ML Lab

Assignment-1

Q1. Load a CSV dataset and display the total number of missing values in each column.

import pandas as pd

df = pd.read_csv("AmesHousing.csv")

print(df.isnull().sum())

```
Order              0
PID                0
MS SubClass        0
MS Zoning          0
Lot Frontage     490
                ...
Mo Sold            0
Yr Sold            0
Sale Type          0
Sale Condition     0
SalePrice          0
Length: 82, dtype: int64
```

Q2. Replace missing numerical values with:mean,median,mode Compare results.

```
num_col = 'Lot Frontage'

df_mean = df.copy()
df_median = df.copy()
df_mode = df.copy()

df_mean[num_col].fillna(df_mean[num_col].mean(), inplace=True)
df_median[num_col].fillna(df_median[num_col].median(), inplace=True)
df_mode[num_col].fillna(df_mode[num_col].mode()[0], inplace=True)

print("Mean value:", df_mean[num_col].mean())
print("Median value:", df_median[num_col].median())
print("Mode value:", df_mode[num_col].mode()[0])
```

```
Mean value: 69.22459016393442
Median value: 68.0
Mode value: 60.0
```

Q3. Replace missing categorical values with the most frequent category.

```
cat_col = 'Alley'
df[cat_col].fillna(df[cat_col].mode()[0], inplace=True)
```

Q4. Drop rows where more than 50% of columns contain missing values.

```
threshold = len(df.columns) * 0.5
df_dropped = df.dropna(thresh=threshold)
print(df_dropped.shape)
```

```
(2930, 82)
```

Q5. Create a function that reports:% missing per column, total missing rows

```
def missing_report(data):
    percent_missing = (data.isnull().sum() / len(data)) * 100
    total_missing_rows = data.isnull().any(axis=1).sum()

    print(percent_missing.sort_values(ascending=False))
    print("Total rows with missing values:", total_missing_rows)

missing_report(df)
```

```
Pool QC           99.556314
Misc Feature      96.382253
Fence             80.477816
Mas Vnr Type      60.580205
Fireplace Qu      48.532423
                    ...
Mo Sold            0.000000
Yr Sold            0.000000
Sale Type          0.000000
Sale Condition     0.000000
SalePrice          0.000000
Length: 82, dtype: float64
Total rows with missing values: 2929
```

Q6. Identify all categorical columns in a dataset automatically.

```
categorical_cols = df.select_dtypes(include=['object']).columns
print(list(categorical_cols))
```

```
['MS Zoning', 'Street', 'Alley', 'Lot Shape', 'Land Contour', 'Utilities', 'Lot Config',
```

Q7. Apply Label Encoding to a single categorical column.

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['Neighborhood_encoded'] = le.fit_transform(df['Neighborhood'])
```

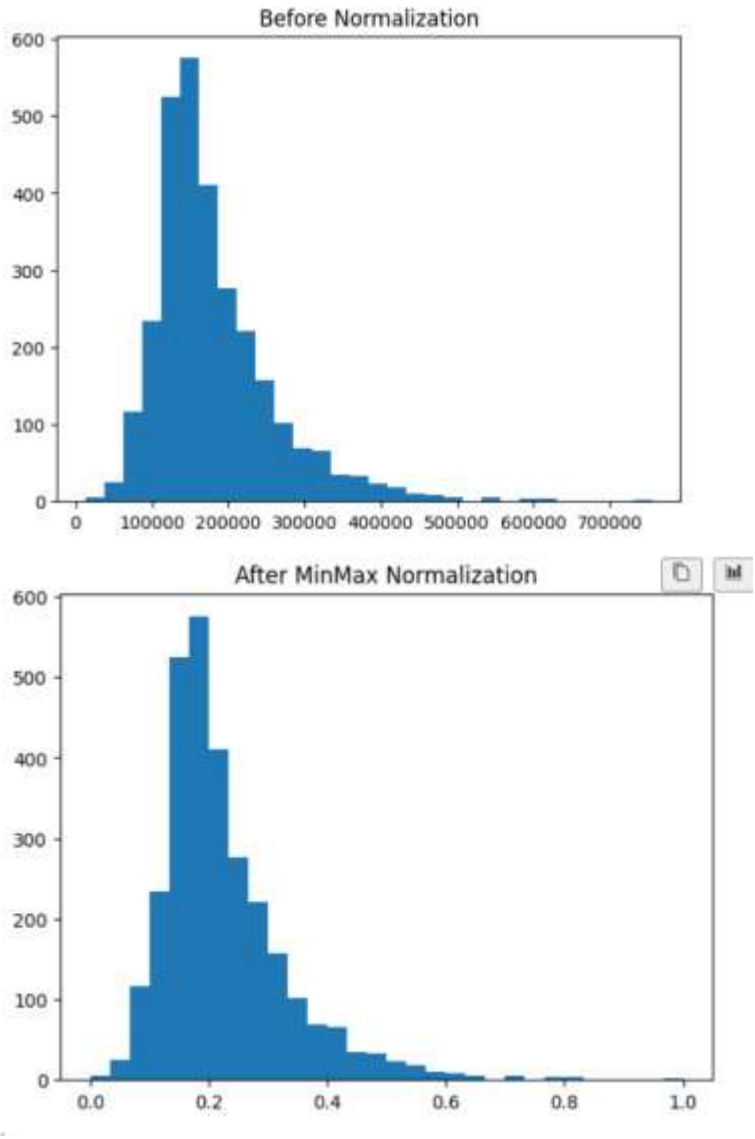
	Neighborhood	Neighborhood_encoded
0	NAmes	15
1	NAmes	15
2	NAmes	15
3	NAmes	15
4	Gilbert	8

Q8. Write a reusable function for Min-Max normalization and z score standardization

```
def scale_column(series, method='minmax'):
    if method == 'minmax':
        return (series - series.min()) / (series.max() - series.min())
    elif method == 'zscore':
        return (series - series.mean()) / series.std()
```

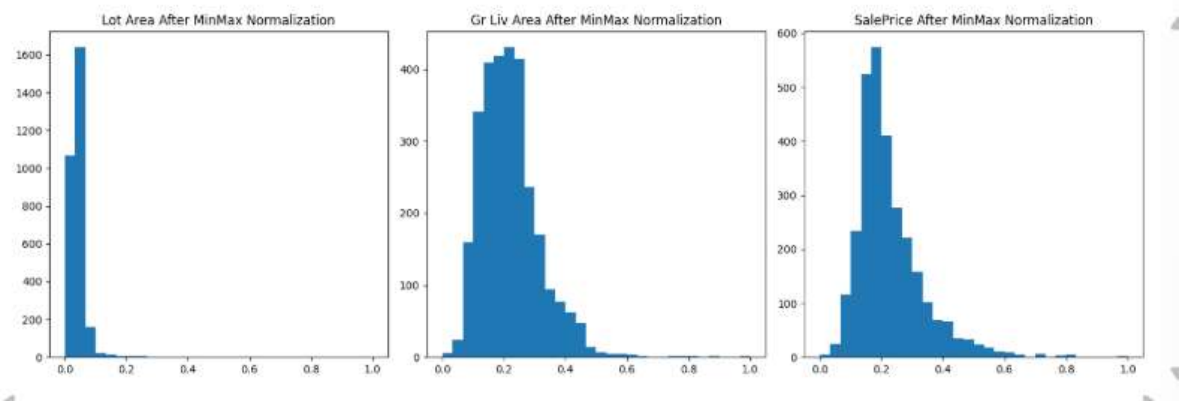
Q9. Plot before vs after normalization distributions.

```
import matplotlib.pyplot as plt

df['SalePrice_minmax'] = scale_column(df['SalePrice'], 'minmax')

plt.figure()
plt.hist(df['SalePrice'], bins=30)
plt.title("Before Normalization")
plt.show()

plt.figure()
plt.hist(df['SalePrice_minmax'], bins=30)
plt.title("After MinMax Normalization")
plt.show()
```



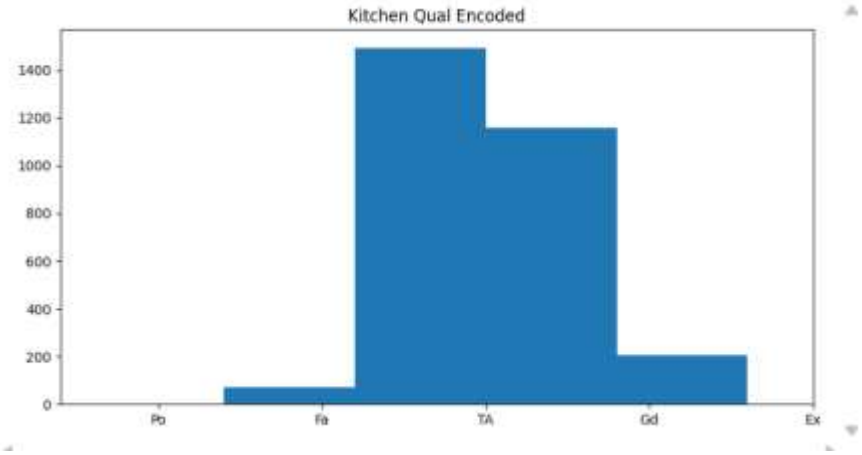
Q10. Normalize only selected columns while keeping others unchanged.

```
cols = ['Lot Area', 'Gr Liv Area', 'SalePrice']
for col in cols:

    df[col + '_norm'] = scale_column(df[col], 'minmax')
```



Q11. Implement Ordinal Encoding for ordered categories (e.g., low–medium– high).

```
order = {'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5}
df['Kitchen Qual_encoded'] = df['Kitchen Qual'].map(order)
```



Q12. Add a binary indicator column showing whether a value was originally missing.

```
df['Lot Frontage_missing'] = df['Lot Frontage'].isnull().astype(int)
```

Q13. Design an automated preprocessing workflow that adapts based on column data type.

```
def auto_preprocess(data):
    data = data.copy()

    for col in data.columns:
        if data[col].dtype == 'object':
            data[col].fillna(data[col].mode()[0], inplace=True)
        else:
            data[col].fillna(data[col].mean(), inplace=True)
            data[col] = scale_column(data[col], 'minmax')
```

```
    return data

df_processed = auto_preprocess(df)
```

```
df_processed
✓ 0.0s
```

	Order	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	...	Sale Type	Sale Condition	SalePrice	Neighborhood_encoded	SalePrice_min
0	0.000000	0.000000	0.000000	RL	0.410959	0.142420	Pave	Grvl	IR1	Lvl	...	WD	Normal	0.272444	0.555556	0.27
1	0.000341	0.000102	0.000000	RH	0.202055	0.048246	Pave	Grvl	Reg	Lvl	...	WD	Normal	0.124238	0.555556	0.12
2	0.000683	0.000104	0.000000	RL	0.205479	0.060609	Pave	Grvl	IR1	Lvl	...	WD	Normal	0.214509	0.555556	0.21
3	0.001024	0.000108	0.000000	RL	0.246575	0.046087	Pave	Grvl	Reg	Lvl	...	WD	Normal	0.311517	0.555556	0.31
4	0.001366	0.001672	0.235294	RL	0.181507	0.058566	Pave	Grvl	IR1	Lvl	...	WD	Normal	0.238626	0.296296	0.23
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2925	0.998634	0.825655	0.352941	RL	0.054795	0.031022	Pave	Grvl	IR1	Lvl	...	WD	Normal	0.174763	0.518519	0.17
2926	0.998976	0.825657	0.000000	RL	0.165153	0.035453	Pave	Grvl	IR1	Low	...	WD	Normal	0.159269	0.518519	0.15
2927	0.999317	0.825915	0.382353	RL	0.140411	0.042726	Pave	Grvl	Reg	Lvl	...	WD	Normal	0.160616	0.518519	0.16
2928	0.999659	0.827371	0.000000	RL	0.191781	0.040711	Pave	Grvl	Reg	Lvl	...	WD	Normal	0.211814	0.518519	0.21
2929	1.000000	0.827477	0.235294	RL	0.181507	0.038921	Pave	Grvl	Reg	Lvl	...	WD	Normal	0.236066	0.518519	0.23

2930 rows × 89 columns