UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

CENTRO DE TECNOLOGIA

ESCOLA POLITÉCNICA

EEL891 - Introdução ao Aprendizado de Maquina

Trabalho 02

2024.2

Abraão Carvalho Gomes - 121066101

1 . Introdução

Este relatório documenta o processo de análise e modelagem preditiva realizado a partir dos dados fornecidos para o trabalho de disciplina, abrangendo as etapas de pré-processamento dos dados, seleção de atributos, experimentação com diferentes modelos preditivos e ajustes de hiperparâmetros, além das técnicas de validação utilizadas e resultados intermediários alcançados.

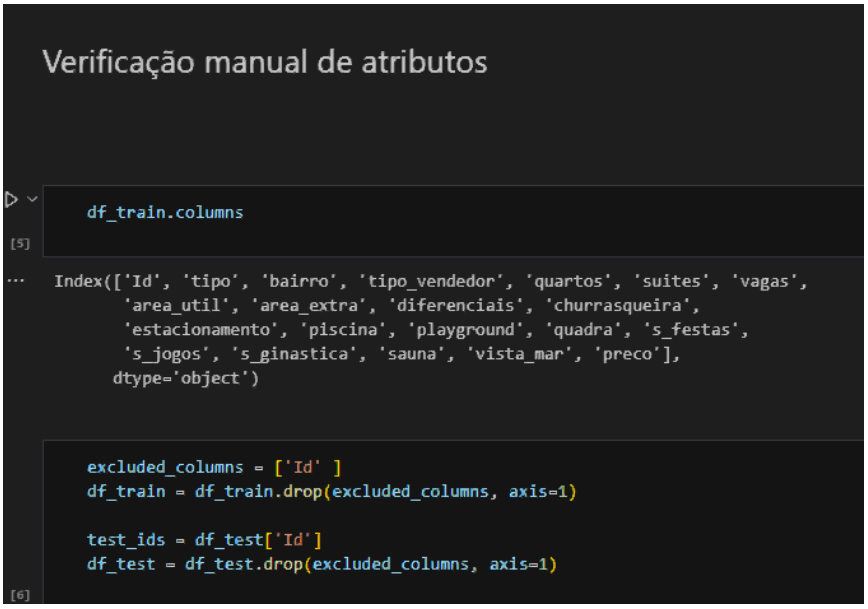
O trabalho foi primeiramente desenvolvido com notebook jupyter que permite melhor interação para visualizar dados e alterar trechos e código rapidamente. Após certo ponto, diversas versões de scripts python foram desenvolvidas.

---

2. Pré-Processamento dos Dados

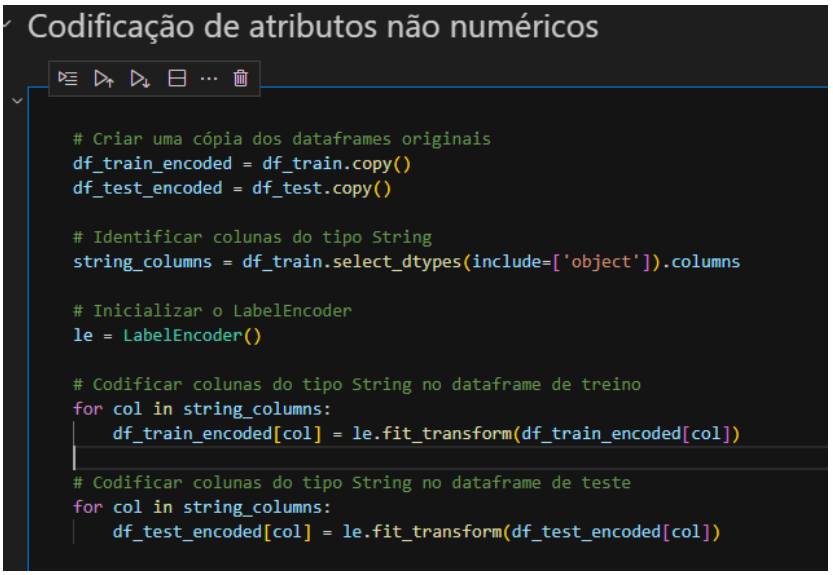
2.1. Seleção Manual de Atributos Não Relacionados

A princípio, apenas o ‘Id’ foi descartado sem prévia visualização, pois é um valor único para cada imóvel e não reflete o valor desse.



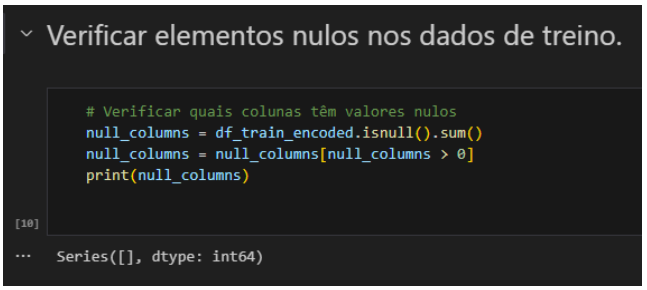
2.2. Codificação de Atributos Não Numéricos

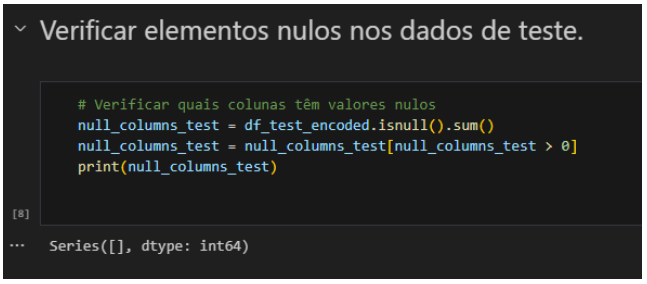
Utilizando o LabelEncoder, cada coluna não numérica é codificada para que se possa visualizar melhor com plots e permitir que o atributo seja utilizado por certos modelos. Essas características não numéricas codificadas foram: 'tipo', 'bairro', 'tipo_vendedor', 'diferenciais'.



2.3. Verificação de Elementos Nulos nos Dados de Teste

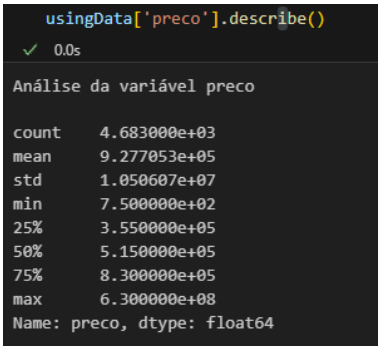
Utilizando a agregação de funções “.isnull().sum()” do pandas, é possível visualizar quais colunas têm valores nulos / NaN e quantos valores nulos elas têm. Nesse dataset (treino e teste), não haviam valores nulos a serem tratados.

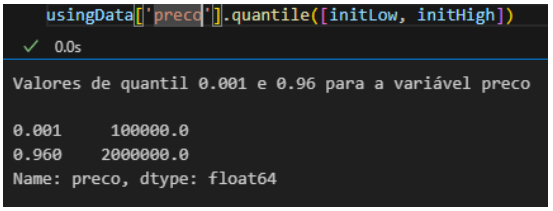




2.4. Análise e Remoção de Outliers

A partir de dois parâmetros reguláveis, é possível remover uma baixa porcentagem de linhas que contêm valores irreais, possivelmente preenchidos incorretamente. Exemplo: Um outlier evidente na variável alvo ‘preço’ é um imóvel custando 750 reais, o que é impossível e deve ser retirado. Desse modo, nesse exemplo, todos os valores até algo em torno de 60 - 100 mil reais poderiam ser desconsiderados. Para esse pré-processamento, foram utilizadas as funções ‘.describe()’, ‘.quantile([initLow, initHigh])’, dentre outras, para observar os valores nos extremos de cada atributo e filtrar o que foi julgado necessário.





2.5. Normalização de Dados

A fim de regularizar as variáveis, mantendo a escala das variáveis próximas e aproximando a distribuição de dados de uma distribuição normal. Isso é o ideal para alguns modelos treinarem de forma eficaz.

Foi utilizado o StandartScaler do scikit-learn.

```
# Normalizar os dados
scaler = StandardScaler()
# scaler_test = StandardScaler()
predictors = df_train.drop(columns=['preco']).columns
df_train[predictors] = scaler.fit_transform(df_train[predictors])
df_test[predictors] = scaler.transform(df_test[predictors])
```

2.6. Análise de Correlação Linear.

Foi feito um filtro de correlação linear para determinar as variáveis com maior correlação em relação à variável objetivo. Após testes concluiu-se que filtrar as variáveis de modo a selecionar as de maior correlação afetava a acurácia do modelo positivamente.

```
## Encontrar variáveis de maior correlação

corr_scaled = df_train.corr()

# Identificar colunas com correlação maior que k
k = 0.5
high_correlation_columns = corr_scaled.columns[corr_scaled['preco'] > k].drop('preco').tolist()

# Criar um novo DataFrame apenas com essas colunas
X_scaled = df_train[high_correlation_columns].to_numpy()
X_test_scaled = df_test[high_correlation_columns].to_numpy()

print(len(df_train),len(X_scaled))

print(f'Colunas com alta correlação: {high_correlation_columns}')
```

2.7. Uso de redução de dimensionalidade.

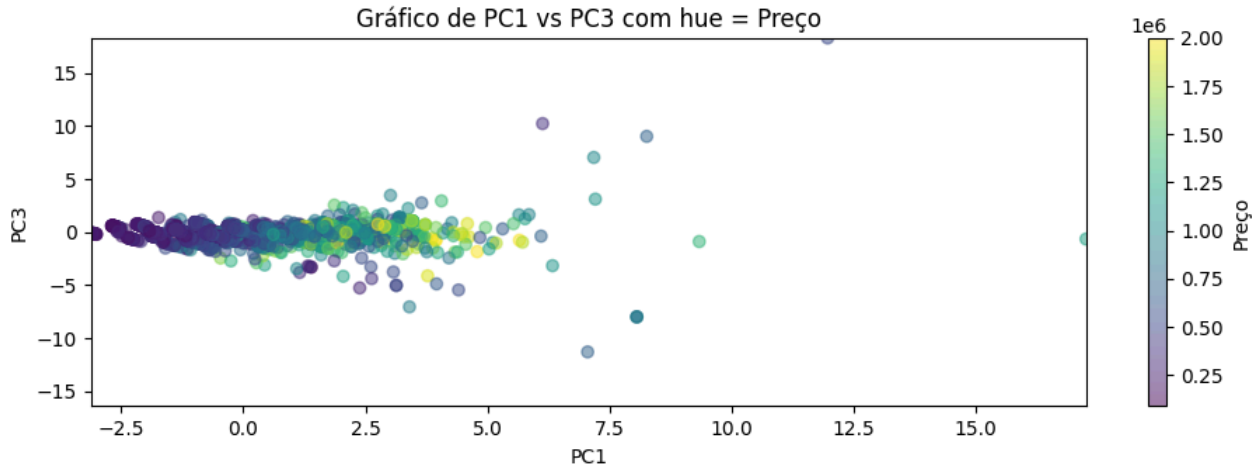
Foram testados alguns métodos de redução de dimensionalidade como PCA e variações, ICA, UMAP, dentre outros. Concluiu-se com testes que a seleção de componentes principais por meio do PCA tinha um impacto positivo na acurácia do modelo.

```python
# Aplicar PCA

# Fixar número de componentes
n_components = 0.95
pca = PCA(n_components=n_components)

X_pca = pca.fit_transform(X_scaled)

pca_df = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in range(X_pca.shape[1])], index = df_train.index)
pca_df['preco'] = df_train['preco']
```

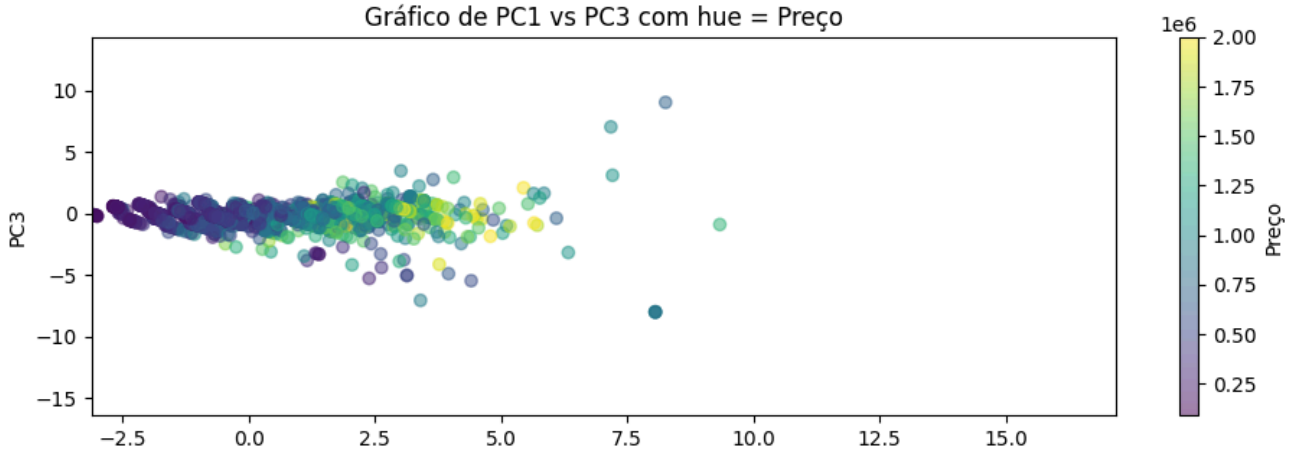
2.8. Análise e Remoção de Outliers após PCA

Percebeu-se que após a transformação, havia poucos valores de componentes principais bem distantes do restante, sendo reconhecidos como outliers. Esses valores foram removidos antes do treinamento. Apesar dos resultados não serem tão significativos, de fato isso diminuiu a métrica de RMSPE em torno de 5 % durante validação.

Antes:



Após:



Trecho do código:

```
# Segunda camada de remoção de outliers (Dentro dos componentes principais)
lowQ = 0.00001 # 0#
HighQ = 0.9999 # 1#

quantile_low = pca_df.quantile(lowQ)
quantile_high = pca_df.quantile(HighQ)

# print('Análise de Quantil -- Segunda redução de outliers')
# print(f'Low quantil 2: \n {quantile_low} ; High quantil 2: \n {quantile_high}')


if(lowQ == 0 and HighQ == 1):
    pca_df = pca_df
else:
    for column in pca_df.columns.drop('preco'):
        pca_df = pca_df[(pca_df[column] >= quantile_low[column]) & (pca_df[column] <= quantile_high[column])]

X_pca_df = pca_df.drop(columns=['preco']).to_numpy()
Y_pca_series = pca_df['preco']
```

2.9. Divisão em dados de treino e validação

Por fim, é uma etapa importante dividir os dados de treino nos que realmente serão usados para treinar o modelo e um segmento aleatório dos dados que será separado para validar o modelo. As métricas de erro para o segmento treino são enviesadas, pois o modelo “conhece” esses dados, mas não para o de validação, que são dados inéditos para o modelo treinado.

```
# Dividir os dados em treino e validate
X_train_pca, X_validate_pca, y_train, y_validate = train_test_split(X_pca_df, Y_pca_series, test_size=0.1,

X_train_pca_df = pd.DataFrame(X_train_pca, columns=[f'PC{i+1}' for i in range(X_train_pca.shape[1])])

X_validate_pca_df = pd.DataFrame(X_validate_pca, columns=[f'PC{i+1}' for i in range(X_validate_pca.shape[1]
```

---

4. Modelos Preditivos e métricas de desempenho

4.1. Modelos Testados e seleção

- Lista dos modelos
    - Regressão Linear
    - Regressor XGB
    - Regressor KNN
    - Random Forest
    - Extra Trees
    - Gradient Boosting Regressor

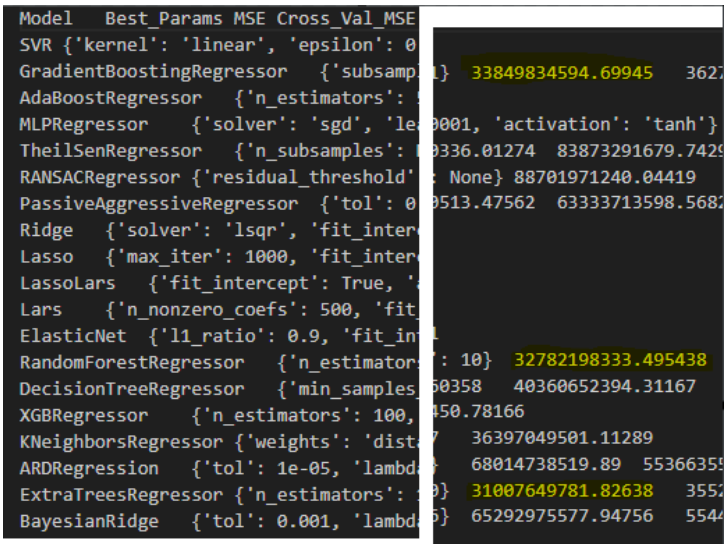
- Kernel Ridge
- Lasso
- LassoLars
- SVR
- AdaBoostRegressor
- MLPRegressor
- TheilSenRegressor
- RANSACRegressor
- PassiveAggressiveRegressor
- Lars
- ElasticNet
- DecisionTreeRegressor
- ARDRegression

Foi realizado um benchmark extenso em um script específico nomeado no respositório como *"script_test_6_benchmark.py".* Nesse script, determinou-se um conjunto de hyperparâmetros possíveis para cada modelo acima. Em seguida, para cada modelo uma execução do "RandomizedSearch" foi realizada, buscando melhores parâmetros dentre esse conjunto limitado. Os melhores hyperparâmetros foram utilizados para realizar uma predição com os dados de validação, assim como validação cruzada, e esses resultados armazenados em um dataFrame, que foi ao fim escrito em um arquivo .txt.

Trecho do código:

```
# Realizar RandomizedSearchCV para cada modelo
for model_name in models:
    model = models[model_name]
    param_grid = param_grids[model_name]

    random_search = RandomizedSearchCV(model, param_distributions=pa
    random_search.fit(X_pca_df, y_train)

    best_model = random_search.best_estimator_
    best_params = random_search.best_params_
    y_pred = best_model.predict(X_validate_pca_df)
    mse = mean_squared_error(y_validate, y_pred)
    cross_val_mse = -cross_val_score(best_model, X_pca_df, y_train,
    print(f"Mean Squared Error: {mse}")
    results_df = pd.concat([results_df, pd.DataFrame([{
        'Model': model_name,
        'Best_Params': best_params,
        'MSE': mse,
        'Cross_Val_MSE': cross_val_mse
    }])], ignore_index=True)

# Imprimir os resultados
print(results_df)

# Escrever os resultados em um arquivo txt
results_df.to_csv('benchmark_results.txt', sep='\t', index=False)
```

A partir do output desse código, foi possível ter uma noção por alto quais modelos tiveram melhor desempenho com base nas métricas de desempenho:



A imagem foi cortada omitindo os hyperparametros e permitiu a conclusão que os modelos Gradient Bossting Reg., Random Forest Regressor e Extra Trees Regressor foram os que melhor desempenharam com esse conjunto de dados.

4.2. Ajustes de Hiperparâmetros

Foi utilizada a função do scikit-learn “GridSearch” em cada modelo dos três acima e em alguns outros, e para diversos parâmetros que controlam os dados (diferentes quantis de outliers, diferentes valores mínimos de correlação linear, diferentes métodos de redução de dimensionalidade, etc.). O GridSearch testa todas as combinações possíveis, sendo um método mais confiável do que a função RandomizedSearch
Dessa forma, foi possível incluir outras possibilidades de hyperparâmetros e melhorar ainda mais o modelo.

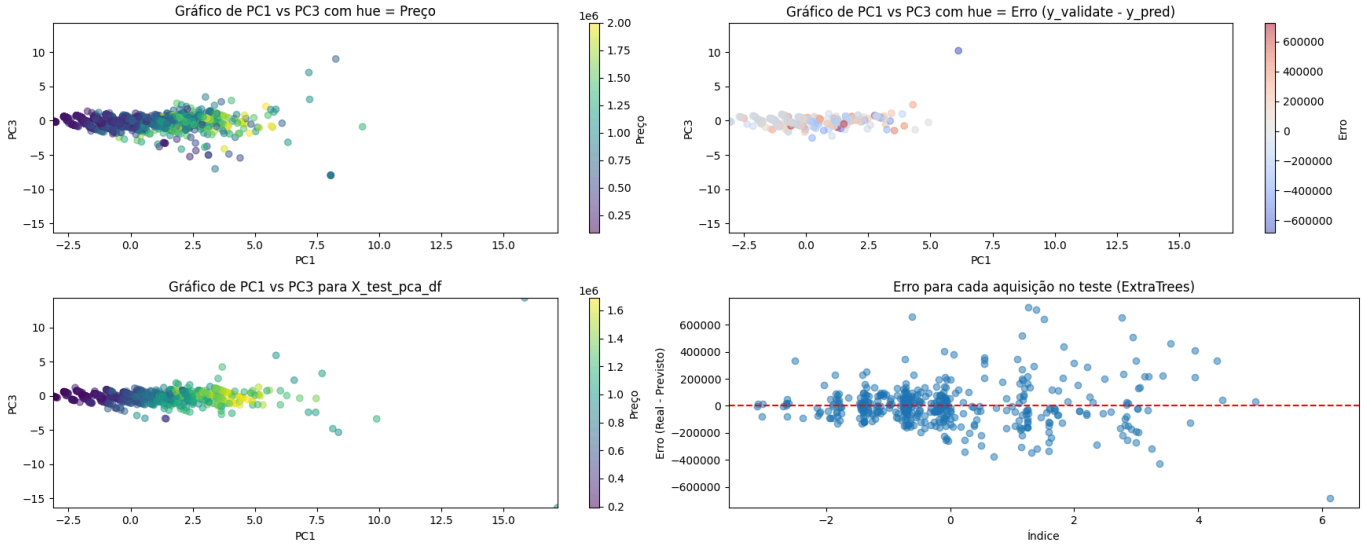
Trecho do código:

```python
# Benchmark
# Definir o modelo
model = ExtraTreesRegressor(random_state=42 ) # ,n_estimators =  500, min_sam

# Definir o grid de hiperparâmetros
param_grid = {
    'n_estimators': [100, 200, 500 , 800],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [None, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 6, 8],
}

# Configurar o GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
                           cv=5, n_jobs=-1, scoring='neg_mean_squared_error')

# Realizar o fit do GridSearchCV
grid_search.fit(X_pca_df, y_train)

# Obter os melhores parâmetros e o melhor modelo
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
```

4.3 Visualização dos resultados

Alguns plots foram utilizados para visualizar os resultados, mas o principal utilizado no final foi o seguinte, que mostra:

- Gráfico dos dados de treino PCX X PCY (Duas componentes principais distintas), com pontos coloridos com base no valor do imóvel. Esse gráfico permite visualizar a distribuição de dados nessas duas componentes e possíveis agrupamentos formados.
- Gráfico dos dados de validação PCX X PCY (Duas componentes principais distintas), com pontos coloridos com base no erro de predição dos pontos de validação. Esse gráfico permite visualizar a distribuição de dados nessas duas componentes e como o erro se distribui em função das componentes.
- Gráfico dos dados de teste PCX X PCY (Duas componentes principais distintas), com pontos coloridos com base no valor do imóvel.
- Gráfico dos índices/Id pelo erro de predição dos pontos de validação dispostos. Permite visualizar o erro de predição em torno do 0.



É possível observar em função da componente principal nº1 três principais grupamentos de valores de imóveis, o que certamente ajuda no treinamento do modelo.

4.3. Resultados intermediários e finais

Ao longo do desenvolvimento do modelo final, diversos modelos intermediários foram produzidos, de forma que algumas etapas inclusas no relatório foram sendo incluidas aos poucos.

Dito isso, o modelo final usado é de Extra Trees tunado, utilizando:

- Como filtro inicial de outliers:

```
quantile_ranges['preco'] = [0.0001, 0.96]
quantile_ranges['diferenciais'] = [0.001, 0.999]
```

- Limite de correlação linear mínima: 0.5
- filtro final de outliers:

```
# Segunda camada de remoção de outliers
lowQ = 0.00001 # 0#
HighQ = 0.9999 # 1#
```

- Hyperparâmetros do modelo:

  N_estimator = 100 ; min_samples_split = 4 ; min_samples_leaf = 5 ; max_depth = 40

```
= ExtraTreesRegressor(random_state=42 ,n_estimators =  100, min_samples_split = 4, min_samples_leaf =  5, max_depth =  40)
```

O script com modelo final tem como output as seguintes informações:

Análise de Quantil -- Primeira redução de outliers
preco - Low quantil 1: 42536.85 ; High quantil 1: 2000000.0
diferenciais - Low quantil 1: 2.0 ; High quantil 1: 81.0

Colunas com alta correlação: ['quartos', 'suites', 'vagas', 'area_util']
Number of components: 4

Mean Squared Error: 23764890800.648567
Root Mean Squared Error: 154158.65464075824

***Root Mean Squared Percentage Error:** 0.2521897131990370*
5

Cross-validated Mean Squared Error: 32702525731.353474

Ou seja, o RMSPE esperado é de **0.252189.**

O resultado inicial obtido no Kaggle é de **0.3077.**

Repositório dos trabalhos:

https://github.com/AbraaoCG/EEL891_ML_projects