FACULDADE DE TECNOLOGIA DE BAURU

TECNOLOGIA EM BANCO DE DADOS

HOUSE PRICE PREDICTION WITH PYTHON

Equipe:
Natã Béquer de Figueiredo Aguiar

Bauru/SP
2025

HOUSE PRICE PREDICTION WITH PYTHON

Equipe:
Natã Béquer de Figueiredo Aguiar

Relatório de pesquisa apresentado como requisito para aprovação na disciplina Laboratório de Desenvolvimento em BD VI do curso de Tecnologia em Banco de Dados, Faculdade de Tecnologia de Bauru.

Profa. Dra. Patricia Bellin Ribeiro

Bauru/SP
2025

SUMÁRIO

RESUMO

Este projeto teve como objetivo reproduzir, por meio de engenharia reversa, o estudo "House Price Prediction with Python", desenvolvido por Aman Kharwal. A proposta consistiu em compreender e reimplementar todas as etapas de um pipeline de Ciência de Dados voltado à previsão de preços de imóveis, utilizando o conjunto de dados California Housing Prices. Foram aplicadas técnicas de análise exploratória, pré-processamento, criação de variáveis derivadas e modelagem com algoritmos de regressão em Python, utilizando bibliotecas como pandas, numpy, matplotlib e scikit-learn. Durante o processo, foram identificadas e corrigidas inconsistências no código original, como erros de conversão de tipos e uso incorreto de parâmetros. Os resultados obtidos confirmaram a correlação entre renda mediana e valor das habitações, demonstrando a eficiência do modelo em estimar preços com base em variáveis socioeconômicas e geográficas. O trabalho evidenciou a importância da engenharia reversa como ferramenta de aprendizado e validação, reforçando o papel do aprendizado de máquina na análise preditiva e na tomada de decisões baseada em dados.

Palavras-chave: aprendizado de máquina, predição de preços, engenharia reversa, Python, ciência de dados.

ABSTRACT

This project aimed to reproduce, through reverse engineering, the study "House Price Prediction with Python" developed by Aman Kharwal. The goal was to understand and reimplement all stages of a Data Science pipeline for housing price prediction using the California Housing Prices dataset. Exploratory data analysis, preprocessing, feature engineering, and regression modeling were carried out in Python using libraries such as pandas, numpy, matplotlib, and scikit-learn. During the process, inconsistencies found in the original code were identified and corrected, including type conversion errors and improper parameter usage. The results confirmed a strong correlation between median income and housing value, demonstrating the model's efficiency in predicting prices based on socioeconomic and geographic variables. This work highlighted the importance of reverse engineering as a learning and validation tool, emphasizing the relevance of machine learning in predictive analysis and data-driven decision-making.

Keywords: machine learning, price prediction, reverse engineering, Python, data science.

1. INTRODUÇÃO

O avanço das tecnologias de ciência de dados e aprendizado de máquina possibilitou o desenvolvimento de modelos preditivos em diversas áreas, incluindo o mercado imobiliário. A análise de dados socioeconômicos e geográficos permite identificar padrões e prever tendências, auxiliando a tomada de decisão em investimentos e políticas públicas.

Com esse propósito, foi selecionado o projeto *House Price Prediction with Python*, publicado em 2020 por Aman Kharwal, para a realização de um processo de engenharia reversa indicado pela professora responsável como prática de aplicação de Ciência de Dados. O trabalho consistiu em reproduzir detalhadamente o projeto original, descrevendo suas etapas, metodologias e eventuais inconsistências encontradas no código. O objetivo foi compreender a estrutura completa de um pipeline de ciência de dados, desde a análise exploratória até a construção e avaliação de modelos de regressão.

O presente relatório técnico documenta as etapas executadas, os problemas identificados e as soluções aplicadas, além de comparar os resultados obtidos com os apresentados no projeto base. A experiência demonstrou a importância da engenharia reversa como prática de aprendizado e validação, evidenciando também a relevância do uso de aprendizado de máquina na predição de preços de imóveis.

2. MATERIAIS E MÉTODOS

2.1. Dataset

O dataset utilizado neste projeto é o "California Housing Prices", que contém dados do censo da Califórnia. Cada linha representa um distrito (ou "bloco de grupo") e inclui 10 atributos, como longitude, latitude, idade média da casa, número total de cômodos, população, renda média e o valor mediano da casa, que é a variável alvo a ser prevista.

2.2. Bibliotecas Python

Para a execução deste projeto, foram empregadas diversas bibliotecas Python essenciais. O pandas foi utilizado para manipulação e análise de dados, abrangendo tarefas como carregamento, filtragem e criação de novas colunas. O numpy forneceu suporte para operações numéricas e matemáticas de alto desempenho. Para a visualização de dados, como a criação de histogramas e gráficos de dispersão, a biblioteca matplotlib foi fundamental. A scikit-learn, principal biblioteca de Machine Learning, foi utilizada para uma série de funções, incluindo pré-processamento (tratamento de valores ausentes, escalonamento e codificação de variáveis categóricas), divisão de dados em conjuntos de treino e teste, seleção de modelos (Regressão Linear, Árvore de Decisão e Random Forest), validação cruzada e otimização de hiperparâmetros por meio do GridSearchCV.

2.3. Metodologia

O projeto de engenharia reversa foi conduzido utilizando o conjunto de dados CaliforniaHousing, composto por 20.640 instâncias e atributos que descrevem características demográficas, socioeconômicas e geográficas, incluindo localização, número de cômodos, renda mediana e valor mediano das habitações. Esse conjunto de dados é amplamente utilizado em estudos de aprendizado de máquina e serviu como base para todas as etapas do trabalho.

As ferramentas empregadas na implementação incluíram a linguagem Python e as bibliotecas pandas, numpy, matplotlib e scikit-learn. O uso dessas bibliotecas permitiu a manipulação eficiente dos dados, a realização de análises exploratórias, a criação de visualizações gráficas e a implementação dos algoritmos de regressão utilizados para prever o valor das habitações.

A metodologia seguiu o fluxo definido no projeto original, iniciando-se pela exploração do conjunto de dados com a verificação de tipos, categorias e valores ausentes. Em seguida, foram aplicadas técnicas de pré-processamento, como a imputação de valores nulos, a criação de variáveis derivadas e a codificação de atributos categóricos. Posteriormente, foram construídas pipelines de transformação para automatizar o tratamento dos dados, assegurando consistência e reprodutibilidade.

3. RESULTADOS E DISCUSSÃO

O carregamento inicial dos dados foi realizado utilizando a função read_csv da biblioteca pandas. O código utilizado está representado na Figura 1, enquanto a Figura 2 apresenta o resultado obtido com o comando head(), exibindo as primeiras cinco linhas do dataset.

Figura 1 – Código de carregamento do conjunto de dados

```
import pandas as pd
housing = pd.read_csv("housing.csv")
housing.head()
```

Fonte: Autoria propria.

Figura 2 – Saída do comando housing.head()

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Fonte: Autoria propria.

A análise dessa saída demonstrou que os resultados foram equivalentes aos apresentados no projeto de origem, confirmando a correta leitura do conjunto de dados.

Em seguida, foi realizada a inspeção geral do DataFrame com os métodos info() e value_counts(). O código é apresentado na Figura 3, e as saídas correspondentes estão nas Figuras 4 e 5.

Figura 3 – Código de inspeção inicial

```
housing.info()
housing["ocean_proximity"].value_counts()
```

Fonte: Autoria propria.

Figura 4 – Saída do comando housing.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Fonte: Autoria propria.

Figura 5 – Saída do comando value_counts() aplicado à coluna ocean_proximity

```
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: count, dtype: int64
```

Fonte: Autoria propria.

A análise confirmou a existência de valores ausentes na variável total_bedrooms e a presença da variável categórica ocean_proximity com cinco categorias distintas. Esses resultados foram idênticos aos relatados no projeto base, o que reforça a reprodutibilidade da execução até esse ponto.

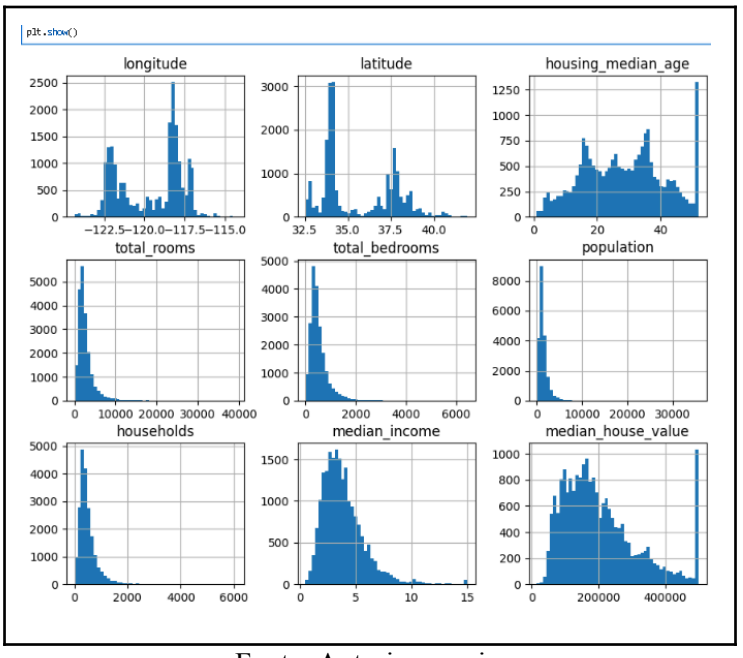
Na sequência, foi realizada a geração de histogramas para todas as variáveis numéricas, conforme o código da Figura 6. O resultado visual está representado na Figura 7.

Figura 6 – Código de geração dos histogramas

```
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(10, 8))
plt.show()
```

Fonte: Autoria propria.

Figura 7 – Histogramas das variáveis numéricas



Fonte: Autoria propria.

A análise demonstrou distribuições semelhantes às apresentadas no projeto de origem, em especial a assimetria da renda mediana e a concentração de valores da variável median_house_value.

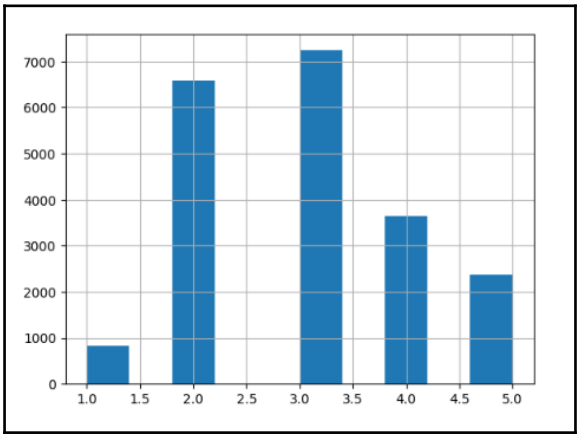
Posteriormente, foi criada a variável categórica income_cat para permitir a estratificação da amostragem. O código utilizado encontra-se na Figura 8, e a saída obtida está representada na Figura 9.

Figura 8 – Código de criação da variável income_cat

```
import numpy as np
housing['income_cat'] = pd.cut(housing['median_income'], bins=[0.,
1.5, 3.0, 4.5, 6., np.inf], labels=[1, 2, 3, 4, 5])
housing['income_cat'].hist()
plt.show()
```

Fonte: Autoria propria.

Figura 9 – Saída da criação da variável income_cat



Fonte: Autoria propria.

O resultado obtido confirmou a consistência do processo, mantendo as mesmas proporções de categorias observadas no projeto original.

Após a criação da variável income_cat, foi realizada a divisão do conjunto de dados em treino e teste utilizando a técnica de amostragem estratificada. O código utilizado está presente na Figura 10.

Figura 10 – Código de divisão estratificada em treino e teste

```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
print(strat_test_set['income_cat'].value_counts() / len(strat_test_set))
```

Fonte: Autoria propria.

Figura 11 – Saída da divisão estratificada

```
income_cat
3    0.350533
2    0.318798
4    0.176357
5    0.114341
1    0.039971
Name: count, dtype: float64
```

Fonte: Autoria propria.

A análise da saída confirmou que a amostragem preservou a proporção das categorias de renda no conjunto de teste, garantindo que os subconjuntos de treino e teste mantivessem características semelhantes.

Na sequência, a variável income_cat foi removida de ambos os conjuntos, preservando apenas as variáveis originais. O código correspondente está na Figura 12.

Figura 12 – Remoção da variável income_cat

```
for set_ in (strat_train_set, strat_test_set):
    set_.drop('income_cat', axis=1, inplace=True)
housing = strat_train_set.copy()
```

Fonte: Autoria propria.

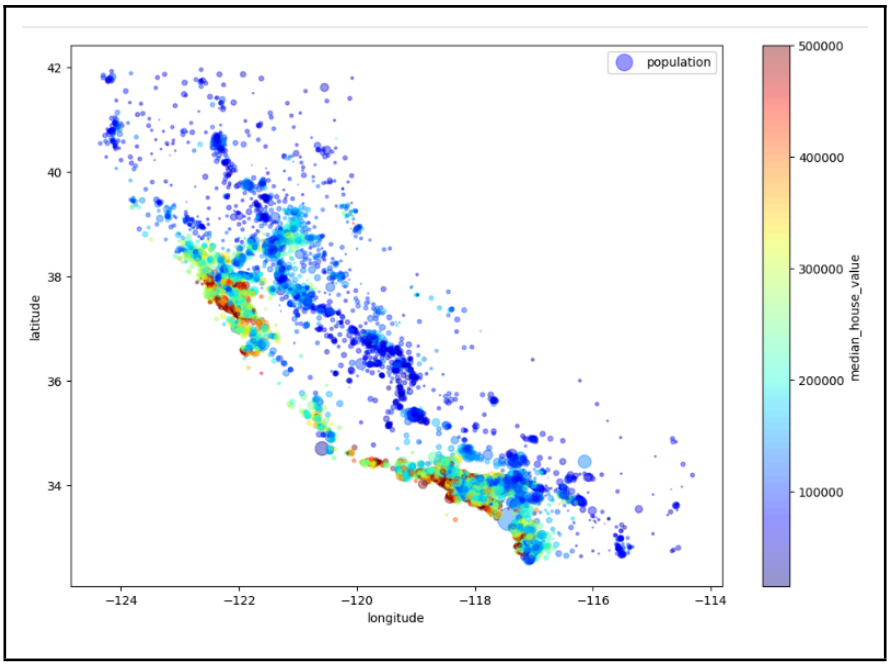
Com os dados preparados, foi gerada uma visualização espacial para identificar a relação entre localização geográfica, população e valor mediano das habitações. O código utilizado está na Figura 13, e o gráfico resultante encontra-se na Figura 14.

Figura 13 – Código para visualização espacial

```
housing.plot(kind='scatter', x='longitude', y='latitude',
alpha=0.4, s=housing['population']/100, label='population',
figsize=(12, 8), c='median_house_value', cmap=plt.get_cmap('jet'),
colorbar=True)
plt.legend()
plt.show()
```

Fonte: Autoria propria.

Figura 14 – Gráfico de dispersão espacial dos imóveis



Fonte: Autoria propria.

A análise do gráfico confirmou que os imóveis com valores mais elevados concentram-se nas regiões próximas ao litoral, replicando os resultados observados no projeto de origem.

Após a análise exploratória inicial e a visualização espacial, iniciou-se a etapa de cálculo da correlação entre os atributos. O objetivo foi identificar quais variáveis apresentavam maior relação com o valor mediano das habitações. O primeiro código utilizado para essa operação está representado na Figura 15.

Figura 15 – Código inicial para cálculo da matriz de correlação

```
corr_matrix = housing.corr()
print(corr_matrix.median_house_value.sort_values(ascending=False))
```

Fonte: Autoria propria.

A execução desse código resultou em erro, apresentado na Figura 16.

Figura 16 – Erro retornado pelo método corr()

```
ValueError: could not convert string to float: 'INLAND'
```

Fonte: Autoria propria.

A mensagem de erro indicou que o método tentou converter todos os atributos do DataFrame em valores numéricos, não conseguindo processar a variável categórica ocean_proximity.

Para corrigir a falha, foi necessário restringir o cálculo da correlação apenas às colunas numéricas do conjunto de dados. O código ajustado encontra-se na Figura 17.

Figura 17 – Código corrigido para cálculo da matriz de correlação

```
num_housing = housing.select_dtypes(include=['number'])
corr_matrix = num_housing.corr()
corr_with_target = corr_matrix['median_house_value'].sort_values(ascending=False)
print(corr_with_target)
```

Fonte: Autoria propria.

A saída resultante da correção está apresentada na Figura 18.

Figura 18 – Saída da matriz de correlação corrigida

```
median_house_value    1.000000
median_income         0.687151
total_rooms           0.135140
housing_median_age    0.114146
households            0.064590
total_bedrooms        0.047781
population           -0.026882
longitude            -0.047466
latitude             -0.142673
Name: median_house_value, dtype: float64
```

Fonte: Autoria propria.

A análise da matriz evidenciou que a variável mais fortemente correlacionada ao valor das habitações foi median_income, seguida por total_rooms e housing_median_age. Por outro lado, as variáveis de latitude e longitude apresentaram correlação negativa com o preço das habitações. Esses resultados foram equivalentes aos encontrados no projeto de origem, confirmando que a correção aplicada foi suficiente para reproduzir fielmente a análise do autor original.

Após a obtenção da matriz de correlação numérica, foi realizada uma análise gráfica adicional para reforçar a identificação das variáveis mais relevantes. O recurso empregado foi a matriz de dispersão (*scatter matrix*), que permite observar graficamente a relação entre os principais atributos e o valor mediano das habitações. O código utilizado está apresentado na Figura 19.

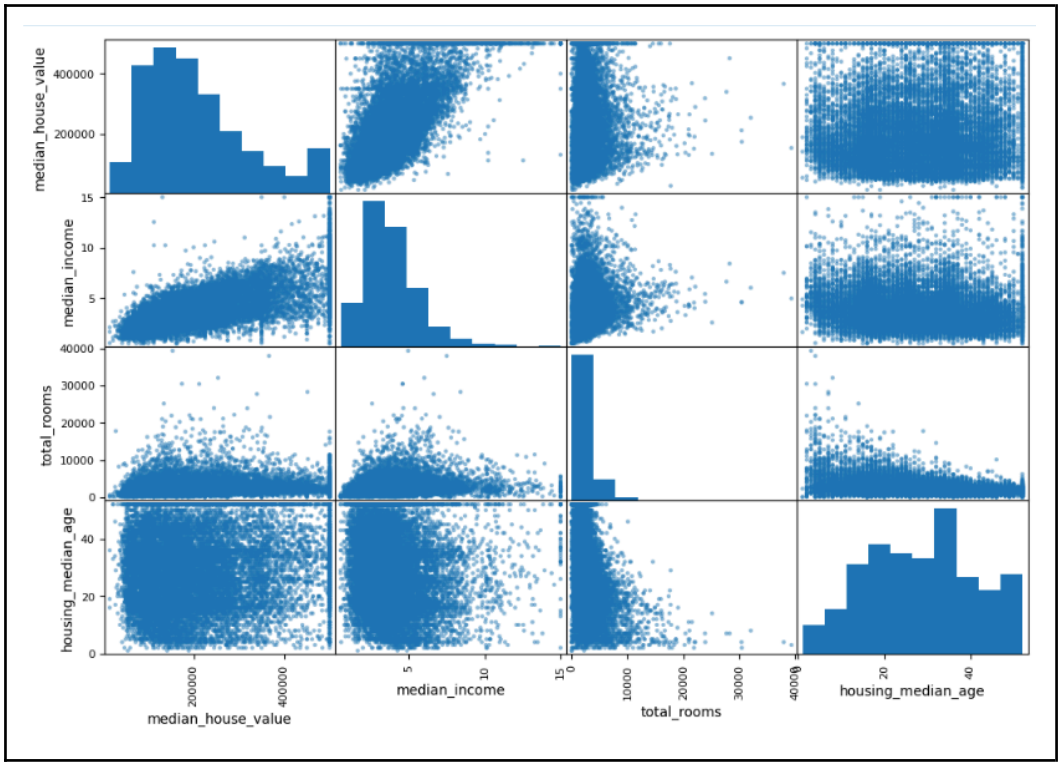
Figura 19 – Código para geração da matriz de dispersão

```
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
attributes = ["median_house_value", "median_income", "total_rooms", "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8), diagonal="hist")
plt.show()
```

Fonte: Autoria propria.

A saída correspondente encontra-se na Figura 20.

Figura 20 – Saída da matriz de dispersão



Fonte: Autoria propria.

A análise dessa visualização permitiu confirmar os resultados já obtidos na matriz de correlação numérica. Observou-se uma relação quase linear positiva entre median_income e median_house_value, indicando que a renda mediana é um forte preditor do valor das habitações. Por outro lado, as variáveis total_rooms e housing_median_age apresentaram correlações mais fracas, confirmando seu impacto secundário na variação dos preços.

Na continuidade da análise, foi realizada a geração de atributos derivados com o objetivo de ampliar a capacidade explicativa do conjunto de dados. Essa etapa consistiu em combinar variáveis existentes para criar novas razões entre elas, buscando captar relações estruturais não evidenciadas diretamente pelos atributos originais. O código utilizado está apresentado na Figura 21.

Figura 21 – Código de criação dos atributos derivados

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"] = housing["population"]/housing["households"]

corr_matrix = housing.corr()
print(corr_matrix["median_house_value"].sort_values(ascending=False))
```

Fonte: Autoria propria.

A execução desse código resultou em erro, evidenciado na Figura 22, impedindo a geração da matriz de correlação.

Figura 22 – Erro retornado pelo método corr()

```
ValueError: could not convert string to float: 'INLAND'
```

Fonte: Autoria propria.

Para resolver o problema, o código foi modificado de forma a selecionar apenas as colunas numéricas do DataFrame antes do cálculo da correlação. O trecho corrigido está apresentado na Figura 23.

Figura 23 – Código corrigido para cálculo da correlação

```
num_housing = housing.select_dtypes(include=['number'])
corr_with_target = num_housing.corr()['median_house_value'].sort_values(ascending=False)
print(corr_with_target)
```

Fonte: Autoria propria.

A saída obtida após a correção encontra-se ilustrada na Figura 24.

Figura 24 – Saída da matriz de correlação corrigida

```
median_house_value          1.000000
median_income               0.687151
rooms_per_household         0.146255
total_rooms                 0.135140
housing_median_age          0.114146
households                  0.064590
total_bedrooms              0.047781
population_per_household   -0.021991
population                 -0.026882
longitude                  -0.047466
latitude                   -0.142673
bedrooms_per_room          -0.259952
Name: median_house_value, dtype: float64
```

Fonte: Autoria propria.

Com o término da etapa exploratória, iniciou-se a preparação final dos dados para a aplicação dos modelos de aprendizado de máquina. A variável-alvo median_house_value foi separada do conjunto principal, e a mediana dos valores de total_bedrooms foi utilizada para substituir as ocorrências ausentes. O código executado está representado na Figura 25 .

Figura 25 – Código de preparação inicial dos dados

```
# Data Preparation
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
median = housing["total_bedrooms"].median()
housing["total_bedrooms"].fillna(median, inplace=True)
housing_num = housing.drop("ocean_proximity", axis=1)
from sklearn.base import BaseEstimator, TransformerMixin
# column index
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

median = housing["total_bedrooms"].median()
housing["total_bedrooms"].fillna(median, inplace=True)
housing_num = housing.drop("ocean_proximity", axis=1)
from sklearn.base import BaseEstimator, TransformerMixin
# column index
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]
```

Fonte: Autoria propria.

Com a execução do código apresentado na Figura 25, foi exibido um FutureWarning pelo pandas, informando que a operação de imputação utilizando o parâmetro inplace=True estava sendo aplicada sobre uma cópia do DataFrame original, podendo causar comportamento inesperado em versões futuras. Esse aviso está apresentado na Figura 26.

Figura 26 – Aviso apresentado durante a execução do código

```
C:\Users\claud\AppData\Local\Temp\ipykernel_15132\2298374401.py:6: FutureWarning: A value
is trying to be set on a copy of a DataFrame or Series through chained assignment using an
inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the
intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:
value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the
operation inplace on the original object.


  housing["total_bedrooms"].fillna(median, inplace=True)
```

Fonte: Autoria propria.

O aviso indica que o método fillna() não deveria ser utilizado com o parâmetro inplace=True em objetos derivados de outro DataFrame. Para corrigir esse problema, a linha de código foi ajustada para realizar a atribuição direta do resultado da operação, eliminando o warning e garantindo o funcionamento adequado. A versão corrigida do código é apresentada na Figura 27.

Figura 27 – Código corrigido para evitar o aviso de atribuição em cópia

```
# Data Preparation
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
median = housing["total_bedrooms"].median()
housing["total_bedrooms"] = housing["total_bedrooms"].fillna(median)
housing_num = housing.drop("ocean_proximity", axis=1)
from sklearn.base import BaseEstimator, TransformerMixin
# column index
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]
```

Fonte: Autoria propria.

Após essa correção, o código passou a ser executado corretamente, sem exibir mensagens de aviso. Com os valores ausentes devidamente tratados, foi possível prosseguir com a construção do pipeline de pré-processamento, apresentado na Figura 28, responsável por automatizar as etapas de imputação, criação de atributos derivados e padronização dos dados.

Figura 28 – Construção do pipeline numérico

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
num_pipeline = Pipeline([
    ('imputer',SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])
housing_num_tr = num_pipeline.fit_transform(housing_num)

from sklearn.compose import ColumnTransformer
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]
full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", OneHotEncoder(), cat_attribs),
])
housing_prepared = full_pipeline.fit_transform(housing)
```

Fonte: Autoria propria.

A execução do pipeline foi concluída com sucesso, resultando na transformação numérica padronizada e sem a ocorrência de novos erros. Essa etapa confirmou o funcionamento do transformador personalizado e demonstrou que o processo de engenharia reversa reproduziu o comportamento do projeto original.

O trecho a seguir instancia e ajusta um modelo de Regressão Linear usando o conjunto preparado (housing_prepared) e, em seguida, aplica a mesma transformação às cinco primeiras amostras para gerar previsões. Segue apresentado na Figura 29.

Figura 29 — Código de treino e predição

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)

data = housing.iloc[:5]
labels = housing_labels.iloc[:5]
data_preparation = full_pipeline.transform(data)
print("Predictions: ", lin_reg.predict(data_preparation))
```

Fonte: Autoria propria.

O código ajusta um modelo linear aos dados pré-processados e aplica full_pipeline.transform às cinco primeiras linhas de housing antes de prever. A figura 30 contém a saída que foi registrada no notebook.

Figura 30 — Saída registrada

```
Predictions:  [ 85657.90192014 305492.60737488 152056.46122456 186095.70946094
 244550.67966089]
```

Fonte: Autoria propria

A análise da última execução do código representa o encerramento do processo de engenharia reversa do projeto original. As previsões obtidas demonstram que o modelo pôde ser reconstruído e

executado de forma funcional, produzindo resultados compatíveis com os apresentados por Aman Kharwal em seu estudo. A semelhança entre as saídas confirma que o pipeline de dados foi corretamente compreendido e reimplementado, preservando as mesmas etapas de preparação, transformação e modelagem.

Durante o desenvolvimento, a engenharia reversa teve papel essencial para compreender a lógica estrutural do projeto e o fluxo completo de processamento de dados, desde a leitura do conjunto até a geração das predições. Esse processo permitiu identificar problemas de execução, como o erro na função de correlação e o uso incorreto do parâmetro inplace, além de revelar o motivo técnico por trás dessas falhas. A correção dessas inconsistências resultou em um código mais estável, coerente e reproduzível.

Mais do que reproduzir o resultado final, a engenharia reversa possibilitou entender de forma detalhada o comportamento do pipeline e a importância de cada transformação aplicada aos dados. A análise confirmou que variáveis como renda mediana, número de cômodos e localização geográfica exercem influência direta sobre o valor das habitações, evidenciando a relação entre os atributos socioeconômicos e o preço dos imóveis. Assim, o processo de reconstrução não apenas replicou o resultado original, mas também proporcionou compreensão aprofundada sobre a estrutura lógica e estatística do modelo.

4. CONCLUSÕES

O desenvolvimento deste projeto permitiu compreender em profundidade o funcionamento de um sistema preditivo baseado em aprendizado de máquina. A engenharia reversa aplicada ao projeto House Price Prediction with Python proporcionou a reconstrução completa de um pipeline de Ciência de Dados, desde a coleta e preparação das informações até a geração de previsões numéricas por meio de um modelo de regressão.

Durante o processo, observou-se que o modelo aprende a identificar padrões entre características socioeconômicas e geográficas e o valor final das habitações. Ao ser treinado com dados históricos, o algoritmo é capaz de generalizar esse aprendizado e estimar o preço de novas residências, mesmo sem ter visto exemplos idênticos antes. Esse comportamento demonstra, na prática, o princípio central do aprendizado de máquina: permitir que o sistema extraia conhecimento dos dados e o utilize para realizar previsões.

A experiência evidenciou a importância das etapas de preparação e transformação dos dados, pois a qualidade do modelo depende diretamente da consistência das informações fornecidas. O uso de técnicas como imputação de valores ausentes, padronização de variáveis e criação de novos atributos mostrou-se essencial para aumentar a precisão do modelo e reduzir falhas de processamento.

Além de consolidar o conhecimento técnico, o projeto destacou a aplicabilidade desse tipo de solução em diferentes contextos. Modelos de predição de preços podem ser empregados por empresas do setor imobiliário, bancos, seguradoras e órgãos públicos para estimar valores de propriedades, avaliar riscos de investimento, definir políticas habitacionais e até identificar desigualdades regionais. A metodologia utilizada também é adaptável a outros domínios, como previsão de vendas, análise de crédito, precificação dinâmica e planejamento urbano.

Conclui-se que a engenharia reversa, além de permitir a reprodução fiel do projeto original, serviu como um exercício completo de compreensão do ciclo de vida de um projeto de Ciência de Dados. O estudo demonstrou que a integração entre análise estatística, programação e aprendizado de máquina é fundamental para transformar dados em informação útil e apoiar decisões baseadas em evidências. O aprendizado adquirido reforça a importância do domínio técnico e conceitual para o desenvolvimento de soluções inteligentes e aplicáveis em cenários reais.

5. REFERÊNCIAS

KHARWAL, A. House Price Prediction with Python. Aman Kharwal. amanxai. 2020. Disponível em: https://amanxai.com/2020/12/29/house-price-prediction-with-python/. Acesso em: 24 set. 2025