RP Tumba College

Module Name: Machine Learning

Level 8 Learning Outcome 1 REGNo:25RP18677

Data set imported and well libraries imported

Import Libraries and data set

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")

file_path = "Housing.xls"
df = pd.read_csv(file_path)
df_original = df.copy()
df
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking
0	13300000	7420.0	4	2.0	3	yes	no	no	no	yes	2
1	12250000	8960.0	4	4.0	4	yes	no	no	no	yes	3
2	12250000	9960.0	3	2.0	2	yes	no	yes	no	no	2
3	12215000	NaN	4	2.0	2	yes	no	yes	no	yes	3
4	11410000	NaN	4	1.0	2	yes	yes	yes	no	yes	2
...	...	...	...	...	...	...	...	...	...	...	...
540	1820000	3000.0	2	1.0	1	yes	no	yes	no	no	2
541	1767150	2400.0	3	1.0	1	no	no	no	no	no	0
542	1750000	3620.0	2	1.0	1	yes	no	no	no	no	0
543	1750000	2910.0	3	1.0	1	no	no	no	no	no	0
544	1750000	3850.0	3	1.0	2	yes	no	no	no	no	0

Q1. Statistical Summary of Numerical Variables

Step 1 – Statistical Summary of Numerical Variables

```
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
df[numeric_cols].describe(percentiles=[0.01, .05, .10, .25, .5, .75, .90, .95, .99])

summary_stats = df[numeric_cols].describe().T
summary_stats["median"] = df[numeric_cols].median()
summary_stats["skew"] = df[numeric_cols].skew()
summary_stats["kurtosis"] = df[numeric_cols].kurtosis()
summary_stats
```

Explain what the summary reveals about the distribution and characteristics of each variable.

	count	mean	std	min	25%	50%	75%	max	median	skew	kurtosis
price	545.0	4.766729e+06	1.870440e+06	1750000.0	3430000.0	4340000.0	5740000.0	13300000.0	4340000.0	1.212239	1.960130
area	542.0	5.127168e+03	2.143733e+03	1650.0	3588.0	4540.0	6360.0	16200.0	4540.0	1.308216	2.755046
bedrooms	545.0	3.691743e+00	1.702314e+01	1.0	2.0	3.0	3.0	400.0	3.0	23.279477	542.949099
bathrooms	544.0	1.284926e+00	5.019967e-01	1.0	1.0	1.0	2.0	4.0	1.0	1.599262	2.203253
stories	545.0	1.805505e+00	8.674925e-01	1.0	1.0	2.0	2.0	4.0	2.0	1.082088	0.679404
parking	545.0	6.935780e-01	8.615858e-01	0.0	0.0	0.0	1.0	3.0	0.0	0.842062	-0.573063

Q2. Detects missing values across the dataset.

Step 2 – Handling Missing Values

```python
df2 = df.copy()

for col in df2.columns:
    if df2[col].isnull().sum() == 0:
        continue
    if df2[col].dtype in ['int64','float64']:
        if abs(df2[col].skew()) > 1:
            df2[col] = df2[col].fillna(df2[col].median())
        else:
            df2[col] = df2[col].fillna(df2[col].mean())
    else:
        df2[col] = df2[col].fillna(df2[col].mode()[0])

df2.isnull().sum()
```

```
price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
prefarea            0
furnishingstatus    0
dtype: int64
```

C. • **Numerical variables:**

- **Mean imputation:** Used when data is approximately symmetric (low skew).
- **Median imputation:** Used for skewed variables to avoid bias from extreme values.

• **Categorical variables:**

- **Mode imputation:** Replaces missing with the most frequent category.

• **Justification:** Choosing mean/median ensures the central tendency of data remains reasonable without distorting distributions. Mode maintains the most common categorical information.

Q3. Detecting and Handling Duplicate Records

a. Check for duplicate observations in the dataset.

```python
# Show duplicate rows
duplicates = df2[df2.duplicated(keep=False)]  # keep=False marks all duplicates as True
print(duplicates)
# Rows that were removed
removed = df2[~df2.index.isin(df3.index)]
print(removed)
```

Step 3 – Detecting and Handling Duplicate Records

```python
df3 = df2.drop_duplicates().reset_index(drop=True)
df3.shape
```

✓ 0.0s

b. decides whether to remove or retain duplicates.

```
df3.shape
26]  ✓ 0.0s
(545, 13)
```

c. Explain and justify your decision.

Why: Duplicates may cause bias, overfitting, or distort statistics.

• **Action:** Duplicates removed using. `..drop_duplicates().`

• **Justification:** No two observations should be counted twice in a supervised model; this ensures unique entries.

Q4. Detecting and Handling Data Inconsistency

a. Identify any inconsistencies (e.g., incorrect data types, spelling variations in categorical values, unrealistic values, mixed units, format inconsistencies).

Step 4 – Detecting and Handling Data Inconsistency

```
df4 = df3.copy()

for col in df4.select_dtypes(include="object"):
    df4[col] = df4[col].astype(str).str.strip()
    df4[col] = df4[col].replace({"N/A":np.nan, "NA":np.nan, "None":np.nan})

for col in df4.select_dtypes(include="object"):
    sample = df4[col].dropna().astype(str).head(50)
    num_like = sample.str.replace(',','').str.replace('$','').str.replace('.','').str.isdigit()
    if num_like.mean() > 0.6:
        df4[col] = df4[col].str.replace(',','').str.replace('$','')
        df4[col] = pd.to_numeric(df4[col], errors="coerce")

df4.dtypes
```

b. Clean, correct, or unify the inconsistent data

```
price                 int64
area                float64
bedrooms              int64
bathrooms           float64
stories               int64
mainroad             object
guestroom            object
basement             object
hotwaterheating      object
airconditioning      object
parking               int64
prefarea             object
furnishingstatus     object
dtype: object
```

Strip spaces & unify missing values: Ensures categories like `'NA'`, `'None'`, `'N/A'` are standardized.

• **Convert numeric-like strings:** Converts columns stored as text into numbers (e.g., `$120,000` → `120000`).

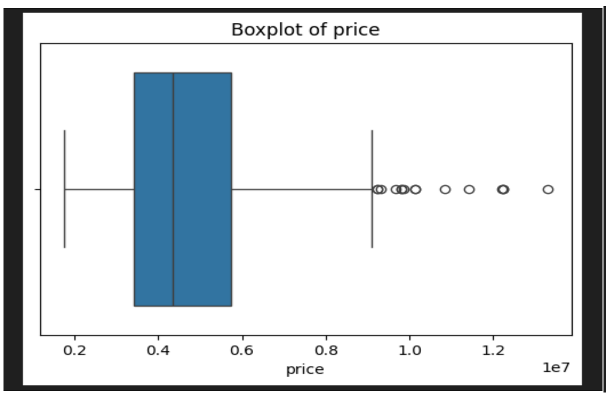
• **Justification:** Consistent data types and standardized missing values are required for analysis and modeling.

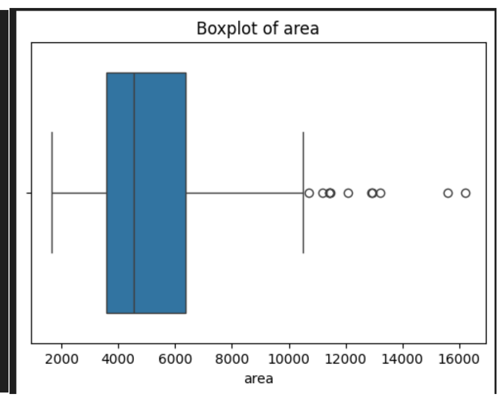
Q5. Detecting and Handling Outliers

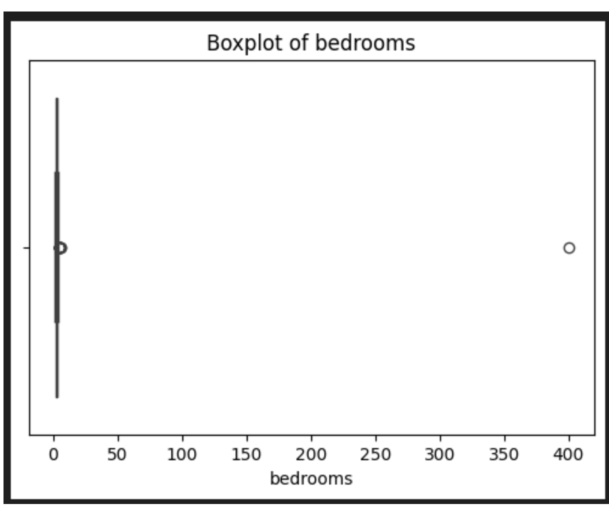
a. Use appropriate outlier detection methods (IQR, Z-Score, visualization techniques, or domain rules).

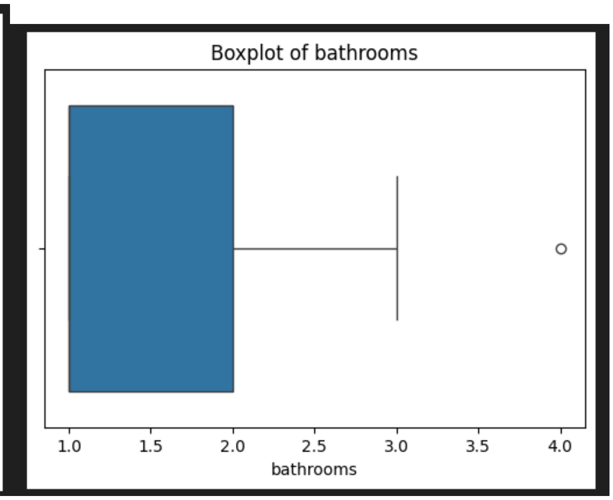
Step 5 – Detecting and Handling Outliers

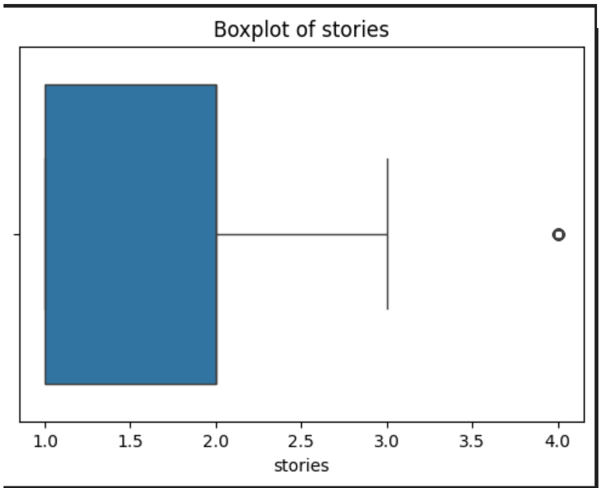
```python
import seaborn as sns
import matplotlib.pyplot as plt

numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns

for col in numerical_cols:
    plt.figure(figsize=(6,4))
    sns.boxplot(data=df, x=col)
    plt.title(f"Boxplot of {col}")
    plt.show()
```

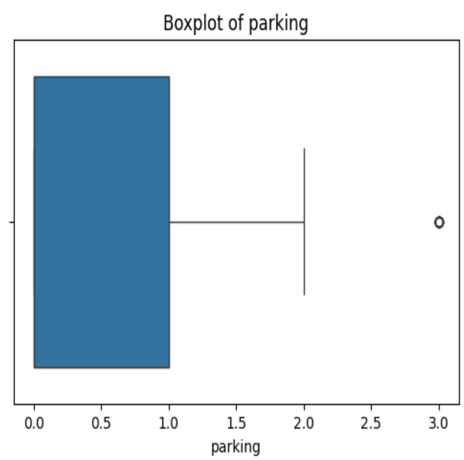












• **Method:** Clipped values at 1st and 99th percentiles.

• **Justification:** Extreme outliers can distort mean, variance, and model predictions. Clipping reduces their impact while retaining most data.

Q6. Normalization and Scaling

Step 6 – Normalization and Scaling

```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler

df6 = df5.copy()
numeric_cols = df6.select_dtypes(include=[np.number]).columns

scalers = {
    "std": StandardScaler(),
    "minmax": MinMaxScaler(),
    "robust": RobustScaler()
}

for name, scaler in scalers.items():
    scaled = scaler.fit_transform(df6[numeric_cols])
    scaled_df = pd.DataFrame(scaled, columns=[f"{c}_{name}" for c in numeric_cols])
    df6 = pd.concat([df6, scaled_df], axis=1)

df6.head()
```

b. Apply appropriate techniques such as Min-Max Scaling, Standardization (Z-score scaling), Robust Scaling

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	...	bedrooms_minmax	bathrooms_minmax
0	10542000.0	7420.0	4	2.0	3	yes	no	no	no	yes	...	0.666667	0.5
1	10542000.0	8960.0	4	3.0	4	yes	no	no	no	yes	...	0.666667	1.0
2	10542000.0	9960.0	3	2.0	2	yes	no	yes	no	no	...	0.333333	0.5
3	10542000.0	4540.0	4	2.0	2	yes	no	yes	no	yes	...	0.666667	0.5
4	10542000.0	4540.0	4	1.0	2	yes	yes	yes	no	yes	...	0.666667	0.0

5 rows × 31 columns

bedrooms_minmax	bathrooms_minmax	stories_minmax	parking_minmax	price_robust	area_robust	bedrooms_robust	bathrooms_robust	stories_robust	parking_robust
0.666667	0.5	0.666667	0.666667	2.684848	1.043478	1.0	1.0	1.0	2.0
0.666667	1.0	1.000000	1.000000	2.684848	1.601449	1.0	2.0	2.0	3.0
0.333333	0.5	0.333333	0.666667	2.684848	1.963768	0.0	1.0	0.0	2.0
0.666667	0.5	0.333333	1.000000	2.684848	0.000000	1.0	1.0	0.0	3.0
0.666667	0.0	0.333333	0.666667	2.684848	0.000000	1.0	0.0	0.0	2.0

c. **Techniques:** StandardScaler, MinMaxScaler, RobustScaler.

• **Why:**

- **StandardScaler:** Centers data at 0 with unit variance (good for symmetric distributions).
- **MinMaxScaler:** Scales to 0–1 (useful for bounded features or neural networks).
- **RobustScaler:** Handles skewed data and reduces influence of outliers.

• Justification**:** Ensures numerical features are comparable and improves convergence of ML models.

Q7. Encoding Categorical Variables
Research, document them theoretically and apply different data encoding techniques to relevant categorical variables in the dataset, including but not limited to

Label Encoding

```
cat_cols = df6.select_dtypes(include="object").columns.tolist()
df7 = df6.copy()

# Label encoding
from sklearn.preprocessing import LabelEncoder
for col in cat_cols:
    if df7[col].nunique() == 2:
        df7[col+"_label"] = LabelEncoder().fit_transform(df7[col])
```

One hot encoding

```
# One-hot
for col in cat_cols:
    if 2 < df7[col].nunique() <= 10:
        df7 = pd.concat([df7, pd.get_dummies(df7[col], prefix=col)], axis=1)
```

Binary encoding and ordinal encoding

```
# Binary encoding
def binary_hash_encoder(series, n_bits=8):
    hashed = series.fillna("NA").apply(lambda x: abs(hash(x)) % (2**n_bits))
    bin_df = hashed.apply(lambda x: list(map(int,list(f"{x:0{n_bits}b}"))))
    return pd.DataFrame(bin_df.tolist(),
                        columns=[f"{series.name}_bit_{i}" for i in range(n_bits)])

for col in cat_cols:
    if 10 < df7[col].nunique() <= 50:
        df7 = pd.concat([df7, binary_hash_encoder(df7[col])], axis=1)

# Ordinal encoding
ordinal_columns = [c for c in cat_cols if any(x in c.lower()
                                              for x in ['level','rating','quality','grade'])]
for col in ordinal_columns:
    mapping = {cat: i for i, cat in enumerate(sorted(df7[col].dropna().unique()))}
    df7[col+"_ord"] = df7[col].map(mapping)
```

```python
# Target encoding
target = "price"

def target_encode(df, column, target, smoothing=10):
    agg = df.groupby(column)[target].agg(['mean','count'])
    global_mean = df[target].mean()
    agg["smooth"] = (agg["count"]*agg["mean"] + smoothing*global_mean) / (agg["count"] + smoothing)
    return df[column].map(agg["smooth"])

for col in cat_cols:
    if df7[col].nunique() > 1:
        df7[col + "_target"] = target_encode(df7, col, target)

df7.head()
```

Target Encoding (with and without smoothing)

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	...	furnishingstatus_furnished
0	10542000.0	7420.0	4	2.0	3	yes	no	no	no	yes	...	True
1	10542000.0	8960.0	4	3.0	4	yes	no	no	no	yes	...	True
2	10542000.0	9960.0	3	2.0	2	yes	no	yes	no	no	...	False
3	10542000.0	4540.0	4	2.0	2	yes	no	yes	no	yes	...	True
4	10542000.0	4540.0	4	1.0	2	yes	yes	yes	no	yes	...	True

5 rows × 47 columns

furnishingstatus_unfurnished	mainroad_target	guestroom_target	basement_target	hotwaterheating_target	airconditioning_target	prefarea_target	furnishingstatus_target
False	4.968476e+06	4.532409e+06	4.505641e+06	4.712995e+06	5.903684e+06	5.750596e+06	5.400149e+06
False	4.968476e+06	4.532409e+06	4.505641e+06	4.712995e+06	5.903684e+06	4.428139e+06	5.400149e+06
False	4.968476e+06	4.532409e+06	5.195743e+06	4.712995e+06	4.203465e+06	5.750596e+06	4.892855e+06
False	4.968476e+06	4.532409e+06	5.195743e+06	4.712995e+06	5.903684e+06	5.750596e+06	5.400149e+06
False	4.968476e+06	5.687425e+06	5.195743e+06	4.712995e+06	5.903684e+06	4.428139e+06	5.400149e+06