

**Autores:** Prof. João Nogueira e Prof. Dino Magri

**Contato:** joao.nogueira@fisica.ufc.br e professor.dinomagri@gmail.com

**Licença deste notebook:**



[Clique aqui para saber mais sobre a licença CC BY v4.0](#)

## ▼ Projeto - Parte 3 - Aprendizagem Supervisionada - Regressão - Exercícios

NOTA MÁXIMA: 25 pontos

NOME COMPLETO: MARCIO FERNANDES CRUZ, EDUARDO BOLYHOS, ALISSON SALES



```
1 from google.colab import drive
2 drive.mount('/content/drive')

Mounted at /content/drive
```

## ▼ Carregando a base de dados

```
1 from google.colab import drive
2 drive.mount('/content/drive')
3
4 from google.colab import auth
5 auth.authenticate_user()
6
7 from IPython.display import clear_output
8
9 import requests
10 gcloud_token = !gcloud auth print-access-token
11 gcloud_tokeninfo = requests.get('https://www.googleapis.com/oauth2/v3/tokeninfo?access_token=' + gcloud_token[0]).json()
12
13 email_logado = gcloud_tokeninfo['email']
14
15 clear_output()
```

```
1 import numpy as np
2 import pandas as pd
```

```
1 if email_logado=='marcio@marciofcruz.com':
2   pasta_raiz = '/content/drive/MyDrive/Colab Notebooks/Data Set Data Science/ifood'
3 elif email_logado=='alissondeandrade@gmail.com':
4   pasta_raiz = '/content/drive/MyDrive/MBA_FIA_DS_IA/DISCIPLINAS/3.IA/2021_2022_projeto_IA/projeto-ia-datasets/ifood'
5 elif email_logado=='sonekabolyhos@gmail.com':
6   pasta_raiz = '/content/drive/MyDrive/Aulas Fia/Dataset'
7 else:
8   pasta_raiz = "Não encontrado o email"
9
10 print(f'A pasta raiz do {email_logado} é {pasta_raiz}')
```

A pasta raiz do [alissondeandrade@gmail.com](mailto:alissondeandrade@gmail.com) é /content/drive/MyDrive/MBA\_FIA\_DS\_IA/DISCIPLINAS/3.IA/2021\_2022\_projeto\_IA/projeto-ia-

```
1 df_orders = pd.read_csv(f'{pasta_raiz}/df_orders.csv', parse_dates=['order_timestamp_local'])
2 df_orders.head(3)
```

	order_id	order_number	order_timestamp_local	order_shift	last_status_date_local	order_total	credit	paid_amount	deliver
0	630e2af0-b456-4b3a-b964-4d66ce5cc5df	1290139943	2019-12-11 13:22:06.497000+00:00	weekday lunch	2019-12-11T15:22:39.156Z	28.90	14.0	24.9	DEl
1	66f8163d-f081-4fbdb-adb8-cfbff9213f7	1141271215	2019-11-06 17:57:17.967000+00:00	weekday dinner	2019-11-06T18:18:48.680Z	56.35	7.9	50.4	DEl
2	e8c55557-81d4-4159-bd2b-~11a5fbab3	694742752	2019-06-11 20:42:32.058000+00:00	weekday dinner	2019-06-11T22:42:54.005Z	53.80	4.0	49.8	DEl

## ▼ Criação da ABT (10 pontos)

### ▼ Agregando receita por dia

**(1.5 ponto) q1** Crie um DataFrame chamado `df` que deverá somar o valor total dos pedidos. Utilize a variável `order_timestamp_local` para recuperar a data. Faça o agrupamento por dia e por fim faça a ordenação do menor para o maior.

O resultado esperado é um DataFrame com duas colunas (`order_date` e `receita`).

order_date	receita
2019-06-01	209061.01
2019-06-02	213995.65
2019-06-03	108973.96

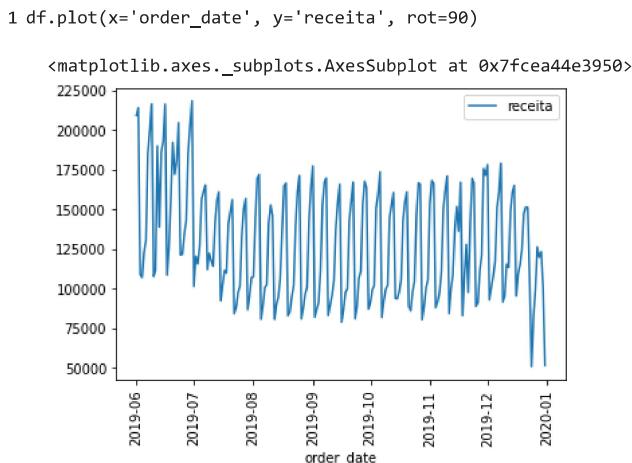
```

1 df_orders['order_date'] = df_orders['order_timestamp_local'].dt.date
2
3 df = (
4     df_orders
5     .groupby("order_date")
6     .agg(receita = ('order_total', 'sum'))
7     .reset_index()
8     .sort_values(by="order_date")
9 )
10
11 df.head(3)
12

```

order_date	receita
0 2019-06-01	209061.01
1 2019-06-02	213995.65
2 2019-06-03	108973.96

**(0.5 ponto) q1.1** Faça um plot da série temporal. Utilize o método `.plot()`.



### ▼ Train-Test Split

**(2.0 ponto) q2** Crie dois dataframes: `df_train` com dados de junho até novembro e `df_test` com dados de dezembro.

```

1 df["order_date"] = pd.to_datetime(df["order_date"])
2
3 #df['order_date'].min()
4 # df['order_date'].max()
5
6 filtro = (df["order_date"]<'2019-12-01')
7 df_train = df[filtro]
8
9 filtro = (df["order_date"]>='2019-12-01')
10 df_test = df[filtro]
11
12 print(f'Linhas do df_train é {df_train.shape[0]} e do df_test é {df_test.shape[0]}')

Linhas do df_train é 183 e do df_test é 31

```

**CHECKPOINT** - A resposta do exercício deverá ser:

df\_train deverá ter 183 linhas e df\_test deverá ter 31 linhas.

▼ Feature Engineering

▼ Variáveis Sazonais

**(2.0 ponto) q3** Crie as seguintes variáveis sazonais para ambos os DataFrames df\_train e df\_test:

- mes : mês
- dia : dia do mês
- weekday : dia da semana
- dayofyear : dia do ano

```

1 df_train = (df_train
2 .assign(mes = df_train["order_date"].dt.month,
3         dia = df_train["order_date"].dt.day,
4         weekday = df_train["order_date"].dt.weekday,
5         dayofyear = df_train["order_date"].dt.dayofyear))
6
7 df_test = (df_test
8 .assign(mes = df_test["order_date"].dt.month,
9         dia = df_test["order_date"].dt.day,
10        weekday = df_test["order_date"].dt.weekday,
11        dayofyear = df_test["order_date"].dt.dayofyear))
12
13
14

```

1 df\_train.head(3)

	order_date	receita	mes	dia	weekday	dayofyear
0	2019-06-01	209061.01	6	1	5	152
1	2019-06-02	213995.65	6	2	6	153
2	2019-06-03	108973.96	6	3	0	154

1 df\_test.head(3)

	order_date	receita	mes	dia	weekday	dayofyear
183	2019-12-01	178085.12	12	1	6	335
184	2019-12-02	92833.10	12	2	0	336
185	2019-12-03	99995.42	12	3	1	337

▼ Variáveis Lag

**(2.0 ponto) q4** Crie as seguintes variáveis de lag para ambos os DataFrames df\_train e df\_test:

- receita\_atual : receita do dia em que a previsão é feita
- receita\_anterior : receita do dia anterior ao dia em que a previsão é feita
- diff\_receitas : diferença de receita do dia atual para o dia anterios

Utilize o método `.shift()` da coluna `df['receita']`. O método `shift()` é utilizada para deslocar o índice pelo número desejado de períodos com uma frequencia de tempo opcional.

```

1 df_train = (df_train
2 .assign(receita_atual = df_train["receita"].shift(1))
3 .assign(receita_anterior = df_train["receita"].shift(2))
4 .assign(diff_receitas = lambda df_em_memoria: df_em_memoria['receita_atual']-df_em_memoria['receita_anterior'])
5 )
6
7 df_train.head(5)

```

	order_date	receita	mes	dia	weekday	dayofyear	receita_atual	receita_anterior	diff_receitas
0	2019-06-01	209061.01	6	1	5	152	NaN	NaN	NaN
1	2019-06-02	213995.65	6	2	6	153	209061.01	NaN	NaN
2	2019-06-03	108973.96	6	3	0	154	213995.65	209061.01	4934.64
3	2019-06-04	106973.23	6	4	1	155	108973.96	213995.65	-105021.69
4	2019-06-05	122957.05	6	5	2	156	106973.23	108973.96	-2000.73

```

1 df_test = (df_test
2 .assign(receita_atual = df_test["receita"].shift(1))
3 .assign(receita_anterior = df_test["receita"].shift(2))
4 .assign(diff_receitas = lambda df_em_memoria: df_em_memoria['receita_atual']-df_em_memoria['receita_anterior'])
5 )
6
7 df_test.head(5)

```

	order_date	receita	mes	dia	weekday	dayofyear	receita_atual	receita_anterior	diff_receitas
183	2019-12-01	178085.12	12	1	6	335	NaN	NaN	NaN
184	2019-12-02	92833.10	12	2	0	336	178085.12	NaN	NaN
185	2019-12-03	99995.42	12	3	1	337	92833.10	178085.12	-85252.02
186	2019-12-04	107698.24	12	4	2	338	99995.42	92833.10	7162.32
187	2019-12-05	117057.18	12	5	3	339	107698.24	99995.42	7702.82

```

1 df_train.shape, df_test.shape
((183, 9), (31, 9))

```

#### ▼ Média Móvel dos Últimos 3 dias

**(2.0 ponto) q5** Cria uma variável que representa a média móvel dos últimos 3 dias anteriores ao dia de previsão para ambos os DataFrames df\_train e df\_test.

- media\_movel: média móvel dos últimos 3 dias.

Utilize o método `.rolling(3)` que possibilita realizar cálculos para uma determinada janela deslizante. O valor 3, significa o tamanho da janela que será utilizada para deslizar.

```

1 # Adicione seu código aqui para criar as variáveis solicitadas no exercício para o df_train:
2 df_train = (
3     df_train
4     .assign(media_movel = df_train['receita_atual'].rolling(3).mean())
5 )
6 df_train.head()
7

```

	order_date	receita	mes	dia	weekday	dayofyear	receita_atual	receita_anterior	diff_receitas	media_movel
0	2019-06-01	209061.01	6	1	5	152	NaN	NaN	NaN	NaN
1	2019-06-02	213995.65	6	2	6	153	209061.01	NaN	NaN	NaN
2	2019-06-03	108973.96	6	3	0	154	213995.65	209061.01	4934.64	NaN
3	2019-06-04	106973.23	6	4	1	155	108973.96	213995.65	-105021.69	177343.54
4	2019-06-05	122957.05	6	5	2	156	106973.23	108973.96	-2000.73	143314.28

```

1 # Adicione seu código aqui para criar as variáveis solicitadas no exercício para o df_test:
2 df_test = (
3     df_test
4     .assign(media_movel = df_test['receita_atual'].rolling(3).mean())
5 )
6 df_test.head()
7

```

	order_date	receita	mes	dia	weekday	dayofyear	receita_atual	receita_anterior	diff_receitas	media_movel
183	2019-12-01	178085.12	12	1	6	335	NaN	NaN	NaN	NaN
184	2019-12-02	92833.10	12	2	0	336	178085.12	NaN	NaN	NaN
185	2019-12-03	99995.42	12	3	1	337	92833.10	178085.12	-85252.02	NaN
186	2019-12-04	107698.24	12	4	2	338	99995.42	92833.10	7162.32	123637.880000
187	2019-12-05	117057.18	12	5	3	339	107698.24	99995.42	7702.82	100175.586667

## ▼ Modelagem (10 pontos)

Iremos testar diferentes algoritmos de regressão para o problema em questão.

## ▼ Criando os conjuntos de treino e teste

**(1.0 ponto) Q6** Crie os seguintes dataframes selecionando apenas as colunas `['mes', 'dia', 'weekday', 'dayofyear', 'receita_atual', 'receita_anterior', 'diff_receitas', 'media_movel']` como **features** e a coluna `receita` como **target**. Remova todas as linhas com valores faltantes.

- `X_train`
- `X_test`
- `y_train`
- `y_test`

```

1 features = ['mes', 'dia', 'weekday', 'dayofyear', 'receita_atual', 'receita_anterior', 'diff_receitas', 'media_movel']
2 target    = 'receita'

```

Remova os valores faltantes `.dropna()` existentes que foram criados na parte de Feature Engineering e faça a filtragem correspondente utilizando as variáveis `features` e `target`.

```

1 X_train = df_train.dropna().filter(features).copy()
2 y_train = df_train.dropna()[target]
3
4 X_test = df_test.dropna().filter(features).copy()
5 y_test = df_test.dropna()[target]
6
7 print('X_train: ', X_train.shape)
8 print('y_train: ', y_train.shape)
9
10 print('X_test: ', X_test.shape)
11 print('y_test: ', y_test.shape)

X_train: (180, 8)
y_train: (180,)
X_test: (28, 8)
y_test: (28,)

```

 **CHECKPOINT** - Verifique a quantidade de linhas e colunas para as variáveis `X_train`, `X_test`, `y_train` e `y_test`:

```

X_train - (180, 8)
y_train - (180,)

X_test - (28, 8)
y_test - (28,)

```

## ▼ Regressão Linear

**(0.5 ponto) Q7** Treine uma regressão linear para o conjunto de treino e salve na variável `lr_model`.

**Obs:** Não esquecer de normalizar as variáveis para modelos lineares, como Regressão Linear e Ridge Regression.

Utilize o `StandardScaler` para normalizar as variáveis. Utilize a classe `SklearnTransformerWrapper` da biblioteca `feature_engine`.

```
1 !pip install feature_engine
2
3 clear_output()
```

```
1 X_train
```

	mes	dia	weekday	dayofyear	receita_atual	receita_anterior	diff_receitas
3	6	4	1	155	108973.96	213995.65	-105021.69
4	6	5	2	156	106973.23	108973.96	-2000.73
5	6	6	3	157	122957.05	106973.23	15983.82
6	6	7	4	158	130615.30	122957.05	7658.25
7	6	8	5	159	185314.25	130615.30	54698.95
...	...	...	...	...	...	...	...
178	11	26	1	330	88803.48	166519.97	-77716.49
179	11	27	2	331	91214.85	88803.48	2411.37
180	11	28	3	332	111371.52	91214.85	20156.67
181	11	29	4	333	121050.43	111371.52	9678.91
182	11	30	5	334	175600.49	121050.43	54550.06

180 rows × 8 columns

```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.preprocessing import StandardScaler
3 from feature_engine.wrappers import SklearnTransformerWrapper
4 from sklearn.pipeline import Pipeline
5
6
7 #####
8 # Adicione o código da Regressão Linear via Pipeline #
9 #####
10
11 X = X_train
12 y = y_train
13
14 cat_vars = []
15
16 modelos_lineares = [
17     ('lr', LinearRegression())
18 ]
19
20 steps_modelos_lineares = [
21     ('numeric_scaler', SklearnTransformerWrapper(variables=features, transformer=StandardScaler()))
22 ]
23
24
25 aux_steps = steps_modelos_lineares + [modelos_lineares[0]]
26 pipeline = Pipeline(steps=aux_steps)
27
28 lr_model = Pipeline(steps=aux_steps)

1 # Execute para criar o modelo
2 lr_model.fit(X, y)
3

Pipeline(steps=[('numeric_scaler',
                 SklearnTransformerWrapper(transformer=StandardScaler(),
                                           variables=['mes', 'dia', 'weekday',
                                                       'dayofyear',
                                                       'receita_atual',
                                                       'receita_anterior',
                                                       'diff_receitas',
                                                       'media_movel']),
              ('lr', LinearRegression())])
```

## ▼ Criando Função que retorna todas as Métricas

(1.0 ponto) q8 Crie uma função chamada `log_results` que deverá retornar um DataFrame com todas as métricas de regressão. A função deverá conter os seguintes parâmetros:

- `modelo_objeto` esse parâmetro é obrigatório e tem que ser a variável que representa o objeto do modelo já "fitado".
- `nome_modelo` deverá ser o nome do modelo
- `X_train, X_test, y_train, y_test` são os parâmetros referentes aos conjuntos de treino e teste que serão avaliados.

Lembre-se de criar funções auxiliares para facilitar a criação das métricas.

Exemplo de chamada da função:

```
1 lr_results = log_results(lr_model, 'Regressão Linear', X_train, X_test, y_train, y_test)
```

A saída da função para a regressão linear deverá ser a seguinte:

	modelo	modo	r2	r2_ajustado	mse	rmse	rmsle	mae	medae	mape	max_error
0	Regressão Linear	treino	0.8518	0.8449	184683302.0703	13589.8235	0.1024	10438.1981	8183.7347	0.0814	57688.0425
1	Regressão Linear	teste	0.5458	0.3545	447617025.0598	21156.9616	0.2217	18787.5085	16495.3045	0.1762	38496.7851

```
1 lr_results = pd.DataFrame(columns=['r2', 'r2_ajustado', 'mse', 'rmse', 'rmsle', 'mae', 'medae', 'mape', 'max_error'])
```

```
1 from sklearn.metrics import r2_score
2 from sklearn.metrics import mean_squared_error
3 from sklearn.metrics import mean_absolute_error
4 from sklearn.metrics import median_absolute_error
5 from sklearn.metrics import mean_absolute_percentage_error
6 from sklearn.metrics import max_error
7 from sklearn.metrics import mean_squared_log_error
8
9 pd.options.display.float_format = '{:.4f}'.format
10
11 def adjusted_r2(y Esperado, y Previsto, X_treino):
12     R2 = r2_score(y Esperado, y Previsto)
13     N = len(y Esperado)
14     p = X_treino.shape[1]
15
16     r2_ajustado = (1 - ((1 - R2) * (N - 1)) / (N - p - 1))
17     return r2_ajustado
18
19 def get_metricas(modelo, modo, y_previsto, y_esperado):
20     return dict(modelo = modelo,
21                 modo = modo,
22                 r2 = r2_score(y_esperado, y_previsto),
23                 r2_ajustado = adjusted_r2(y_esperado, y_previsto, X_train),
24                 mse = mean_squared_error(y_esperado, y_previsto),
25                 rmse = mean_squared_error(y_esperado, y_previsto, squared=False),
26                 rmsle = mean_squared_log_error(y_esperado, y_previsto, squared=False),
27                 mae = mean_absolute_error(y_esperado, y_previsto),
28                 medae = median_absolute_error(y_esperado, y_previsto),
29                 mape = mean_absolute_percentage_error(y_esperado, y_previsto),
30                 maximo_error = max_error(y_esperado, y_previsto)
31     )
32
33
34 def log_results(modelo_objeto, nome_modelo, X_train, X_test, y_train, y_test):
35     y_previsto = modelo_objeto.predict(X_train)
36     y_esperado = y_train.copy()
37     df_treino = pd.DataFrame([get_metricas(nome_modelo, 'treino', y_previsto, y_esperado)])
38
39     y_previsto = modelo_objeto.predict(X_test)
40     y_esperado = y_test.copy()
41     df_teste = pd.DataFrame([get_metricas(nome_modelo, 'teste', y_previsto, y_esperado)])
42
43     return pd.concat([df_treino, df_teste])
44
45
46 def log_validacao(modelo_objeto, nome_modelo, X_test, y_test):
47     y_previsto = modelo_objeto.predict(X_test)
48     y_esperado = y_test.copy()
49
50     return pd.DataFrame([get_metricas(nome_modelo, 'validação', y_previsto, y_esperado)])
51
```

**(0.5 ponto) Q8.1** Aplique a função desenvolvida anteriormente para o modelo `lr_model` criado no exercício q7. Salve o retorno da função na variável `lr_results`.

```
1 lr_results = log_results(lr_model, 'Regressão Linear', X_train, X_test, y_train, y_test)
2 lr_results
```

	modelo	modo	r2	r2_ajustado	mse	rmse	rmsle	
0	Regressão Linear	treino	0.8518	0.8449	184683302.0703	13589.8235	0.1024	10438.1

## ▼ Ridge Regression

**(1.0 ponto) q9** Treine uma [Ridge Regression](#) (Regressão Linear menos suscetível ao overfitting) para o conjunto de treino e salve na variável `ridge_model`. Avalie os resultados na base de treino e na base de teste utilizando a função `log_results` desenvolvida anteriormente e salve o retorno da função na variável `ridge_results`.

**Obs:** Não esquecer de normalizar as variáveis para modelos lineares, como `Regressão Linear` e `Ridge Regression`.

```

1 from sklearn.linear_model import Ridge
2 from feature_engine.imputation import ArbitraryNumberImputer, MeanMedianImputer
3 from sklearn.preprocessing import PolynomialFeatures
4
5 ridge_model = Pipeline(steps=[
6     ('numeric_imputer', MeanMedianImputer(variables=features, imputation_method='mean')),
7     ('scaling', SklearnTransformerWrapper(variables=features, transformer=StandardScaler())),
8     ('model', Ridge())])
9 ridge_model.fit(X, y)
10
11 # continue o código abaixo
12 ridge_results = log_results(ridge_model, 'Ridge Regression', X_train, X_test, y_train, y_test)

1 # Configurações para os modelos de árvore
2 steps_modelos_arvores = [
3     ('numeric_imputer', MeanMedianImputer(variables=features, imputation_method='mean'))
4 ]

```

## ▼ Decision Tree

**(1.0 ponto) q10** Treine uma Decision Tree para o conjunto de treino e salve na variável `tree_model`. Avalie os resultados na base de treino e na base de teste utilizando a função `log_results` desenvolvida anteriormente e salve o retorno da função na variável `tree_results`.

Utilize os parâmetros `random_state=30` e `max_depth=3`.

```

1 from sklearn.tree import DecisionTreeRegressor
2 from sklearn.preprocessing import PolynomialFeatures
3
4 tree_model = Pipeline(steps=[('model', DecisionTreeRegressor(random_state=30, max_depth=3))])
5 tree_model.fit(X, y)
6
7 # continue o código abaixo
8 tree_results = log_results(tree_model, 'Decision Tree', X_train, X_test, y_train, y_test)

```

## ▼ Random Forest

**(1.0 ponto) q11** Treine uma Random Forest para o conjunto de treino e salve na variável `rf_model`. Avalie os resultados na base de treino e na base de teste utilizando a função `log_results` desenvolvida anteriormente e salve o retorno da função na variável `rf_results`.

Utilize os parâmetros `random_state=30` e `max_depth=3`.

```

1 from sklearn.ensemble import RandomForestRegressor
2
3 rf_model = Pipeline(steps=[('model', RandomForestRegressor(random_state=30, max_depth=3))])
4 rf_model.fit(X, y)
5
6 # continue o código abaixo
7 random_forest_results = log_results(rf_model, 'Random Forest', X_train, X_test, y_train, y_test)

```

## ▼ LGBM

**(1.0 ponto) q12** Treine um LightGBM para o conjunto de treino e salve na variável `lgbm_model`. Avalie os resultados na base de treino e na base de teste utilizando a função `log_results` desenvolvida anteriormente e salve o retorno da função na variável `lgbm_results`.

Utilize os parâmetros `random_state=30`

```

1 from lightgbm import LGBMRegressor
2
3 lgbm_model = Pipeline(steps=[...])

```

```

4 ('scaling', SklearnTransformerWrapper(transformer=StandardScaler(), variables=features)),
5 ('model', LGBMRegressor(random_state=30))])
6 lgbm_model.fit(X, y)
7
8 # continue o código abaixo
9 lgbm_results = log_results(lgbm_model, 'LGBM', X_train, X_test, y_train, y_test)

```

## ▼ XGBoost

**(1.0 ponto) q13** Treine um XGBoost para o conjunto de treino e salve na variável `xgb_model`. Avalie os resultados na base de treino e na base de teste utilizando a função `log_results` desenvolvida anteriormente e salve o retorno da função na variável `xgb_results`.

Utilize os parâmetros `random_state=30`

```

1 from xgboost import XGBRegressor
2
3 xgb_pipeline = Pipeline(steps=[('model', XGBRegressor(random_state=30))])
4 xgb_pipeline.fit(X, y)
5
6 # continue o código abaixo
7 xgb_results = log_results(xgb_pipeline, 'XGBoost', X_train, X_test, y_train, y_test)

```

⚠ [19:30:07] WARNING: /workspace/src/objective/regression\_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

## ▼ CatBoost

**(1.0 ponto) q14** Treine um CatBoost para o conjunto de treino e salve na variável `cat_model`. Avalie os resultados na base de treino e na base de teste utilizando a função `log_results` desenvolvida anteriormente e salve o retorno da função na variável `cat_results`.

Utilize os parâmetros `random_state=30`

PS: não esqueça de executar a linha para instalar a biblioteca `catboost`.

```

1 !pip install catboost==0.25.1
2
3 clear_output()
4
5
6 from catboost import CatBoostRegressor
7
8 cat_pipeline = Pipeline(steps=[('model', CatBoostRegressor(random_state=30))])
9 cat_pipeline.fit(X, y)
10
11 clear_output()
12
13 # continue o código abaixo
14 cat_results = log_results(cat_pipeline, 'CatBoost', X_train, X_test, y_train, y_test)

```

## ▼ Juntando todos os resultados

**(0.5 ponto) q15** Consolide todos os resultados acima em uma única tabela. Dica: use a função `pd.concat()`.

```

1 df_resultado_consolidado = pd.concat([lr_results, ridge_results, tree_results, random_forest_results, lgbm_results, xgb_results, cat_results])
2
3 lista = []
4 def retorna_descricao(df_teste):
5     lista.append("No modo {}, o modelo na {}a posição é {}".format(df_teste["modo"], int(df_teste["ranking"]), df_teste["modelo"]))
6
7 df_resultado_consolidado.query("modo == 'teste'")

```

	modelo	modo	r2	r2_ajustado	mse	rmse	rmsle
0	Regressão Linear	teste	0.5458	0.3545	447617025.0598	21156.9616	0.2217
0	Ridge Regression	teste	0.6553	0.5102	339686747.4264	18430.5927	0.1958
0	Decision Tree	teste	0.6025	0.4351	391733754.2567	19792.2650	0.2104
0	Random Forest	teste	0.6497	0.5022	345215267.3708	18579.9695	0.2025
0	LGFM	teste	0.6670	0.5268	328178439.0652	18115.6959	0.2041

**(0.5 ponto) q16** Qual o modelo que apresentou o melhor R2 ajustado na base de treino?

```

1 df_resultado_teste = (
2     df_resultado_consolidado[df_resultado_consolidado["modo"]=="treino"]
3     .assign(ranking = lambda df_em_memoria: df_em_memoria["r2_ajustado"].rank(ascending=False))
4     .sort_values(by="ranking")
5     .query("ranking==1")
6 )
7
8 lista = []
9 df_resultado_teste.apply(retorna_descricao, axis=1)
10 lista

['No modo treino, o modelo na 1a posição é CatBoost']

```

**(0.5 ponto) q17** Qual o modelo que apresentou o melhor R2 ajustado na base de teste?

```

1 df_resultado_teste = (
2     df_resultado_consolidado[df_resultado_consolidado["modo"]=="teste"]
3     .assign(ranking = lambda df_em_memoria: df_em_memoria["r2_ajustado"].rank(ascending=False))
4     .sort_values(by="ranking")
5     .query("ranking==1")
6 )
7
8 lista = []
9 df_resultado_teste.apply(retorna_descricao, axis=1)
10 lista

['No modo teste, o modelo na 1a posição é LGBM']

```

**(0.5 ponto) q18** Qual o modelo que devemos considerar para realizar as otimizações e por que?

```

1 (
2     df_resultado_consolidado
3     .query('modo=="treino"')
4     .sort_values(by="r2_ajustado", ascending=False)
5     .head(7)
6 )

    modelo  modo      r2   r2_ajustado        mse      rmse    rmsle
0   CatBoost  treino  0.9992      0.9991  1045179.7029  1022.3403  0.0083  813.3
0   XGBoost   treino  0.9880      0.9875  14920027.9569  3862.6452  0.0296  2791.0
0    LGBM     treino  0.9723      0.9710  34574302.9413  5879.9917  0.0420  3772.2
0  Random Forest  treino  0.9372      0.9343  78288006.4591  8848.0510  0.0676  6261.5
0   Decision Tree  treino  0.9237      0.9202  95070777.7824  9750.4245  0.0737  6746.0
0       Regressão  ...  0.6510      0.6510  101000000.0700  10500.0005  0.1001  10100.0

```

```

1 (
2     df_resultado_consolidado
3     .query('modo=="teste"')
4     .sort_values(by="r2_ajustado", ascending=False)
5     .head(10)
6 )

    modelo  modo      r2   r2_ajustado        mse      rmse    rmsle
0     LGBM   teste  0.6670      0.5268  328178439.0652  18115.6959  0.2041  12544.3
0  Ridge Regression  teste  0.6553      0.5102  339686747.4264  18430.5927  0.1958  15391.8
0  Random Forest   teste  0.6497      0.5022  345215267.3708  18579.9695  0.2025  13758.3
0   Decision Tree   teste  0.6025      0.4351  391733754.2567  19792.2650  0.2104  14953.6
0   Regressão Linear  teste  0.5458      0.3545  447617025.0598  21156.9616  0.2217  18787.5

```

**Resposta:** O melhor modelo é o LGBM pois apresentou um melhor resultado na base de Teste.  
Apesar de todos os indicadores de erros estarem próximos, o MEDAE deste modelo está melhor.

## ▼ Otimização dos hiperparâmetros (5 pontos)

### ▼ Grid Search do Melhor Modelo

(1.5 ponto) q19 Realize um grid search do melhor modelo de acordo com o R2 ajustado na base de teste.

Escolha pelo menos 5 hiperparâmetros para serem otimizados. Cada um desses hiperparâmetros deverá testar ao menos dois valores.

O algoritmo que melhor ajustou o modelo foi LightGBM. Para identificar os hiperparâmetros veja na documentação -

<https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMRegressor.html>

Utilize o random\_state=30.

**Os parâmetros do GridSearchCV ficam a seu critério.**

```

1 from sklearn.model_selection import GridSearchCV
2 import time
3
4 def get_df_comparacao(best_model, X, y):
5     y_previsto = best_model.predict(X)
6     y_esperado = y.copy()
7
8     validacao_results = log_validacao(best_model, 'LGBM', X, y)
9
10    return (
11        pd.concat([validacao_results, lgbm_results])
12        .sort_values(by="r2_ajustado", ascending=False)
13    )
14
15 random_state=30
16
17 t1 = time.time()
18 grid_search = GridSearchCV(lgbm_model, parametros, scoring=adjusted_r2, cv=5, n_jobs=-1, verbose=1)
19 grid_search.fit(X,y)
20
21 clear_output()
22 print('Melhores parâmetros: ', grid_search.best_params_)
23 print('\nTempo Execução: {} segundos'.format(time.time() - t1))
24
25 best_pipeline = grid_search.best_estimator_
26
27 best_pipeline.fit(X, y)
28
29 df_comparacao = get_df_comparacao(best_pipeline, X, y)
30 df_comparacao.head(3)

Melhores parâmetros: {'model_alpha': 5, 'model_class_weight': 'balanced', 'mode
Tempo Execução: 66.04484939575195 segundos

```

modelo	modo	r2	r2_ajustado	mse	rmse	rmsle
0	LGBM	validação	0.9951	0.9949	6099561.3531	2469.7290
0	LGBM	treino	0.9723	0.9710	34574302.9413	5879.9917
0	LGBM	teste	0.6670	0.5268	328178439.0652	18115.6959

**(1.0 ponto) q20** Veja se o melhor modelo tunado apresenta uma melhor performance do que o modelo original. Faça as considerações sobre os resultados obtidos.

```

1 valor_validacao = df_comparacao[df_comparacao["modo"]=="validação"]["r2_ajustado"]
2 valor_teste = df_comparacao[df_comparacao["modo"]=="teste"]["r2_ajustado"]
3
4 print(f'Na Validação o resultado foi {valor_validacao[0]} e no Teste foi {valor_teste[0]}')
5
6 if (valor_validacao[0]>valor_teste[0]):
7     print('Portanto, o modelo pode ser considerado para ser posto em Produção')
8 else:
9     print('Não é aconselhável por este Modelo em Produção. Aconselhamos revisar a configuração dos hiperparâmetros e/ou melhorar a Quali
Na Validação o resultado foi 0.9948774289864032 e no Teste foi 0.5267743593734386
Portanto, o modelo pode ser considerado para ser posto em Produção

```

## ▼ Feature Importance

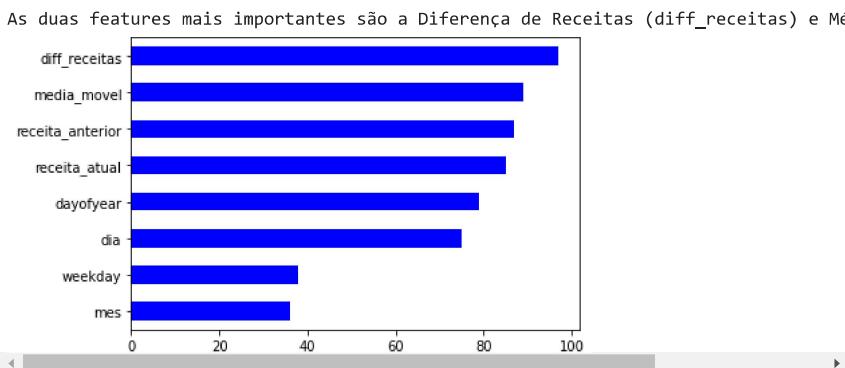
**(0.5 ponto) q21** Quais são as duas features mais importantes para o modelo (`lgbm_model`)? Crie um gráfico de barras na horizontal com a feature mais importante no topo e a menos importante na base do gráfico.

Clique duas vezes (ou pressione "Enter") para editar

```

1 import matplotlib.pyplot as plt
2 from matplotlib.pyplot import figure
3
4 encoder = best_pipeline.named_steps["scaling"]
5 columns = encoder.transform(X).columns
6 model = best_pipeline.named_steps["model"]
7
8 importance = model.feature_importances_
9
10 combination = pd.Series(importance, columns)
11
12 combination.sort_values().plot.barh(color="blue")
13
14
15 print("As duas features mais importantes são a Diferença de Receitas (diff_receitas) e Média Móvel (media_movel)")

```



**Diff\_receitas e Media Movel**

## ▼ Shap Values

**(0.5 ponto) q22** Execute as linhas de código abaixo e interprete o gráfico `shap values`.

```

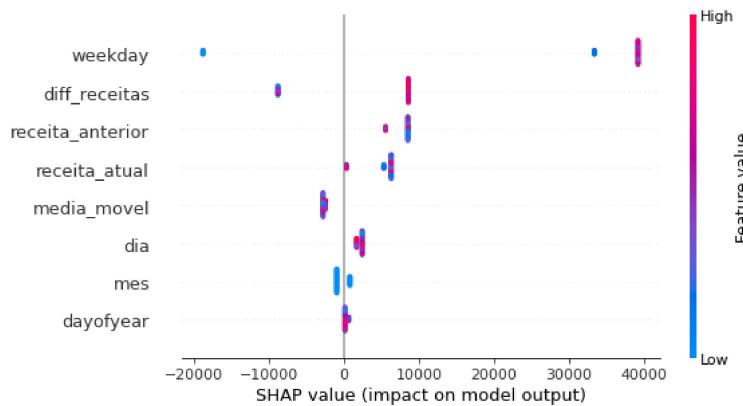
1 !pip install shap
2 clear_output()
3
4 import shap

1 model = best_pipeline.named_steps["model"]
2 explainer = shap.TreeExplainer(model)

1 shap_values = explainer.shap_values(X_test)

1 shap.summary_plot(shap_values, X_test)

```



```
1 print('O dia da semana (weekday) é um fator de suma importância que explica o total de Receita, mas não é a feature mais importante do modelo.')
2 print('O segundo fator que mais explica é a media móvel mas, em suma, é um reflexo dos últimos 3 dias.'
```

O dia da semana (weekday) é um fator de suma importância que explica o total de Receita, mas não é a feature mais importante do modelo.  
O segundo fator que mais explica é a media móvel mas, em suma, é um reflexo dos últimos 3 dias.