', label='close')
#plt.plot(df[df.symbol == 'EQIX'].low.values, color='blue', label='low')
#plt.plot(df[df.symbol == 'EQIX'].high.values, color='black', label='high')
plt.title('Retorno VALE3')
plt.xlabel('tempo [dias]')
plt.ylabel('retorno')
```

```
plt.legend(loc='best')
#plt.show()

plt.subplot(1,2,2);
plt.plot(vale_df_norm.SMA9.values, color='blue', label='SMA9')
plt.plot(vale_df_norm.SMA20.values, color='red', label='SMA20')
plt.plot(vale_df_norm.EMA14.values, color='yellow', label='EMA14')
plt.plot(vale_df_norm.RSI14.values, color='purple', label='RSI14')
plt.plot(vale_df_norm.OBV.values, color='black', label='OBV')
plt.title('Indicadores VALE3')
plt.xlabel('tempo [dias]')
plt.ylabel('valor')
plt.legend(loc='best');
```





[104]:
```
fig, ax = plt.subplots(figsize=(14,7))
ax = sns.distplot(vale_df_norm.RET.values, color='green', label='RET')
ax = sns.distplot(vale_df_norm.SMA9.values, color='blue', label='SMA9')
ax = sns.distplot(vale_df_norm.SMA20.values, color='red', label='SMA20')
ax = sns.distplot(vale_df_norm.EMA14.values, color='yellow', label='EMA14')
ax = sns.distplot(vale_df_norm.RSI14.values, color='purple', label='RSI14')
ax = sns.distplot(vale_df_norm.OBV.values, color='black', label='OBV')
ax.set_xlabel("Indicadores",fontsize=16)
ax.set_ylabel("Frequência",fontsize=16)
plt.title('Distribuição',fontsize=16)
plt.legend();
```



[105]:
```
vale_df_norm.describe()
```

[105]:

	SMA9	SMA20	EMA14	RSI14	OBV	RET
count	3121.000000	3121.000000	3121.000000	3121.000000	3121.000000	3121.000000
mean	0.343424	0.344246	0.342857	0.490206	0.375727	0.592987
std	0.211003	0.212822	0.212044	0.153664	0.226176	0.055513
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.204993	0.203191	0.203012	0.380122	0.288530	0.563385
50%	0.316229	0.318558	0.316781	0.488840	0.350822	0.592816
75%	0.418046	0.416093	0.415714	0.593756	0.404671	0.621207
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

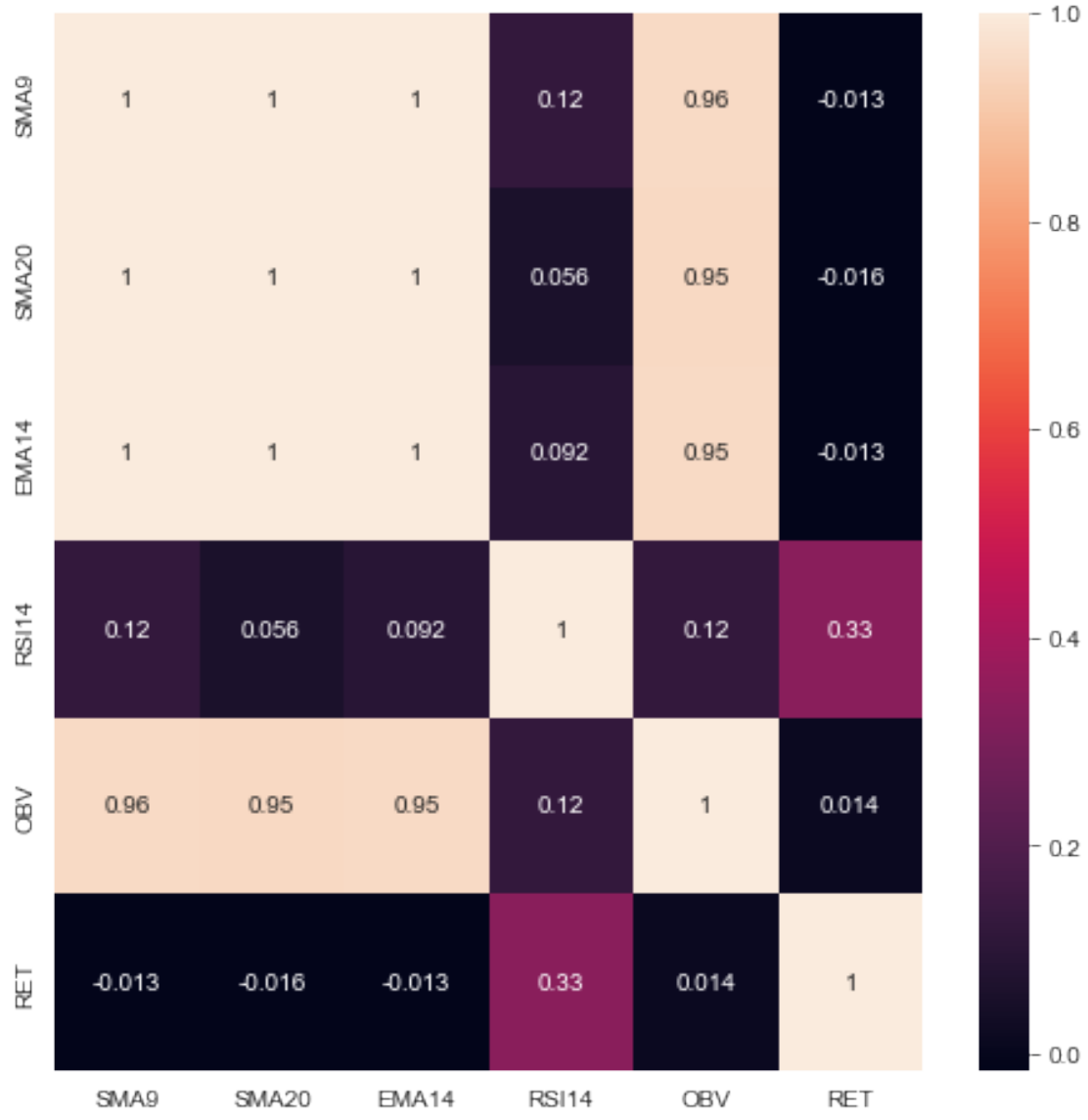
[106]:
```
vale_df_norm.corr()
```

[106]:

	SMA9	SMA20	EMA14	RSI14	OBV	RET
SMA9	1.000000	0.996162	0.999145	0.120603	0.955329	-0.012841
SMA20	0.996162	1.000000	0.998856	0.056181	0.951899	-0.016099

```
EMA14  0.999145  0.998856  1.000000  0.092068  0.954929 -0.012947
RSI14  0.120603  0.056181  0.092068  1.000000  0.119380  0.333849
OBV    0.955329  0.951899  0.954929  0.119380  1.000000  0.014057
RET   -0.012841 -0.016099 -0.012947  0.333849  0.014057  1.000000
```

[107]:
```
plt.figure(figsize=(8,8))
sns.heatmap(vale_df_norm.corr(), annot=True);
```



[108]:
```
vale_df_norm.cov()
```

[108]:

	SMA9	SMA20	EMA14	RSI14	OBV	RET
SMA9	0.044522	0.044734	0.044704	0.003910	0.045592	-0.000150
SMA20	0.044734	0.045293	0.045076	0.001837	0.045820	-0.000190
EMA14	0.044704	0.045076	0.044963	0.003000	0.045798	-0.000152
RSI14	0.003910	0.001837	0.003000	0.023613	0.004149	0.002848
OBV	0.045592	0.045820	0.045798	0.004149	0.051155	0.000176
RET	-0.000150	-0.000190	-0.000152	0.002848	0.000176	0.003082

6.4 Transformações dos Dados para o Modelo

[109]:
```
# Date preparation
vale_df_norm_dt = vale_df_norm.copy()
vale_df_norm_dt['YYYY'] = vale_df_norm_dt['Date'].dt.year
vale_df_norm_dt['MM'] = vale_df_norm_dt['Date'].dt.month
vale_df_norm_dt['DD'] = vale_df_norm_dt['Date'].dt.day
vale_df_norm_dt = vale_df_norm_dt.drop('Date', axis=1)
vale_df_norm_dt
```

[109]:

	SMA9	SMA20	EMA14	RSI14	OBV	RET	YYYY	MM	DD
1	0.381214	0.406331	0.393848	0.373046	0.359215	0.588219	2010	2	3
2	0.378617	0.404328	0.389597	0.215186	0.356603	0.475829	2010	2	4
3	0.376396	0.401643	0.385406	0.198368	0.354191	0.575050	2010	2	5
4	0.374674	0.399013	0.382205	0.234969	0.357014	0.607927	2010	2	8
5	0.374122	0.396601	0.380697	0.333994	0.359686	0.636641	2010	2	9
…	…	…	…	…	…	…	…	…	…
3117	0.555914	0.567686	0.562358	0.426067	0.588728	0.523992	2022	8	24
3118	0.555174	0.567118	0.563137	0.470491	0.604673	0.633287	2022	8	25
3119	0.555174	0.566001	0.562494	0.438349	0.586458	0.560995	2022	8	26
3120	0.554084	0.564711	0.560266	0.399365	0.572320	0.551722	2022	8	29
3121	0.551406	0.563298	0.555877	0.347067	0.572320	0.530925	2022	8	30

[3121 rows x 9 columns]

[110]:
```
# Labels are the values we want to predict
labels = np.array(vale_df_norm_dt['RET'])
# Remove the labels from the features
# axis 1 refers to the columns
#features= vale_df_norm.drop(labels=['Date','RET'], axis = 1)
features= vale_df_norm_dt.drop(labels=['RET'], axis = 1)
# Saving feature names for later use
feature_list = list(features.columns)
# Convert to numpy array
features = np.array(features)
```

6.5 Seleção de Bases de Treino e Teste

[111]:
```
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
train_features, test_features, train_labels, test_labels =␣
 ↪train_test_split(features, labels, test_size = 0.33, random_state=42)

print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

```
Training Features Shape: (2091, 8)
Training Labels Shape: (2091,)
Testing Features Shape: (1030, 8)
Testing Labels Shape: (1030,)
```

[112]:
```
# The baseline predictions are the historical SMA9
baseline_preds = test_features[:, feature_list.index('SMA9')]
# Baseline errors, and display average baseline error
baseline_errors = abs(baseline_preds - test_labels)
print('Average baseline error: ', round(np.mean(baseline_errors), 2), 'degrees.
 ↪')
```

```
Average baseline error:  0.3 degrees.
```

6.6 Criação do Modelo Random Forest

Random Forest é um algoritmo de aprendizado de máquina comumente usado, marca registrada de Leo Breiman e Adele Cutler, que combina a saída de várias árvores de decisão para alcançar um único resultado. Sua facilidade de uso e flexibilidade impulsionaram sua adoção, pois lida com problemas de classificação e regressão.

[113]:
```
# Import the model we are using
from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(train_features[:,:-3], train_labels);
```

[114]:
```
# Use the forest's predict method on the test data
predictions = rf.predict(test_features[:,:-3])
# Calculate the absolute errors
errors = abs(predictions - test_labels)
```

6.6.1 Métricas do Modelo

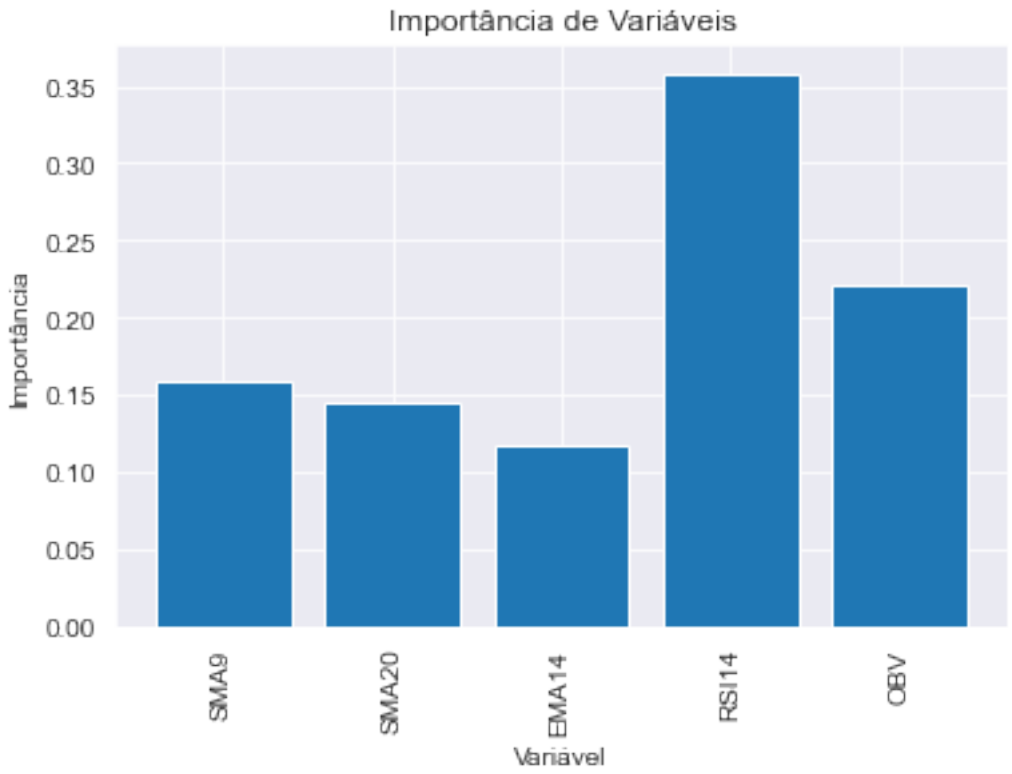
[115]:
```
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(test_labels,␣
 ↪predictions))
print('Mean Squared Error:', metrics.mean_squared_error(test_labels,␣
 ↪predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.
 ↪mean_squared_error(test_labels, predictions)))
```

```
Mean Absolute Error: 0.03836361970747581
Mean Squared Error: 0.003179022645190427
Root Mean Squared Error: 0.056382822252796345
```

[116]:
```
# Get numerical feature importances
importances = list(rf.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance␣
 ↪in zip(feature_list, importances)]
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse␣
 ↪= True)
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in␣
 ↪feature_importances];
```

```
Variable: RSI14                Importance: 0.36
Variable: OBV                  Importance: 0.22
Variable: SMA9                 Importance: 0.16
Variable: SMA20                Importance: 0.15
Variable: EMA14                Importance: 0.12
```

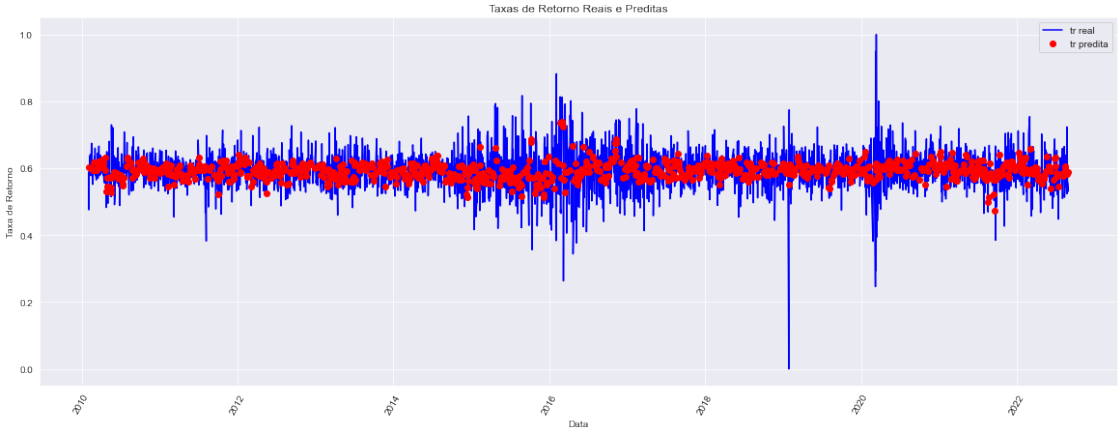
[117]:
```
# list of x locations for plotting
x_values = list(range(len(importances)))
# Make a bar chart
plt.bar(x_values, importances, orientation = 'vertical')
# Tick labels for x axis
plt.xticks(x_values, feature_list[:-3], rotation='vertical')
# Axis labels and title
plt.ylabel('Importância'); plt.xlabel('Variável'); plt.title('Importância de␣
 ↪Variáveis');
```



6.7 Forecast - Predição das Taxas de Retorno de VALE3

[118]:
```
# Use datetime for creating date objects for plotting
import datetime
# Dates of training values
months = features[:, feature_list.index('MM')]
days = features[:, feature_list.index('DD')]
years = features[:, feature_list.index('YYYY')]
# List and then convert to datetime object
dates = [str(int(year)) + '-' + str(int(month)) + '-' + str(int(day)) for year,␣
 ↪month, day in zip(years, months, days)]
dates = [datetime.datetime.strptime(date, '%Y-%m-%d') for date in dates]
# Dataframe with true values and dates
true_data = pd.DataFrame(data = {'date': dates, 'actual': labels})
# Dates of predictions
months = test_features[:, feature_list.index('MM')]
days = test_features[:, feature_list.index('DD')]
years = test_features[:, feature_list.index('YYYY')]
# Column of dates
test_dates = [str(int(year)) + '-' + str(int(month)) + '-' + str(int(day)) for␣
 ↪year, month, day in zip(years, months, days)]
```

```
# Convert to datetime objects
test_dates = [datetime.datetime.strptime(date, '%Y-%m-%d') for date in␣
 ↪test_dates]
# Dataframe with predictions and dates
predictions_data = pd.DataFrame(data = {'date': test_dates, 'prediction':␣
 ↪predictions})
# Plot the actual values
plt.figure(figsize=(20, 7));
plt.plot(true_data['date'], true_data['actual'], 'b-', label = 'tr real')
# Plot the predicted values
plt.plot(predictions_data['date'], predictions_data['prediction'], 'ro', label␣
 ↪= 'tr predita')
plt.xticks(rotation = '60');
plt.legend()
# Graph labels
plt.xlabel('Data'); plt.ylabel('Taxa de Retorno'); plt.title('Taxas de Retorno␣
 ↪Reais e Preditas');
```

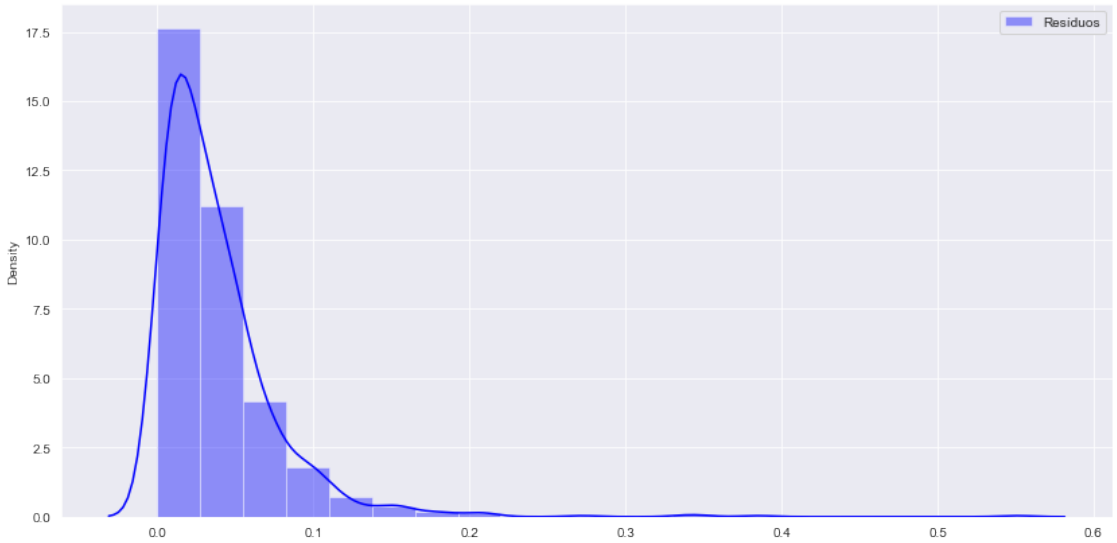


[119]:
```
# Make the data accessible for plotting
true_data['SMA9'] = features[:, feature_list.index('SMA9')]
true_data['SMA20'] = features[:, feature_list.index('SMA20')]
true_data['EMA14'] = features[:, feature_list.index('EMA14')]
true_data['RSI14'] = features[:, feature_list.index('RSI14')]
true_data['OBV'] = features[:, feature_list.index('OBV')]
# Plot all the data as lines
plt.figure(figsize=(20, 7));
plt.plot(true_data['date'], true_data['actual'], 'b-', label  = 'actual', alpha␣
 ↪= 1.0)
plt.plot(predictions_data['date'], predictions_data['prediction'], 'ro', label␣
 ↪= 'prediction', alpha = 1.0)
```

```
plt.plot(true_data['date'], true_data['SMA9'], 'r-', label  = 'SMA9', alpha = 0.
↪3)
plt.plot(true_data['date'], true_data['SMA20'], 'k-', label = 'SMA20', alpha =␣
↪0.3)
plt.plot(true_data['date'], true_data['EMA14'], 'r-', label = 'EMA14', alpha =␣
↪0.3)
plt.plot(true_data['date'], true_data['RSI14'], 'y-', label = 'RSI14', alpha =␣
↪0.3)
plt.plot(true_data['date'], true_data['OBV'], 'k-', label = 'OBV', alpha = 0.3)
# Formatting plot
plt.legend(); plt.xticks(rotation = '60');
# Lables and title
plt.xlabel('Data'); plt.ylabel('Valor'); plt.title('Taxa de Retorno e Variáveis␣
↪Importantes');
```

6.8 Análise dos Resíduos

[128]:
```
fig, ax = plt.subplots(figsize=(14,7))
ax = sns.distplot(errors, bins=20, label='Resíduos', color = 'blue')
plt.legend();
```



6.8.1 Durbin-Watson

- Se Durbin–Watson for inferior a 1,0, pode haver motivo para preocupação.
- Pequenos valores de d indicam que termos de erro sucessivos estão positivamente correlacionados.
- Se d $>$ 2, os termos de erro sucessivos são negativamente correlacionados.

[129]:
```
sm.stats.durbin_watson(errors)
```

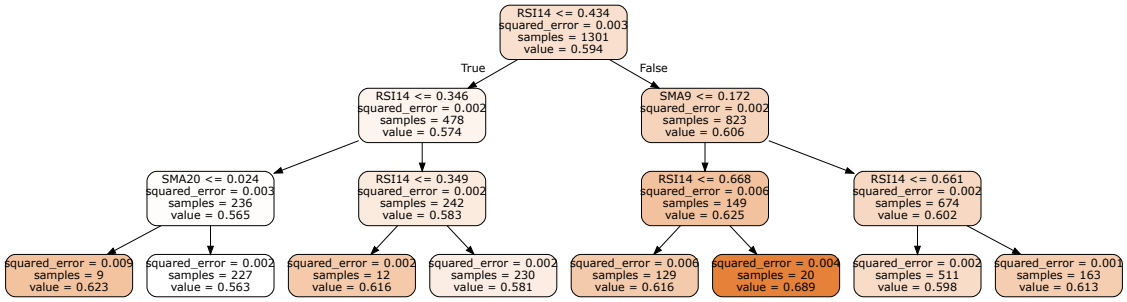
[129]:
```
1.0924090737476402
```

6.9 Árvore de Decisão Reduzida

De forma a tornar mais fácil a viasualização de uma árvore de decisão gerada pelo algoritmo Random Forest, criamos um novo modelo simplificado, de profundidade 3 e a seguir podemos visualizar a árvore gerada.

[130]:
```
# Limit depth of tree to 3 levels
rf_small = RandomForestRegressor(n_estimators=10, max_depth = 3)
rf_small.fit(train_features[:,:-3], train_labels)
dot_data = export_graphviz(rf_small.estimators_[5],
                           feature_names=['SMA9','SMA20','EMA14','RSI14','OBV'],
                           filled=True, impurity=True,
                           rounded=True)

graph = graphviz.Source(dot_data, format='png')
graph
```

[130]:



7 Comparação Entre Modelos Preditivos

[131]:
```
print('==============    Linear Regression     ===================')
print('Modelo: TR[VALE3] ~ TR[BOVA11]')
print('Mean Absolute Error:', metrics.mean_absolute_error(retornos_teste['VALE3.
 ↪SA'], retornos_teste['VALE3.SA P']))
print('Mean Squared Error:', metrics.mean_squared_error(retornos_teste['VALE3.
 ↪SA'], retornos_teste['VALE3.SA P']))
print('Root Mean Squared Error:', np.sqrt(metrics.
 ↪mean_squared_error(retornos_teste['VALE3.SA'], retornos_teste['VALE3.SA␣
 ↪P'])))
print()
print('=============  Random Forest Regressor   =================')
print('Modelo: TR[VALE3] ~ SMA9[VALE3], SMA20[VALE3], EMA14[VALE3],␣
 ↪RSI14[VALE3], OBV[VALE3]')
print('Mean Absolute Error:', metrics.mean_absolute_error(test_labels,␣
 ↪predictions))
print('Mean Squared Error:', metrics.mean_squared_error(test_labels,␣
 ↪predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.
 ↪mean_squared_error(test_labels, predictions)))
```

```
==============    Linear Regression     ===================
Modelo: TR[VALE3] ~ TR[BOVA11]
Mean Absolute Error: 0.015296046005004869
Mean Squared Error: 0.0004626643241046447 6
Root Mean Squared Error: 0.02150963328614983

=============  Random Forest Regressor   =================
Modelo: TR[VALE3] ~ SMA9[VALE3], SMA20[VALE3], EMA14[VALE3], RSI14[VALE3],
OBV[VALE3]
Mean Absolute Error: 0.03836361970747581
Mean Squared Error: 0.003179022645190427
Root Mean Squared Error: 0.056382822252796345
```

8 Conclusão

Neste trabalho foram executadas diversas técnicas de análise estatística, tendo como tema o mercado financeiro brasileiro. Foram realizadas coletas automatizadas de dados históricos do mercado financeiro, provenientes de fonte pública na web utilizando-se de recursos do pacote Pandas no Python.

Em seguida foram montados e analisados alguns portfólios simulados de ações, onde buscamos a maximização do índice sharpe para um portfólio otimizado. Foram realizadas análises visuais e descritivas desses portfólios por meio de indicadores como taxa de retorno e risco.

Após a análise de portfólios foi realizado um passo de análise de variância (ANOVA) e testes de hipóteses, onde pudemos perceber que nem sempre a simples inspeção visual é confiável quando assumimos a normalidade em uma distribuição.

Em seguida foi elaborado um modelo preditivo simplificado, utilizando Regressão Linear, de forma a prever a taxa de retorno de um papel (VALE3), a partir da variação da taxa de retorno de um índice (BOVA11). Neste ponto utilizamos técnicas de análise de modelos de regressão, como testes de normalidade dos resíduos e variância.

Por fim, foi elaborado um novo modelo preditivo, utilizando-se de técnica de Machine Learning chamada árvores de decisão, com apoio do algoritmo Random Forest. Neste experimento realizamos a predição das taxas de retorno de um papel (VALE3), a partir de alguns indicadores de análise técnica comumente utilizados no mercado financeiro. De forma a exemplificar o funcionamento de uma árvore de decisão, geramos também um modelo simplificado, demonstrando visualmente o resultado do algoritmo utilizado.

De forma a analisar a performance de ambos os modelos, trouxemos ao término desse trabalho os indicadores Mean Absolute Error, Mean Squared Error e Root Mean Squared Error para os dois cenários, onde concluímos a partir das análises realizadas, que o modelo preditivo da taxa de retorno elaborado com regressão linear, com base na taxa de retorno de um índice, tem menor erro. Todavia, vale ressaltar que este cenário é inviável no mundo real, uma vez que a variável predita é parte do próprio índice, influenciando nos seus preços e tendo a mesma temporalidade de aquisição de dados e fechamento.