RUSTY BARGAIN

DESCRIÇÃO DO PROJETO

Rusty Bargain é um serviço de venda de carros usados que está desenvolvendo um aplicativo para atrair novos clientes. Nesse aplicativo, você pode descobrir rapidamente o valor de mercado do seu carro. Temos acesso a dados históricos, especificações técnicas, versões de acabamento e preços. É necessário construir o modelo para determinar o valor.

Rusty Bargain está interessado em:

- a qualidade da predição
- a velocidade da predição
- o tempo necessário para o treinamento

DESCRIÇÃO DOS DADOS

Características:

- DateCrawled — data em que o perfil foi baixado do banco de dados
- VehicleType — tipo de carroçaria do veículo
- RegistrationYear — ano de matrícula do veículo
- Gearbox — tipo de caixa de transmissão
- Power — potência (hp)
- Model — modelo do veículo
- Mileage — quilometragem (medida em km devido às especificidades regionais do conjunto de dados)
- RegistrationMonth — mês de registro do veículo
- FuelType — tipo de combustível
- Brand — marca do veículo
- NotRepaired — veículo reparado ou não
- DateCreated — data de criação do perfil
- NumberOfPictures — número de fotos do veículo
- PostalCode — código postal do proprietário do perfil (usuário)
- LastSeen — data da última atividade do usuário

Objetivo:

- Price — preço (Euro)

PREPARAÇÃO DOS DADOS

In [6]:

```
import pandas as pd
```

In [2]:

```
df= pd.read_csv("C:/Users/Felipe/Documents/car_data.csv")
```

In [3]:

```
df.head(2)
```

Out[3]:

	DateCrawled	Price	VehicleType	RegistrationYear	Gearbox	Power	Model	Mileage	Regi
0	24/03/2016 11:52	480	NaN	1993	manual	0	golf	150000	
1	24/03/2016 10:58	18300	coupe	2011	manual	190	NaN	125000	

In [4]:

```
# informações do df
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 354369 entries, 0 to 354368
Data columns (total 16 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   DateCrawled        354369 non-null  object
 1   Price              354369 non-null  int64 
 2   VehicleType        316879 non-null  object
 3   RegistrationYear   354369 non-null  int64 
 4   Gearbox            334536 non-null  object
 5   Power              354369 non-null  int64 
 6   Model              334664 non-null  object
 7   Mileage            354369 non-null  int64 
 8   RegistrationMonth  354369 non-null  int64 
 9   FuelType           321474 non-null  object
 10  Brand              354369 non-null  object
 11  NotRepaired        283215 non-null  object
 12  DateCreated        354369 non-null  object
 13  NumberOfPictures   354369 non-null  int64 
 14  PostalCode         354369 non-null  int64 
 15  LastSeen           354369 non-null  object
dtypes: int64(7), object(9)
memory usage: 43.3+ MB
```

ANÁLISE PRIMÁRIA DAS INFORMAÇÕES DO DATAFRAME ORIGINAL:

- Possui 354369 linhas e 16 colunas;
- Tipagem incorreta de algumas colunas, entretanto iremos avaliar se iremos utilizá-las. Caso necessário, será realizado a alteração de tipagem;
- Algumas colunas com valores ausentes;

AVALIAÇÃO DAS COLUNAS NECESSÁRIAS

Algumas colunas já poderemos descartar, pois não utilizaremos no treinamento. São elas:

- Datecrawled;
- DateCreated;
- LastSeen;

Como vamos exluir as colunas de data, não será necessário transformar a tipagem das mesmas

In [5]:

```
# Observando o número de fotos
df['NumberOfPictures'].value_counts()
```

Out[5]:

```
0    354369
Name: NumberOfPictures, dtype: int64
```

Todos os valores da coluna são 0, vamos exluí-la, pois será indiferente.

In [6]:

```
# exclusão das colunas
df.drop(['DateCrawled','DateCreated','LastSeen','NumberOfPictures'],axis= 1, inplace= Tru
```

In [7]:

```
# revisando as informações
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 354369 entries, 0 to 354368
Data columns (total 12 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   Price              354369 non-null  int64 
 1   VehicleType        316879 non-null  object
 2   RegistrationYear   354369 non-null  int64 
 3   Gearbox            334536 non-null  object
 4   Power              354369 non-null  int64 
 5   Model              334664 non-null  object
 6   Mileage            354369 non-null  int64 
 7   RegistrationMonth  354369 non-null  int64 
 8   FuelType           321474 non-null  object
 9   Brand              354369 non-null  object
 10  NotRepaired        283215 non-null  object
 11  PostalCode         354369 non-null  int64 
dtypes: int64(6), object(6)
memory usage: 32.4+ MB
```

In [8]:

```
#descrição numérica das colunas
df.describe()
```

Out[8]:

	Price	RegistrationYear	Power	Mileage	RegistrationMonth	
count	354369.000000	354369.000000	354369.000000	354369.000000	354369.000000	354
mean	4416.656776	2004.234448	110.094337	128211.172535	5.714645	50
std	4514.158514	90.227958	189.850405	37905.341530	3.726421	25
min	0.000000	1000.000000	0.000000	5000.000000	0.000000	1
25%	1050.000000	1999.000000	69.000000	125000.000000	3.000000	30
50%	2700.000000	2003.000000	105.000000	150000.000000	6.000000	49
75%	6400.000000	2008.000000	143.000000	150000.000000	9.000000	71
max	20000.000000	9999.000000	20000.000000	150000.000000	12.000000	99

Pode-se perceber alguns erros:

- Valor mínimo e máximo do ano de matrícula do veículo (RegistrationYear)
- Valor mínimo de potência (Power)
- Valor mínimo de mês de registro do veículo (RegistrationMonth)

Os valores fora de contexto serão excluídos

In [9]:

```
# criação de filtros para exlusão dos valores

#valores de ano menor que o atual
df = df[df['RegistrationYear'] < 2023]
#valores de ano maior que um ano real
df = df[df['RegistrationYear'] > 1900]
# potência acima de 0
df = df[df['Power'] > 0]
#meses acima de 0
df = df[df['RegistrationMonth'] > 0]

df.reset_index(drop = True, inplace = True)
#visualizando o df
df.head(5)
```

Out[9]:

	Price	VehicleType	RegistrationYear	Gearbox	Power	Model	Mileage	RegistrationMonth
0	18300	coupe	2011	manual	190	NaN	125000	5
1	9800	suv	2004	auto	163	grand	125000	8
2	1500	small	2001	manual	75	golf	150000	6
3	3600	small	2008	manual	69	fabia	90000	7
4	650	sedan	1995	manual	102	3er	150000	10

Verificação de valores ausentes:

In [10]:

```
df.isna().sum()
```

Out[10]:

```
Price                   0
VehicleType         16693
RegistrationYear        0
Gearbox              4379
Power                   0
Model               10563
Mileage                 0
RegistrationMonth       0
FuelType            15476
Brand                   0
NotRepaired         38792
PostalCode              0
dtype: int64
```

Preenchimento dos valores ausentes por: unknown

In [11]:

```
df.fillna('unknown', inplace= True)
```

In [12]:

```
df.isna().sum()
```

Out[12]:

```
Price               0
VehicleType         0
RegistrationYear    0
Gearbox             0
Power               0
Model               0
Mileage             0
RegistrationMonth   0
FuelType            0
Brand               0
NotRepaired         0
PostalCode          0
dtype: int64
```

MODELO DE TREINAMENTO

In [13]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 292084 entries, 0 to 292083
Data columns (total 12 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   Price              292084 non-null  int64 
 1   VehicleType        292084 non-null  object
 2   RegistrationYear   292084 non-null  int64 
 3   Gearbox            292084 non-null  object
 4   Power              292084 non-null  int64 
 5   Model              292084 non-null  object
 6   Mileage            292084 non-null  int64 
 7   RegistrationMonth  292084 non-null  int64 
 8   FuelType           292084 non-null  object
 9   Brand              292084 non-null  object
 10  NotRepaired        292084 non-null  object
 11  PostalCode         292084 non-null  int64 
dtypes: int64(6), object(6)
memory usage: 26.7+ MB
```

Para não atrapalhar o desempenho do modelo, vamos retirar as colunas que possuem mais diversidades de valores para executar o OHE, como:

- Price;
- RegistrationYear;
- Power;
- Mileage;
- RegistrationMonth;
- PostalCode;

In [14]:

```
new = ['VehicleType',
       'Gearbox',
       'Model',
       'FuelType',
       'Brand',
       'NotRepaired']
```

In [15]:

```
for i in new:

    print(f'{i} Há {df[i].nunique()} valores únicos')
```

```
VehicleType Há 9 valores únicos
Gearbox Há 3 valores únicos
Model Há 250 valores únicos
FuelType Há 8 valores únicos
Brand Há 40 valores únicos
NotRepaired Há 3 valores únicos
```

A coluna Model e Brand também serão exluídas para a aplicação do OHE:

In [16]:

```
#OHE
data_ohe = df.drop(['Model', 'Brand'], axis = 1)
data_ohe = pd.get_dummies(data_ohe)
data_ohe.shape
```

Out[16]:

```
(292084, 29)
```

In [17]:

```
data_ohe.head()
```

Out[17]:

	Price	RegistrationYear	Power	Mileage	RegistrationMonth	PostalCode	VehicleType_bus
0	18300	2011	190	125000	5	66954	0
1	9800	2004	163	125000	8	90480	0
2	1500	2001	75	150000	6	91074	0
3	3600	2008	69	90000	7	60437	0
4	650	1995	102	150000	10	33775	0

5 rows × 29 columns

In [18]:

```
#codificação ordinal

from sklearn.preprocessing import OrdinalEncoder

df[new] = OrdinalEncoder().fit_transform(df[new])

df.head()
```

Out[18]:

	Price	VehicleType	RegistrationYear	Gearbox	Power	Model	Mileage	RegistrationMonth
0	18300	2.0	2011	1.0	190	227.0	125000	5
1	9800	6.0	2004	0.0	163	117.0	125000	8
2	1500	5.0	2001	1.0	75	116.0	150000	6
3	3600	5.0	2008	1.0	69	101.0	90000	7
4	650	4.0	1995	1.0	102	11.0	150000	10

OHE NO MODELO

In [19]:

```
from sklearn.model_selection import train_test_split

index_train_valid, index_test = train_test_split(df.index, test_size = 0.2, random_state=
index_train, index_valid = train_test_split(index_train_valid, test_size = 0.25, random_s

train = df.loc[index_train]
valid = df.loc[index_valid]
test = df.loc[index_test]

train_ohe = data_ohe.loc[index_train]
valid_ohe = data_ohe.loc[index_valid]
test_ohe = data_ohe.loc[index_test]

print(train.shape)
print(valid.shape)
print(test.shape)
print()
print(train_ohe.shape)
print(valid_ohe.shape)
print(test_ohe.shape)
```

```
(175250, 12)
(58417, 12)
(58417, 12)

(175250, 29)
(58417, 29)
(58417, 29)
```

In [20]:

```
# fórmula mean_squared_error

from sklearn.metrics import mean_squared_error

def rmse(y, a):
    return mean_squared_error(y, a)**0.5
```

In [21]:

```
# aplicação OHE

feature_train = train_ohe.drop(['Price'], axis = 1)
target_train = train_ohe['Price']
feature_valid = valid_ohe.drop(['Price'], axis = 1)
target_valid = valid_ohe['Price']
feature_test = test_ohe.drop(['Price'], axis = 1)
target_test = test_ohe['Price']
```

In [22]:

```
%%time
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(feature_train, target_train)
```

```
CPU times: total: 844 ms
Wall time: 1.01 s
```

Out[22]:

```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [23]:

```
%%time

pred_train = model.predict(feature_train)
pred_valid = model.predict(feature_valid)
pred_test = model.predict(feature_test)
```

```
CPU times: total: 359 ms
Wall time: 201 ms
```

In [24]:

```
# cálculo do rmse

print('Train RMSE: ', rmse(target_train, pred_train).round(3))
print('Valid RMSE: ', rmse(target_valid, pred_valid).round(3))
print('Test RMSE: ', rmse(target_test, pred_test).round(3))
```

```
Train RMSE:  3230.484
Valid RMSE:  3231.555
Test RMSE:  3239.434
```

UTILIZANDO CODIFICAÇÃO ORDINAL NO MODELO

In [25]:

```
feature_train = train.drop(['Price'], axis = 1)
target_train = train['Price']
feature_valid = valid.drop(['Price'], axis = 1)
target_valid = valid['Price']
feature_test = test.drop(['Price'], axis = 1)
target_test = test['Price']
```

UTILIZANDO FLORESTA ALEATÓRIA

In [26]:

```


from sklearn.ensemble import RandomForestRegressor

for depth in [1, 3, 5, 7]:
    model = RandomForestRegressor(max_depth = depth, n_estimators=100)
    model.fit(feature_train, target_train)

    pred_train = model.predict(feature_train)
    pred_valid = model.predict(feature_valid)
    print('Depth: ', depth)
    print('Train RMSE: ', rmse(target_train, pred_train).round(3))
    print('Valid RMSE: ', rmse(target_valid, pred_valid).round(3))
    print()
```

```
Depth:  1
Train RMSE:  3784.281
Valid RMSE:  3784.968

Depth:  3
Train RMSE:  3026.156
Valid RMSE:  3024.556

Depth:  5
Train RMSE:  2511.603
Valid RMSE:  2494.367

Depth:  7
Train RMSE:  2204.429
Valid RMSE:  2207.043
```

UTILIZAREMOS MAX_DEPTH = 7 NO MODELO

In [27]:

```
%%time

model = RandomForestRegressor(max_depth=7, n_estimators=100)
model.fit(feature_train, target_train)
```

```
CPU times: total: 1min 5s
Wall time: 1min 6s
```

Out[27]:

```
RandomForestRegressor(max_depth=7)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [28]:

```
%%time

pred_train = model.predict(feature_train)
pred_valid = model.predict(feature_valid)
pred_test = model.predict(feature_test)

print('Train RMSE: ', rmse(target_train, pred_train).round(3))
print('Valid RMSE: ', rmse(target_valid, pred_valid).round(3))
print('Test RMSE: ', rmse(target_test, pred_test).round(3))
```

```
Train RMSE:  2205.036
Valid RMSE:  2207.593
Test RMSE:  2224.478
CPU times: total: 4.3 s
Wall time: 4.44 s
```

APLICANDO O GRADIENT BOOSTING

In [29]:

```
pip install lightgbm
```

```
Requirement already satisfied: lightgbm in c:\users\felipe\anaconda3\lib\s
ite-packages (4.0.0)
Requirement already satisfied: numpy in c:\users\felipe\anaconda3\lib\site
-packages (from lightgbm) (1.24.3)
Requirement already satisfied: scipy in c:\users\felipe\anaconda3\lib\site
-packages (from lightgbm) (1.10.1)
Note: you may need to restart the kernel to use updated packages.
```

In [30]:

```
%%time



import lightgbm as lgb

model = lgb.LGBMRegressor(num_iterators=1000, vebose=1, metric='rmse')
model.fit(feature_train, target_train,
          eval_set=(feature_valid, target_valid),
          categorical_feature=new)
```

```
[LightGBM] [Warning] Unknown parameter: num_iterators
[LightGBM] [Warning] Unknown parameter: vebose
[LightGBM] [Warning] Unknown parameter: num_iterators
[LightGBM] [Warning] Unknown parameter: vebose
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.036968 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 926
[LightGBM] [Info] Number of data points in the train set: 175250, number o
f used features: 11
[LightGBM] [Warning] Unknown parameter: num_iterators
[LightGBM] [Warning] Unknown parameter: vebose
[LightGBM] [Info] Start training from score 4896.342631
CPU times: total: 9.86 s
Wall time: 8.23 s
```

Out[30]:

```
LGBMRegressor(metric='rmse', num_iterators=1000, vebose=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [31]:

```
%%time

pred_train = model.predict(feature_train)
pred_valid = model.predict(feature_valid)
pred_test = model.predict(feature_test)

print('Train RMSE: ', rmse(target_train, pred_train).round(3))
print('Valid RMSE: ', rmse(target_valid, pred_valid).round(3))
print('Test RMSE: ', rmse(target_test, pred_test).round(3))
```

```
[LightGBM] [Warning] Unknown parameter: num_iterators
[LightGBM] [Warning] Unknown parameter: vebose
[LightGBM] [Warning] Unknown parameter: num_iterators
[LightGBM] [Warning] Unknown parameter: vebose
[LightGBM] [Warning] Unknown parameter: num_iterators
[LightGBM] [Warning] Unknown parameter: vebose
Train RMSE:  1666.977
Valid RMSE:  1716.395
Test RMSE:  1736.301
CPU times: total: 5.55 s
Wall time: 2.8 s
```

GRADIENT BOOSTING COM CATBOOST

In [ ]:

```
pip install catboost
```

In [158]:

```
%%time


from catboost import CatBoostRegressor

model = CatBoostRegressor(iterations=1000, learning_rate=0.1, metric_period=50)
model.fit(feature_train, target_train, eval_set=(feature_valid, target_valid))
```

```
0:      learn: 4345.9372862     test: 4343.1108192      best: 4343.1108192
(0)     total: 57ms     remaining: 57s
50:     learn: 2039.8024627     test: 2035.7346450      best: 2035.7346450
(50)    total: 2.57s    remaining: 47.9s
100:    learn: 1911.1644899     test: 1913.5605425      best: 1913.5605425
(100)   total: 5s       remaining: 44.5s
150:    learn: 1855.8624343     test: 1864.2384817      best: 1864.2384817
(150)   total: 7.43s    remaining: 41.8s
200:    learn: 1816.1901524     test: 1829.8868225      best: 1829.8868225
(200)   total: 10.4s    remaining: 41.3s
250:    learn: 1786.4337159     test: 1808.3502785      best: 1808.3502785
(250)   total: 13.1s    remaining: 39s
300:    learn: 1764.0968577     test: 1791.5971146      best: 1791.5971146
(300)   total: 15.9s    remaining: 36.9s
350:    learn: 1745.1668179     test: 1777.9386074      best: 1777.9386074
(350)   total: 18.4s    remaining: 33.9s
400:    learn: 1729.1075901     test: 1767.8942190      best: 1767.8942190
(400)   total: 21s      remaining: 31.3s
450:    learn: 1715.7655993     test: 1759.4658152      best: 1759.4658152
(450)   total: 23.6s    remaining: 28.8s
500:    learn: 1702.7174841     test: 1751.4345254      best: 1751.4345254
(500)   total: 26.3s    remaining: 26.2s
550:    learn: 1690.1337806     test: 1743.6636925      best: 1743.6636925
(550)   total: 28.8s    remaining: 23.5s
600:    learn: 1678.7901562     test: 1737.7977375      best: 1737.7977375
(600)   total: 31.2s    remaining: 20.7s
650:    learn: 1668.3191930     test: 1732.5392346      best: 1732.5392346
(650)   total: 33.7s    remaining: 18.1s
700:    learn: 1659.2895806     test: 1727.9583205      best: 1727.9583205
(700)   total: 36.1s    remaining: 15.4s
750:    learn: 1649.3646952     test: 1722.8185448      best: 1722.8185448
(750)   total: 38.5s    remaining: 12.8s
800:    learn: 1641.4108158     test: 1719.5950059      best: 1719.5950059
(800)   total: 40.9s    remaining: 10.2s
850:    learn: 1632.9961133     test: 1716.2826712      best: 1716.2826712
(850)   total: 43.4s    remaining: 7.59s
900:    learn: 1625.6610435     test: 1713.1808734      best: 1713.1808734
(900)   total: 45.8s    remaining: 5.03s
950:    learn: 1618.3856887     test: 1710.2227820      best: 1710.2227820
(950)   total: 48.2s    remaining: 2.48s
999:    learn: 1611.6464908     test: 1707.7718146      best: 1707.7718146
(999)   total: 50.6s    remaining: 0us

bestTest = 1707.771815
bestIteration = 999

CPU times: total: 3min 1s
Wall time: 51 s
```

Out[158]:

<catboost.core.CatBoostRegressor at 0x1e4a847f410>

In [159]:

```
%%time

pred_train = model.predict(feature_train)
pred_valid = model.predict(feature_valid)
pred_test = model.predict(feature_test)

print('Train RMSE: ', rmse(target_train, pred_train).round(3))
print('Valid RMSE: ', rmse(target_valid, pred_valid).round(3))
print('Test RMSE: ', rmse(target_test, pred_test).round(3))
```

```
Train RMSE:  1611.646
Valid RMSE:  1707.772
Test RMSE:  1718.731
CPU times: total: 1.38 s
Wall time: 639 ms
```

GRADIENT BOOSTING COM XGBOOST

In [161]:

```
pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-1.7.6-py3-none-win_amd64.whl (70.9 MB)
                                              0.0/70.9 MB ? eta -:--:--
                                              0.1/70.9 MB 3.3 MB/s eta
0:00:22
                                              0.3/70.9 MB 2.8 MB/s eta
0:00:25
                                              0.4/70.9 MB 2.8 MB/s eta
0:00:26
                                              0.6/70.9 MB 2.9 MB/s eta
0:00:24
                                              0.7/70.9 MB 3.1 MB/s eta
0:00:23
                                              1.0/70.9 MB 3.5 MB/s eta
0:00:21
                                              1.0/70.9 MB 3.5 MB/s eta
0:00:21
                                              1.2/70.9 MB 3.2 MB/s eta
0:00:22
```

In [162]:

```
%%time
# gradient boosting with XGBoost

from xgboost import XGBRegressor

model = XGBRegressor(n_estimators=500, max_depth=7,
                     eta=0.1, subsample=1,
                     colsample_bytree=1)

model.fit(feature_train, target_train)
```

```
CPU times: total: 4min 15s
Wall time: 2min 9s
```

Out[162]:

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None, colsample_bytr
ee=1,
             early_stopping_rounds=None, enable_categorical=False, eta=0.
1,
             eval_metric=None, feature_types=None, gamma=None, gpu_id=Non
e,
             grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=Non
e,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=7, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=Non
e,
             n_estimators=500, n_jobs=None, num_parallel_tree=None,
             predictor=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [163]:

```
%%time

pred_train = model.predict(feature_train)
pred_valid = model.predict(feature_valid)
pred_test = model.predict(feature_test)

print('Train RMSE: ', rmse(target_train, pred_train).round(3))
print('Valid RMSE: ', rmse(target_valid, pred_valid).round(3))
print('Test RMSE: ', rmse(target_test, pred_test).round(3))
```

```
Train RMSE:  1363.081
Valid RMSE:  1667.792
Test RMSE:  1668.962
CPU times: total: 11.6 s
Wall time: 5.82 s
```

ANÁLISE DOS MODELOS

- **REGRESSÃO LINEAR:**
  -RMSE NO CONJUNTO DE TREINO= 3230.484
  -RMSE NO CONJUNTO DE VALIDAÇÃO= 3231.555
  -RMSE NO CONJUNTO DE TESTES= 3239.434

O TEMPO DE TREINAMENTO DO MODELO PARA MODEL.FIT() É **844 MS**
O TEMPO DE TREINAMENTO DO MODELO PARA MODEL.PREDICT() É **359 MS**

- **FLORESTA ALEATÓRIA:**
  -RMSE NO CONJUNTO DE TREINO= 2205.036
  -RMSE NO CONJUNTO DE VALIDAÇÃO= 2207.593
  -RMSE NO CONJUNTO DE TESTES= 2224.478

O TEMPO DE TREINAMENTO DO MODELO PARA MODEL.FIT() É **1MIN 5S**
O TEMPO DE TREINAMENTO DO MODELO PARA MODEL.PREDICT() É **4.3 S**

- **GRADIENT BOOSTING COM LIGHTGBM:**
  -RMSE NO CONJUNTO DE TREINO= 1666.977
  -RMSE NO CONJUNTO DE VALIDAÇÃO= 1716.395
  -RMSE NO CONJUNTO DE TESTES= 1736.301

O TEMPO DE TREINAMENTO DO MODELO PARA MODEL.FIT() É **9.86 S**
O TEMPO DE TREINAMENTO DO MODELO PARA MODEL.PREDICT() É **5.55 S**

- **GRADIENT BOOSTING COM CATBOOST:**
  -RMSE NO CONJUNTO DE TREINO= 1611.646
  -RMSE NO CONJUNTO DE VALIDAÇÃO= 1707.772
  -RMSE NO CONJUNTO DE TESTES= 1718.731

O TEMPO DE TREINAMENTO DO MODELO PARA MODEL.FIT() É **3MIN 1S**
O TEMPO DE TREINAMENTO DO MODELO PARA MODEL.PREDICT() É **1.38 S**

- **GRADIENT BOOSTING COM XGBOOST:**
  -RMSE NO CONJUNTO DE TREINO= 1363.081
  -RMSE NO CONJUNTO DE VALIDAÇÃO= 1667.792
  -RMSE NO CONJUNTO DE TESTES= 1668.962

O TEMPO DE TREINAMENTO DO MODELO PARA MODEL.FIT() É **4MIN 15S**
O TEMPO DE TREINAMENTO DO MODELO PARA MODEL.PREDICT() É **11.6 S**

O MENOR RMSE NO CONJUNTE DE TESTE É DA APLICAÇÃO DE GRADIENT BOOSTING COM XGBOOST (1668.962), O QUE SIGNIFICA TER O MELHOR DESEMPENHO DE PREDIÇÃO. PORÉM, POSSUI O MAIOR TEMPO DE TREINAMENTO (4MIN 15S) E O TEMPO DE PREVISÃO (11.6S) TAMBÉM RELATIVAMENTE ALTO.

GRADIENT BOOSTING COM CATBOOST, POSSUI UM TEMPO DE TREINAMENTO(3MIN 1S) MENOR EM RELAÇÃO AO XGBOOST E UM BOM DESEMPENHO EM TERMOS DE TESTE (RMSE 1718.731).

OS TEMPOS DE PREVISÃO E TREINAMENTO MAIS CURTOS, SÃO DE REGRESSÃO LINEAR, RESPECTIVAMENTE 359 MS E 844 MS. PORÉM APRESENTAM VALORES DE RMSE ELEVADOS,INDICANDO MENOR PRECISÃO QUE OS MODELOS DE AUMENTO DE GRADIENTE

CONCLUSÃO

VISANDO A QUALIDADE DA PREVISÃO, O MODELO MAIS ADEQUADO É O DE GRADIENT BOOSTING COM XGBOOST. APESAR DESTE MODELO APRESENTAR TEMPOS MAIS LONGOS QUE OS OUTROS MODELOS, ELE POSSUI TAMBÉM UMA MELHOR QUALIDADE DE PREVISÃO, O QUE É O NOSSO PRINCIPAL OBJETIVO.

CHECK LIST

Digite 'x' para verificar. Em seguida, pressione Shift + Enter.

- ☑ O Jupyter Notebook está aberto
- ☑ O código está livre de erros
- ☑ As células com o código foram organizadas em ordem de execução
- ☑ Os dados foram baixados e preparados
- ☑ Os modelos foram treinados
- ☑ A análise de velocidade e qualidade dos modelos foi realizada