Data Cleaning and Preprocessing

his code cleans a dataset by handling outliers and missing values. It identifies numerical outliers using Z-scores and replaces extreme values in the MSSubClass column with its median. Missing values are imputed with the most frequent value (for categorical columns) or the mean (for numerical columns). Finally, the cleaned dataset is saved as cleaned.csv.

import pandas as pd

from scipy import stats

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

#identify missing values

df = pd.read\_csv('train.csv')

# Load your dataset

df = pd.read\_csv('train.csv')

#\*\*\*\*\*\*\*\*\*OUTLIERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Select only the numerical columns

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

# Calculate the Z-scores for each numerical column

z\_scores = np.abs(stats.zscore(numerical\_df))

# Set a threshold for identifying outliers

threshold = 3

# Assuming you want to replace outliers based on z-score

# and replace them with median for the column

upper\_limit = numerical\_df['MSSubClass'].mean() + threshold \* numerical\_df['MSSubClass'].std()

median = numerical\_df['MSSubClass'].median()

df['MSSubClass'] = df['MSSubClass'].apply(lambda x: median if x > upper\_limit else x)

#print(df['MSSubClass'])

# Recalculate the Z-scores after imputation

z\_scores\_post\_imputation = np.abs(stats.zscore(df[['MSSubClass']]))

# Identify outliers after imputation

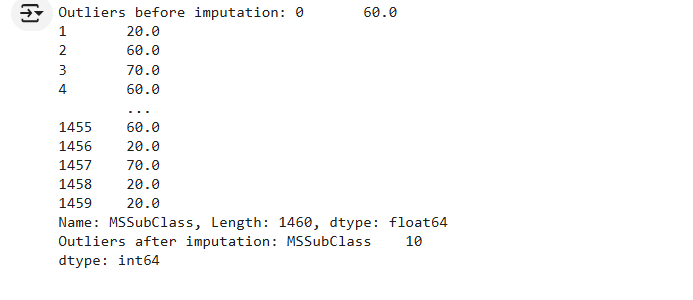
outliers\_after = (z\_scores\_post\_imputation > threshold).sum(axis=0)

# Print out the number of outliers before and after imputation

print(f"Outliers before imputation: {df['MSSubClass']}")

print(f"Outliers after imputation: {outliers\_after}")

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTLIERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MISSING VALUES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

missing\_values = df.isnull().sum()

# Instead of dropping rows with missing values, impute with most frequent value

for column in df.columns:

    if df[column].dtype == 'object':  # Check if column is categorical

        df[column] = df[column].fillna(df[column].mode()[0]) # Fill missing values with mode

    else:

        df[column] = df[column].fillna(df[column].mean()) # Fill missing values with mean

#Save the Cleaned Dataset

df.to\_csv("cleaned.csv")

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MISSING VALUES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Encode Train Data

This code processes a cleaned dataset by identifying non-numerical features and applying one-hot encoding to convert them into numerical format. It separates features and the target variable (SalePrice), encodes the categorical features, combines the encoded data with the target, and saves the resulting dataset to a CSV file named encoded.csv.

from re import X

#Import Necessary Libraries

from sklearn.preprocessing import OneHotEncoder

import pandas as pd

from sklearn.model\_selection import train\_test\_split

#Load cleaned dataset

train\_encode = pd.read\_csv('cleaned.csv')

train\_encode = pd.DataFrame(train\_encode)

#Find non-numerical values in the cleaned dataset

def find\_non\_numerical(train\_encode):

    non\_numerical = []

    for column in df.columns:

        if df[column].dtype == 'object':

            non\_numerical.append(column)

    return non\_numerical

#print(find\_non\_numerical(train\_encode))

#Separate features and Target

X = train\_encode[find\_non\_numerical(train\_encode)]

y = train\_encode['SalePrice']

#Initialize OneHotEncoder

encoder = OneHotEncoder(sparse\_output=False, drop='first')

#Fit and Transform the Features

X\_encoded = encoder.fit\_transform(X)

#Convert to dataframes and column names

encoded\_df = pd.DataFrame(X\_encoded, columns=encoder.get\_feature\_names\_out(X.columns))

df\_encoded = pd.concat([encoded\_df, y], axis=1)

#print(df\_encoded)

#Save the Encoded Dataset

df\_encoded.to\_csv("encoded.csv")

Model Training

This code trains a **Gradient Boosting Regressor** on a dataset to predict house prices. It separates features and the target variable, fits the model, evaluates it using **Mean Squared Error (MSE)** and **R-squared (R²)** metrics, and saves the trained model to a file using joblib. The model is trained on the entire dataset without splitting for testing.

import pandas as pd

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

import joblib

#Load and prepare dataset

data = pd.read\_csv('encoded.csv')

#Specify the name of the target column

target\_column = 'SalePrice'

#Separate features and target

X = data.drop(columns=[target\_column])

y = data[target\_column]

# Initialize the Random Forest model

#rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

#Initialize the model here

rf\_model = GradientBoostingRegressor()

#Fit the model with the entire dataset

rf\_model.fit(X, y)

#Make predictions on the same dataset since we are not splitting

y\_pred = rf\_model.predict(X)

# Evaluate the model using regression metrics

# Calculate Mean Squared Error (MSE)

mse = mean\_squared\_error(y, y\_pred)

# Calculate R-squared (R2)

r2 = r2\_score(y, y\_pred)

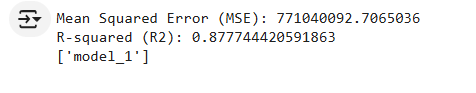
# Print the evaluation metrics

print(f"Mean Squared Error (MSE): {mse}")

print(f"R-squared (R2): {r2}")

# Save the trained model

joblib.dump(rf\_model, 'model\_1')



Clean Test Dataset

Cleaning a dataset ensures the removal of errors, inconsistencies, and irrelevant information, improving the quality and reliability of the data. This step is essential for accurate analysis and better-performing machine learning models.

import pandas as pd

#identify missing values

df = pd.read\_csv('test.csv')

missing\_values = df.isnull().sum()

# Instead of dropping rows with missing values, impute with most frequent value

for column in df.columns:

    if df[column].dtype == 'object':  # Check if column is categorical

        df[column] = df[column].fillna(df[column].mode()[0]) # Fill missing values with mode

    else:

        df[column] = df[column].fillna(df[column].mean()) # Fill missing values with mean

#Save the Cleaned Dataset

df.to\_csv("cleaned\_test.csv")

Encode Test Dataset

Data encoding is the process of converting categorical or text data into numerical formats so that machine learning algorithms can process it effectively. A common technique is **one-hot encoding**, which creates binary columns for each category.

from re import X

#Import Necessary Libraries

from sklearn.preprocessing import OneHotEncoder

import pandas as pd

from sklearn.model\_selection import train\_test\_split

#Load cleaned dataset

test\_encode = pd.read\_csv('cleaned\_test.csv')

test\_encode = pd.DataFrame(test\_encode)

#Find non-numerical values in the cleaned dataset

def find\_non\_numerical(test\_encode):

    non\_numerical = []

    for column in df.columns:

        if df[column].dtype == 'object':

            non\_numerical.append(column)

    return non\_numerical

#print(find\_non\_numerical(train\_encode))

#Separate features and Target

X = test\_encode[find\_non\_numerical(test\_encode)]

y = test\_encode['Id']

#Initialize OneHotEncoder

encoder = OneHotEncoder(sparse\_output=False, drop='first')

#Fit and Transform the Features

X\_encoded = encoder.fit\_transform(X)

#Convert to dataframes and column names

encoded\_df = pd.DataFrame(X\_encoded, columns=encoder.get\_feature\_names\_out(X.columns))

df\_encoded = pd.concat([encoded\_df, y], axis=1)

#print(df\_encoded)

#Save the Encoded Dataset

df\_encoded.to\_csv("encoded\_test.csv")

Predict Housing Prices

This code uses a pre-trained machine learning model to predict house prices. It loads the model and test data, aligns the test data's features with the training data, makes predictions, and stores the results in a DataFrame with corresponding IDs. Finally, it saves the predictions to a CSV file.

import pandas as pd

import joblib

# Load the model

model = joblib.load('model\_1')

# Load the training data to get the original feature names

predict\_data = pd.read\_csv('encoded.csv')  # Replace with the path to your encoded training data

predict\_features = predict\_data.drop(columns=['SalePrice']).columns  # Get the feature names from the training data

# Load the test data

test\_data = pd.read\_csv('encoded\_test.csv')

# Specify the name of the target column (if it exists in test data, remove it)

target\_column = 'Id'  # Replace with your actual target column name, if needed

# \*\*Store 'Id' before reindexing\*\*

index\_ids = test\_data['Id'].tolist()

# Remove the target column if it exists in test data

if target\_column in test\_data.columns:

    test\_data = test\_data.drop(columns=[target\_column])

# Reindex the test data to match the training data columns

test\_data = test\_data.reindex(columns=predict\_features, fill\_value=0) # Fill missing values with 0

# Make predictions for each row in the test data

predictions = model.predict(test\_data)

# Create a DataFrame with Item\_Identifier and Item\_Outlet\_Sales columns

results = pd.DataFrame({

    'Id': index\_ids,

    'SalePrice': predictions

})

# Display the results

print(results)

# Optionally, save the results to a CSV file

results.to\_csv('HousingPricePrediction2.csv', index=False)

