import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import StratifiedShuffleSplit

#cargar datos

df = pd.read\_csv('/content/housing - borr.csv')

# Crear una nueva columna 'income\_cat' basada en 'median\_income'

df['income\_cat'] = pd.cut(df['median\_income'],

                         bins=[0, 1.5, 3.0, 4.5, 6.0, np.inf],

                         labels=[1, 2, 3, 4, 5])

df['income\_cat'].hist()

# Definir la partición estratificada

split = StratifiedShuffleSplit(n\_splits=1, test\_size=0.2, random\_state=42)

# Realizar la división estratificada

for train\_index, test\_index in split.split(df, df['income\_cat']):

    strat\_train\_set = df.loc[train\_index]

    strat\_test\_set = df.loc[test\_index]

strat\_test\_set['income\_cat'].value\_counts()/len(strat\_test\_set)

#Remover atributo income\_cat

for set\_ in(strat\_train\_set,strat\_test\_set):

  set\_.drop('income\_cat',axis=1, inplace=True)

#Crear nuevos atributos

df["rooms\_per\_household"] = df["total\_rooms"]/df["households"]

df["bedrooms\_per\_room"] = df["total\_bedrooms"]/df["total\_rooms"]

df["population\_per\_household"] = df["population"]/df["households"]

# Seleccionar solo columnas numéricas para calcular la correlación

numeric\_df = df.select\_dtypes(include=[np.number])

# Calcular la matriz de correlación

corr\_matrix = numeric\_df.corr()

# Mostrar las correlaciones con 'median\_house\_value'

print(corr\_matrix["median\_house\_value"].sort\_values(ascending=False))

#Volver a un conjunto de entrenamiento limpio

df = strat\_train\_set.drop("median\_house\_value", axis=1)

df\_labels = strat\_test\_set["median\_house\_value"].copy()

#Reemplazar los valores faltantes de cada atributo con la mediana de ese atrinuto

from sklearn.impute import SimpleImputer

imputer = SimpleImputer (strategy="median")

#Crear una copia sin ocean proximity

df\_num = df.drop("ocean\_proximity", axis=1)

imputer.fit(df\_num)

imputer.statistics\_

df\_num.median().values

#Transformar el conjunto de entrenamiento reemplazando los valores faltantes con las medianas aprendidas

X = imputer.transform(df\_num)

#El resultado es una matriz de numpy con las caracteristicas transformadas

df\_tr = pd.DataFrame(X, columns=df\_num.columns, index=df\_num.index)

# Seleccionar la columna categórica

df\_cat = df[['ocean\_proximity']]

# Crear un codificador one-hot

cat\_encoder = OneHotEncoder(sparse=False)  # Usar sparse=False para obtener un array en lugar de una matriz dispersa

# Ajustar y transformar los datos

df\_cat\_1hot = cat\_encoder.fit\_transform(df\_cat)

# Convertir a un DataFrame con nombres de columnas

df\_cat\_1hot\_df = pd.DataFrame(df\_cat\_1hot, columns=cat\_encoder.get\_feature\_names\_out(['ocean\_proximity']))

print(df\_cat\_1hot\_df)

#transformador que agrega atributos combinados

from sklearn.base import BaseEstimator, TransformerMixin

rooms\_ix, bedrooms\_ix, population\_ix, households\_ix = 3,4,5,6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):

  def \_\_init\_\_(self,add\_bedrooms\_per\_room = True):

    self.add\_bedrooms\_per\_room = add\_bedrooms\_per\_room

  def fit(self, X, y=None):

    return self

  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]

attr\_adder = CombinedAttributesAdder(add\_bedrooms\_per\_room=False)

df\_extra\_attribs = attr\_adder.transform(df.values)

# Crear el pipeline para datos numéricos

num\_pipeline = Pipeline([

    ('imputer', SimpleImputer(strategy="median")),

    ('attribs\_adder', CombinedAttributesAdder()),

    ('std\_scaler', StandardScaler())

])

# Crear el transformador para todas las columnas

num\_attribs = list(df\_num)

cat\_attribs = ["ocean\_proximity"]

full\_pipeline = ColumnTransformer([

    ("num", num\_pipeline, num\_attribs),

    ("cat", OneHotEncoder(), cat\_attribs),

])

# Volver a un conjunto de entrenamiento limpio

df = strat\_train\_set.drop("median\_house\_value", axis=1)

df\_labels = strat\_train\_set["median\_house\_value"].copy()  # Asegúrate de usar strat\_train\_set aquí

# Preparar el conjunto de entrenamiento

df\_prepared = full\_pipeline.fit\_transform(df)

# Asegurarse que df\_labels sea del mismo tamaño que df\_prepared

print(f"df\_prepared shape: {df\_prepared.shape}")

print(f"df\_labels shape: {df\_labels.shape}")

# Entrenamiento de regresión lineal y evaluación en el conjunto de entrenamiento

lin\_reg = LinearRegression()

lin\_reg.fit(df\_prepared, df\_labels)

# Predicciones

some\_data = df.iloc[:5]

some\_labels = df\_labels.iloc[:5]

some\_data\_prepared = full\_pipeline.transform(some\_data)

print("Predictions:", lin\_reg.predict(some\_data\_prepared))

print("Labels:", list(some\_labels))