

Relatório de entrega Inteligência Artificial

Neste começo do código importamos algumas bibliotecas e fizemos o processamento dos dados tratados anteriormente para continuar com o Machine Learning.

```
import pandas as pd
import datetime
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

URL = '/content/export (20).csv'

df = pd.read_csv(URL)
df = df.drop(columns=['cod_produto'])
display(df.head())
```

Agora iremos tratar algumas colunas que podem atrapalhar nosso processo, e dividir os dados em treino e teste para seguir o ML. Deste modo conseguimos uma base melhor para ter uma melhor acurácia e precisão.

```
import numpy as np
from sklearn.model_selection import train_test_split

colunas_com_virgula = ['num_latitude_origem', 'num_longitude_origem', 'num_latitude_destino', 'num_longitude_destino']
for col in colunas_com_virgula:
    df[col] = df[col].astype(str).str.replace(',', '.').astype(float)

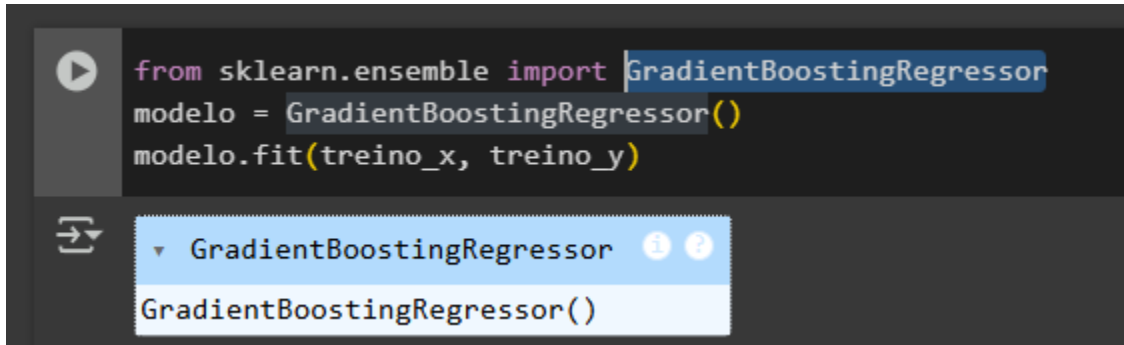
x = df.drop(['num_preco', 'dsc_produto', 'dat_criacao', 'dat_atualizacao', 'dsc_carro'], axis=1)
y = df['num_preco']

SEED = 15
np.random.seed(SEED)
treino_x, teste_x, treino_y, teste_y = train_test_split(x,
                                                         y,
                                                         test_size = 0.3)

print(f"Treino: {len(treino_x)} e Teste:{len(teste_x)}")
```

Treino: 83200 e Teste:35658

Após isso iremos escolher um modelo para seguir, neste primeiro momento escolhemos o modelo **GradientBoostingRegressor**. (porém informamos que estamos fazendo outros destes para assim definir qual modelo iremos seguir)

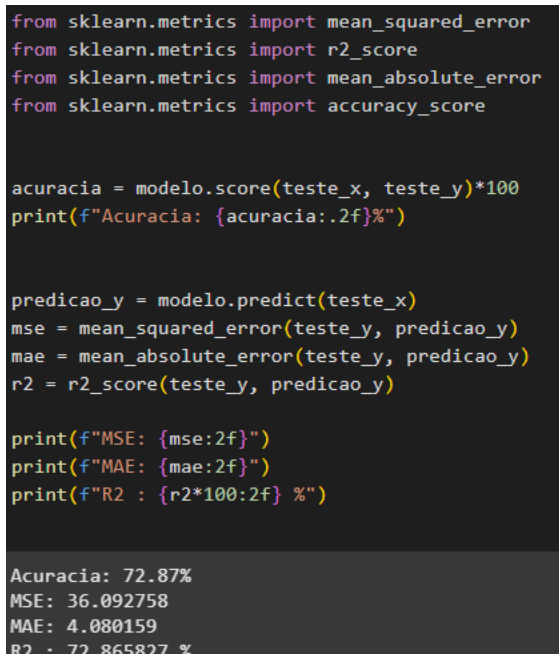


```
from sklearn.ensemble import GradientBoostingRegressor
modelo = GradientBoostingRegressor()
modelo.fit(treino_x, treino_y)
```

The screenshot shows a Jupyter Notebook interface. The top part is a code cell with the following Python code: `from sklearn.ensemble import GradientBoostingRegressor`, `modelo = GradientBoostingRegressor()`, and `modelo.fit(treino_x, treino_y)`. The bottom part is a variable inspector showing the `GradientBoostingRegressor` object, with its class `GradientBoostingRegressor()` listed below it.

Agora iremos testar a acurácia deste modelo para entender se podemos seguir com ele ou não. Temos alguns outros modelos no código que usamos para testar também este modelo, optamos por colocar todos.

Segue abaixo:



```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score

acuracia = modelo.score(teste_x, teste_y)*100
print(f"Acuracia: {acuracia:.2f}%")

predicao_y = modelo.predict(teste_x)
mse = mean_squared_error(teste_y, predicao_y)
mae = mean_absolute_error(teste_y, predicao_y)
r2 = r2_score(teste_y, predicao_y)

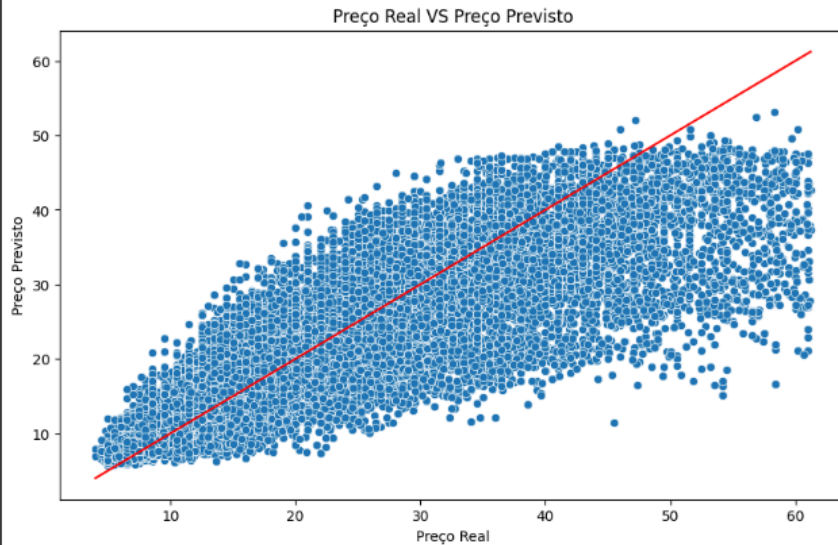
print(f"MSE: {mse:2f}")
print(f"MAE: {mae:2f}")
print(f"R2 : {r2*100:2f} %")
```

The screenshot shows a Jupyter Notebook cell with Python code for evaluating the model. The code imports `mean_squared_error`, `r2_score`, `mean_absolute_error`, and `accuracy_score` from `sklearn.metrics`. It then calculates the accuracy, MSE, MAE, and R2 score. The output of the cell is: `Acuracia: 72.87%`, `MSE: 36.092758`, `MAE: 4.080159`, and `R2 : 72.865827 %`.

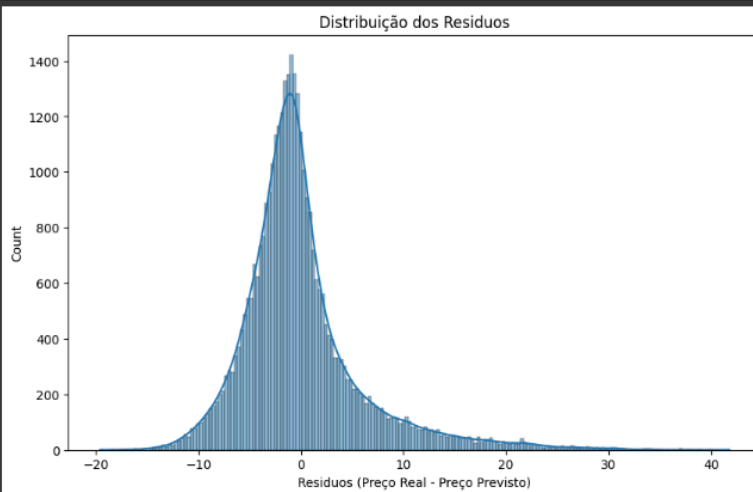
Abaixo segue alguns gráficos que produzimos para entender melhor a relação entre os dados.

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10,6))
sns.scatterplot(x=teste_y, y=predicao_y)
plt.plot([min(teste_y), max(teste_y)], [min(teste_y), max(teste_y)],
         color='red')
plt.xlabel("Preço Real")
plt.ylabel("Preço Previsto")
plt.title("Preço Real VS Preço Previsto")
plt.show()
```



```
plt.figure(figsize=(10,6))
sns.histplot(teste_y - predicao_y, kde=True)
plt.xlabel("Resíduos (Preço Real - Preço Previsto)")
plt.title("Distribuição dos Resíduos")
plt.show()
```





Este último gráfico é extremamente interessante para entender qual os dados mais correlacionados entre eles. Percebemos que distância e tempo de corrida estão muito relacionados com o num_preco, que seria assim interpretado como valor da corrida.

Agora criamos também uma função que será usada para prever o valor e entender se nosso ML está ou não funcionando com o modelo que escolhemos.

```
def preverPreco(distancia, tempo_corrida_seg):
    novo_dado = pd.DataFrame(columns=treino_x.columns)

    novo_dado.loc[0, 'num_distancia_m'] = distancia
    novo_dado.loc[0, 'num_tempo_estimado_seg_1'] = tempo_corrida_seg

    return modelo.predict(novo_dado)[0]

horario = 1305.8
distancia = 2500

preco_previsto = preverPreco(distancia, horario)

print("\nEXEMPLO DE PREDIÇÃO:\n")
print(f"Distância: {distancia}")
print(f"Tempo de corrida: {horario}")
print(f"VALOR PREVISTO R$: {preco_previsto:3.2f}")
```

Assim basicamente usamos tempo de corrida e Distância para prever qual seria o valor comparado ao Uber.

- O modelo testado não é o modelo apresentado no aplicativo final.