

```
!pip install pandas
!pip install tqdm
!pip install geopandas
!pip install autorank
!pip install scikit-learn==0.24.2
```

```
↳ Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (1.1.5)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (4.41.1)
Requirement already satisfied: geopandas in /usr/local/lib/python3.7/dist-packages (0.9
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyproj>=2.2.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: fiona>=1.8 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: shapely>=1.6 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: munch in /usr/local/lib/python3.7/dist-packages (from fi
Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: six>=1.7 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: click-plugins>=1.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: attrs>=17 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: autorank in /usr/local/lib/python3.7/dist-packages (1.1.
Requirement already satisfied: pandas>=0.25.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scipy>=1.3.0 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: matplotlib>=3.1.3 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from au
Requirement already satisfied: statsmodels>=0.10.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: baycomp>=1.0.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pyparsing!=2.0.4,!>=2.1.2,!>=2.1.6,>=2.0.1 in /usr/local/l
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from cycl
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: scikit-learn==0.24.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-packages
```

```
%matplotlib inline

import pandas as pd
import geopandas as gpd
from google.colab import drive
import logging
import matplotlib.pyplot as plt
import numpy as np
import scipy

logging.basicConfig(format='%(asctime)s - %(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S',
                    level=logging.INFO)

#evitar formatação em notação científica
pd.options.display.float_format = ',. 2f' format
```

```
drive.mount('/content/drive')
HOME = '/content/drive/MyDrive/TCC RSM Novo AE RJ'
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount(..., force_remount=True)

▼ Dados

Os dados do Auxílio Emergencial, BPC e Bolsa Família são lidos facilmente pelo pandas, apenas explicitando que o separador utilizado no arquivo CSV é um ";"

```
auxilio_emergencial = pd.read_csv(HOME + '/datasets/auxilio_emergencial.csv', sep=";")
bpc = pd.read_csv(HOME + '/datasets/bpc.csv', sep=";")
# só conseguimos obter os dados de bolsa família até dezembro de 2020
bolsa_familia = pd.read_csv(HOME + '/datasets/bolsa_familia.csv', sep=";")
```

Por outro lado, os dados de FPM estão em arquivos separados para cada mês, tem uma codificação diferente, que precisa ser explicitada ao usar o pandas na leitura dos arquivos. Ainda fazemos outros tratamentos para deixar os dados de FPM compatível com os demais, como:

- Criar um coluna que agregue os decendios como um valor mensal;
- Criar uma coluna Mês/Ano que junte os dados que estão separados aqui, mas que nos demais dados estão juntos;
- Colocar os nomes dos municípios em caixa alta e alterar a grafia de alguns municípios que não bate com os dados dos outros repasses.

```
from os import listdir
fpm_home = HOME + '/datasets/fpm'
fpm = None
# os dados dos fpm estão em arquivos separados
# então é necessário juntar todos num dataframe só
for f in listdir(fpm_home):
    # há uma dificuldade pelo encoding diferente desse dado
    df = pd.read_csv(fpm_home + '/' + f, sep=';', encoding = "ISO-8859-1")
    # como há dados de todos os estados e de outras transferencias, é preciso filtrar para manter
    fpm_data = df[ (df.UF == 'RJ') & (df['Transferência'] == 'FPM') ]
    try:
        fpm = pd.concat([fpm , fpm_data])
    except:
        fpm = fpm_data
# todos os outros dados tem a coluna Mês/Ano só esse tem os dados separados
# além disso, para deixar mais fácil o merge com os outros dados foi colocado o mês com 2 dígitos
fpm['Mês/Ano'] = fpm.apply (lambda row: (str(row['Mês'])) if row['Mês'] >= 10 else '0' + str(row['Mês']))
fpm['Soma Decêndio'] = fpm.apply (lambda row: row['1º Decêndio'] + row['2º Decêndio'] + row['3º Decêndio'])
# foi realizado upcase pra conseguir fazer o merge com os outros dados
fpm['Município'] = fpm['Município'].str.upper()
# esses municípios só tem o nome errado pra essa base
fpm['Município'] = fpm['Município'].replace(['PARATI'], 'PARATY')
fpm['Município'] = fpm['Município'].replace(['ARMAÇÃO DE BÚZIOS'], 'ARMAÇÃO DOS BÚZIOS')
fpm = fpm.reset_index(drop=True)
```

▼ Dados IBGE

Temos também os dados do IBGE. Os dados extraídos originalmente do site possuam uma linha no inicio e no final do arquivo que não permitia que o dados fossem lido pelo pandas, então removemos manualmente essas duas linhas para ser possível ler com o pandas

```
dados_ibge = pd.read_csv(HOME + '/datasets/dados_ibge_rj.csv', sep=",")
dados_ibge
```

	Município	Código	Genre	Prefeito	Área Territorial	População
	[-]	[-]	[-]	[2021]	- km² [2020]	[2020]
0	Angra dos Reis	3300100	angrense	FERNANDO ANTÔNIO CECILIANO JORDÃO	813.420	
1	Aperibe;	3300159	aperibeense	RONALD DE CARLOS DAIBES MOREIRA	94.542	
2	Araruama	3300209	araruamense	LÍVIA SOARES BELLO DA SILVA	638.276	
3	Areal	3300225	arealense	JOSÉ AUGUSTO BERNARDES LIMA	110.724	
4	Armação dos Búzios	3300233	buziano	ALEXANDRE DE OLIVEIRA MARTINS	70.977	
...
87	Três Rios	3306008	trirriense	JOACIR BARBAGLIO PEREIRA	322.843	
88	Valençada	3306107	valenciano	LUIZ FERNANDO FURTADO DA GRAÇA	1.300.767	
89	Varre-Sai	3306156	varresaiense	SILVESTRE JOSE GORINI	201.938	
90	Vassouras	3306206	vassourense	SEVERINO ANANIAS DIAS FILHO	536.073	
91	Volta Redonda	3306305	volta-redondense	ANTONIO FRANCISCO NETO	182.105	

92 rows × 13 columns

Podemos notar que os dados possuem uma codificação estranha, mas dessa vez não conseguimos passar um parametro na leitura do arquivo pelo pandas, permitindo a leitura correta dos dados. Precisamos fazer o

parser das strings HTML.

```
import io
import html

with open(HOME + '/datasets/dados_ibge_rj.csv') as f:
    s = f.read()
f = io.StringIO(html.unescape(s))
dados_ibge = pd.read_csv(f)
dados_ibge
```

Município	Código	Gentílico	Prefeito	Territorial	Área estimada	População demográfica	Densidade km^{-2}	Esco
-----------	--------	-----------	----------	-------------	---------------	-----------------------	---------------------------------------	------

Ainda notamos alguns caracteres estranhos nos nomes das colunas e o nome dos municípios em camelcase. Vamos fazer alguns pré-processamentos

```
dados_ibge.rename(columns=lambda x: x.replace('[-]', '').replace('<span>', '-').replace('</span>', ''))  
dados_ibge['Município'] = dados_ibge['Município'].str.upper()  
dados_ibge
```

Área População Densidade ...

▼ Dados PIB 2018

```
pib_2018 = pd.read_csv(HOME + '/datasets/pib_rj_2018.csv', sep=";")
pib_2018['Município'] = pib_2018['Município'].str.upper()
pib_2018['Município'] = pib_2018['Município'].str.replace(' \(\RJ\) ', '')
pib_2018
```

Município PIB (2018)		
0	ANGRA DOS REIS	8936325
1	APERIBÉ	181448
2	ARARUAMA	2671410
3	AREAL	314273
4	ARMAÇÃO DOS BÚZIOS	2550287
...
87	TRÊS RIOS	3956601
88	VALENÇA	2564836
89	VARRE-SAI	187033
90	VASSOURAS	977186
91	VOLTA REDONDA	13764889

92 rows × 2 columns

FERNANDO

▼ Dados de ICMS de 2020 e 2019

Os dados extraídos originalmente passaram primeiro por um processamento manual, onde células contendo informações de título como 'Governo do Estado do Rio de Janeiro', e a linha que separava os dados por região foram removidas. O arquivo resultante é o lido aqui como um csv.

```
icms_2020 = pd.read_csv(HOME + '/datasets/icms_2020_modificado.csv', sep=';', encoding = "ISO-8859-1")
```

	Município	Janeiro	Fevereiro	Março	Abril	Ma
0	ARARUAMA	R\$ 2.063.886,94	R\$ 1.556.280,63	R\$ 1.735.636,50	R\$ 1.333.368,64	R\$ 1.849.151
1	ARMAÇÃO DOS BÚZIOS	R\$ 650.436,92	R\$ 677.400,24	R\$ 569.249,91	R\$ 281.072,82	R\$ 327.138
2	ARRAIAL DO CABO	R\$ 278.131,00	R\$ 270.669,61	R\$ 231.316,25	R\$ 172.449,53	R\$ 147.884
3	CABO FRIO	R\$ 5.265.748,44	R\$ 4.011.183,02	R\$ 3.348.951,56	R\$ 1.928.653,65	R\$ 3.205.321
4	CACHOEIRAS DE MACACU	R\$ 2.213.240,74	R\$ 3.585.562,73	R\$ 8.288.873,45	R\$ 835.552,95	R\$ 2.513.105
...
89	SUMIDOURO	R\$ 374.001,00	R\$ 400.510,70	R\$ 377.051,57	R\$ 411.000,05	R\$ 355.010

```
icms_2019 = pd.read_csv(HOME+='/datasets/icms_2019_modificado.csv', sep=';', encoding = "ISO-8859-1")
```

	Município	Janeiro	Fevereiro	Março	Abril	Ma
0	ARARUAMA	R\$ 1.680.762,44	R\$ 1.234.234,95	R\$ 1.251.259,27	R\$ 1.502.526,13	R\$ 1.472.140
1	ARMAÇÃO DOS BÚZIOS	R\$ 588.880,35	R\$ 703.604,50	R\$ 377.881,90	R\$ 519.919,50	R\$ 395.434
2	ARRAIAL DO CABO	R\$ 98.478,26	R\$ 140.222,68	R\$ 77.438,63	R\$ 174.613,73	R\$ 156.554
3	CABO FRIO	R\$ 4.849.423,30	R\$ 3.526.584,64	R\$ 2.778.119,67	R\$ 3.166.833,00	R\$ 3.057.977
4	CACHOEIRAS DE MACACU	R\$ 3.194.763,44	R\$ 3.169.333,01	R\$ 5.603.932,14	R\$ 3.654.732,04	R\$ 3.337.154
...
89	SUMIDOURO	R\$ 330.593,42	R\$ 408.716,64	R\$ 501.376,24	R\$ 432.360,50	R\$ 508.604
90	TERESÓPOLIS	R\$ 11.475.239,17	R\$ 8.861.268,62	R\$ 8.729.071,25	R\$ 10.423.800,35	R\$ 8.438.394
91	TRAJANO DE MORAES	R\$ 16.018,83	R\$ 10.822,55	R\$ 15.843,94	R\$ 14.993,46	R\$ 1.935
92	OUTROS	R\$ 551.188.371,78	R\$ 510.120.239,21	R\$ 487.078.133,18	R\$ 494.594.197,20	R\$ 510.985.976
93	Total Mês	R\$ 3.534.884.547,15	R\$ 3.143.835.402,09	R\$ 2.718.260.370,89	R\$ 3.143.630.312,47	R\$ 2.938.160.845

94 rows x 14 columns

▼ Concatenando dados de repasse

```

dados = bpc[['Município', 'Mês/Ano']]
dados['BPC'] = bpc[['Valor Transferido']]

tmp = bolsa_familia[['Município', 'Mês/Ano']]
tmp['BOLSA FAMILIA'] = bolsa_familia[['Valor Transferido']]
dados = pd.merge(dados, tmp, on=['Município', 'Mês/Ano'])

tmp = auxilio_emergencial[['Município', 'Mês/Ano']]
tmp['auxilio emergencial'] = auxilio_emergencial[['Valor Transferido']]
dados = pd.merge(dados, tmp, on=['Município', 'Mês/Ano'])

```

```

tmp[ AUXILIO_EMERGENCIAL ] = auxilio_emergencial[ valor_transferido ]
dados = pd.merge(dados, tmp, on=['Município', 'Mês/Ano'])

tmp = fpm[['Município', 'Mês/Ano']]
tmp['FPM'] = fpm[['Soma Decêndio']]
dados = pd.merge(dados, tmp, on=['Município', 'Mês/Ano'])

dados

```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```

if __name__ == '__main__':
    /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:13: SettingWithCopyWarning
    A value is trying to be set on a copy of a slice from a DataFrame.
    Try using .loc[row_indexer,col_indexer] = value instead

```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
del sys.path[0]

	Município	Mês/Ano	BPC	BOLSA FAMILIA	AUXILIO EMERGENCIAL	FPM
0	COMENDADOR LEVY GASPARIAN	12/2020	78.375,00	127396	1040044	914002.63
1	COMENDADOR LEVY GASPARIAN	10/2020	79.420,00	127436	862543	430504.81
2	COMENDADOR LEVY GASPARIAN	11/2020	79.434,18	127437	1120382	570601.49
3	COMENDADOR LEVY GASPARIAN	09/2020	80.465,00	127595	1314776	319936.06
4	COMENDADOR LEVY GASPARIAN	08/2020	81.510,00	128108	1518000	397185.06
...
823	RIO DE JANEIRO	06/2020	117.769.537,63	46919909	740594400	17304695.76
824	RIO DE JANEIRO	09/2020	117.941.013,21	47913112	1149490204	15264153.44
825	RIO DE JANEIRO	10/2020	118.194.948,75	47846856	760531244	20539389.07

Percebemos que as colunas referentes a valores possuem algumas discrepâncias entre os diferentes repasses, vamos uniformizá-los

```

#convertendo todas as colunas relativas a valor para float
dados.BPC = dados.BPC.str.replace('.','').str.replace(',','.').astype(float)
dados['BOLSA FAMILIA'] = dados['BOLSA FAMILIA'].astype(float)
dados['AUXILIO EMERGENCIAL'] = dados['AUXILIO EMERGENCIAL'].astype(float)

```

dados

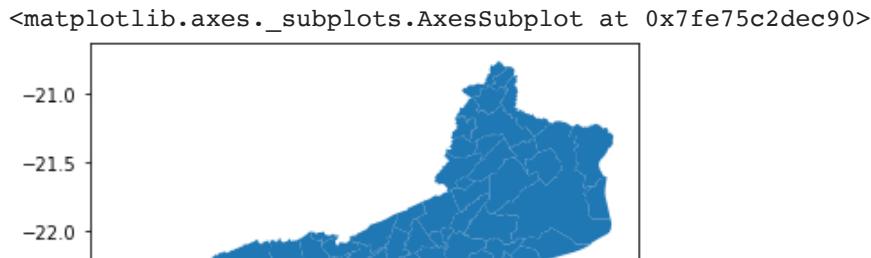
	Município	Mês/Ano	BPC	BOLSA FAMILIA	AUXILIO EMERGENCIAL	FPM
0	COMENDADOR LEVY GASPARIAN	12/2020	78375.00	127396.00	1040044.00	914002.63
1	COMENDADOR LEVY GASPARIAN	10/2020	79420.00	127436.00	862543.00	430504.81
2	COMENDADOR LEVY GASPARIAN	11/2020	79434.18	127437.00	1120382.00	570601.49
3	COMENDADOR LEVY GASPARIAN	09/2020	80465.00	127595.00	1314776.00	319936.06
4	COMENDADOR LEVY GASPARIAN	08/2020	81510.00	128108.00	1518000.00	397185.06
...
823	RIO DE JANEIRO	06/2020	117769537.63	46919909.00	740594400.00	17304695.76
824	RIO DE JANEIRO	09/2020	117941013.21	47913112.00	1149490204.00	15264153.44
825	RIO DE JANEIRO	10/2020	118194948.75	47846856.00	760531244.00	20539389.07

▼ Carregamento de dados geográficos para exibição de mapas

```
# fonte de dados do mapa: https://caelum-online-public.s3.amazonaws.com/985-geopandas/01/Mapa\_rj.shp
mapa_rj = gpd.read_file(HOME+'/Mapas/RJ/33MUE250GC_SIR.shp')
mapa_rj.tail(10)
```

ID	CD_GEOCODM	NM_MUNICIP	geometry
82	1550	SILVA JARDIM	POLYGON ((-42.33565 -22.42905, -42.33522 -22.4...
83	1551	SUMIDOURO	POLYGON ((-42.73429 -22.21886, -42.73415 -22.2...
84	1552	TANGUÁ	POLYGON ((-42.75893 -22.71853, -42.75877 -22.7...
85	1553	TERESÓPOLIS	POLYGON ((-42.76504 -22.12872, -42.76460 -22.1...
86	1554	TRAJANO DE MORAES	POLYGON ((-42.24055 -22.03420, -42.24048 -22.0...
87	1555	TRÊS RIOS	POLYGON ((-43.13666 -22.10933, -43.13182 -22.1...
88	1556	VALENÇA	POLYGON ((-43.82120 -22.08844, -43.82101 -22.0...
89	1557	VARRE-SAI	POLYGON ((-41.82695 -20.84164, -41.82686 -20.8...
90	1558	VASSOURAS	POLYGON ((-43.61208 -22.30841, -43.61113 -22.3...
91	1559	VOLTA REDONDA	POLYGON ((-44.13912 -22.53062, -44.13878 -22.5...

```
mapa_rj.plot()
```



▼ Mais pré-processamento

```
# há uma inconsistÊncia entre os municípios do dados do ibge e do governo
print(set(dados['Município'].unique()) - set(mapa_rj['NM_MUNICIP'].unique()))

print(sorted(dados['Município'].unique()))
print(sorted(mapa_rj['NM_MUNICIP'].unique()))

# descoberta a inconsistencia, foi feita a correção
dados['Município'] = dados['Município'].replace(['TRAJANO DE MORAIS'], 'TRAJANO DE MORAES')

{'TRAJANO DE MORAIS'}
['ANGRA DOS REIS', 'APERIBÉ', 'ARARUAMA', 'AREAL', 'ARMAÇÃO DOS BÚZIOS', 'ARRAIAL DO CAI
['ANGRA DOS REIS', 'APERIBÉ', 'ARARUAMA', 'AREAL', 'ARMAÇÃO DOS BÚZIOS', 'ARRAIAL DO CAI
```

▼ Concatenação de dados de repasse e dados geográficos

```
media_fpm_por_municipio = dados[['Município', 'Mês/Ano', 'FPM']].groupby(['Município']).mean()
media_bpc_por_municipio = dados[['Município', 'Mês/Ano', 'BPC']].groupby(['Município']).mean()
media_bolsa_por_municipio = dados[['Município', 'Mês/Ano', 'BOLSA FAMILIA']].groupby(['Município'])
media_auxilio_por_municipio = dados[['Município', 'Mês/Ano', 'AUXILIO EMERGENCIAL']].groupby(['Município'])

mapa_rj = mapa_rj.merge(media_fpm_por_municipio, left_on='NM_MUNICIP', right_on='Município')
mapa_rj = mapa_rj.merge(media_bpc_por_municipio, left_on='NM_MUNICIP', right_on='Município')
mapa_rj = mapa_rj.merge(media_bolsa_por_municipio, left_on='NM_MUNICIP', right_on='Município')
mapa_rj = mapa_rj.merge(media_auxilio_por_municipio, left_on='NM_MUNICIP', right_on='Município')
mapa_rj = mapa_rj.merge(dados_ibge, left_on='NM_MUNICIP', right_on='Município')

mapa_rj
```

ID	CD_GEOCODM	NM_MUNICIP	geometry	FPM	BPC	BOLSA FAMILIA	AUX_EMERGENC	
0	1468	3300100	ANGRA DOS REIS	MULTIPOINT ((-44.33208 -23.02384, -44.33221...))	4413445.12	3252821.51	2222303.22	2943807
1	1469	3300159	APERIBÉ	POLYGON ((-42.08950 -21.62893, -42.08912 -21.6...))	674895.68	172080.00	125623.22	182734
2	1470	3300209	ARARUAMA	POLYGON ((-42.22526 -22.59253, -42.22521 -22.5...))	3037030.48	3873568.67	2083571.89	2612338
3	1471	3300225	AREAL	POLYGON ((-43.16007 -22.20005, -43.16003 -22.1...))	674895.68	261604.17	170855.56	224346
4	1472	3300233	ARMAÇÃO DOS BÚZIOS	MULTIPOINT ((-41.98427 -22.74458, -41.98296...))	1349791.33	652906.00	306118.78	974496
...	
87	1555	3306008	TRÊS RIOS	POLYGON ((-43.13666 -22.10933, -43.13182 -22.1...))	2362134.82	1242999.83	1190230.22	1269959
88	1556	3306107	VALENÇA	POLYGON ((-43.82120 -22.08844, -43.82101 -22.0...))	2193410.91	1659844.14	1063391.11	1105836

▼ Análise e Exploração dos Dados

-41.82686

▼ Gráfico ICMS x AE

```
90 1558 3306206 VASSOURAS "2020-01-01" 1349791.33 1123990.37 305963.56 559467
```

```
def processa_icms(icms):
    total_icms_mes = icms.iloc[-1:] #ultima linha é a soma total de icms por mês
    total_icms_mes = total_icms_mes.loc[:, total_icms_mes.columns != 'Município'] # a primeira
    total_icms_mes = total_icms_mes.loc[:, total_icms_mes.columns != 'Total Anual'] # a ultima
    total_icms_mes = total_icms_mes.T.rename(columns={93: "Total Arrecadado"}).reset_index().re
    total_icms_mes['Total Arrecadado'] = total_icms_mes['Total Arrecadado'].apply(lambda x: float(x))
    return total_icms_mes
```

```
total_icms_2020_mes = processa_icms(icms_2020)
```

```

ae_mes = auxilio_emergencial.groupby(['Mês/Ano']).sum().reset_index() # agrupo todos os repas
ae_mes = ae_mes[1:] # Descarto o primeiro registro pq é relativo a janeiro de 2021

INT_TO_MES = {4: 'Abril', 5:'Maio', 6: 'Junho', 7:'Julho', 8:'Agosto', 9:'Setembro', 10:'Outubro', 11:'Novembro', 12:'Dezembro'}

ae_mes['Mês'] = ae_mes['Mês/Ano'].apply(lambda x: INT_TO_MES[int(x.split('/')[0])])#crio uma coluna com o mês
ae_mes['Valor Transferido'] = ae_mes['Valor Transferido'].astype(float)
ae_mes

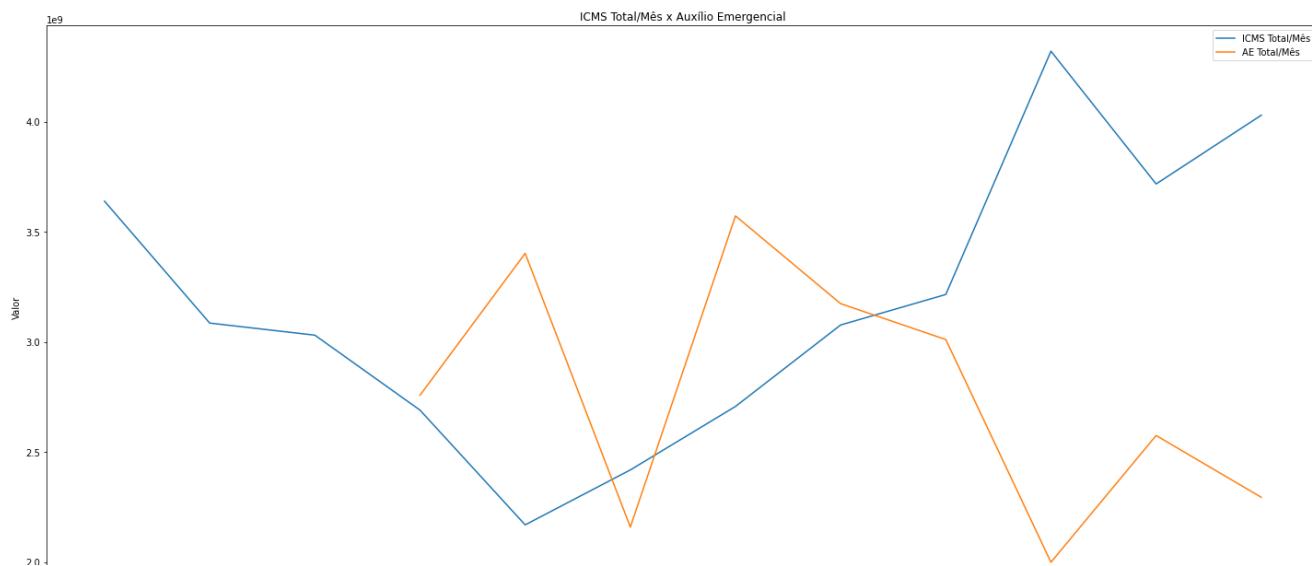
```

	Mês/Ano	Valor Transferido	Mês
1	04/2020	2757376200.00	Abril
2	05/2020	3401640600.00	Maio
3	06/2020	2158593600.00	Junho
4	07/2020	3571626000.00	Julho
5	08/2020	3173173800.00	Agosto
6	09/2020	3010164929.00	Setembro
7	10/2020	1998948825.00	Outubro
8	11/2020	2574479627.00	Novembro
9	12/2020	2293955616.00	Dezembro

```

fig = plt.figure(figsize=(20, 10))
plt.subplot(1, 1, 1)
plt.title('ICMS Total/Mês x Auxílio Emergencial')
plt.plot(total_icms_2020_mes['Mês'], total_icms_2020_mes['Total Arrecadado'],
         label='ICMS Total/Mês')
plt.plot(ae_mes['Mês'], ae_mes['Valor Transferido'],
         label='AE Total/Mês')
plt.legend(loc='upper right')
plt.xlabel('Meses')
plt.xticks(rotation='90')
plt.ylabel('Valor')
fig.tight_layout()
plt.show()

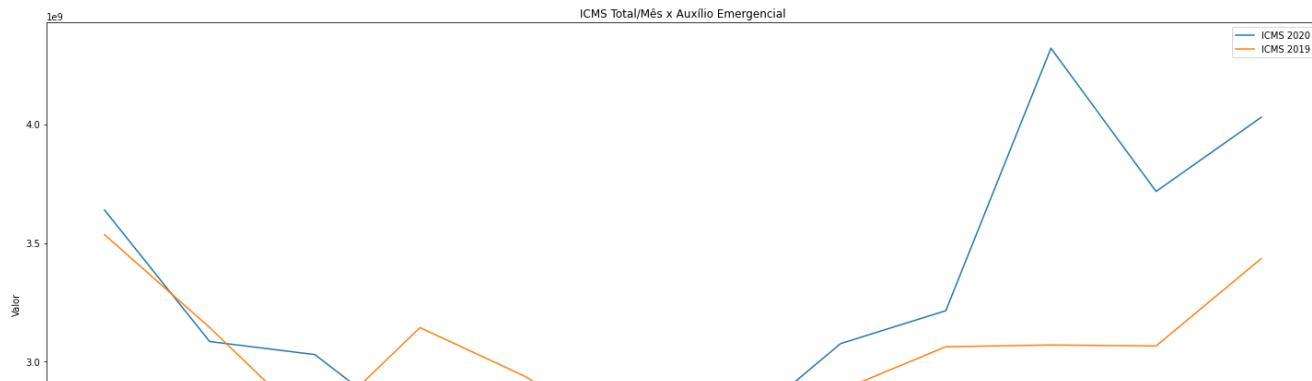
```



▼ Gráfico ICMS 2020 x 2019

```
total_icms_2019_mes = processa_icms(icms_2019)

fig = plt.figure(figsize=(20, 10))
plt.subplot(1, 1, 1)
plt.title('ICMS Total/Mês x Auxílio Emergencial')
plt.plot(total_icms_2020_mes['Mês'], total_icms_2020_mes['Total Arrecadado'],
         label='ICMS 2020')
plt.plot(total_icms_2019_mes['Mês'], total_icms_2019_mes['Total Arrecadado'],
         label='ICMS 2019')
plt.legend(loc='upper right')
plt.xlabel('Meses')
plt.xticks(rotation='90')
plt.ylabel('Valor')
fig.tight_layout()
plt.show()
```



▼ Gráfico IDH x AE e PIB x AE

Ainda nas análises preliminares, não foi possível fazer um gráfico com os três indicadores de uma vez por conta da escala de valores ser muito diferente.

```
ibge_idh = dados_ibge[dados_ibge.filter(regex='Município|IDH').columns] # seleção das colunas
ibge_idh = ibge_idh.rename(columns={"IDHM - Índice de desenvolvimento humano municipal [2010]": "IDHM"})
ibge_idh
```

	Município	IDHM (2010)
0	ANGRA DOS REIS	724
1	APERIBÉ	692
2	ARARUAMA	718
3	AREAL	684
4	ARMAÇÃO DOS BÚZIOS	728
...
87	TRÊS RIOS	725
88	VALENÇA	738
89	VARRE-SAI	659
90	VASSOURAS	714
91	VOLTA REDONDA	771

92 rows x 2 columns

```
ae_total_municipio = auxilio_emergencial[auxilio_emergencial['Mês/Ano'] != '01/2021'][['Município', 'Valor Transferido']]
ae_total_municipio = ae_total_municipio.groupby(['Município']).sum().reset_index()
ae_total_municipio['Valor Transferido'] = ae_total_municipio['Valor Transferido'].astype(float)
ae_total_municipio
```

	Município	Valor Transferido
0	ANGRA DOS REIS	264942690.00
1	APERIBÉ	16446098.00
2	ARARUAMA	235110439.00
3	AREAL	20191463.00
4	ARMAÇÃO DOS BÚZIOS	87704452.00
...

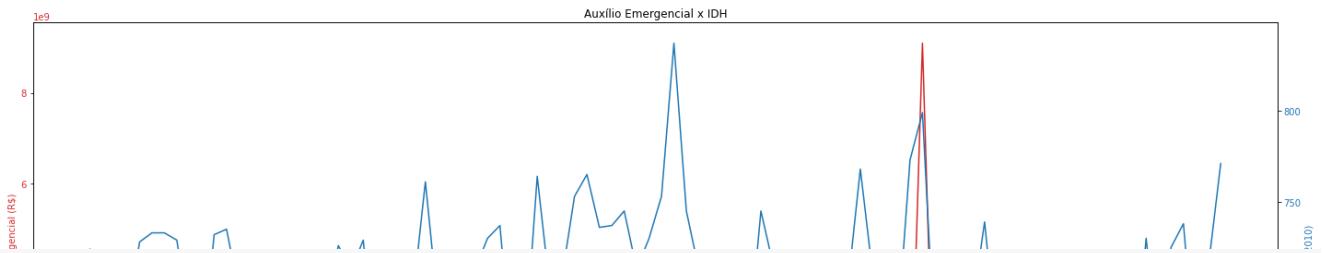
```
fig, ax1 = plt.subplots()
fig.set_size_inches(20, 10)
plt.xticks(rotation='90')

color = 'tab:red'
ax1.set_xlabel('Município')
ax1.set_ylabel('Auxílio Emergencial (R$)', color=color)
ax1.plot(ae_total_municipio['Município'], ae_total_municipio['Valor Transferido'], color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx() # instantiate a second axes that shares the same x-axis

color = 'tab:blue'
ax2.set_ylabel('IDH (2010)', color=color) # we already handled the x-label with ax1
ax2.plot(ibge_idh['Município'], ibge_idh['IDHM (2010)'], color=color)
ax2.tick_params(axis='y', labelcolor=color)

plt.title('Auxílio Emergencial x IDH')
fig.tight_layout() # otherwise the right y-label is slightly clipped
plt.show()
```



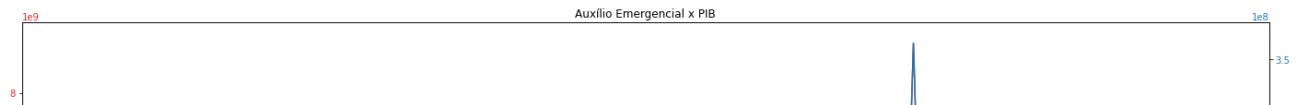
```
fig, ax1 = plt.subplots()
fig.set_size_inches(20, 10)
plt.xticks(rotation='90')

color = 'tab:red'
ax1.set_xlabel('Município')
ax1.set_ylabel('Auxílio Emergencial (R$)', color=color)
ax1.plot(ae_total_municipio['Município'], ae_total_municipio['Valor Transferido'], color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx() # instantiate a second axes that shares the same x-axis

color = 'tab:blue'
ax2.set_ylabel('PIB (2018)', color=color) # we already handled the x-label with ax1
ax2.plot(pib_2018['Município'], pib_2018['PIB (2018)'], color=color)
ax2.tick_params(axis='y', labelcolor=color)

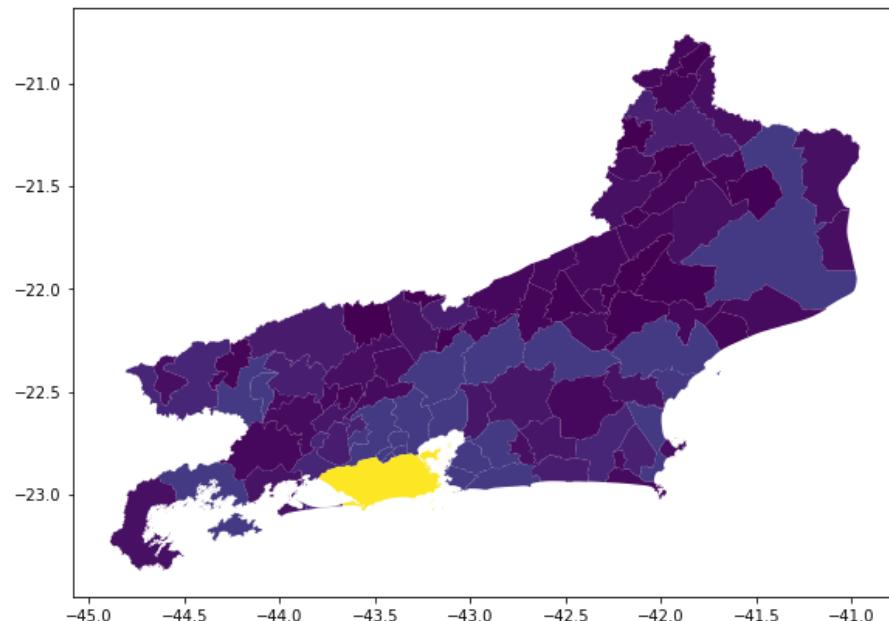
plt.title('Auxílio Emergencial x PIB')
fig.tight_layout() # otherwise the right y-label is slightly clipped
plt.show()
```



▼ Mapas por tipo de repasse

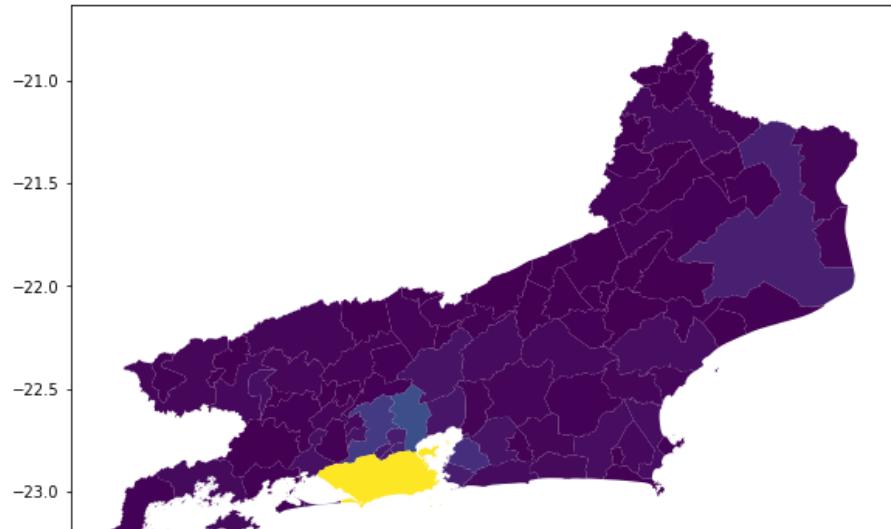
```
mapa_rj.plot(column='FPM', legend=True,
               legend_kwds={'label': "Valor Médio de FPM",
                            'orientation': "horizontal"}, figsize=(15,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe75228f790>
```



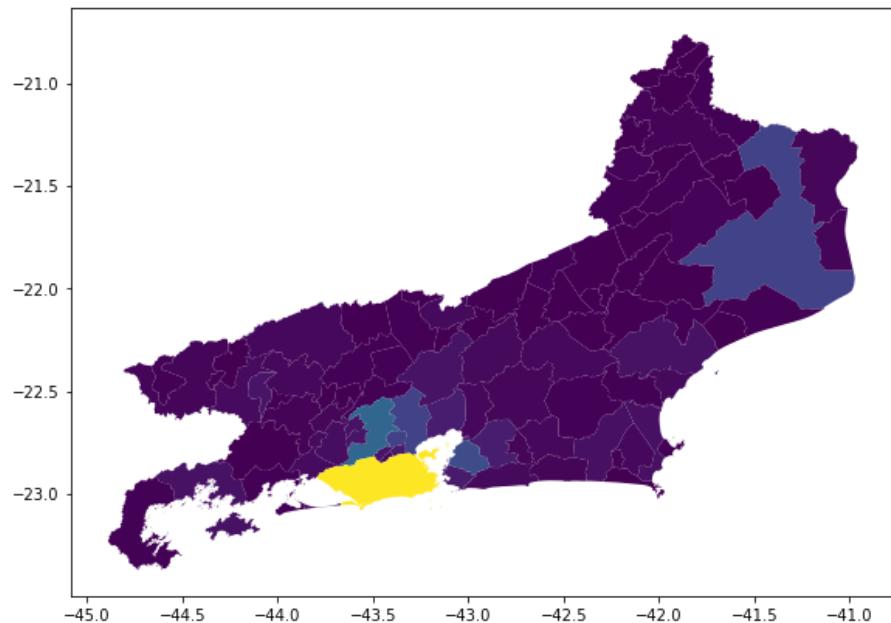
```
mapa_rj.plot(column='BPC', legend=True,
               legend_kwds={'label': "Valor Médio de BPC",
                            'orientation': "horizontal"}, figsize=(15,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe751d7b890>
```



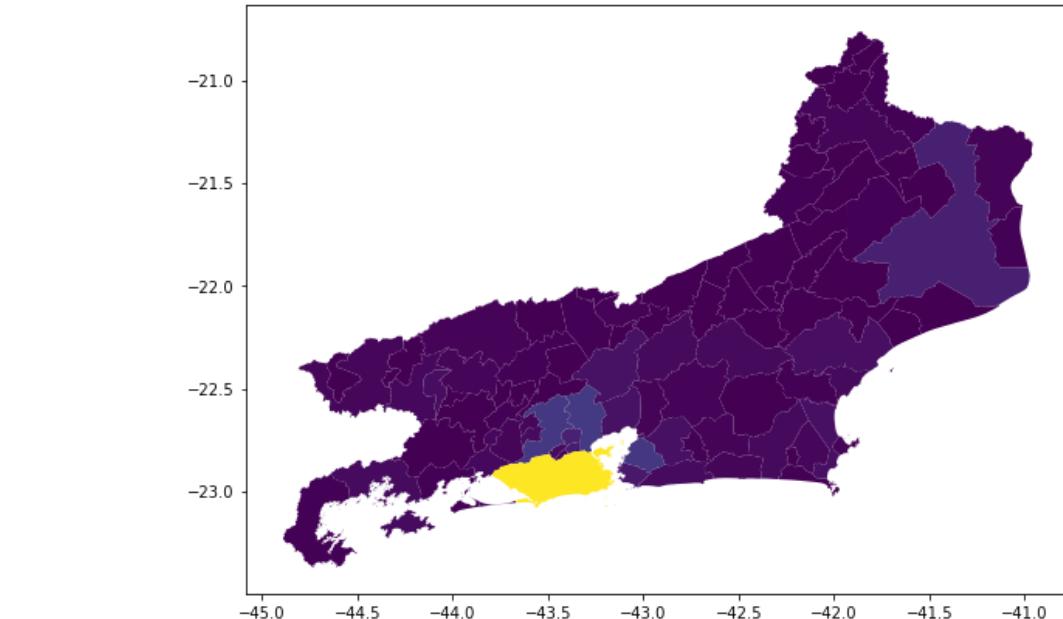
```
mapa_rj.plot(column='BOLSA FAMILIA', legend=True,  
              legend_kwds={'label': "Valor Médio de BOLSA FAMILIA",  
                           'orientation': "horizontal"}, figsize=(15,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe751c2d710>
```



```
mapa_rj.plot(column='AUXILIO EMERGENCIAL', legend=True,  
              legend_kwds={'label': "Valor Médio de AUXILIO EMERGENCIAL",  
                           'orientation': "horizontal"}, figsize=(15,10))
```

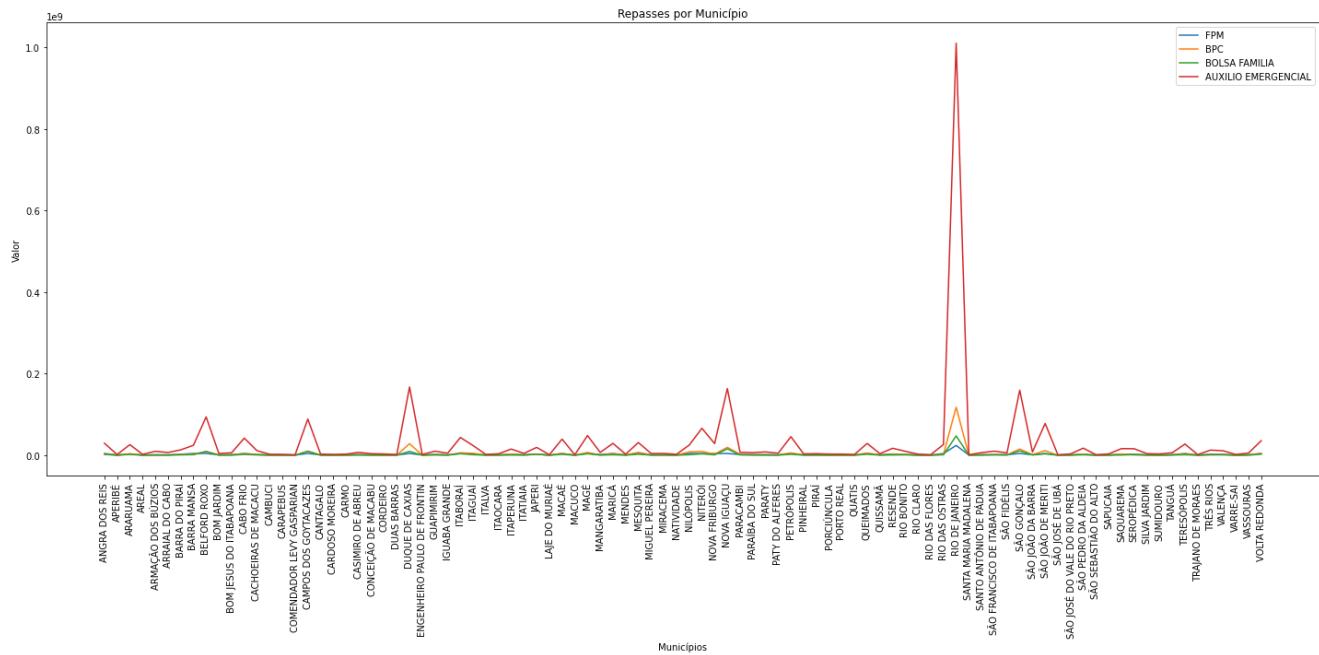
<matplotlib.axes._subplots.AxesSubplot at 0x7fe751cc1490>



▼ Comparação dos valores de diferentes repasses por município

```
fig = plt.figure(figsize=(20, 10))
plt.subplot(1, 1, 1)
plt.title('Repasses por Município')
for repasse in ['FPM', 'BPC', 'BOLSA FAMILIA', 'AUXILIO EMERGENCIAL']:
    dado = mapa_rj[['NM_MUNICIP', repasse]]
    plt.plot(dado['NM_MUNICIP'], dado[repasse],
              label=repasse)

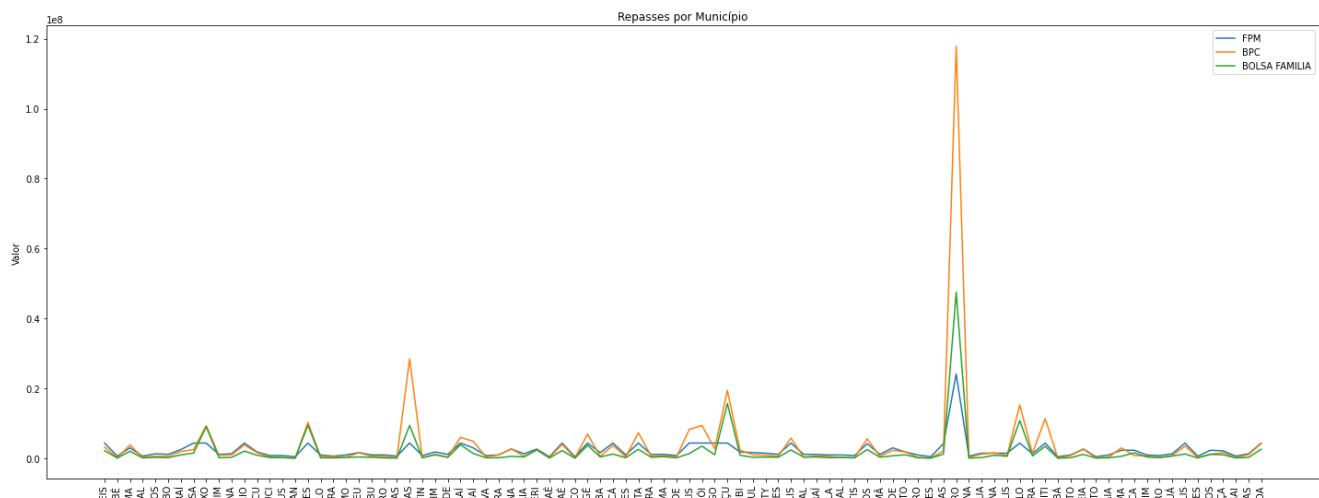
plt.legend(loc='upper right')
plt.xlabel('Municípios')
plt.xticks(rotation='90')
plt.ylabel('Valor')
fig.tight_layout()
plt.show()
```



O Auxílio Emergencial é um valor muito discrepante e dificulta a visualização da relação dos demais benefícios com o FPM. Vamos avaliar plotando sem o Auxílio Emergencial.

```
fig = plt.figure(figsize=(20, 10))
plt.subplot(1, 1, 1)
plt.title('Repasses por Município')
for repasse in ['FPM', 'BPC', 'BOLSA FAMILIA']:
    dado = mapa_rj[['NM_MUNICIP', repasse]]
    plt.plot(dado['NM_MUNICIP'], dado[repasse],
              label=repasse)

plt.legend(loc='upper right')
plt.xlabel('Municípios')
plt.xticks(rotation='90')
plt.ylabel('Valor')
fig.tight_layout()
plt.show()
```



A lista dos municípios que valores de BPC ou BOLSA FAMILIA são maiores que FPM é:

```
mapa_rj.iloc[[i
    for i, row in mapa_rj.iterrows()
    if row['BPC'] > row['FPM'] or
    row['BOLSA FAMILIA'] > row['FPM']
]] .NM_MUNICIP
```

2	ARARUAMA
8	BELFORD ROXO
16	CAMPOS DOS GOYTACAZES
24	DUQUE DE CAXIAS
28	ITABORAÍ
29	ITAGUAÍ
31	ITAOCARA
32	ITAPERUNA
38	MAGÉ
42	MESQUITA
46	NILÓPOLIS
47	NITERÓI
49	NOVA IGUAÇU
50	PARACAMBI
54	PETRÓPOLIS
60	QUEIMADOS
63	RIO BONITO
67	RIO DE JANEIRO
70	SÃO FRANCISCO DE ITABAPOANA
72	SÃO GONÇALO
73	SÃO JOÃO DA BARRA
74	SÃO JOÃO DE MERITI
80	SAQUAREMA

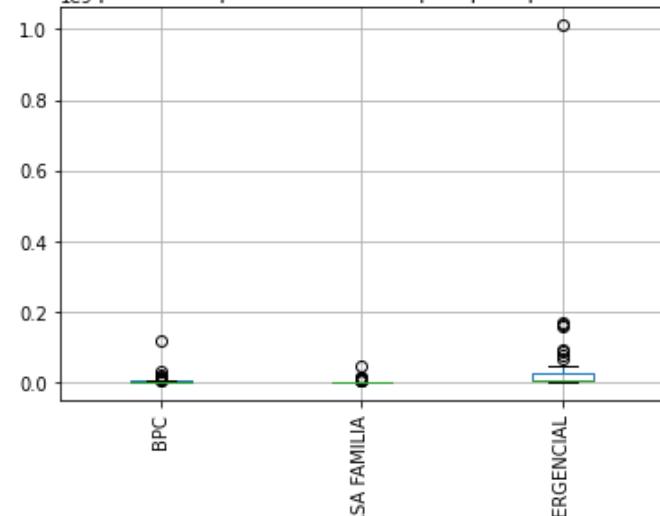
Name: NM_MUNICIP, dtype: object

Boxplot para avaliar a distribuição dos valores médio de repasses entre os municípios

```
ax = mapa_rj.boxplot(column=['BPC', 'BOLSA FAMILIA', 'AUXILIO EMERGENCIAL'])
ax.set_title('Boxplot dos Repasses aos municípios por tipo de auxílio')
plt.xticks(rotation=90)
```

```
(array([1, 2, 3]), <a list of 3 Text major ticklabel objects>)
```

Boxplot dos Repasses aos municípios por tipo de auxílio

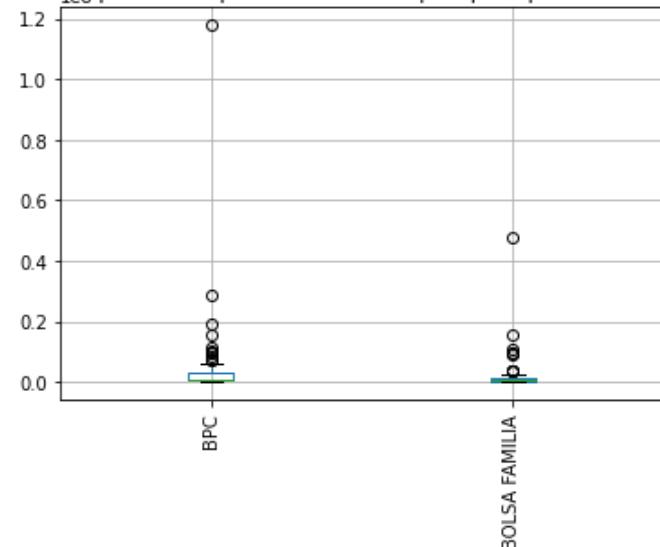


Percebemos que um município recebeu repasses de Auxílio Emergencial muito acima dos demais municípios, além disso, este repasse é maior do que todos os outros. Vamos analisar os boxplots dos repasses, desconsiderando o Auxílio Emergencial.

```
ax = mapa_rj.boxplot(column=['BPC', 'BOLSA FAMILIA'])
ax.set_title('Boxplot dos Repasses aos municípios por tipo de auxílio')
plt.xticks(rotation=90)
```

```
(array([1, 2]), <a list of 2 Text major ticklabel objects>)
```

Boxplot dos Repasses aos municípios por tipo de auxílio

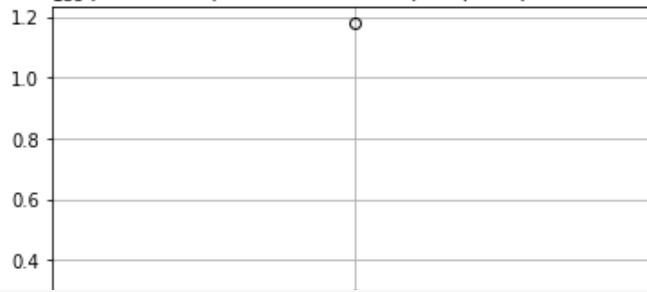


▼ BOXPLOTS INDIVIDUAIS

```
ax = mapa_rj.boxplot(column=['BPC'])
ax.set_title('Boxplot dos Repasses aos municípios por tipo de auxílio')
plt.xticks(rotation=90)
```

```
(array([1]), <a list of 1 Text major ticklabel objects>)
```

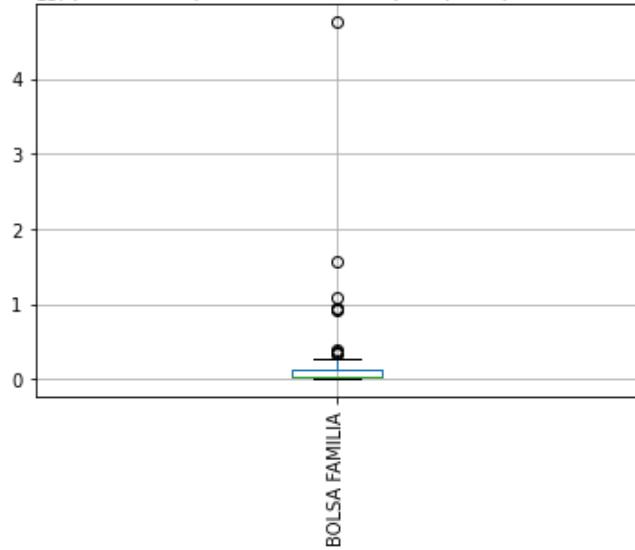
Boxplot dos Repasses aos municípios por tipo de auxílio



```
ax = mapa_rj.boxplot(column=['BOLSA FAMILIA'])  
ax.set_title('Boxplot dos Repasses aos municípios por tipo de auxílio')  
plt.xticks(rotation=90)
```

```
(array([1]), <a list of 1 Text major ticklabel objects>)
```

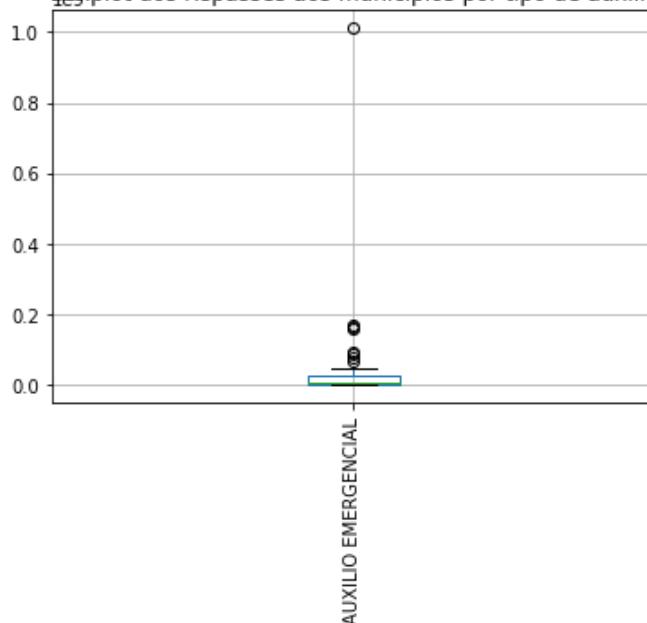
Boxplot dos Repasses aos municípios por tipo de auxílio



```
ax = mapa_rj.boxplot(column=['AUXILIO EMERGENCIAL'])  
ax.set_title('Boxplot dos Repasses aos municípios por tipo de auxílio')  
plt.xticks(rotation=90)
```

```
(array([1]), <a list of 1 Text major ticklabel objects>)
```

Boxplot dos Repasses aos municípios por tipo de auxílio

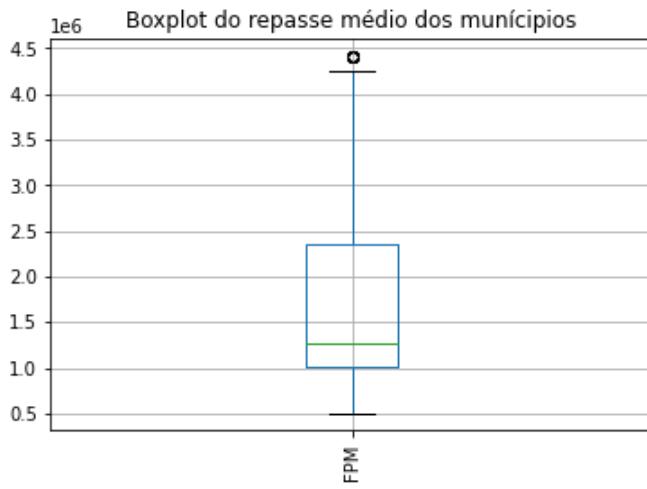


Ainda sim vemos municípios com médias de repasses muito maiores que outros. Vamos plotar boxplots de cada tipo de repasse excluindo os 10 maiores recebedores.

```
from scipy.stats import variation

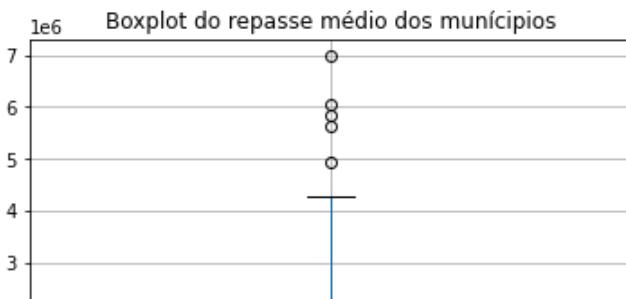
def boxplot_exceto_top_n(df, column, top=10):
    ax = df[[column]].sort_values(column, ascending=False)[top: ].boxplot(column=[column])
    ax.set_title('Boxplot do repasse médio dos municípios')
    plt.xticks(rotation=90)
    plt.show()
    print(df[[column]].describe())
    print('Coeficiente de Variação: ' + str(variation(df[[column]]), axis = 0)[0]))
```

```
boxplot_exceto_top_n(mapa_rj, 'FPM')
```

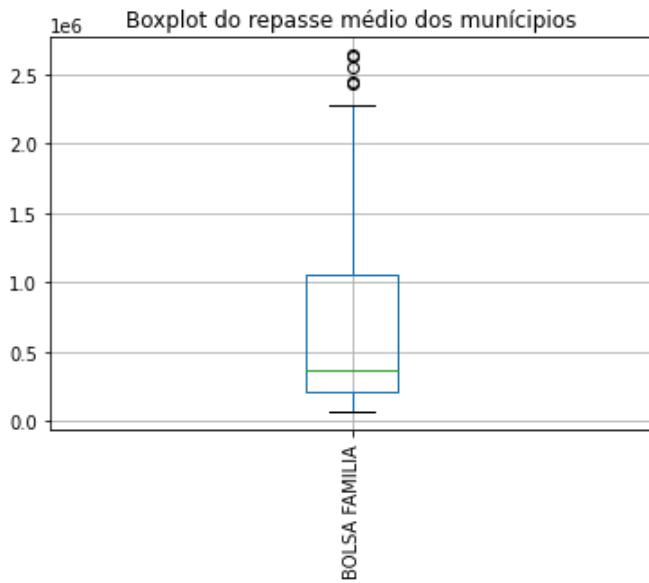


	FPM
count	92.00
mean	2315555.05
std	2713498.20
min	506171.76
25%	1012343.50
50%	1349791.33
75%	3338953.16
max	24149461.22
Coeficiente de Variação:	1.1654702951238058

```
boxplot_exceto_top_n(mapa_rj, 'BPC')
```



```
boxplot_exceto_top_n(mapa_rj, 'BOLSA FAMILIA')
```

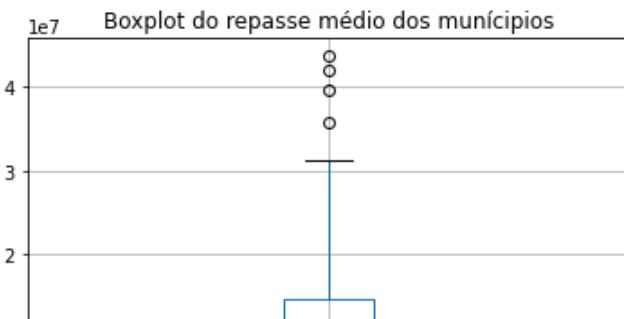


BOLSA FAMILIA

	BOLSA FAMILIA
count	92.00
mean	1905535.30
std	5437716.81
min	69426.78
25%	216876.00
50%	425620.94
75%	1310976.00
max	47533532.78

Coeficiente de Variação: 2.838091346781201

```
boxplot_exceto_top_n(mapa_rj, 'AUXILIO EMERGENCIAL')
```



▼ Quais municípios recebem mais e menos repasses?

↓

A partir dos mapas, podemos inferir que um dos municípios que recebe muito acima da média de repasses é o Rio de Janeiro, mas os demais municípios que recebem repasses acima da média não ficam tão claros. Vamos descobrir quem são.

≡

```
colunas_para_avaliar = ['NM_MUNICIP', 'BPC', 'Área Territorial - km² [2020]', 'População estimada - pessoas [2020]']
mapa_rj[colunas_para_avaliar].sort_values('BPC', ascending=False).head(5)
```

	NM_MUNICIP	BPC	Área Territorial - km² [2020]	População estimada - pessoas [2020]	Densidade demográfica - hab/km² [2010]
67	RIO DE JANEIRO	117852859.67	1.200.329	6747815	5265.82
24	DUQUE DE CAXIAS	28416651.60	467.319	924624	1828.51
49	NOVA IGUACU	19415190.33	520.581	823302	1527.60

```
colunas_para_avaliar = ['NM_MUNICIP', 'FPM', 'Área Territorial - km² [2020]', 'População estimada - pessoas [2020]']
mapa_rj[colunas_para_avaliar].sort_values('FPM', ascending=False).head(5)
```

	NM_MUNICIP	FPM	Área Territorial - km² [2020]	População estimada - pessoas [2020]	Densidade demográfica - hab/km² [2010]
67	RIO DE JANEIRO	24149461.22	1.200.329	6747815	5265.82
42	MESQUITA	4413445.33	41.169	176569	4310.48
46	NILÓPOLIS	4413445.12	19.393	162693	8117.62
22	DUQUE DE CAXIAS	4413445.10	467.319	924624	1828.51

```
colunas_para_avaliar = ['NM_MUNICIP', 'BOLSA FAMILIA', 'Área Territorial - km² [2020]', 'População estimada - pessoas [2020]']
mapa_rj[colunas_para_avaliar].sort_values('BOLSA FAMILIA', ascending=False).head(5)
```

			Área		Densidade
			Territorial - km ² [2020]	População estimada - pessoas [2020]	demográfica - hab/km ² [2010]
67	RIO DE JANEIRO	1010316090.67	1.200.329	6747815	5265.82
24	DUQUE DE CAXIAS	167405029.11	467.319	924624	1828.51
49	NOVA IGUACU	163562353.22	520.581	823302	1527.60

Além do Rio de Janeiro, quais outros municípios estão entre os que mais recebem repasses de todos os tipos?

maiores = [mapa_rj[['NM_MUNICIP', 'BPC']].sort_values('BPC', ascending=False).head(5).NM_MUNICIP]
mapa_rj[['NM_MUNICIP', 'FPM']].sort_values('FPM', ascending=False).head(5).NM_MUNICIP.tolist()
mapa_rj[['NM_MUNICIP', 'BOLSA FAMILIA']].sort_values('BOLSA FAMILIA', ascending=False).head(5)
mapa_rj[['NM_MUNICIP', 'AUXILIO EMERGENCIAL']].sort_values('AUXILIO EMERGENCIAL', ascending=False)
set.intersection(*map(set,maiores))
{'DUQUE DE CAXIAS', 'RIO DE JANEIRO'}

E quais municípios são os que menos recebem repasses?

			Área	População estimada - pessoas [2020]	Densidade demográfica - hab/km ² [2010]
			Territorial - km ² [2020]	pessoas [2020]	hab/km ² [2010]
1	APERIBÉ	172080.00	94.542	11901	107.92
37	MACUCO	155823.78	78.364	5623	67.80
65	RIO DAS FLORES	126099.10	478.783	9344	17.90
75	SÃO JOSÉ DE UBÁ	104384.76	249.688	7206	27.98

			Área	População estimada - pessoas [2020]	Densidade demográfica - hab/km ² [2010]
			Territorial - km ² [2020]	pessoas [2020]	hab/km ² [2010]
37	MACUCO	506171.76	78.364	5623	67.80
65	RIO DAS FLORES	506171.76	478.783	9344	17.90
35	LAJE DO MURIAÉ	506171.76	253.530	7326	29.95
75	SÃO JOSÉ DE UBÁ	506171.76	249.688	7206	27.98

```
colunas_para_avaliar = ['NM_MUNICIP', 'BOLSA FAMILIA', 'Área Territorial - km² [2020]', 'População estimada - pessoas [2020]']
mapa_rj[colunas_para_avaliar].sort_values('BOLSA FAMILIA', ascending=False).tail(5)
```

	NM_MUNICIP	BOLSA FAMILIA	Área Territorial - km² [2020]	População estimada - pessoas [2020]	Densidade demográfica - hab/km² [2010]
78	SÃO SEBASTIÃO DO ALTO	116806.33	397.214	9387	22.35
23	DUAS BARRAS	112691.44	379.619	11528	29.14
-	SANTA MARIA				

```
colunas_para_avaliar = ['NM_MUNICIP', 'AUXILIO EMERGENCIAL', 'Área Territorial - km² [2020]', 'População estimada - pessoas [2020]']
mapa_rj[colunas_para_avaliar].sort_values('AUXILIO EMERGENCIAL', ascending=False).head(5).tail(5)
```

	NM_MUNICIP	AUXILIO EMERGENCIAL	Área Territorial - km² [2020]	População estimada - pessoas [2020]	Densidade demográfica - hab/km² [2010]
67	RIO DE JANEIRO	1010316090.67	1.200.329	6747815	5265.82
24	DUQUE DE CAXIAS	167405029.11	467.319	924624	1828.51
49	NOVA IGUACU	163562353.22	520.581	823302	1527.60

Quais municípios estão entre os que menos recebem repasses de todos os tipos?

```
menores = [mapa_rj[['NM_MUNICIP', 'BPC']].sort_values('BPC', ascending=False).tail(5).NM_MUNICIP,
mapa_rj[['NM_MUNICIP', 'FPM']].sort_values('FPM', ascending=False).tail(5).NM_MUNICIP.tolist(),
mapa_rj[['NM_MUNICIP', 'BOLSA FAMILIA']].sort_values('BOLSA FAMILIA', ascending=False).tail(5).NM_MUNICIP,
mapa_rj[['NM_MUNICIP', 'AUXILIO EMERGENCIAL']].sort_values('AUXILIO EMERGENCIAL', ascending=False).NM_MUNICIP.tolist()]
set.intersection(*map(set, menores))
```

```
{'MACUCO', 'SÃO JOSÉ DE UBÁ'}
```

▼ Qual a correlação entre os valores médios de cada tipo de repasse?

Os valores de repasses estão correlacionados? Para isso vamos analisar a correlação de Pearson.

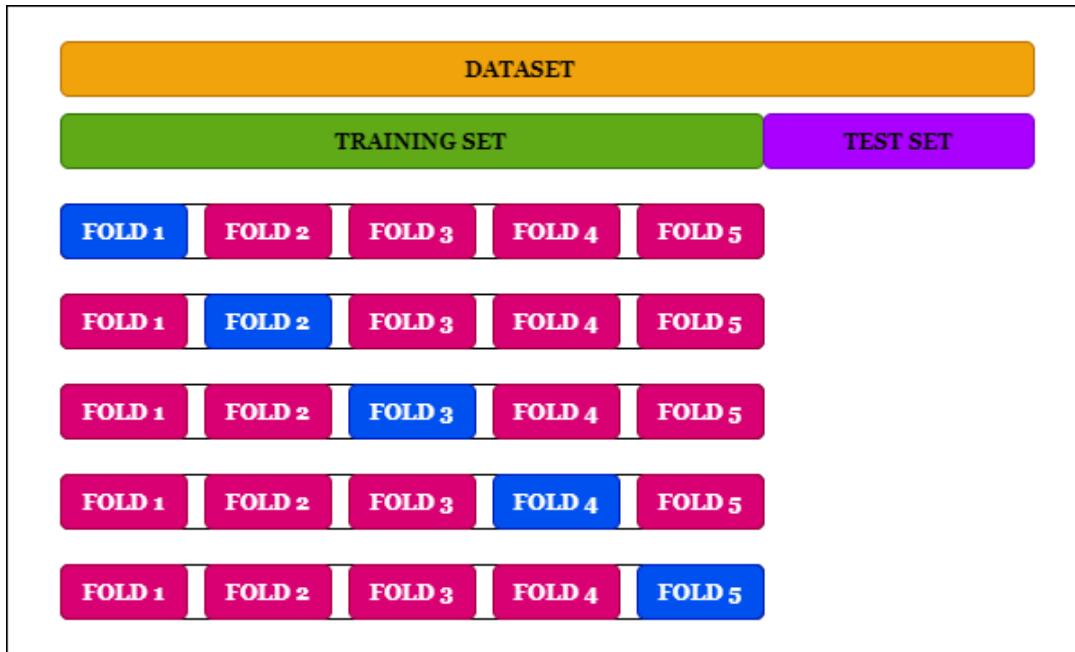
```
v = np.array([mapa_rj['BPC'].tolist(), mapa_rj['FPM'].tolist(),
             mapa_rj['BOLSA FAMILIA'].tolist(), mapa_rj['AUXILIO EMERGENCIAL'].tolist()])
np.corrcoef(v)

array([[1.          , 0.92324116, 0.97137647, 0.99468916],
       [0.92324116, 1.          , 0.90843526, 0.92150122],
       [0.97137647, 0.90843526, 1.          , 0.97730426],
       [0.99468916, 0.92150122, 0.97730426, 1.        ]])
```

	BPC	FPM	BOLSA FAMILIA	AUXILIO EMERGENCIAL
BPC	0.92324116	0.97137647	0.99468916	
FPM	0.92324116	0.90843526	0.92150122	
BOLSA FAMILIA	0.97137647	0.97137647		0.97730426

A correlação de Pearson entre os valores médios de repasses por município mostra uma correlação positiva forte, ou seja, quanto maior o valor de um tipo de repasse, maior o valor de outro tipo de repasse.

▼ Treinamento de Modelos de Machine Learning



Antes de gerarmos nossos modelos precisamos dividir nosso dados entre os dados que vamos treinar e o dado que vamos fazer a predição (ou testar), ou seja **TRAINING SET** e **TEST SET**. Além disso, precisamos de ter um processo que nos auxilie a escolher qual modelo se ajusta melhor aos nossos dados como um todo. Apesar de podermos treinar os modelos escolhidos com todo o dado de treinamento e avaliá-lo apenas no conjunto de teste, esse processo pode esconder algumas informações.

Imagine que um modelo, que seja um pouco mais sofisticado, consiga encontrar um "atalho" nos dados que o faça ter um desempenho melhor que os outros modelos. Além disso, imagine que, num ambiente de produção, os dados usados para predição sejam exatamente os dados que esse modelo "ignorou" ao tomar os atalhos que o dava um melhor desempenho no conjunto de teste (?). Para mitigar esse risco, podemos fazer uso do processo K-Cross Validation, onde avaliamos um mesmo modelo, em diferentes recortes dos dados de treinamento antes de avaliá-lo nos dados de teste.

```
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.svm import LinearSVR
from sklearn.neighbors import KNeighborsRegressor
from statsmodels.api import OLS

class OLSWrapper:

    def __init__(self):
        self.model = None

    def fit(self, X, y):
        self.model = OLS(y,X).fit()

    def predict(self, X):
        return self.model.predict(X)
```

```

def __str__(self):
    return 'OLS'

from sklearn.model_selection import KFold
from sklearn.model_selection import ShuffleSplit

/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
    import pandas.util.testing as tm

```

Vamos usar como modelos de regressão:

- OLS (statsmodel)
- Linear Regression (sklearn)
- KNR (k=5)
- SVR (kernels= [RBF, polinomial e linear])

OBS. 1: Os modelos OLS do statsmodel e Linear Regression do sklearn seguem basicamente o mesmo procedimento, no entanto a fórmula da regressão utilizada tem uma pequena diferença de tratamento. Por exemplo, enquanto no OLS uma fórmula seria utilizada do seguinte modo:

$$y = B_0 + B_1x_1$$

o Linear Regression acrescentaria um termo:

$$y = B_0 + B_1x_1$$

o que em casos de borda podem alterar os resultados obtidos, como será visto nos resultados do K-Fold Cross Validation(<https://stats.stackexchange.com/a/482178>)

OBS. 2: As interfaces utilizadas nos modelos do statsmodels e do sklearn são diferentes, por isso foi criado uma classe que serve como um proxy convertendo as interfaces usadas do sklearn para a forma esperada pelo statsmodels ao usar o OLS.

```

treino = dados[dados['Mês/Ano'] != '12/2020']
teste = dados[dados['Mês/Ano'] == '12/2020']
print('Total de dados de treinamento: '+str(len(treino)))
print('Total de dados de teste: '+str(len(teste)))

```

```

Total de dados de treinamento: 736
Total de dados de teste: 92

```

Os dados de treinamento são de todos os meses, exceto dezembro, que é o mês que vamos tentar prever.

```

X_train = treino[['BPC', 'FPM', 'BOLSA FAMILIA']].values
y_train = treino[['AUXILIO EMERGENCIAL']].values

modelos = [LinearRegression(), OLSWrapper(), SVR(), SVR(kernel='poly'), LinearSVR(max_iter=1000)]

def treino_kfold(X_train, y_train, modelos, n_split=10, shuffle=False):
    if shuffle:
        random_state = 42
    else:
        random_state = None
    kf = KFold(n_splits=n_split, shuffle=shuffle, random_state=random_state)
    resp = {}

    for train_index, test_index in kf.split(X_train):
        X_train_fold, X_test_fold = X_train[train_index], X_train[test_index]
        y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]

        for modelo in modelos:
            modelo.fit(X_train_fold, y_train_fold)
            y_pred = modelo.predict(X_test_fold)
            resp[modelo] = np.append(resp.get(modelo, {}), y_pred)

```

```

for train, test in kf.split(X_train):
    X_, X_test, y_, y_test = X_train[train], X_train[test], y_train[train], y_train[test]
    for modelo in modelos:
        if str(modelo) not in resp.keys():
            resp[str(modelo)] = {'preditos': [], 'reais': []}

        modelo.fit(X_, y_.ravel())
        y_predict = modelo.predict(X_test)

        resp[str(modelo)]['preditos'].append(y_predict.ravel())
        resp[str(modelo)]['reais'].append(y_test.ravel())

return resp

```

X = [A,B,C,D]

[0,1] [2,3]

Os dados de resultados que estamos guardando vão ser parecidos com a seguinte estrutura:

```

{ 'LinearRegression': {
    'preditos': [
        [1,2,3,4],
        [2,4,5,6]
    ],
    'reais': [
        [1,3,3,4],
        [3,4,1,6]
    ]
}
}

```

```

def nome_display_modelo(nome):
    tmp = str(nome).split('(')[0].strip()
    kernel = [k for k in nome.split(',') if 'kernel' in k]
    if len(kernel) > 0:
        return tmp + ' ' + kernel[0].strip()
    return tmp

def plota_metrica(resultados, modelos, metrica, titulo, eixo_x, eixo_y, loc='upper left'):
    resp = []
    fig = plt.figure(figsize=(10, 5))
    plt.subplot(1, 1, 1)

    boxplots_data = []
    boxplots_ticks = []

    for nome_modelo in modelos:
        modelo = resultados[str(nome_modelo)]
        for i, d in enumerate(zip(modelo['reais'], modelo['preditos'])):
            real, predito = d
            resp.append([i, metrica(real, predito)])

    nome_display = nome_display_modelo(str(nome_modelo))
    plt.plot([x[0] for x in resp], [x[1] for x in resp],
             'bo--')

```

```

        label=nome_display)
print(nome_display, resp)

boxplots_data.append([ x[1] for x in resp])
boxplots_ticks.append(nome_display)

resp = []

plt.title(titulo)
plt.legend(loc=loc)
plt.xlabel(eixo_x)
plt.ylabel(eixo_y)
fig.tight_layout()
plt.show()

fig2, ax2 = plt.subplots()
ax2.set_title('Boxplots '+eixo_y)
ax2.boxplot(boxplots_data)
plt.xticks(list(range(1, len(boxplots_ticks)+1)), boxplots_ticks, rotation=90)
plt.show()

```

```

from sklearn.metrics import mean_squared_error

def rmse(y_true, y_pred):
    return mean_squared_error(y_true, y_pred, squared=False)
# R^2 (coefficient of determination) regression score function.
# Best possible score is 1.0 and it can be negative (because the model can be arbitrarily wrong).
# A constant model that always predicts the expected value of y, disregarding the input features,
# would get a R^2 score of 0.0.
# https://www.geeksforgeeks.org/python-coefficient-of-determination-r2-score/
from sklearn.metrics import r2_score

from sklearn.metrics import mean_absolute_percentage_error

```

Vamos fazer uma primeira avaliação dos modelos usando o KFold Cross Validation com K = 10, escolha arbitrária.

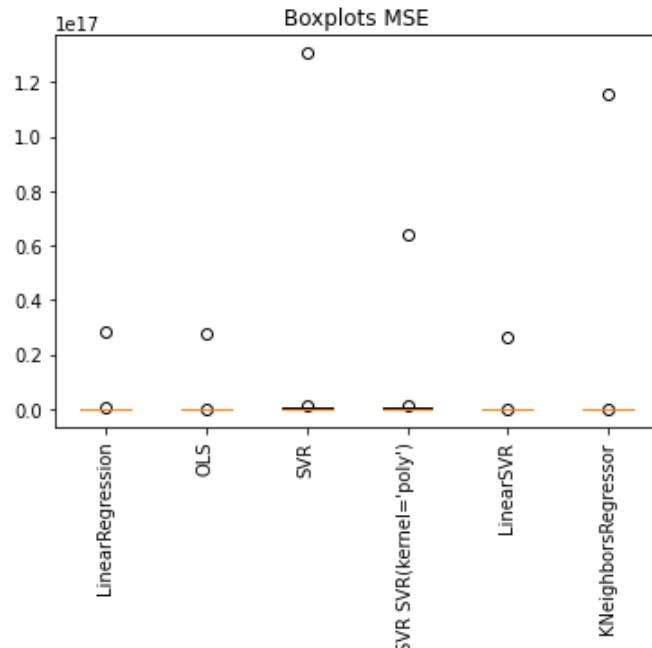
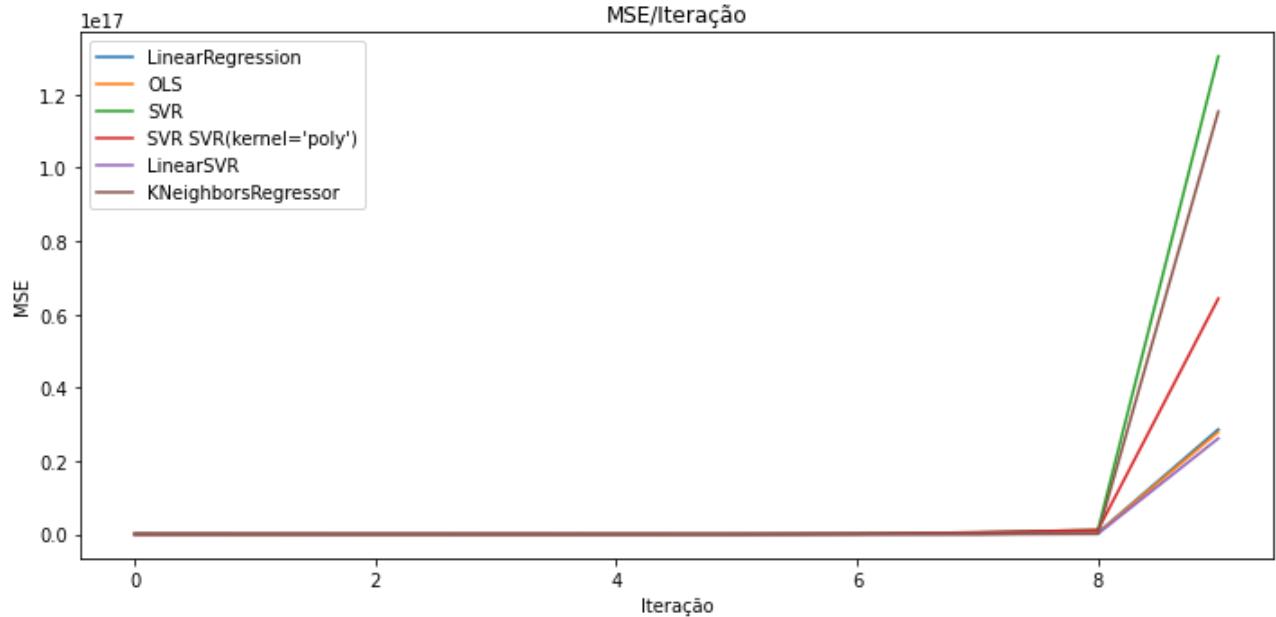
```

resp_10_fold = treino_kfold(X_train, y_train, modelos, n_split=10)

/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:986: ConvergenceWarning: LIBSVM
  "the number of iterations.", ConvergenceWarning)

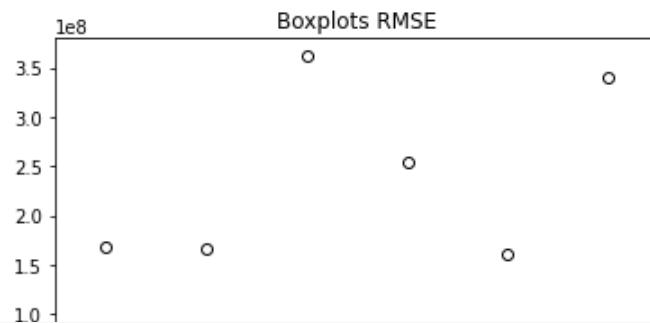
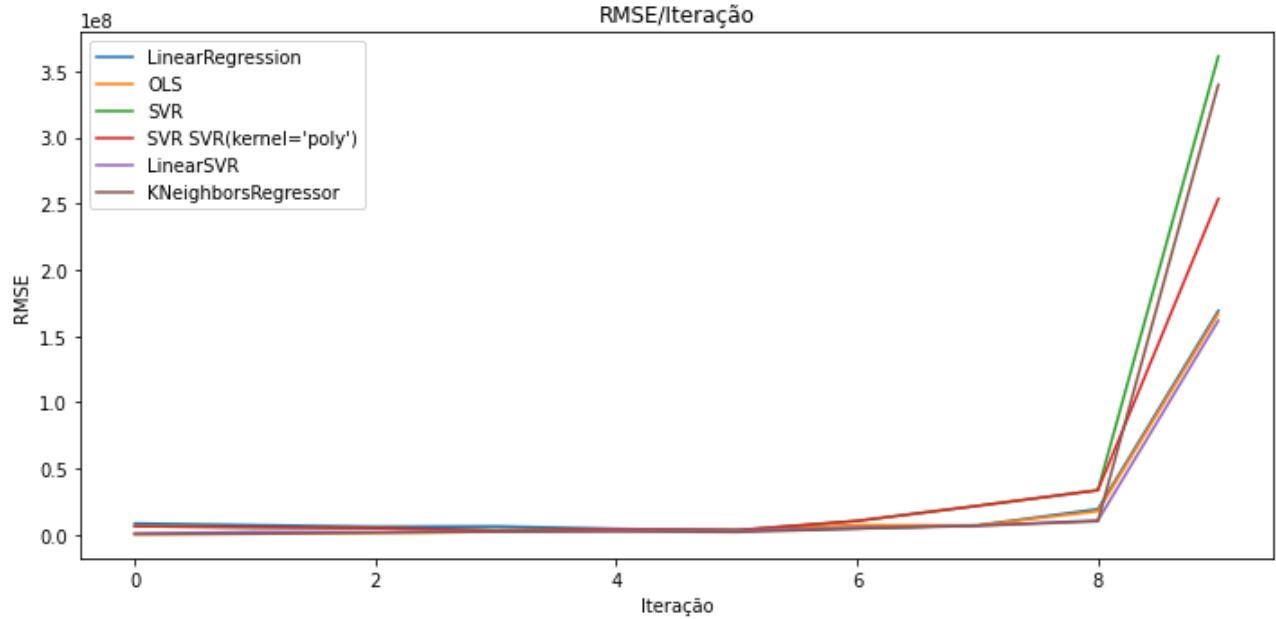
```

```
LinearRegression [[0, 73702815587955.02], [1, 53566928836020.53], [2, 37111744625899.08]
OLS [[0, 250291057811.17578], [1, 737545572427.4434], [2, 2503079906790.4673], [3, 4982
SVR [[0, 40981655484593.375], [1, 32884319932374.21], [2, 25543798662341.54], [3, 11025
SVR SVR(kernel='poly') [[0, 40981709508836.21], [1, 32884351240270.37], [2, 25543810153
LinearSVR [[0, 786684609099.5485], [1, 5109508822346.389], [2, 4886738643444.095], [3,
KNeighborsRegressor [[0, 187458476635.90106], [1, 583728143639.8718], [2, 2347134206359
```



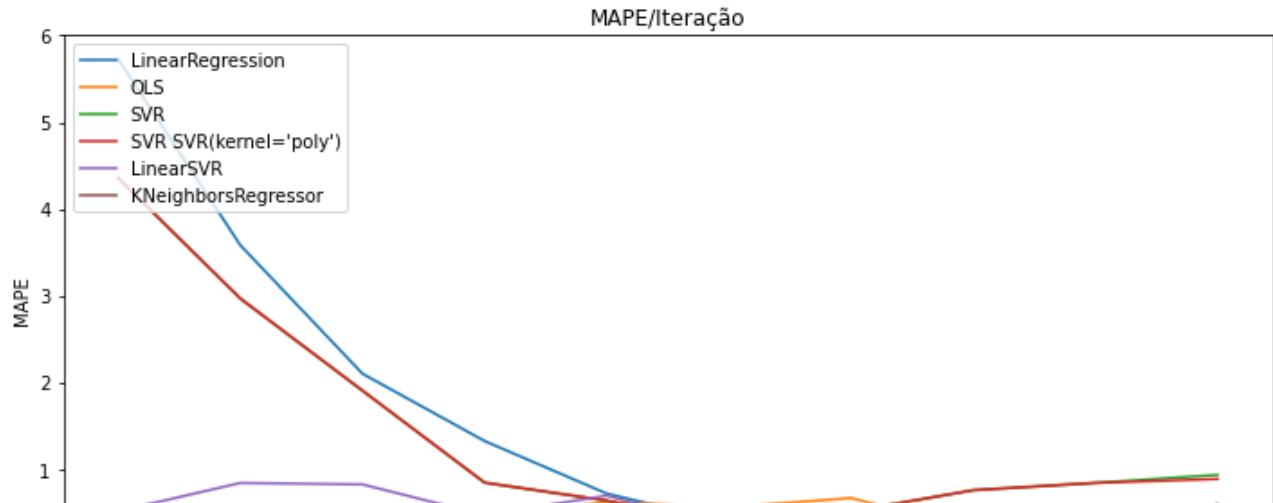
```
plota_metrica(resp_10_fold, modelos, rmse, 'RMSE/Iteração', 'Iteração', 'RMSE', loc='upper left')
```

```
LinearRegression [[0, 8585034.396434007], [1, 7318943.150211001], [2, 6091940.957190827]
OLS [[0, 500290.9731458042], [1, 858804.7347490833], [2, 1582112.4823445606], [3, 22321
SVR [[0, 6401691.611175391], [1, 5734485.149721308], [2, 5054087.322389823], [3, 332044
SVR SVR(kernel='poly') [[0, 6401695.83070269], [1, 5734487.87951203], [2, 5054088.45918
LinearSVR [[0, 886952.42775447], [1, 2260422.266379976], [2, 2210596.897546926], [3, 22
KNeighborsRegressor [[0, 432964.7521864812], [1, 764021.0361239223], [2, 1532035.967710
```



```
plota_metrica(resp_10_fold, modelos, mean_absolute_percentage_error, 'MAPE/Iteração', 'Iteraçõ
```

```
LinearRegression [[0, 5.72630280841444], [1, 3.580870407542419], [2, 2.0986065596372234
OLS [[0, 0.27352033393044317], [1, 0.4297588787347003], [2, 0.5695555298987176], [3, 0.
SVR [[0, 4.3546583996745], [1, 2.9656711268996876], [2, 1.9029195909657666], [3, 0.8415
SVR SVR(kernel='poly') [[0, 4.354661306432743], [1, 2.965672583981489], [2, 1.902920022
LinearSVR [[0, 0.512456643433221], [1, 0.8382914833124673], [2, 0.821131642835244], [3
KNeighborsRegressor [[0, 0.237002310988329], [1, 0.27328485900258703], [2, 0.4074067783
```



```
plota_metrica(resp_10_fold, modelos, r2_score, 'R2/Iteração', 'Iteração', 'R2', loc='lower right')
```

```

LinearRegression [[0, -327.4215144878936], [1, -71.75563893318326], [2, -31.34266782260
OLS [[0, -0.1153029584199099], [1, -0.0017486634069545648], [2, -1.181419461004475], [3
SVR [[0, -181.61524004343255], [1, -43.66411943994582], [2, -21.26127074842776], [3, -0
SVR(SVR(kernel='poly')) [[0, -181.61548077675474], [1, -43.66416196293601], [2, -21.2612
LinearSVR [[0, -2.5054854917511973], [1, -5.9398337198411815], [2, -3.258764072506258],
KNeighborsRegressor [[0, 0.16468053073378852], [1, 0.20716914378631213], [2, -1.0455136

```

R2/Iteração

O que podemos observar avaliando o MSE é que todos os modelos tem um comportamento similar nos primeiros folds, mas nos 2 folds finais o melhor foi SVR com kernel linear seguido dos modelos são o OLS e Linear Regression. O RMSE seguiu a mesma tendência.

Já usando o R2 score, o SVR com kernel linear e o KNR (K=5) têm seus resultados muito próximos em todos os folds. Seguido do OLS.

Contudo, algo que devemos prestar atenção é realizando o split dos dados em 10 folds sequenciais, podemos ter folds desbalanceados, pois o número de folds é maior que o numero de meses dos dados, e algum fold pode conter dados com uma variância muito diferente da distribuição original dos dados. Essa diferença entre a variância dos dados de um fold e os dados originais poderia justificar os resultados ruins de alguns modelos nos folds finais.

Vamos realizar o processo de cross-validation com um k=8, assim teremos folds que batem com os meses dos dados de treinamento, e portanto os dados não estaram desbalanceados.

```
resp_8_fold = treino_kfold(X_train, y_train, modelos, n_split=8)
```

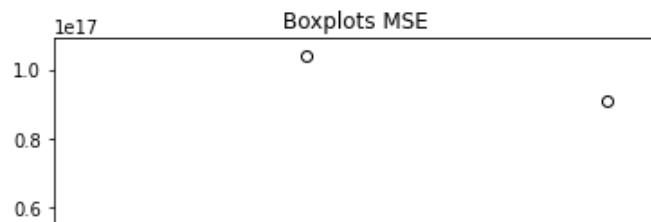
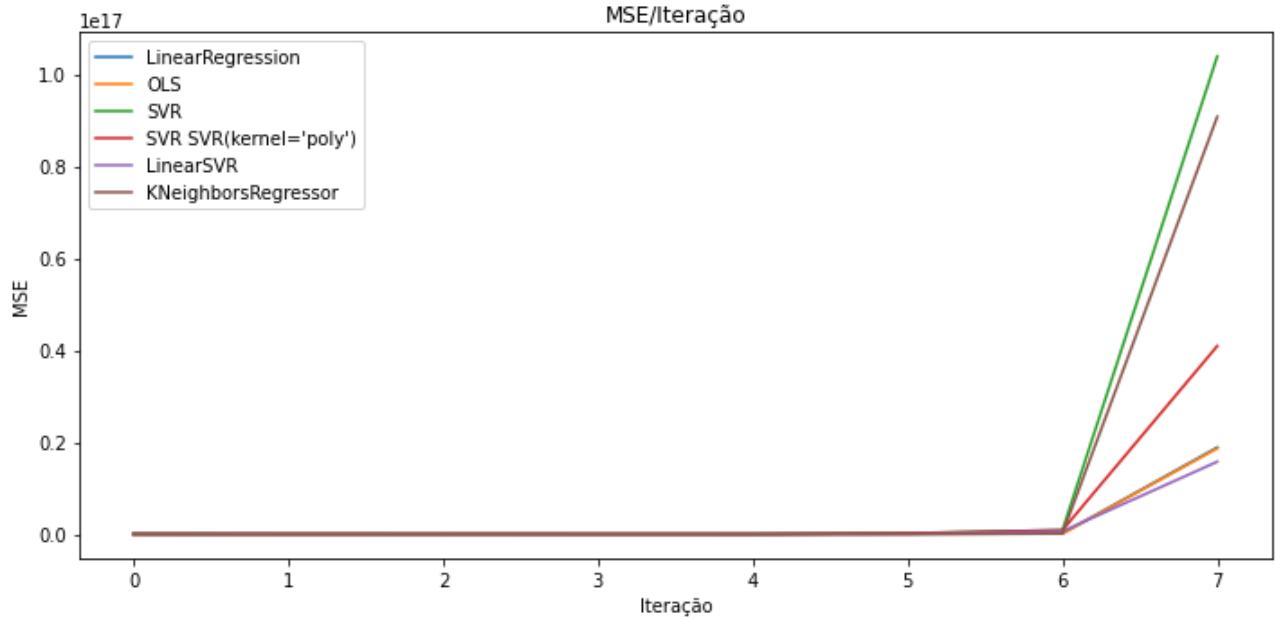
```

/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:986: ConvergenceWarning: Li
    "the number of iterations.", ConvergenceWarning)

```

```
plota_metrica(resp_8_fold, modelos, mean_squared_error, 'MSE/Iteração', 'Iteração', 'MSE', lo
```

```
LinearRegression [[0, 78653690691965.33], [1, 51681951019327.99], [2, 39421937264139.87]
OLS [[0, 282290844466.25793], [1, 1107794079451.6824], [2, 4431052049656.41], [3, 98685
SVR [[0, 43221948722965.21], [1, 30723083946804.64], [2, 20940517513323.76], [3, 178486
SVR SVR(kernel='poly') [[0, 43222054616414.016], [1, 30723153795202.72], [2, 2094053990
LinearSVR [[0, 300571449552.2905], [1, 3058455715449.781], [2, 3805400871955.943], [3,
KNeighborsRegressor [[0, 296076140120.70654], [1, 1299603939717.035], [2, 2553250662870
```



```
plota_metrica(resp_8_fold, modelos, rmse, 'RMSE/Iteração', 'Iteração', 'RMSE', loc='upper lef
```

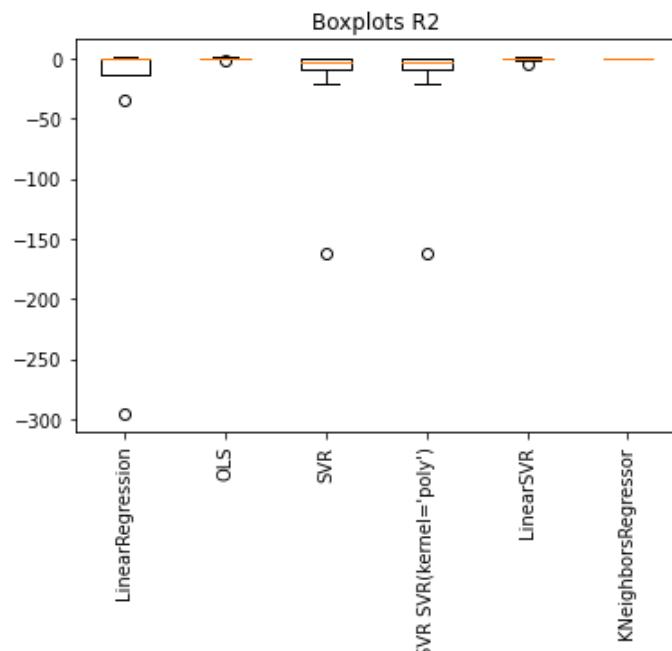
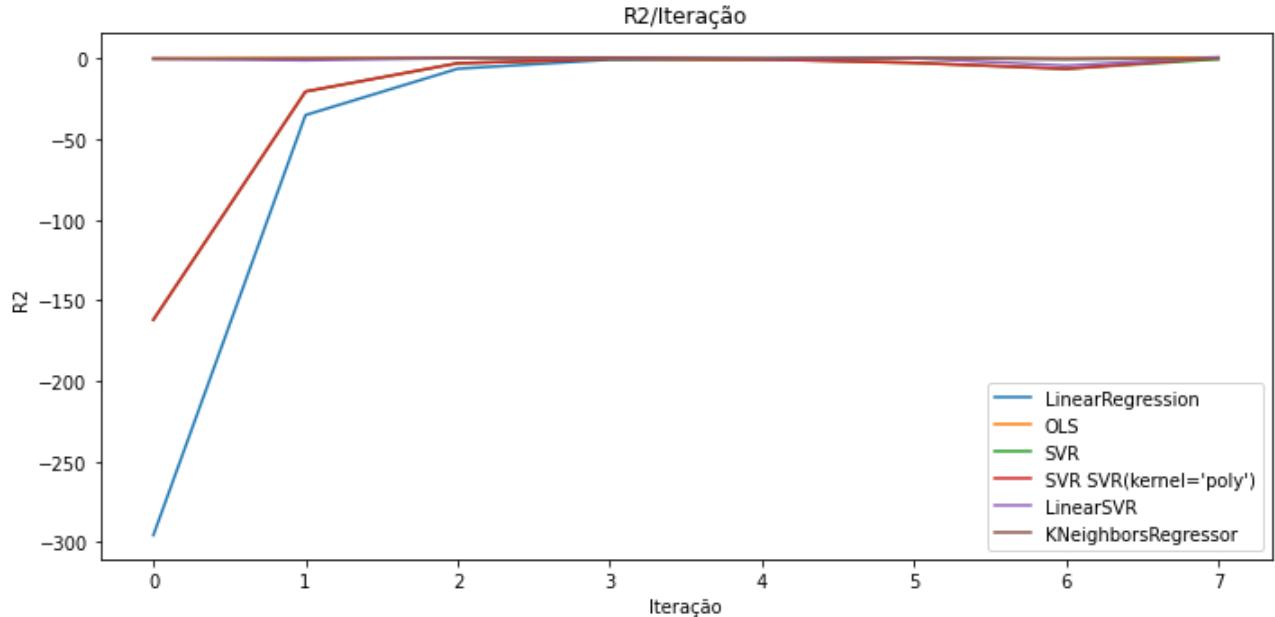
```
LinearRegression [[0, 8868691.599777576], [1, 7189015.9979880415], [2, 6278689.13581010
OLS [[0, 531310.4972294994], [1, 1052517.9710825286], [2, 2105006.4250867288], [3, 3141
SVR [[0, 6574340.173961583], [1, 5542840.783100723], [2, 4576081.02128052], [3, 4224761
SVR SVR(kernel='poly') [[0, 6574348.227498602], [1, 5542847.083873298], [2, 4576083.468
LinearSVR [[0, 548243.9690067648], [1, 1748844.108389819], [2, 1950743.671515031], [3,
KNeighborsRegressor [[0, 544128.7900127198], [1, 1140001.7279447585], [2, 1597889.44012
```



```
plota_metrica(resp_8_fold, modelos, mean_absolute_percentage_error, 'MAPE/Iteração', 'Iteração')
```

```
LinearRegression [[0, 5.631680369108014], [1, 2.9956439779903774], [2, 1.80104451520300
OLS [[0, 0.2798975711316105], [1, 0.423988870971584], [2, 0.598739304479637], [3, 0.580
SVR [[0, 4.253545874306161], [1, 2.5089340690723763], [2, 1.5468298178880688], [3, 0.72
SVR SVR(kernel='poly') [[0, 4.253551085773068], [1, 2.5089369038373537], [2, 1.54683066
TlinearSVR rro 0.249569653833716181 r1 0.72154244099246861 r2 0.28846992511829261
plota_metrica(resp_8_fold, modelos, r2_score, 'R2/Iteração', 'Iteração', 'R2', loc='lower ri
```

```
LinearRegression [[0, -295.5878595363817], [1, -35.04499019554924], [2, -6.310248240142
OLS [[0, -0.06446419221263189], [1, 0.22738159173615125], [2, 0.17832322062347228], [3,
SVR [[0, -161.98161146613558], [1, -20.427466219792276], [2, -2.883126805102604], [3, -
SVR SVR(kernel='poly') [[0, -161.98201076985603], [1, -20.4275149347648], [2, -2.883130
LinearSVR [[0, -0.1333968193506252], [1, -1.1330852280650285], [2, 0.2943415022741954],
KNeighborsRegressor [[0, -0.1164458766734573], [1, 0.09360598156058564], [2, 0.52653528
```



Pelo gráfico, percebemos que diminuir a quantidade de folds (e consequentemente aumentar a quantidade de dados de teste a cada iteração) manteve o mesmo perfil das linhas dos modelos no MSE e RSME. O modelo SVR com kernel linear se destacou positivamente.

Em outra avaliação podemos também alterar como os folds do cross validation são gerados para usarem os dados de forma aleatória e não sequencial. Isso é desejável, já que na regressão não tem como

premissa a dependência temporal. Vamos começar com 10 folds.

```
resp_10_random_fold = treino_kfold(X_train, y_train, modelos, n_split=10, shuffle=True)

/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:986: ConvergenceWarning: Li
  "the number of iterations.", ConvergenceWarning)
```

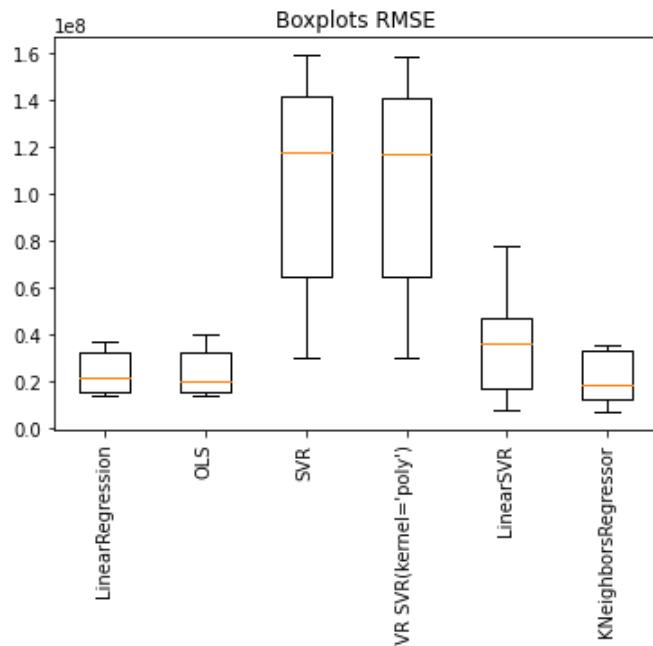
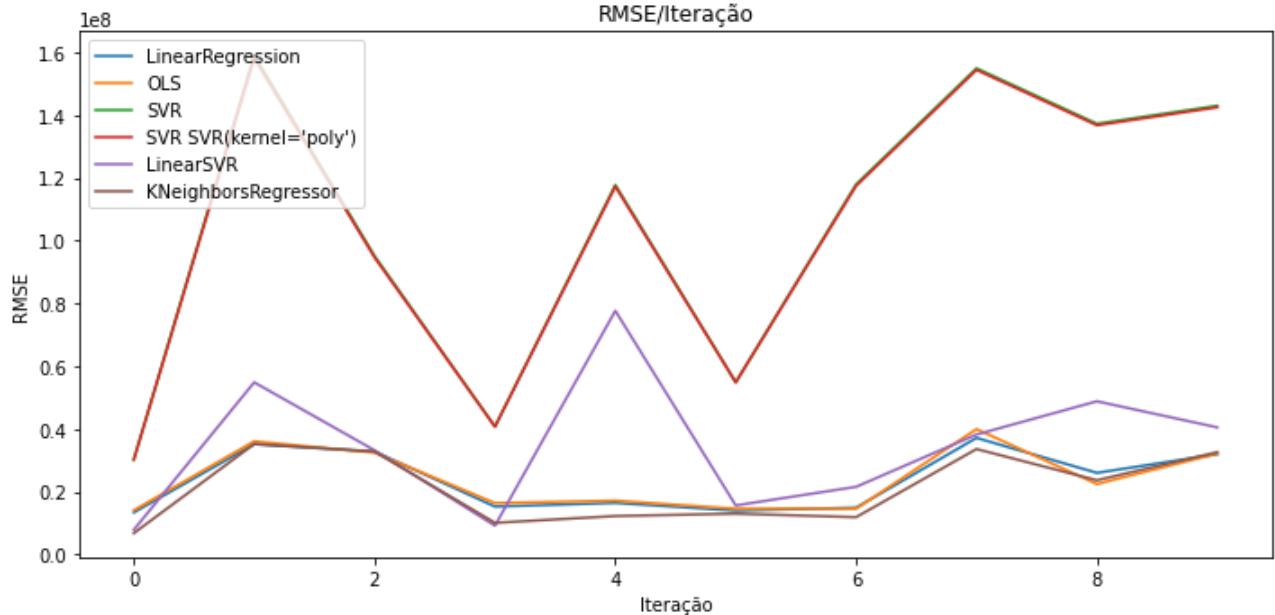
```
plota_metrica(resp_10_random_fold, modelos, mean_squared_error, 'MSE/Iteração', 'Iteração',
```

```
LinearRegression [[0, 179106804596300.3], [1, 1235713578193080.5], [2, 1075247984600578
OLS [[0, 198162523397208.5], [1, 1300295764827767.0], [2, 1052207468267538.6], [3, 2681
SVR [[0, 914526395206907.6], [1, 2.5291552116542692e+16], [2, 9043803083712710.0], [3,
SVR SVR(kernel='poly') [[0, 914234802393585.8], [1, 2.5015989600244772e+16], [2, 895234
LinearSVR [[0, 61678750427241.83], [1, 3007712057916741.0], [2, 1101127437134685.9], [3
KNeighborsRegressor [[0, 46009154960786.766], [1, 1239134715843687.2], [2, 107459675954
```



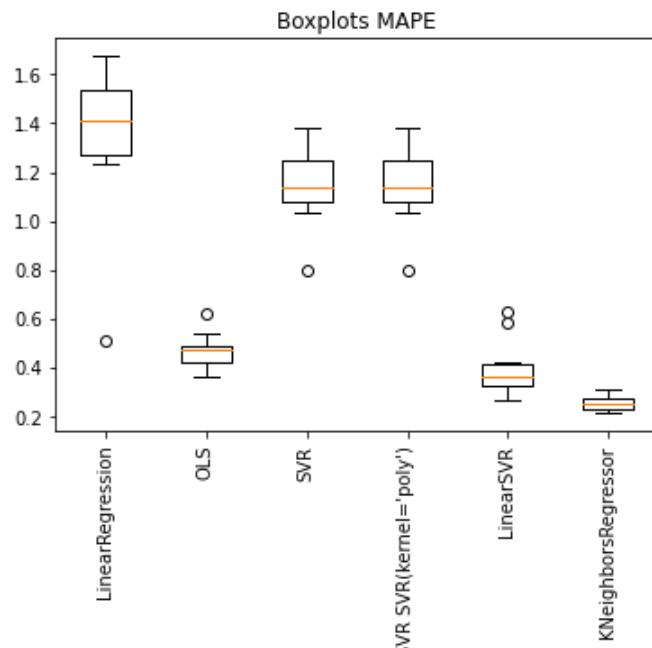
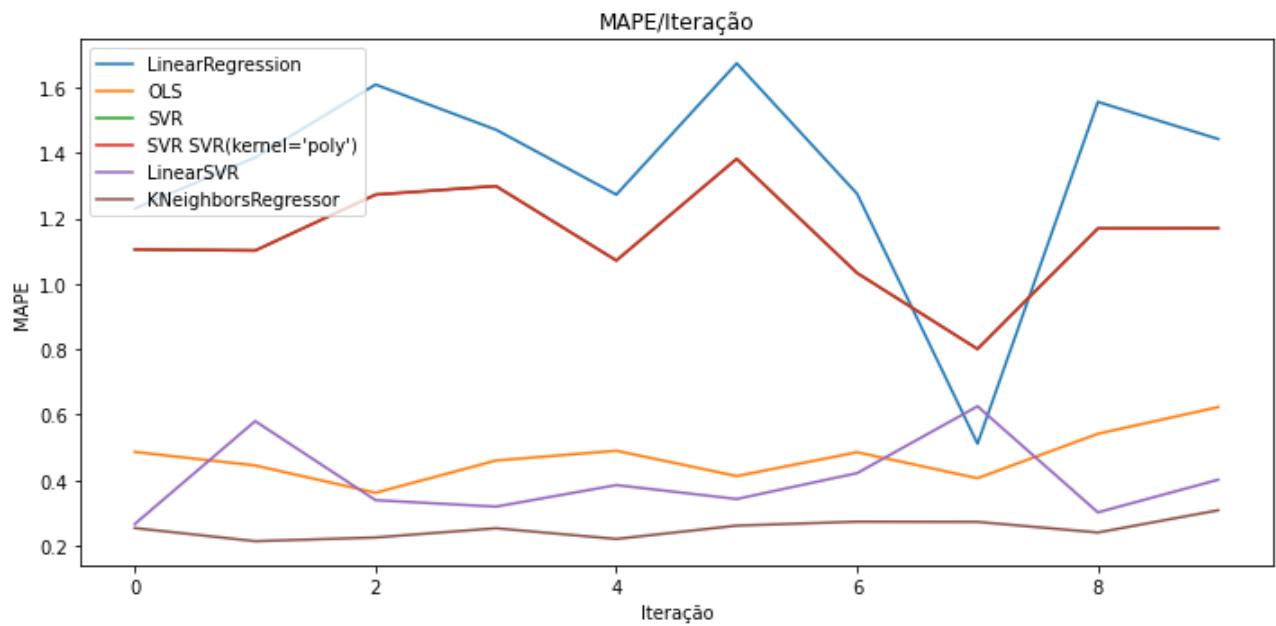
```
plota_metrica(resp_10_random_fold, modelos, rmse, 'RMSE/Iteração', 'Iteração', 'RMSE', loc='right')
```

```
LinearRegression [[0, 13383079.040202232], [1, 35152717.934650235], [2, 32790974.133144
OLS [[0, 14077021.112337954], [1, 36059614.041580744], [2, 32437747.58314052], [3, 1637
SVR [[0, 30241137.465494044], [1, 159033179.2945821], [2, 95098912.10583174], [3, 40725
SVR SVR(kernel='poly') [[0, 30236315.95273448], [1, 158164438.48174208], [2, 94616835.3
LinearSVR [[0, 7853582.01251135], [1, 54842611.69853913], [2, 33183240.304929323], [3,
KNeighborsRegressor [[0, 6783004.862211641], [1, 35201345.36979641], [2, 32781042.68542
```



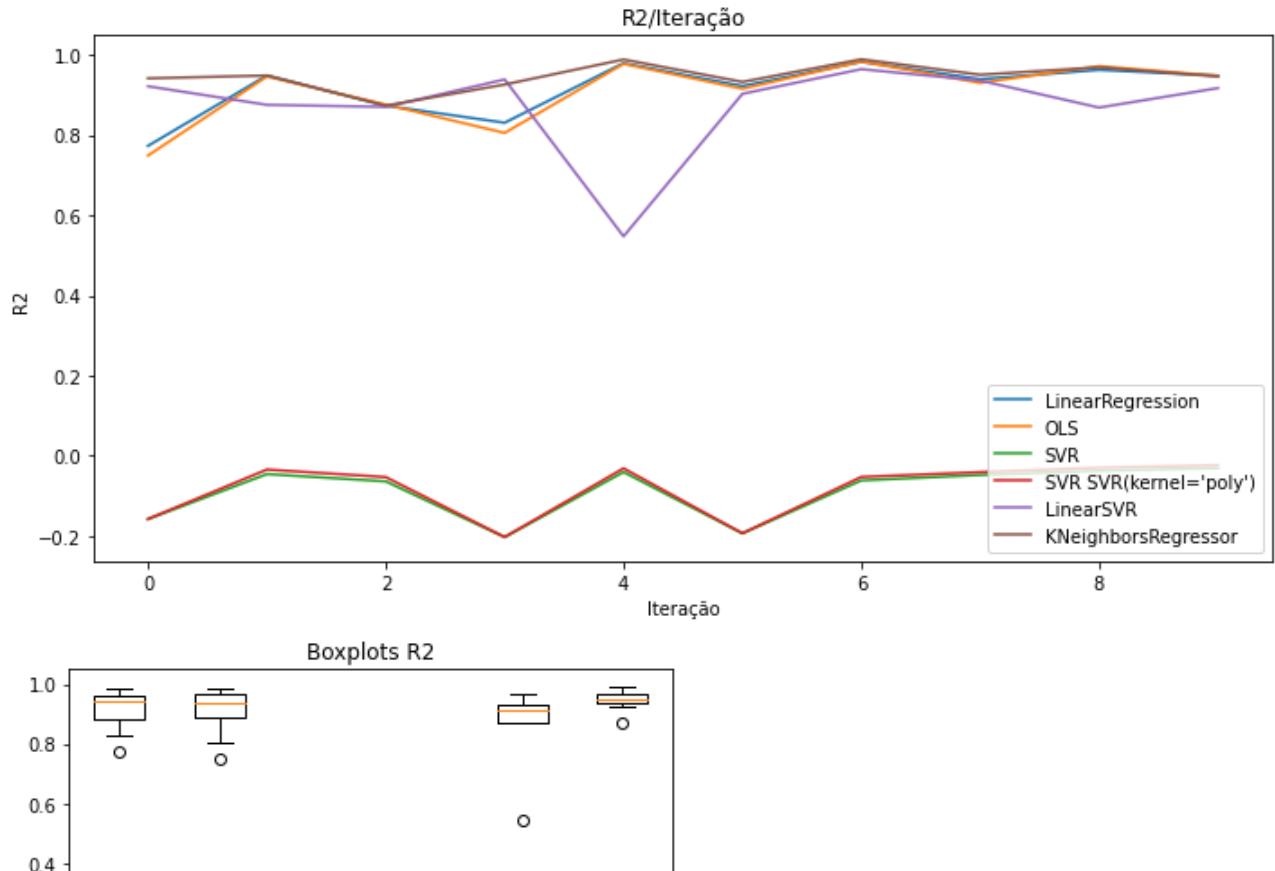
```
plota_metrica(resp_10_random_fold, modelos, mean_absolute_percentage_error, 'MAPE/Iteração', 'Iteração', 'MAPE', loc='right')
```

```
LinearRegression [[0, 1.230431843998861], [1, 1.3863759178577337], [2, 1.60948437192881
OLS [[0, 0.4861915431801641], [1, 0.44471956338642876], [2, 0.3613456477422591], [3, 0.
SVR [[0, 1.1053723122072143], [1, 1.1020529441894462], [2, 1.2728862091367057], [3, 1.2
SVR SVR(kernel='poly') [[0, 1.1053631145005234], [1, 1.1018811076835218], [2, 1.2727992
LinearSVR [[0, 0.26584753958960833], [1, 0.5803437736305144], [2, 0.33870526560085473],
KNeighborsRegressor [[0, 0.2534459242222839], [1, 0.2132734326730556], [2, 0.2245902233
```



```
plota_metrica(resp_10_random_fold, modelos, r2_score, 'R2/Iteração', 'Iteração', 'R2', loc=)
```

```
LinearRegression [[0, 0.7731695887958835], [1, 0.9489000636343143], [2, 0.8734920009409
OLS [[0, 0.7490364100417731], [1, 0.9462294240252432], [2, 0.8762028264066012], [3, 0.8
SVR [[0, -0.15820500929246228], [1, -0.045870763704449535], [2, -0.06404610693584667],
SVR SVR(kernel='poly') [[0, -0.15783572059961215], [1, -0.034475544540327085], [2, -0.0
LinearSVR [[0, 0.9218867404088771], [1, 0.87562336654859], [2, 0.8704471612353576], [3,
KNeighborsRegressor [[0, 0.9417315519506244], [1, 0.9487585907887218], [2, 0.8735686206
```



Agora com 8 folds.

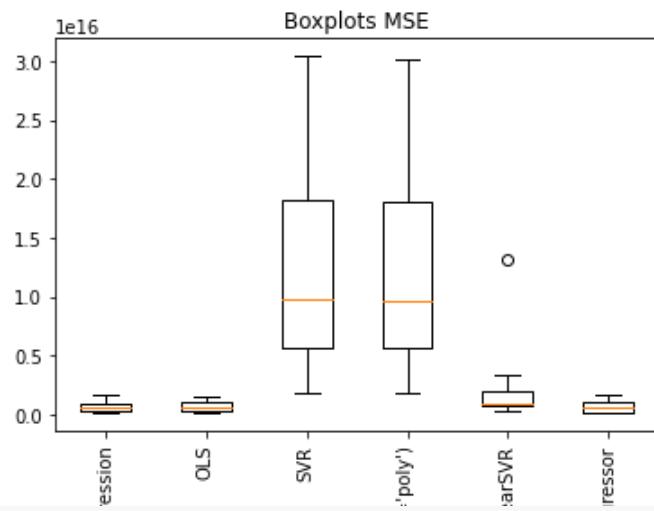
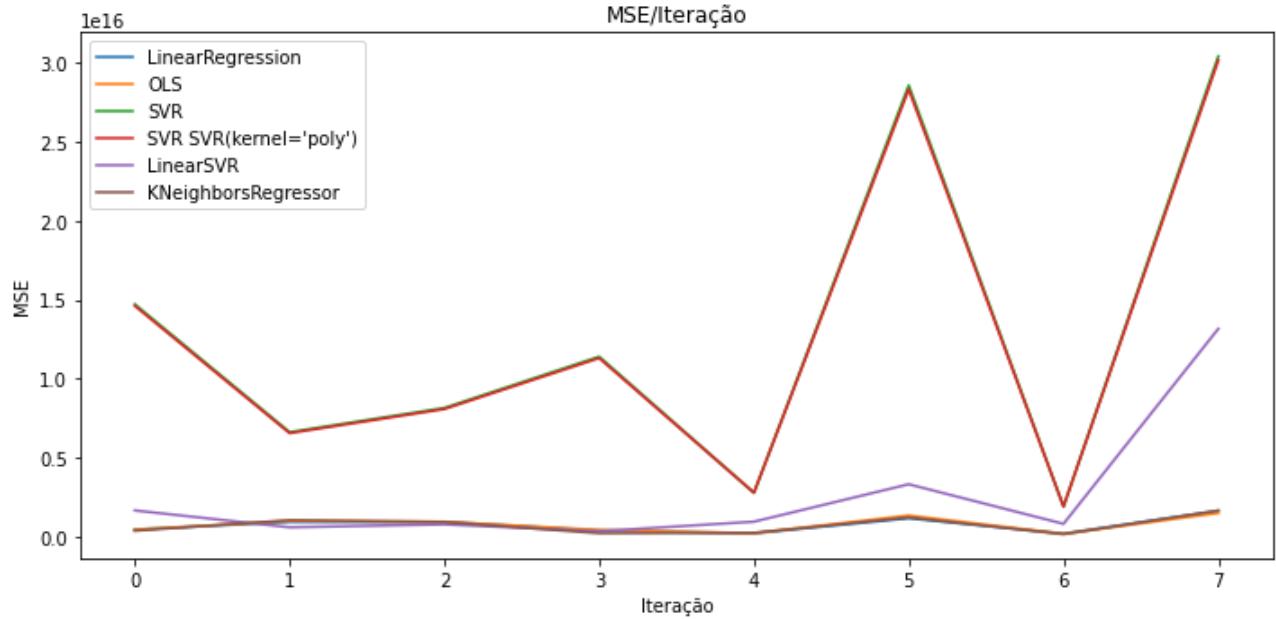
```
|
```

```
resp_8_random_fold = treino_kfold(X_train, y_train, modelos, n_split=8, shuffle=True)

/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:986: ConvergenceWarning: Li
  "the number of iterations.", ConvergenceWarning)
```

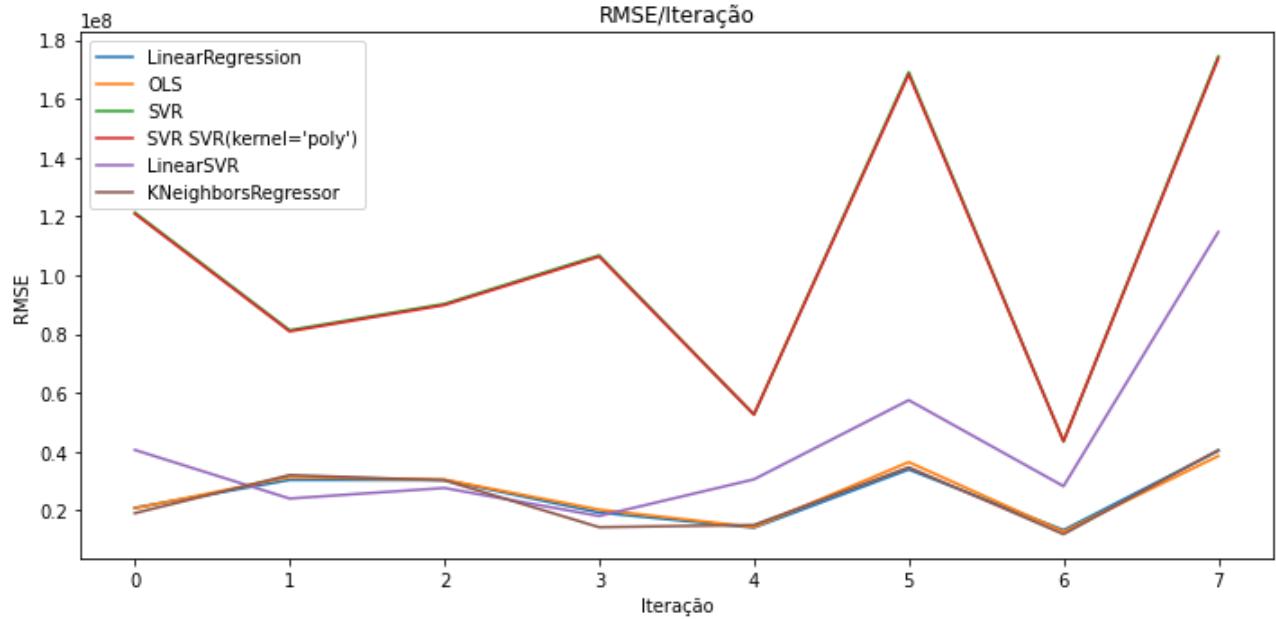
```
plota_metrica(resp_8_random_fold, modelos, mean_squared_error, 'MSE/Iteração', 'Iteração', 'M
```

```
LinearRegression [[0, 436339275429466.5], [1, 922297163634280.9], [2, 921275962975061.8]
OLS [[0, 429115158124454.25], [1, 982177731937673.8], [2, 928624413964610.6], [3, 41068
SVR [[0, 1.4707763388417608e+16], [1, 6606005895303919.0], [2, 8140037172220687.0], [3,
SVR SVR(kernel='poly') [[0, 1.4607189311892146e+16], [1, 6542373659191414.0], [2, 80698
LinearSVR [[0, 1646122375491688.2], [1, 578226903989750.8], [2, 763402890250300.9], [3,
KNeighborsRegressor [[0, 363408356222146.06], [1, 1025277889385633.2], [2, 914552201829
```



```
plota_metrica(resp_8_random_fold, modelos, rmse, 'RMSE/Iteração', 'Iteração', 'RMSE', loc='up')
```

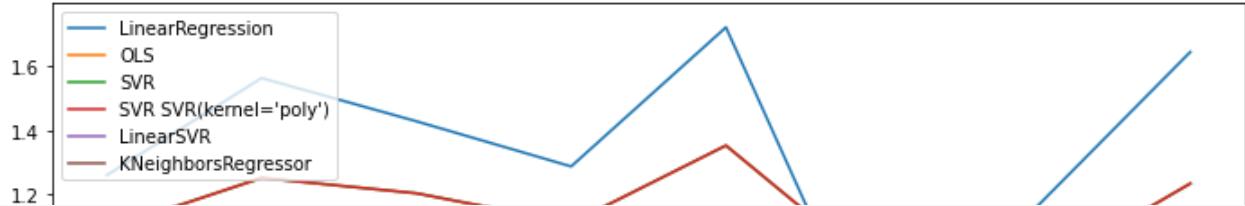
```
LinearRegression [[0, 20888735.61107676], [1, 30369345.78871071], [2, 30352528.11505265
OLS [[0, 20715094.933995698], [1, 31339714.930702128], [2, 30473339.396341365], [3, 202
SVR [[0, 121275567.97812825], [1, 81277339.37146269], [2, 90222154.55319545], [3, 10673
SVR SVR(kernel='poly') [[0, 120860205.6588195], [1, 80884940.86782418], [2, 89832197.34
LinearSVR [[0, 40572433.68953467], [1, 24046349.0781813], [2, 27629746.4745933], [3, 18
KNeighborsRegressor [[0, 19063272.4426355], [1, 32019960.796128925], [2, 30241564.14323
```



```
plota_metrica(resp_8_random_fold, modelos, mean_absolute_percentage_error, 'MAPE/Iteração',
```

```
LinearRegression [[0, 1.2605986807064276], [1, 1.5637698080436697], [2, 1.4288507244987
OLS [[0, 0.497708221948783], [1, 0.41690953543467535], [2, 0.38964814094484856], [3, 0.
SVR [[0, 1.091525919550588], [1, 1.2519331641683318], [2, 1.204052654430522], [3, 1.110
SVR SVR(kernel='poly') [[0, 1.091476958529188], [1, 1.2518707005517016], [2, 1.20397978
LinearSVR [[0, 0.32305061352378234], [1, 0.5367017485602865], [2, 0.2831816898748839],
KNeighborsRegressor [[0, 0.23623776103485464], [1, 0.23094101250000196], [2, 0.22996180
```

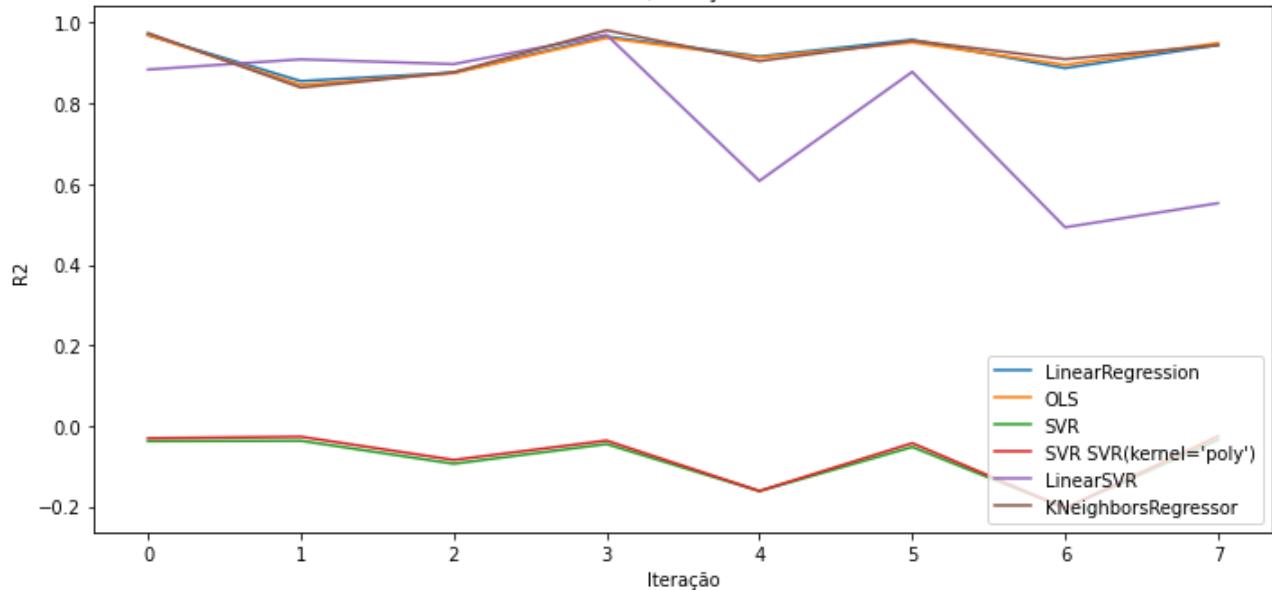
MAPE/Iteração



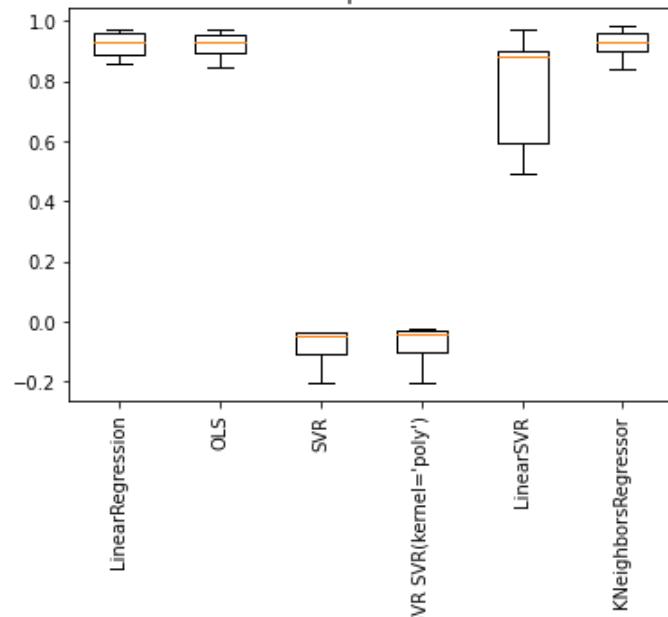
```
plota_metrica(resp_8_random_fold, modelos, r2_score, 'R2/Iteração', 'Iteração', 'R2', loc='left')
```

```
LinearRegression [[0, 0.9692171502935438], [1, 0.8552737502566086], [2, 0.8762173718038
OLS [[0, 0.9697267970977265], [1, 0.8458773318084549], [2, 0.8752300340102029], [3, 0.9
SVR [[0, -0.037602836595818534], [1, -0.03660999589626712], [2, -0.09369530443161933],
SVR SVR(kernel='poly') [[0, -0.0305075397559873], [1, -0.02662486826228405], [2, -0.084
LinearSVR [[0, 0.8838694095705202], [1, 0.9092650236661118], [2, 0.897429196109125], [3
KNeighborsRegressor [[0, 0.9743622785259336], [1, 0.8391140841300024], [2, 0.8771207762
```

R2/Iteração



Boxplots R2



Avaliando os gráficos dos resultados usando 10 e 8 folds, podemos observar o seguinte:

- Os modelos Linear Regression, OLS, KNR e SVR linear continuaram superiores aos SVR com kernels rbf e polinomial;
- No R2, os modelos Linear Regression, OLS, KNR e SVR linear melhoraram se aproximando de 1;
- O modelo SVR linear demonstrou ser mais sensível a quantidade e qualidade dos dados. Observe que os resultados do SVR linear variam bem mais que os modelos Linear Regression e KNR quando usamos 10 folds, o que implica mais dados de treinamento a cada iteração.

Vamos optar por realizar a avaliação com teste de hipótese das diferenças entre os modelos usando o 8-Fold Cross Validation aleatório.

▼ Teste de Hipóteses

Apesar de criarmos diferentes gráficos e computar diferentes métricas, para afirmarmos que um modelo é melhor que outro precisamos realizar um teste de hipótese. Vamos realizar o teste de hipótese com o auxílio da biblioteca [autorank](#), a partir da métricas MSE calculadas com o 8-Fold Cross Validation aleatório.

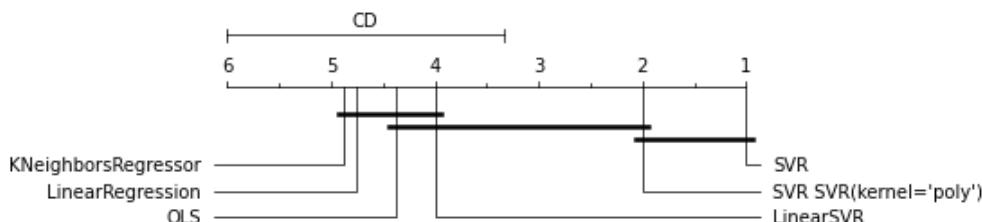
```
from autorank import autorank, plot_stats, create_report, latex_table

resp_dict = {}
for nome_modelo in modelos:
    modelo = resp_8_random_fold[str(nome_modelo)]
    resp = []
    for i, d in enumerate(zip(modelo['reais'], modelo['preditos'])):
        real, predito = d
        resp.append(mean_squared_error(real, predito))
    resp_dict[nome_display_modelo(str(nome_modelo))] = resp

result = autorank(pd.DataFrame.from_dict(resp_dict), alpha=0.05, verbose=False)
create_report(result)
```

The statistical analysis was conducted for 6 populations with 8 paired samples.
The family-wise significance level of the tests is alpha=0.050.
We rejected the null hypothesis that the population is normal for the population LinearRegression.
Because we have more than two populations and the populations and one of them is not normal.
We reject the null hypothesis (p=0.000) of the Friedman test that there is no difference.
Based on the post-hoc Nemenyi test, we assume that there are no significant differences.

```
plot_stats(result)
plt.show()
```



▼ Avaliação em dados de Testes

Nesse passo dos experimentos, os modelos escolhidos serão treinados em todo o corpo de treinamento e avaliados sobre os dados de testes.

```

X_test = teste[['BPC', 'FPM', 'BOLSA FAMILIA']].values
y_test = teste[['AUXILIO EMERGENCIAL']].values

modelos = [LinearRegression(), OLSWrapper(), LinearSVR(max_iter=1000), KNeighborsRegressor(n_
resp_teste = {}
for modelo in modelos:
    if str(modelo) not in resp_teste.keys():
        resp_teste[str(modelo)] = {'preditos': None, 'reais': None}

    modelo.fit(X_train, y_train.ravel())
    y_predict = modelo.predict(X_test)

    resp_teste[str(modelo)]['preditos'] = y_predict.ravel()
    resp_teste[str(modelo)]['reais'] = y_test.ravel()
    resp_teste[str(modelo)]['modelo'] = modelo

/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:986: ConvergenceWarning: Li
    "the number of iterations.", ConvergenceWarning)

def plota_metrica_teste(resultados, modelos, metrica, titulo, eixo_x, eixo_y, loc='upper left'
    resp = []
    fig = plt.figure(figsize=(10, 5))
    plt.subplot(1, 1, 1)

    boxplots_data = []
    boxplots_ticks = []

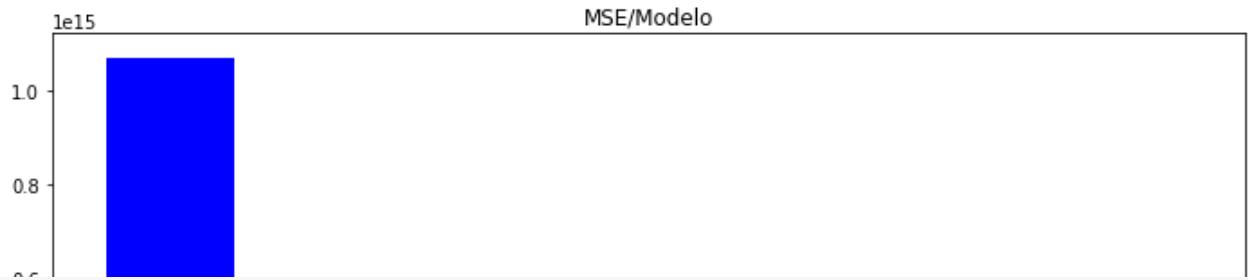
    for nome_modelo in modelos:
        modelo = resultados[str(nome_modelo)]
        nome_display = nome_display_modelo(str(nome_modelo))
        resp.append([nome_display, metrica(modelo['reais'], modelo['preditos'])])

    plt.bar([x[0] for x in resp], [x[1] for x in resp], color ='blue',
            width = 0.4)

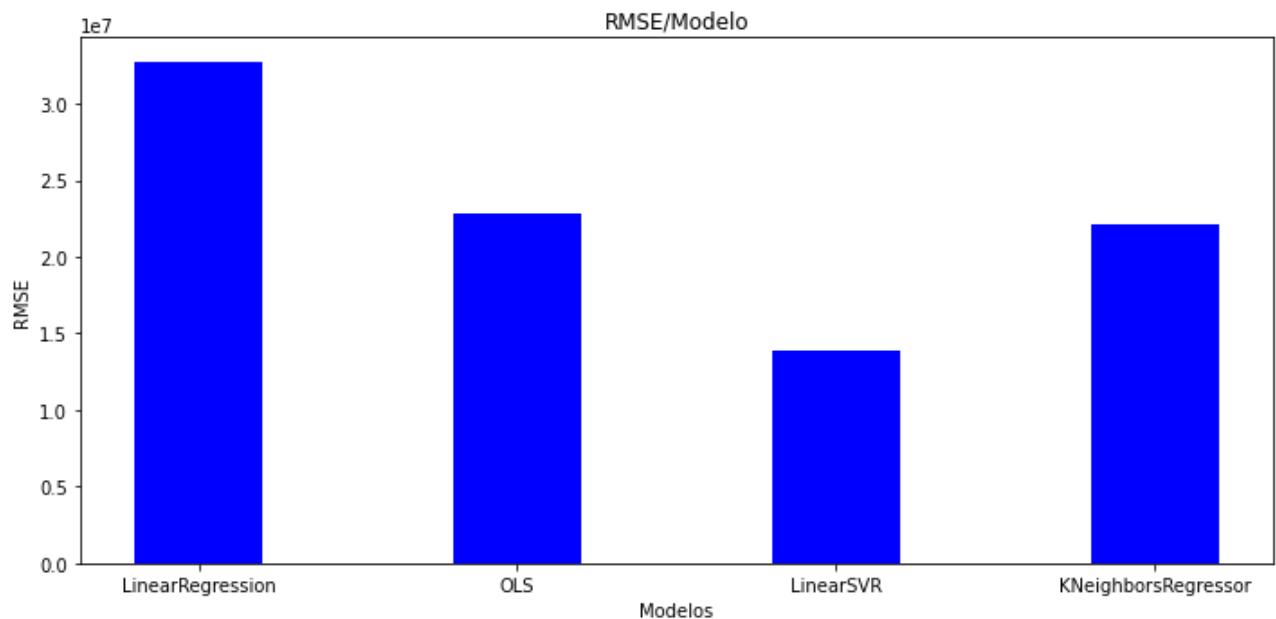
    plt.title(titulo)
    plt.xlabel(eixo_x)
    plt.ylabel(eixo_y)
    fig.tight_layout()
    plt.show()

plota_metrica_teste(resp_teste, modelos, mean_squared_error, 'MSE/Modelo', 'Modelos', 'MSE',

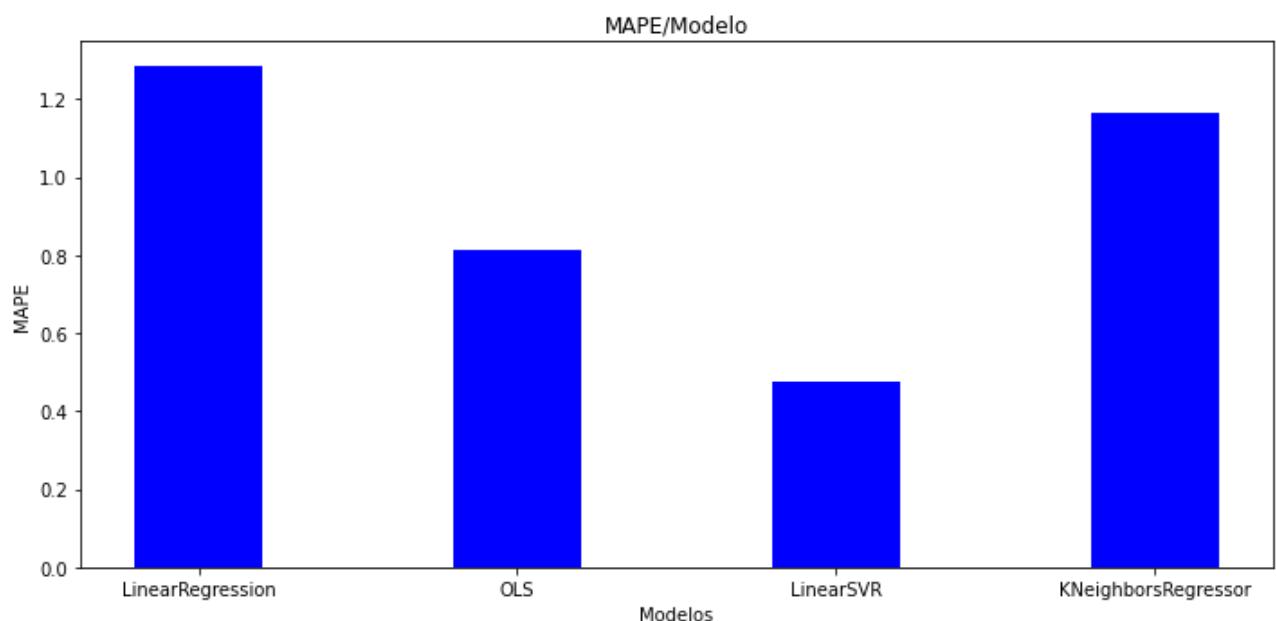
```



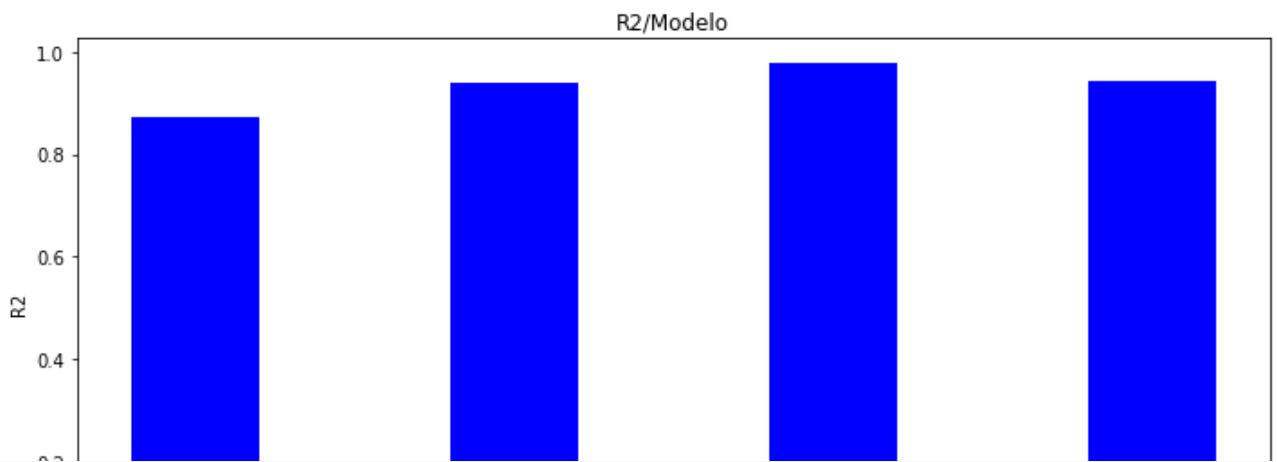
```
plota_metrica_teste(resp_teste, modelos, rmse, 'RMSE/Modelo', 'Modelos', 'RMSE', loc='lower')
```



```
plota_metrica_teste(resp_teste, modelos, mean_absolute_percentage_error, 'MAPE/Modelo', 'Modelos', 'MAPE', loc='lower')
```

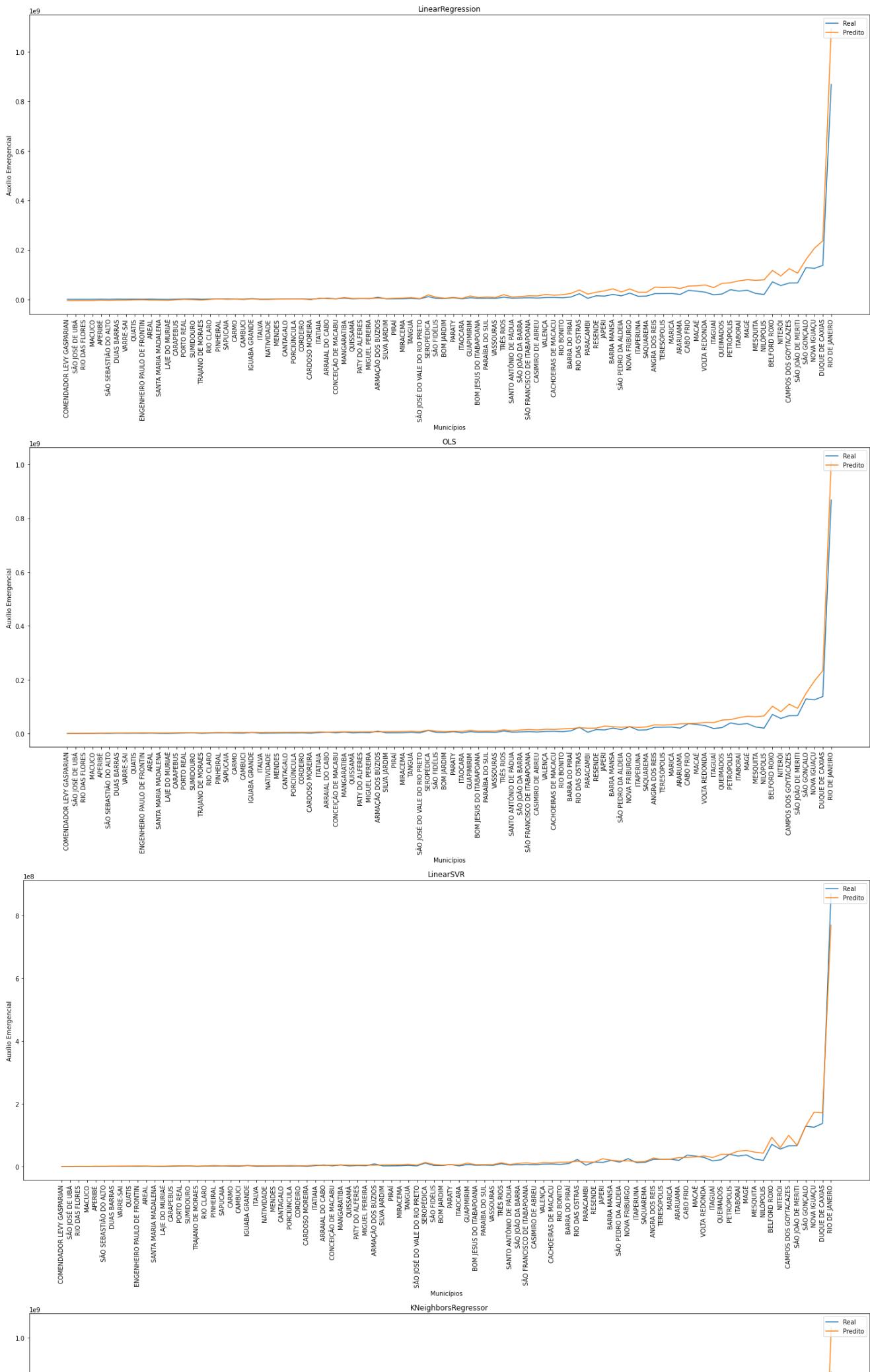


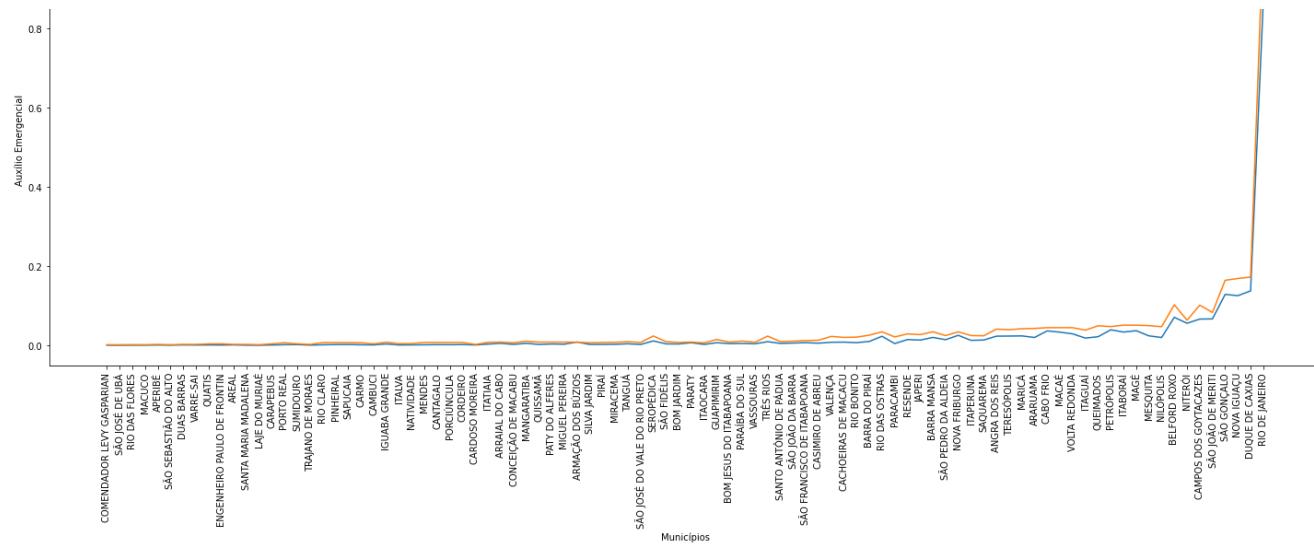
```
plota_metrica_teste(resp_teste, modelos, r2_score, 'R2/Modelo', 'Modelos', 'R2', loc='lower')
```



```
def plota_real_predito(dados_modelo, nome_modelo):
    fig = plt.figure(figsize=(20, 10))
    plt.subplot(1, 1, 1)
    plt.plot(teste['Município'], modelo['reais'],
             label='Real')
    plt.plot(teste['Município'], modelo['preditos'],
             label='Predito')
    plt.title(nome_modelo)
    plt.legend(loc='upper right')
    plt.xlabel('Municípios')
    plt.xticks(rotation='90')
    plt.ylabel('Auxílio Emergencial')
    fig.tight_layout()
    plt.show()
```

```
for nome_modelo in modelos:
    modelo = resp_teste[str(nome_modelo)]
    nome_display = nome_display_modelo(str(nome_modelo))
    plota_real_predito(resp_teste[str(nome_modelo)], nome_display)
```





▼ Análise do OLS induzido com todo o conjunto de treinamento

```
resp_teste['OLS'][['modelo']].model.summary()
```

OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.958			
Model:	OLS	Adj. R-squared (uncentered):	0.958			
Method:	Least Squares	F-statistic:	5591.			
Date:	Sun, 01 Aug 2021	Prob (F-statistic):	0.00			
Time:	14:18:36	Log-Likelihood:	-13543.			
No. Observations:	736	AIC:	2.709e+04			
Df Residuals:	733	BIC:	2.710e+04			
Df Model:	3					
Covariance Type: nonrobust						
coef std err t P> t [0.025 0.975]						
x1	6.9257	0.290	23.901	0.000	6.357	7.495
x2	0.2822	0.498	0.567	0.571	-0.695	1.259
x3	3.6995	0.683	5.420	0.000	2.359	5.040
Omnibus:	380.990	Durbin-Watson:	1.700			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	185962.733			
Skew:	0.938	Prob(JB):	0.00			
Kurtosis:	80.849	Cond. No.	12.4			

A partir da tabela anterior, temos que os valores de coeficiente obtidos pela Regressão Linear com o modelo OLS do **statsmodel** assim como seus respectivos p-valores são:

Coefficiente	Mapeamento	Valor	p-valor
x1	BPC	6.9257	0.000
x2	FPM	0.2822	0.571
x3	BOLSA FAMILIA	3.6995	0.000

Analizando os dados destacados pela tabela acima, percebemos que os valores de **FPM** não são significativos para o modelo pois possui um p-valor > 5%. Neste caso, o ideal a ser feito é induzir novamente o modelo sem essa variável.

```

x_train_sem_fpm = np.delete(X_train, [1], axis=1)
x_test_sem_fpm = np.delete(X_test, [1], axis=1)
modelo = OLSWrapper()
modelo.fit(x_train_sem_fpm, y_train.ravel())
y_predict = modelo.predict(x_test_sem_fpm)

ols_pred = y_predict.ravel()
ols_real = y_test.ravel()

modelo.model.summary()

```

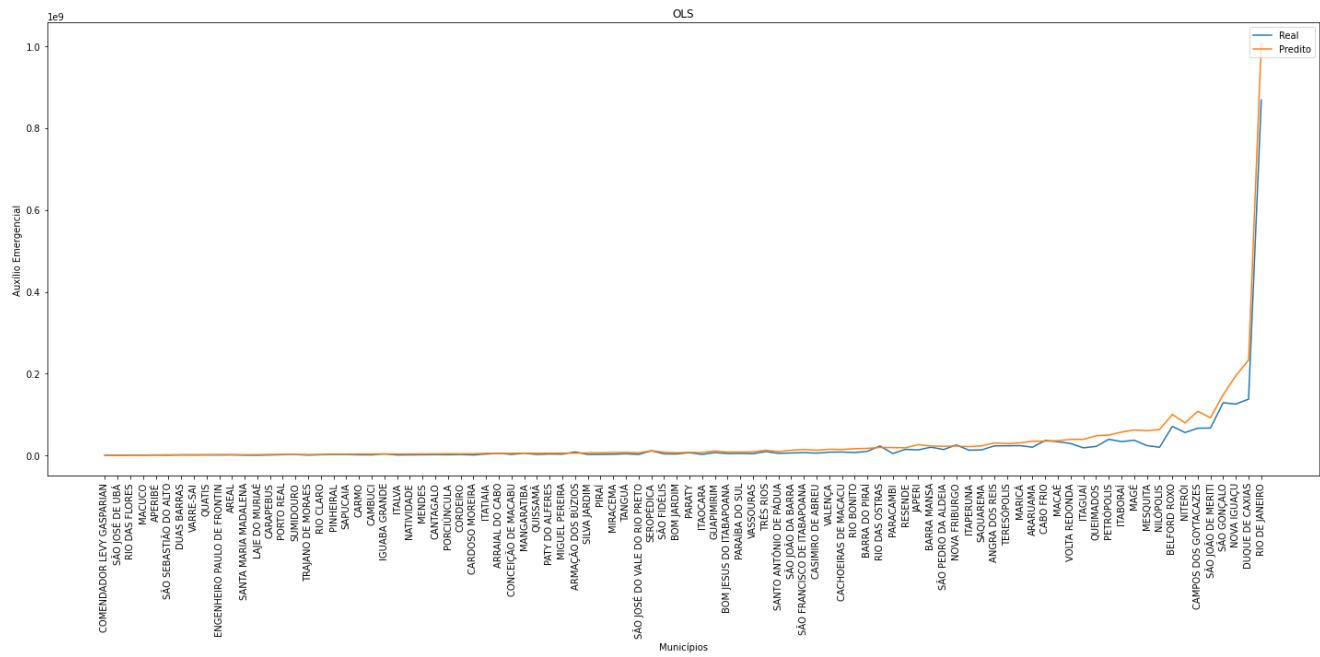
OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.958
Model:	OLS	Adj. R-squared (uncentered):	0.958
Method:	Least Squares	F-statistic:	8394.
Date:	Sun, 01 Aug 2021	Prob (F-statistic):	0.00
Time:	14:18:45	Log-Likelihood:	-13543.
No. Observations:	736	AIC:	2.709e+04
Df Residuals:	734	BIC:	2.710e+04

```

fig = plt.figure(figsize=(20, 10))
plt.subplot(1, 1, 1)
plt.plot(teste['Município'], ols_real,
         label='Real')
plt.plot(teste['Município'], ols_pred,
         label='Predito')
plt.title('OLS')
plt.legend(loc='upper right')
plt.xlabel('Municípios')
plt.xticks(rotation='90')
plt.ylabel('Auxílio Emergencial')
fig.tight_layout()
plt.show()

```



Note que os valores dos coeficientes foram levemente alterados. Daí podemos escrever a função de regressão induzida pelo modelo como:

$$AUXILIO_EMERGENCIAL = 6.9475 \cdot BPC + 3.7869 \cdot BOLSA_FAMILIA$$

Análise do modelo de regressão induzido com o Linear Regression do sklearn

```
print('Coeficientes: ', resp_teste['LinearRegression()']['modelo'].coef_)
print('Constante: ', resp_teste['LinearRegression()']['modelo'].intercept_)
```

```
Coeficientes: [6.25515798 3.81634711 4.03283863]
Constante: -9182542.084970292
```

$$AUXILIO_EMERGENCIAL = 6.25515798 \cdot BPC + 3.81634711 \cdot FPM + \\ 4.03283863 \cdot BOLSA_FAMILIA - 9182542.084970292$$

É interessante notar que o coeficiente relativo ao BPC manteve a mesma faixa de valor quando comparado ao valor obtido com o OLS do statsmodel. Enquanto isso, o coeficiente relativo ao BOLSA FAMILIA teve sua diferença menor que a diferença entre o BPC deste modelo para o modelo do OLS. Por fim, cabe ressaltar que o uso da variável FPM com um valor significativo resultou na adição de uma constante negativa de grande ordem.

Análise do SVR

```
modelo = resp_teste['LinearSVR()']['modelo']
print('Coeficientes: ', modelo.coef_)
print('Constante: ', modelo.intercept_)
```

```
Coeficientes: [3.99171416 0.20234867 5.99001263]
Constante: [-0.09877027]
```

$$AUXILIO_EMERGENCIAL = 1.02850565 \cdot BPC + 0.44230761 \cdot FPM + \\ 4.26802729 \cdot BOLSA_FAMILIA - 0.09803584$$

Note como a função de regressão induzida pelo Linear SVR dá mais peso para o BOLSA_FAMILIA, diminuindo significativamente o peso do BPC, o que reflete numa constante negativa bem menor também.

✓ 0s conclusão: 11:19

