Desafio Data Science

Nome: Luan Donizete Faria

2024

Sumário

QUESTÃO:

Para desenvolver as análises propostas, você deverá utilizar o dadosteste.csv recebido juntamente com esse desafio. Ele contém informações de TCH (Toneladas de cana-de-açúcar por hectare) para diversos talhões de plantação de cana-de-açúcar e para diferentes safras de colheita desses talhões.

O significado de cada coluna do arquivo é explicado a seguir:

• bloco: índice da região de plantação e colheita de cana-de-açúcar

• talhao: índice da sub-região de plantação e colheita de cana-de-açúcar (Um bloco contém diversos talhões);

• area: área do talhão;

• safra: ano que a cana-de-açúcar de cada talhão foi colhida;

• data_colheita: data em que a cana-de-açúcar foi colhida;

• TCH: Toneladas de cana-de-açúcar colhida por hectare;

• NDVI_b01: NDVI é o nome dado a um popular índice de vegetação, e o "b01" corresponde ao índice no primeiro mês antes da colheita;

• NDVI_bN: NDVI no N-ésimo mês antes da colheita;

Análises propostas:

a) Faça uma rápida análise exploratória desse dataset, por exemplo análises estatísticas das variáveis, gráficos de distribuição etc.

b) Desenvolva um modelo capaz de estimar o TCH dos blocos do conjunto de dados para a safra de 2019 (atenção: que a previsão deve ser no nível de blocos e não de talhões)

c) Qual o erro esperado do modelo para a safra de 2019? Utilize a métrica de erro que você julgar necessária, podendo ser utilizada mais de uma.

1 – TRATANDO A BASE DE DADOS

1- Abrindo a base de dados “dadosteste.**CSV**”

```
[5]: #IMPORTANDO BASE DE DADOS
bd_agro = pd.read_csv('dadosteste.csv')
bd_agro
#display(bd_agro)
```

[5]:

	bloco	talhao	area	safra	data_colheita	TCH	NDVI_b01	NDVI_b02	NDVI_b03	NDVI_b04	NDVI_b05	NDVI_b06	NDVI_b07	NDVI_b08	NDVI_b09	N
0	475	1	108.77	2019	2019-06-07	NaN	0.682403	0.759308	0.750978	0.770751	0.751562	0.697348	0.692488	0.521619	0.361398	
1	473	2	95.81	2019	2019-08-07	NaN	0.469399	0.592628	0.672298	0.747850	0.715270	0.722980	0.731796	0.731084	0.695648	
2	473	1	123.98	2019	2019-08-07	NaN	0.428521	0.563982	0.674662	0.770703	0.739079	0.746980	0.700396	0.671140	0.620945	
3	468	5	49.18	2019	2019-05-22	NaN	0.741138	0.747431	0.767039	0.580162	0.761972	0.651611	0.629768	0.460629	0.389895	
4	468	4	39.58	2019	2019-05-22	NaN	0.731495	0.739824	0.767489	0.576364	0.759941	0.651823	0.620141	0.411033	0.315381	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1901	240	1	43.80	2018	2018-09-02	70.929600	0.505531	0.541958	0.593755	0.670426	0.749030	0.763200	0.636833	0.736960	0.407364	
1902	219	2	56.51	2018	2018-08-31	91.035850	0.479797	0.545902	0.605372	0.666031	0.752323	0.787244	0.714353	0.789022	0.637088	
1903	474	1	32.31	2018	2018-06-24	92.629189	0.519261	0.649099	0.761562	0.801162	0.776165	0.712880	0.687334	0.559766	0.543474	

2- Observar as informações apresentadas nos tipos de dados fornecidos

```
[8]: bd_agro.info
```

```
[8]: <bound method DataFrame.info of       bloco  talhao    area  safra data_colheita        TCH  NDVI_b01  \
0       475       1  108.77   2019    2019-06-07        NaN  0.682403
1       473       2   95.81   2019    2019-08-07        NaN  0.469399
2       473       1  123.98   2019    2019-08-07        NaN  0.428521
3       468       5   49.18   2019    2019-05-22        NaN  0.741138
4       468       4   39.58   2019    2019-05-22        NaN  0.731495
...     ...     ...     ...    ...           ...        ...       ...
1901    240       1   43.80   2018    2018-09-02  70.929600  0.505531
1902    219       2   56.51   2018    2018-08-31  91.035850  0.479797
1903    474       1   32.31   2018    2018-06-24  92.629189  0.519261
1904    223       1   32.72   2018    2018-09-03  60.958865  0.428690
1905    223       2  114.63   2018    2018-09-03  64.047401  0.421489

      NDVI_b02  NDVI_b03  NDVI_b04  NDVI_b05  NDVI_b06  NDVI_b07  NDVI_b08  \
0     0.759308  0.750978  0.770751  0.751562  0.697348  0.692488  0.521619
1     0.592628  0.672298  0.747850  0.715270  0.722980  0.731796  0.731084
2     0.563982  0.674662  0.770703  0.739079  0.746980  0.700396  0.671140
3     0.747431  0.767039  0.580162  0.761972  0.651611  0.629768  0.460629
4     0.739824  0.767489  0.576364  0.759941  0.651823  0.620141  0.411033
...        ...       ...       ...       ...       ...       ...       ...
1901  0.541958  0.593755  0.670426  0.749030  0.763200  0.636833  0.736960
1902  0.545902  0.605372  0.666031  0.752323  0.787244  0.714353  0.789022
1903  0.649099  0.761562  0.801162  0.776165  0.712880  0.687334  0.559766
1904  0.443692  0.439647  0.554221  0.652509  0.695369  0.649948  0.709126
1905  0.449245  0.471967  0.573620  0.674816  0.710919  0.683841  0.709820

      NDVI_b09  NDVI_b10  NDVI_b11  NDVI_b12
0     0.361398  0.379454  0.393329  0.449616
1     0.695648  0.511374  0.362315  0.423441
2     0.620945  0.465249  0.305978  0.300977
3     0.389895  0.426775  0.495159  0.507772
4     0.315381  0.374749  0.468938  0.485989
...        ...       ...       ...       ...
1901  0.407364  0.676639  0.550063  0.534069
1902  0.637088  0.722117  0.553598  0.518688
1903  0.543474  0.545682  0.615032  0.628805
1904  0.709535  0.604181  0.470840  0.441193
1905  0.395012  0.592119  0.468342  0.427731
```

3- Agora com o objetivo de encontrar linhas/colunas com informações “vazias”, utilizou-se o comando **.isna().sum()**.

```
[14]: bd_agro.isna().sum()
```

```
[14]: bloco           0
      talhao          0
      area            0
      safra           0
      data_colheita   0
      TCH           274
      NDVI_b01        0
      NDVI_b02        0
      NDVI_b03        0
      NDVI_b04        0
      NDVI_b05        0
      NDVI_b06        0
      NDVI_b07        0
      NDVI_b08        0
      NDVI_b09        0
      NDVI_b10        0
      NDVI_b11        0
      NDVI_b12        0
      dtype: int64
```

No caso em questão foi possível encontrar 274 células vazias para as informações de TCHs.

4- Com o objetivo de observar um com um pouco mais detalhe essa informação, utilizou-se o comando **.info()** apenas para a coluna TCH

```
[12]: bd_agro['TCH'].info()

<class 'pandas.core.series.Series'>
RangeIndex: 1906 entries, 0 to 1905
Series name: TCH
Non-Null Count  Dtype
--------------  -----
1632 non-null   float64
dtypes: float64(1)
memory usage: 15.0 KB
```

Agora podemos observar o tipo da variável, no caso o TCH esta como float64 (números flutuantes), além de mostrar o total de linhas (1906) e o total de linhas **NÃO** vazias (1632). A diferença entre os dois valores serias as linhas vazias que já encontramos anteriormente (1906-1632=274).

5- Em seguida foi realizado um tratamento com alguns caracteres que podem ser considerados erros de preenchimento, transformando-os em dados vazios (“vazio/NaN”).

```
[15]: #TRATAMENTO DE IBNFORMAÇÕES VAZIAS
valores_vazios=['-', '',' ','_','*']
bd_agro = pd.read_csv('dadosteste.csv', na_values=valores_vazios)
```

6- Esses dados vazios foram preenchidos com o valor “0”, utilizando o comando **.fillna**

```
[16]: bd_agro['TCH'] = bd_agro['TCH'].fillna(0)
bd_agro['TCH']

[16]: 0        0.000000
1        0.000000
2        0.000000
3        0.000000
4        0.000000
           ...
1901    70.929600
1902    91.035850
1903    92.629189
1904    60.958865
1905    64.047401
Name: TCH, Length: 1906, dtype: float64
```

7- E para finalizar nosso tratamento da base, foi removido a coluna data, pois estava registrada como string, e como acredito que não vá influenciar diretamente nos meus resultados optei pela remoção.

```
[53]: bd_agro_sem_data=bd_agro.drop(labels="data_colheita", axis=1)
      bd_agro_sem_data.info(20)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1906 entries, 0 to 1905
Data columns (total 17 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   bloco     1906 non-null   int64
 1   talhao    1906 non-null   int64
 2   area      1906 non-null   float64
 3   safra     1906 non-null   int64
 4   TCH       1906 non-null   float64
 5   NDVI_b01  1906 non-null   float64
 6   NDVI_b02  1906 non-null   float64
 7   NDVI_b03  1906 non-null   float64
 8   NDVI_b04  1906 non-null   float64
 9   NDVI_b05  1906 non-null   float64
 10  NDVI_b06  1906 non-null   float64
 11  NDVI_b07  1906 non-null   float64
 12  NDVI_b08  1906 non-null   float64
```

2- DADOS DE CORRELAÇÃO

1- Como o objetivo do trabalho é estimar o TCH da safra de 2019, foram extraídas as informações separando-as em duas planilhas, uma apenas com dados referentes a safra de 2019 e na outra as diversas safras.

```
[26]: safra_2019=bd_agro.loc[bd_agro['safra']==2019]
      outras_safras = bd_agro.loc[bd_agro['safra']!=2019]
      outras_safras
```

[26]:

	bloco	talhao	area	safra	data_colheita	TCH	NDVI_b01	NDVI_b02	NDVI_b03	NDVI_b04	NDVI_b05	NDVI_b06	NDVI_b07	NDVI_b08	NDVI_b09
10	463	6	60.02	2018	2018-07-22	76.739130	0.423142	0.431723	0.559972	0.592814	0.600053	0.681854	0.610406	0.569018	0.425806
12	463	5	82.09	2018	2018-07-22	99.839744	0.431270	0.542980	0.619596	0.657681	0.649257	0.743298	0.669754	0.640421	0.448287
14	463	4	125.36	2018	2018-07-22	107.347368	0.400230	0.532940	0.596022	0.687585	0.727971	0.745753	0.674152	0.647247	0.439842
16	463	3	117.87	2018	2018-07-22	43.585093	0.465889	0.542166	0.622972	0.679962	0.656445	0.651586	0.677720	0.688182	0.492534
18	463	2	121.25	2018	2018-07-22	88.750000	0.474915	0.497241	0.618744	0.650198	0.601881	0.596118	0.656673	0.668282	0.480202
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1901	240	1	43.80	2018	2018-09-02	70.929600	0.505531	0.541958	0.593755	0.670426	0.749030	0.763200	0.636833	0.736960	0.407364
1902	219	2	56.51	2018	2018-08-31	91.035850	0.479797	0.545902	0.605372	0.666031	0.752323	0.787244	0.714353	0.789022	0.637088
1903	474	1	32.31	2018	2018-06-24	92.629189	0.519261	0.649099	0.761562	0.801162	0.776165	0.712880	0.687334	0.559766	0.543474

2- Para diminuir o número de informações, deixei apenas os valores de TCH e NDVI.

```
[143]: analise1 = outras_safras.drop(labels="bloco", axis=1)
       analise2 = analise1.drop(labels="talhao", axis=1)
       analise3 = analise2.drop(labels="area", axis=1)
       analise4 = analise3.drop(labels="safra", axis=1)
       analise4
       #analise = outras_safras[['TCH'],['NDVI_b01'],['NDVI_b02'],['NDVI_b03'],['NDVI_b04'],['NDVI_b05'],['NDVI_b06'],['NDVI_b07'],['NDVI_b08'],['NDVI_b09'],['
```

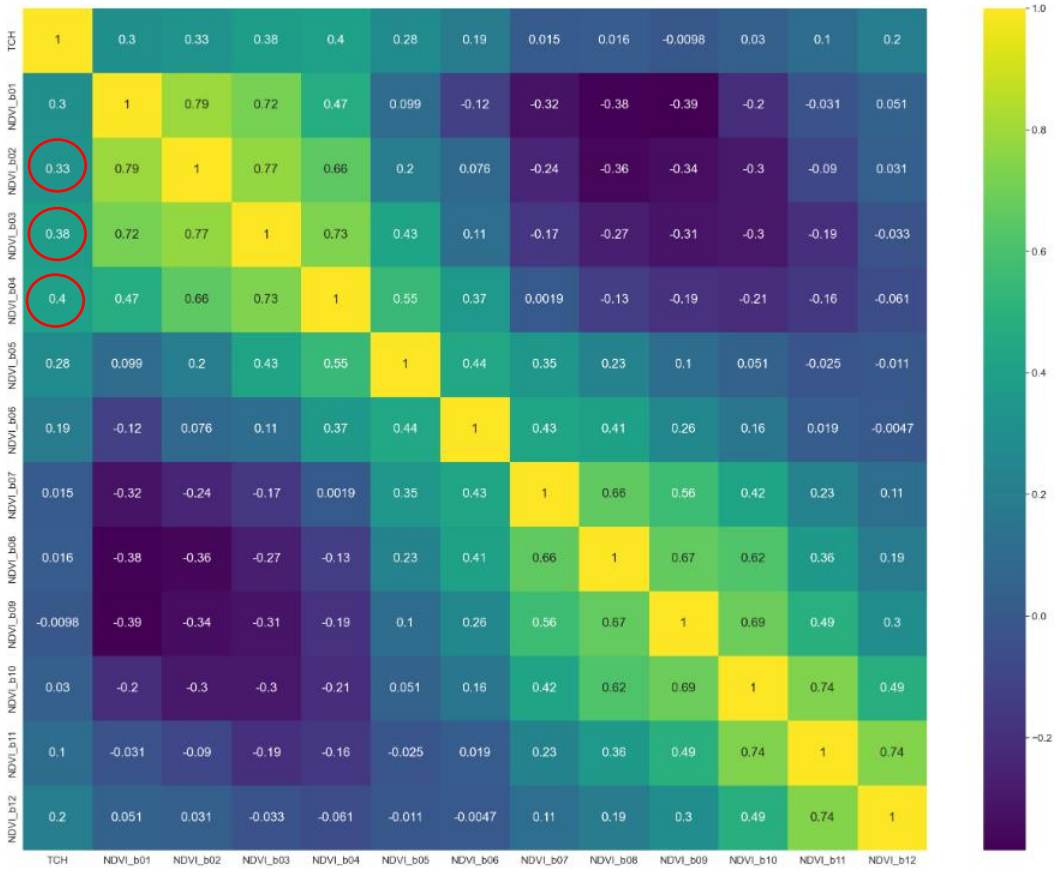
[143]:

	TCH	NDVI_b01	NDVI_b02	NDVI_b03	NDVI_b04	NDVI_b05	NDVI_b06	NDVI_b07	NDVI_b08	NDVI_b09	NDVI_b10	NDVI_b11	NDVI_b12
10	76.739130	0.423142	0.431723	0.559972	0.592814	0.600053	0.681854	0.610406	0.569018	0.425806	0.375776	0.355070	0.406886
12	99.839744	0.431270	0.542980	0.619596	0.657681	0.649257	0.743298	0.669754	0.640421	0.448287	0.381127	0.377751	0.435234
14	107.347368	0.400230	0.532940	0.596022	0.687585	0.727971	0.745753	0.674152	0.647247	0.439842	0.363046	0.360681	0.430465
16	43.585093	0.465889	0.542166	0.622972	0.679962	0.656445	0.651586	0.677720	0.688182	0.492534	0.415071	0.426020	0.517091
18	88.750000	0.474915	0.497241	0.618744	0.650198	0.601881	0.596118	0.656673	0.668282	0.480202	0.428530	0.446913	0.539156
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1901	70.929600	0.505531	0.541958	0.593755	0.670426	0.749030	0.763200	0.636833	0.736960	0.407364	0.676639	0.550063	0.534069
1902	91.035850	0.479797	0.545902	0.605372	0.666031	0.752323	0.787244	0.714353	0.789022	0.637088	0.722117	0.553598	0.518688
1903	92.629189	0.519261	0.649099	0.761562	0.801162	0.776165	0.712880	0.687334	0.559766	0.543474	0.545682	0.615032	0.628805

3- A partir dessa nova Planilha analise4, foi possível criar uma matriz de correlação entre o TCH e os NDVIs utilizando o **.heatmap**.

```
[133]: #Matriz de correlação
fig = pl.gcf()
figsize = fig.get_size_inches()
fig.set_size_inches(figsize * 5.0)
sb.set(font_scale=1.5)
sb.heatmap(analise4.corr(), annot=True,  annot_kws={"size":20 }, cmap="viridis")
```

```
[133]: <Axes: >
```



Pode-se observar que o NDVI_b04 apresentou 40% de relevância quando correlacionado com o aumento de TCH.

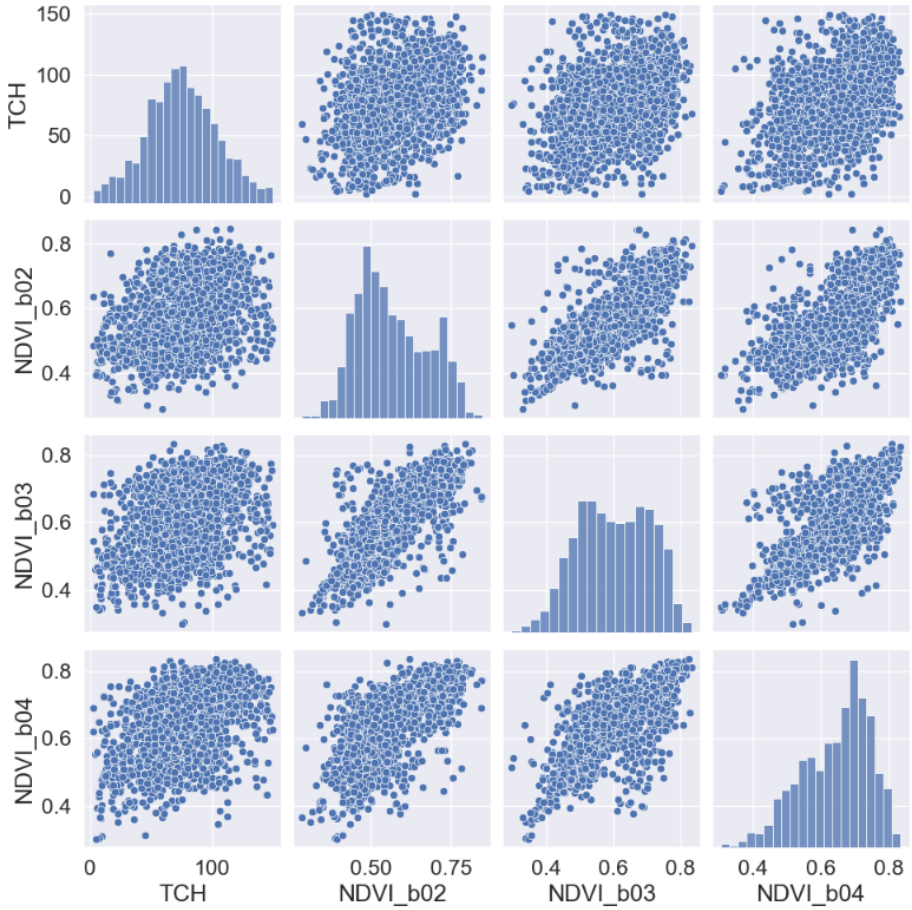
4- A fim de refinar ainda mais a escolha dos dados de input, utilizarei apenas os três dados mais relevantes, NDVI_b02, NDVI_b03 e NDVI_b04.

```
[152]: analise5 = analise4[['TCH','NDVI_b02','NDVI_b03','NDVI_b04']]
analise5
```

[152]:

	TCH	NDVI_b02	NDVI_b03	NDVI_b04
10	76.739130	0.431723	0.559972	0.592814
12	99.839744	0.542980	0.619596	0.657681
14	107.347368	0.532940	0.596022	0.687585
16	43.585093	0.542166	0.622972	0.679962
18	88.750000	0.497241	0.618744	0.650198
...	...	...	...	...
1901	70.929600	0.541958	0.593755	0.670426
1902	91.035850	0.545902	0.605372	0.666031
1903	92.629189	0.649099	0.761562	0.801162

5- A primeira tentativa de se observar padrões foi utilizando gráficos de pontos utilizando o **.pairpoint**, porém não foi possível identificar algo muito significativo, como podemos observar (utilizando já a planilha filtrada).



3- ALGORITMO DE TREINAMENTO

1- Para o algoritmo de treinamento utilizarei regressões, tais como a Linear e a RandomForest. Em primeira mão, foi utilizado 70% dos dados obtidos nas safras anteriores a 2019 como forma de treinamento, e os outros 30% foi utilizado como teste para validarmos o algoritmo.

```
[275]: x = analise4.drop('TCH', axis=1)
       y = analise4['TCH']
       x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.10, random_state=1)
```

2- Treinamento IA

```
[157]: # treino AI
lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)

rf_reg = RandomForestRegressor()
rf_reg.fit(x_train, y_train)
```

```
[157]: RandomForestRegressor
RandomForestRegressor()
```

3- O primeiro teste foi o cálculo do R^2, que como observado foi baixo para a Linear, e um pouco "melhor" para a RandomForest.

```
[336]: test_pred_lin = lin_reg.predict(x_test)
test_pred_rf = rf_reg.predict(x_test)

r2_lin = metrics.r2_score(y_test, test_pred_lin)
print(f"R² da Regressão Linear: {r2_lin}")
r2_rf= metrics.r2_score(y_test, test_pred_rf)
print(f"R² do Random Forest: {r2_rf}")
```

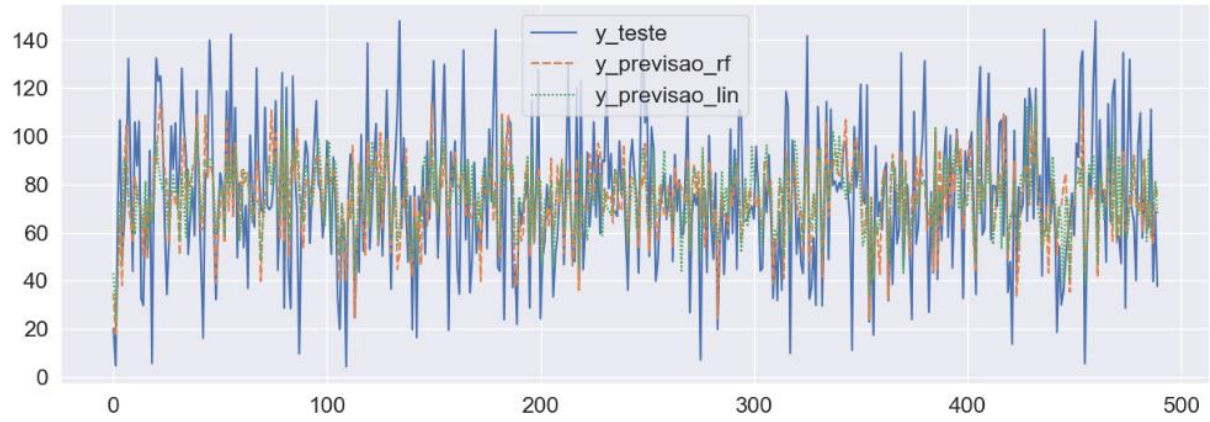
```
R² da Regressão Linear: 0.3304928164263895
R² do Random Forest: 0.44929119539819906
```

4- Foi realizado um comparativo de linhas, comparando os dados teste e as duas regressões.

```
[296]: df_resultado = pd.DataFrame()

# df_resultado.index = x_test
df_resultado['y_teste'] = y_test
df_resultado['y_previsao_rf'] = test_pred_rf
df_resultado['y_previsao_lin'] = test_pred_lin

# display(df_resultado)
df_resultado = df_resultado.reset_index(drop=True)
pl.figure(figsize=(15, 5))
sb.lineplot(data=df_resultado)
pl.show()
display(df_resultado.head(20))
```



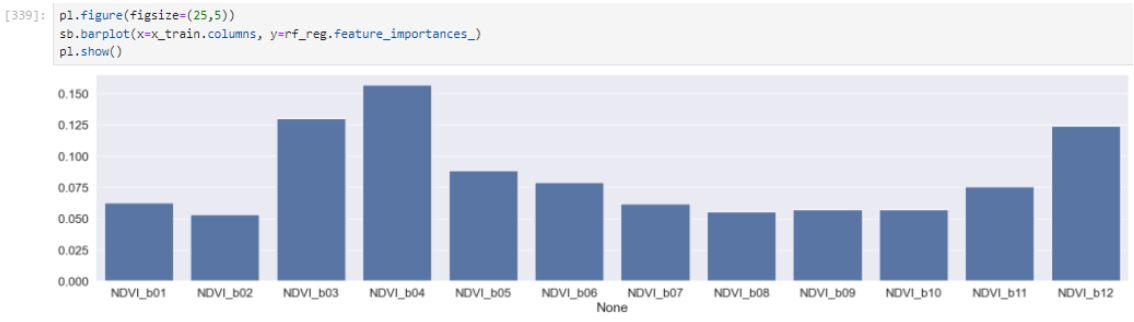
A linha azul representa os dados do teste, a laranja a regressão randomforest e a verde a regressão linear.

Segue a tabela dos 20 primeiros resultados apresentados

	y_teste	y_previsao_rf	y_previsao_lin
0	69.380242	52.058922	44.953379
1	78.636202	85.814188	82.249830
2	62.726115	84.067557	91.224031
3	107.850352	109.387881	99.503599
4	90.505274	86.807980	94.329503
5	97.978231	95.884849	90.890639
6	77.450117	88.010391	81.634496
7	51.610891	73.180067	74.009666
8	77.042017	76.836416	94.924881
9	98.685602	85.536695	75.913803
10	74.680698	81.729676	98.281446
11	78.220933	71.934765	77.375927
12	57.646583	75.985840	75.566529
13	132.367178	88.389187	97.074026
14	65.685498	76.499362	69.020937
15	96.897346	85.343897	83.850804
16	64.511401	69.430415	77.927221
17	71.191525	64.911864	96.611014
18	110.948621	91.513118	78.663420
19	59.508502	76.967419	78.591160

Vale salientar que esses resultados ainda não estão agrupados por blocos.

Podemos observar a ordem de relevância das variáveis utilizadas. Observa-se que a b4 e b3 mantem sua importância como na matriz de correlação, porem ocorre o surgimento da b12 como um dado inconsistente.



4-TCH ESTIMADO 2019 POR BLOCO

1- Aplicar o modelo na planilha 2019 (vale salientar que nessa etapa a analise ainda não esta sendo realizada por bloco)

2- Filtro de alguns valores

```
[359]: safra_1 = safra_2019.drop(labels="bloco", axis=1)
       safra_2  = safra_1 .drop(labels="talhao", axis=1)
       safra_3  = safra_2 .drop(labels="area", axis=1)
       safra_4  = safra_3 .drop(labels="safra", axis=1)
       safra_4
```

[359]:

	TCH	NDVI_b01	NDVI_b02	NDVI_b03	NDVI_b04	NDVI_b05	NDVI_b06	NDVI_b07	NDVI_b08	NDVI_b09	NDVI_b10	NDVI_b11	NDVI_b12
0	0.0	0.682403	0.759308	0.750978	0.770751	0.751562	0.697348	0.692488	0.521619	0.361398	0.379454	0.393329	0.449616
1	0.0	0.469399	0.592628	0.672298	0.747850	0.715270	0.722980	0.731796	0.731084	0.695648	0.511374	0.362315	0.423441
2	0.0	0.428521	0.563982	0.674662	0.770703	0.739079	0.746980	0.700396	0.671140	0.620945	0.465249	0.305978	0.300977
3	0.0	0.741138	0.747431	0.767039	0.580162	0.761972	0.651611	0.629768	0.460629	0.389895	0.426775	0.495159	0.507772
4	0.0	0.731495	0.739824	0.767489	0.576364	0.759941	0.651823	0.620141	0.411033	0.315381	0.374749	0.468938	0.485989
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
398	0.0	0.646574	0.686413	0.691365	0.641561	0.640327	0.474955	0.578015	0.471346	0.378170	0.367619	0.453014	0.487394
399	0.0	0.610348	0.636207	0.643427	0.606883	0.600883	0.470353	0.509756	0.437162	0.366271	0.348088	0.421239	0.456834
400	0.0	0.628287	0.651012	0.653879	0.635426	0.619584	0.502318	0.541027	0.459068	0.381929	0.367744	0.438572	0.472849
401	0.0	0.561317	0.611405	0.674954	0.662746	0.665600	0.525441	0.595787	0.593090	0.375706	0.471425	0.384695	0.375208
402	0.0	0.688200	0.718599	0.732889	0.700157	0.702455	0.479158	0.550311	0.444463	0.409690	0.427989	0.486711	0.501093

274 rows × 13 columns

3- Aplicando o modelo de regressão randomforest

```
[364]: x1 = safra_4.drop('TCH', axis=1)
       y1 = safra_4['TCH']
       #x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.10, random_state=100)

[375]: tch=rf_reg.predict(x1)
```

4- TCH já estimado por talhão (matriz para lista)

```
[381]: #result = pd.concat([TCH, y1], axis=1, sort=False)
       df_resultado = pd.DataFrame()

       # df_resultado.index = x_test
       #df_resultado['y_teste'] = y_test
       df_resultado['y_previsao_rf'] = tch
       #df_resultado['y_previsao_lin'] = test_pred_lin

       # display(df_resultado)
       #df_resultado = df_resultado.reset_index(drop=True)
       #pl.figure(figsize=(15, 5))
       #sb.lineplot(data=df_resultado)
       pl.show()
       display(df_resultado)
```

	y_previsao_rf
0	97.378939
1	77.577104
2	80.576429
3	100.197362
4	99.277738
...	...
269	80.916386
270	85.433371
271	89.447324
272	83.169394
273	86.946375

274 rows × 1 columns

5- Concatenar entra a planilha 2019 e a coluna de resultado de TCH

```
[421]: Resultado = pd.concat([df_resultados, safra_19], axis=1, sort=False)
       Resultado.head(20)
```

[421]:

	y_previsao_rf	bloco	talhao	area	safra	data_colheita	TCH	NDVI_b01	NDVI_b02	NDVI_b03	NDVI_b04	NDVI_b05	NDVI_b06	NDVI_b07	NDVI_b08	NDVI_
0	97.378939	475	1	108.77	2019	2019-06-07	NaN	0.682403	0.759308	0.750978	0.770751	0.751562	0.697348	0.692488	0.521619	0.36
1	77.577104	473	2	95.81	2019	2019-08-07	NaN	0.469399	0.592628	0.672298	0.747850	0.715270	0.722980	0.731796	0.731084	0.69
2	80.576429	473	1	123.98	2019	2019-08-07	NaN	0.428521	0.563982	0.674662	0.770703	0.739079	0.746980	0.700396	0.671140	0.62
3	100.197362	468	5	49.18	2019	2019-05-22	NaN	0.741138	0.747431	0.767039	0.580162	0.761972	0.651611	0.629768	0.460629	0.38
4	99.277738	468	4	39.58	2019	2019-05-22	NaN	0.731495	0.739824	0.767489	0.576364	0.759941	0.651823	0.620141	0.411033	0.31
5	98.877353	468	3	61.38	2019	2019-05-22	NaN	0.758227	0.763782	0.766416	0.553745	0.782921	0.590853	0.659716	0.466268	0.34
6	77.542840	464	3	60.56	2019	2019-05-06	NaN	0.667383	0.741771	0.592049	0.733495	0.664589	0.664214	0.510332	0.422210	0.49
7	88.025841	464	2	79.12	2019	2019-05-06	NaN	0.684474	0.758803	0.614511	0.736995	0.670889	0.691901	0.515743	0.430748	0.51
8	75.275497	464	1	124.61	2019	2019-05-06	NaN	0.680032	0.746286	0.610911	0.717225	0.630382	0.654919	0.477820	0.397485	0.46

```
[ ]:
```

6- Separando TCH e bloco para poder reagrupa-los novamente

```
[425]: Resultado_Fim = Resultado[['y_previsao_rf','bloco']]
       Resultado_Fim
```

[425]:

	y_previsao_rf	bloco
0	97.378939	475
1	77.577104	473
2	80.576429	473
3	100.197362	468
4	99.277738	468
...	...	...
269	80.916386	17
270	85.433371	17
271	89.447324	17
272	83.169394	16
273	86.946375	13

274 rows × 2 columns

7- E para finalizar, foi agrupado os valores de tch MÉDIO por bloco (115 linhas).

```
[457]: ACABOU= Resultado_Fim.groupby('bloco').mean()
       ACABOU.head(200)
```

[457]:

bloco	y_previsao_rf
13	86.946375
16	83.169394
17	83.574473
19	88.404997
20	83.129694
...	...
463	63.271918
464	80.281393
468	99.450818
473	79.076767
475	97.378939

115 rows × 1 columns

5-MÉTRICA DE ERRO

O maior indicativo de erro é o simples fato do NDVI já ser algo que requer cálculo. Esse cálculo já apresenta uma estimativa de erro, portanto quando tenta-se calcular novamente passando por uma rede de arvore de decisões pode ocorrer confusões como as apresentadas no trabalho.