In [4]:

```python
import pandas as pd
df = pd.read_excel("C:/Users/rohit/Downloads/transactions-2025-07-23.csv.xlsx")
df.head()
```

Out[4]:

	TRANSACTION_NUMBER	INSTANCE_DATE	GROUP_EN	PROCEDURE_EN	IS_OFFPLAN_EN
0	102-1-2025	2025-01-07 15:57:40	Sales	Sell - Pre registration	Off-Plan
1	102-10-2025	2025-01-02 08:23:00	Sales	Sell - Pre registration	Off-Plan
2	102-100-2025	2025-01-02 15:18:51	Sales	Sell - Pre registration	Off-Plan
3	102-1000-2025	2025-01-08 11:02:57	Sales	Sell - Pre registration	Off-Plan
4	102-10000-2025	2025-02-07 11:15:01	Sales	Sell - Pre registration	Off-Plan

5 rows × 22 columns

In [5]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110490 entries, 0 to 110489
Data columns (total 22 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   TRANSACTION_NUMBER   110490 non-null  object
 1   INSTANCE_DATE        110490 non-null  datetime64[ns]
 2   GROUP_EN             110490 non-null  object
 3   PROCEDURE_EN         110490 non-null  object
 4   IS_OFFPLAN_EN        110490 non-null  object
 5   IS_FREE_HOLD_EN      110490 non-null  object
 6   USAGE_EN             110490 non-null  object
 7   AREA_EN              110490 non-null  object
 8   PROP_TYPE_EN         110490 non-null  object
 9   PROP_SB_TYPE_EN      104394 non-null  object
 10  TRANS_VALUE          110490 non-null  float64
 11  PROCEDURE_AREA       110489 non-null  float64
 12  ACTUAL_AREA          110490 non-null  float64
 13  ROOMS_EN             94277 non-null   object
 14  PARKING              85646 non-null   object
 15  NEAREST_METRO_EN     56461 non-null   object
 16  NEAREST_MALL_EN      55007 non-null   object
 17  NEAREST_LANDMARK_EN  68028 non-null   object
 18  TOTAL_BUYER          110490 non-null  int64
 19  TOTAL_SELLER         110490 non-null  int64
 20  MASTER_PROJECT_EN    277 non-null     object
 21  PROJECT_EN           100900 non-null  object
dtypes: datetime64[ns](1), float64(3), int64(2), object(16)
memory usage: 18.5+ MB
```

In [6]:

```
#Columns with more than 30% null values
drop_cols = ['MASTER_PROJECT_EN', 'NEAREST_METRO_EN', 'NEAREST_MALL_EN', 'NEAREST_L
df_cleaned = df.drop(columns=drop_cols)
```

In [7]:

```
#Null audit - find percentage of missing values
null_summary = df.isnull().mean().sort_values(ascending=False)
print(null_summary[null_summary > 0])
```

```
MASTER_PROJECT_EN      0.997493
NEAREST_MALL_EN        0.502154
NEAREST_METRO_EN       0.488994
NEAREST_LANDMARK_EN    0.384306
PARKING                0.224853
ROOMS_EN               0.146737
PROJECT_EN             0.086795
PROP_SB_TYPE_EN        0.055172
PROCEDURE_AREA         0.000009
dtype: float64
```

In [8]:

```
#impute missing values for numerical columns
median_val = df_cleaned['PROCEDURE_AREA'].median()
df_cleaned.loc[:, 'PROCEDURE_AREA'] = df_cleaned['PROCEDURE_AREA'].fillna(median_va
```

In [9]:

```
#impute missing values for categorical columns
categorical_impute = ['PARKING', 'ROOMS_EN', 'PROJECT_EN', 'PROP_SB_TYPE_EN']
for col in categorical_impute:
    mode_val = df_cleaned[col].mode()[0]
    df_cleaned.loc[:, col] = df_cleaned[col].fillna(mode_val)
```

In [10]:

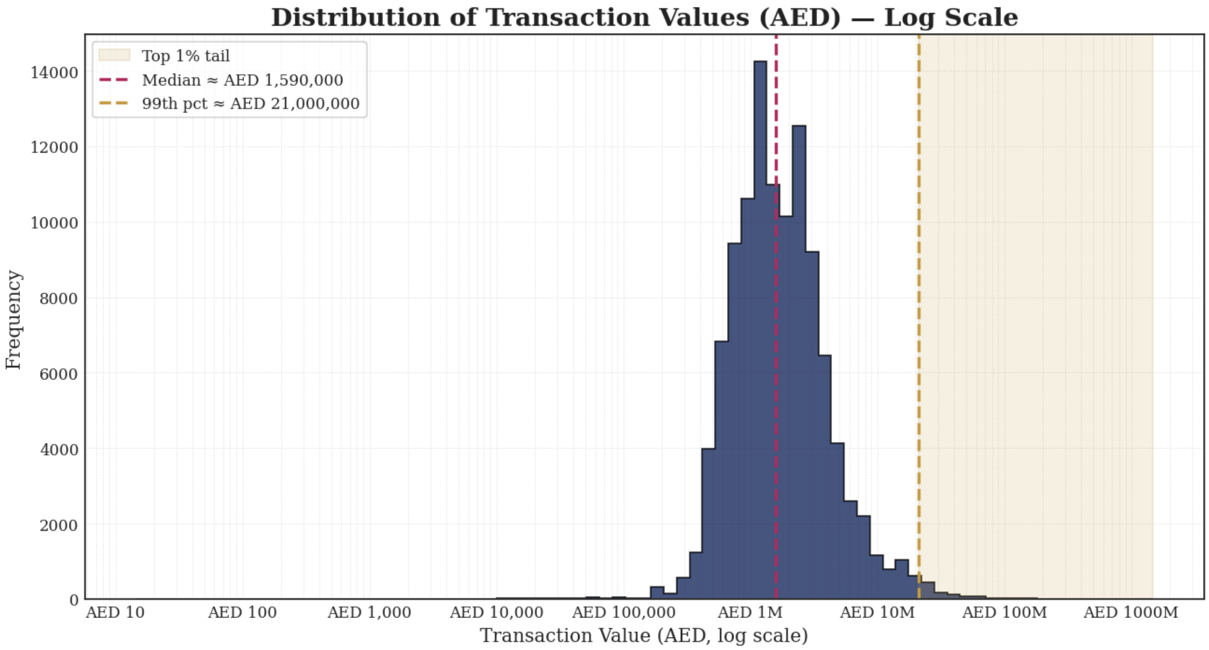
```
#check missing values
print(df_cleaned.isnull().sum().sort_values(ascending=False).head())
```

```
TRANSACTION_NUMBER    0
INSTANCE_DATE         0
GROUP_EN              0
PROCEDURE_EN          0
IS_OFFPLAN_EN         0
dtype: int64
```

In [11]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter, LogLocator

# ---------------------------
# 1) Data prep & key stats
# ---------------------------
s = df_cleaned['TRANS_VALUE'].dropna()
s = s[s > 0]  # Ensure compatibility with log scale

median_val = s.median()
p99_val    = s.quantile(0.99)

bins = np.logspace(np.log10(s.min()), np.log10(s.max()), 80)

# ---------------------------
# 2) Ivy-League Styling
# ---------------------------
plt.style.use('seaborn-v0_8-white')
plt.rcParams.update({
    "font.family": "serif",
    "axes.labelsize": 12,
    "axes.titlesize": 15,
    "figure.dpi": 120,
    "axes.edgecolor": "#2F2F2F",
    "axes.linewidth": 1.1
})

def aed_millions(x, pos):
    return f"AED {x/1e6:.0f}M" if x >= 1e6 else f"AED {x:,.0f}"

# ---------------------------
# 3) Plot
# ---------------------------
fig, ax = plt.subplots(figsize=(11, 6))

# Main histogram with a muted navy tone
ax.hist(s, bins=bins, histtype='stepfilled',
        color='#1A2A5E', alpha=0.8, edgecolor='black')

# Highlight top 1% tail in soft gold
ax.axvspan(p99_val, s.max(), color='#C49E47', alpha=0.15, label='Top 1% tail')
```

```
# Vertical reference lines
ax.axvline(median_val, color='#B03060', linestyle='--', linewidth=2,
           label=f"Median ≈ AED {median_val:,.0f}")
ax.axvline(p99_val, color='#C49E47', linestyle='--', linewidth=2,
           label=f"99th pct ≈ AED {p99_val:,.0f}")

# Log x-axis with AED formatting
ax.set_xscale('log')
ax.xaxis.set_major_locator(LogLocator(base=10.0, numticks=10))
ax.xaxis.set_major_formatter(FuncFormatter(aed_millions))

# Titles & labels
ax.set_title("Distribution of Transaction Values (AED) — Log Scale", fontsize=16, f
ax.set_xlabel("Transaction Value (AED, log scale)")
ax.set_ylabel("Frequency")

# Legend & grid
ax.legend(frameon=True, fontsize=10)
ax.grid(which="both", linestyle=":", linewidth=0.6, alpha=0.5)

plt.tight_layout()
plt.show()

print(f"Median Transaction Value: AED {median_val:,.0f}")
print(f"99th Percentile Transaction Value: AED {p99_val:,.0f}")
```

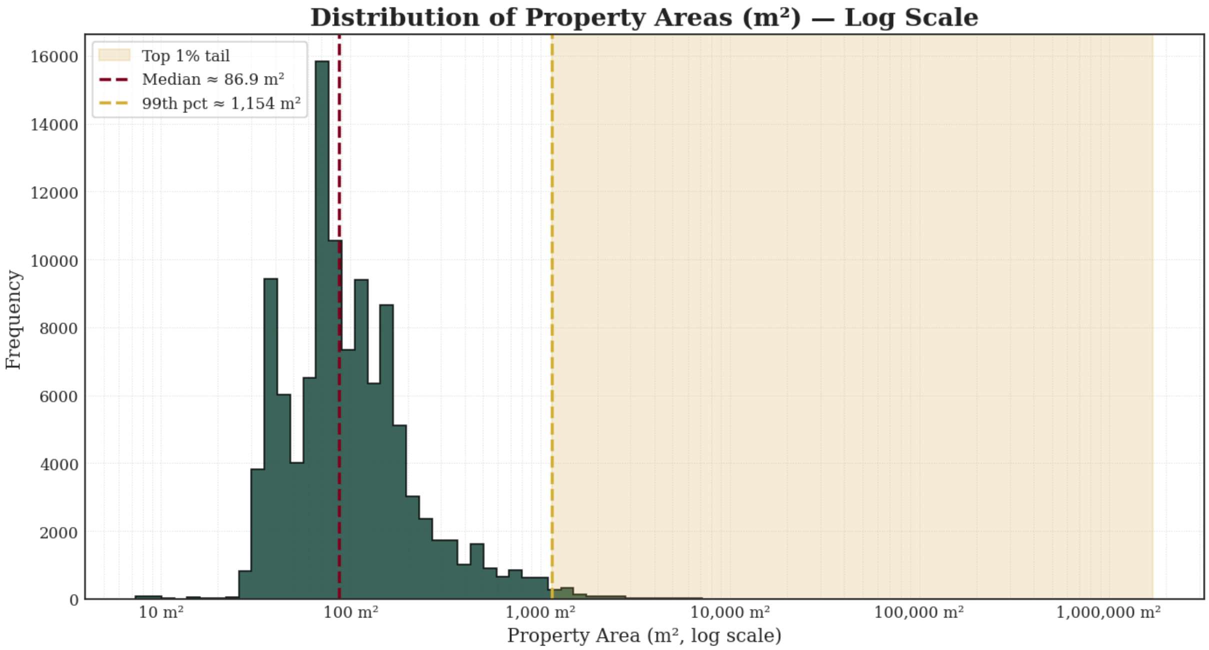


```
Median Transaction Value: AED 1,590,000
99th Percentile Transaction Value: AED 21,000,000
```

In [12]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter, LogLocator

# ---------------------------
# 1) Data Preparation
# ---------------------------
area = df_cleaned['ACTUAL_AREA'].dropna()
```

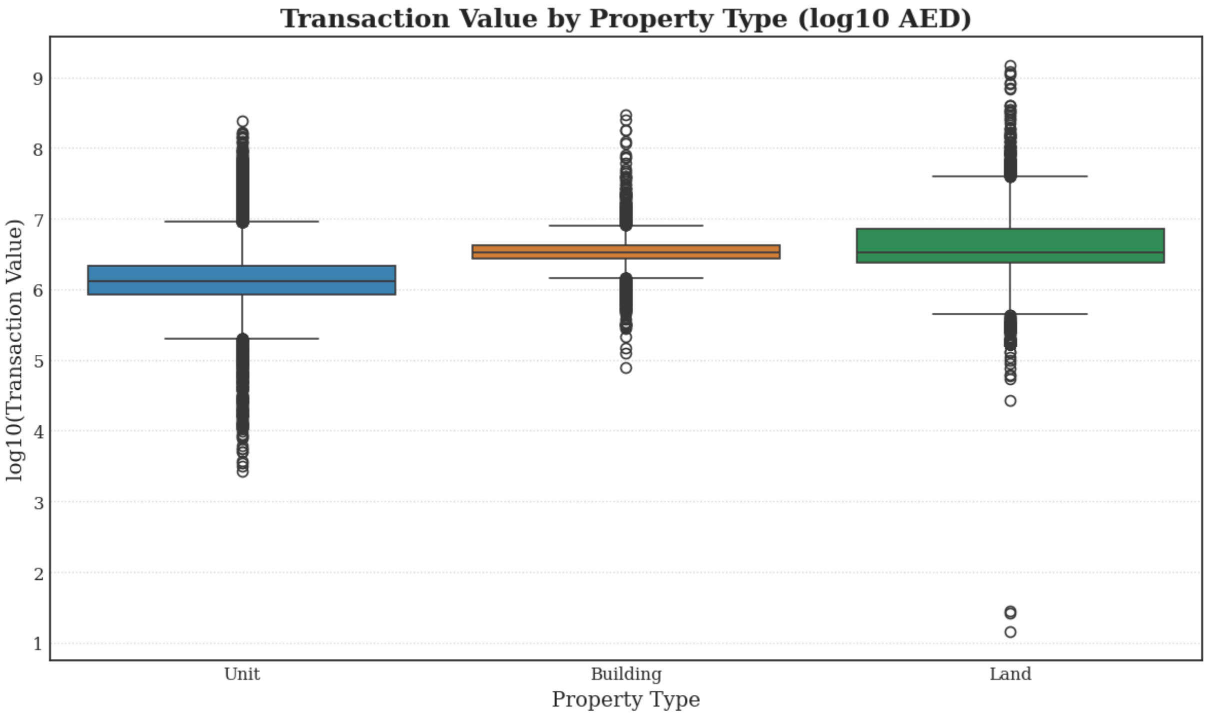
```python
area = area[area > 0]  # Log scale can't handle zero or negatives

median_area = area.median()
p99_area    = area.quantile(0.99)

bins = np.logspace(np.log10(area.min()), np.log10(area.max()), 80)

# ---------------------------
# 2) Professional Styling
# ---------------------------
plt.style.use('seaborn-v0_8-white')
plt.rcParams.update({
    "font.family": "serif",
    "axes.labelsize": 12,
    "axes.titlesize": 15,
    "figure.dpi": 120,
    "axes.edgecolor": "#2F2F2F",
    "axes.linewidth": 1.1
})

def sqm_formatter(x, pos):
    return f"{x:,.0f} m²"

# ---------------------------
# 3) Plot
# ---------------------------
fig, ax = plt.subplots(figsize=(11, 6))

# Main histogram — deep teal
ax.hist(area, bins=bins, histtype='stepfilled',
        color='#1B4D3E', alpha=0.85, edgecolor='black')

# Highlight top 1% tail — champagne gold
ax.axvspan(p99_area, area.max(), color='#D4AF37', alpha=0.2, label='Top 1% tail')

# Reference lines
ax.axvline(median_area, color='#800020', linestyle='--', linewidth=2,
           label=f"Median ≈ {median_area:,.1f} m²")
ax.axvline(p99_area, color='#D4AF37', linestyle='--', linewidth=2,
           label=f"99th pct ≈ {p99_area:,.0f} m²")

# Log x-axis
ax.set_xscale('log')
ax.xaxis.set_major_locator(LogLocator(base=10.0, numticks=10))
ax.xaxis.set_major_formatter(FuncFormatter(sqm_formatter))

# Titles and labels
ax.set_title("Distribution of Property Areas (m²) — Log Scale", fontsize=16, fontwe
ax.set_xlabel("Property Area (m², log scale)")
ax.set_ylabel("Frequency")

# Legend & grid
ax.legend(frameon=True, fontsize=10)
ax.grid(which="both", linestyle=":", linewidth=0.6, alpha=0.5)

plt.tight_layout()
plt.show()

# Key Stats
print(f"Median Property Area: {median_area:,.1f} m²")
print(f"99th Percentile Property Area: {p99_area:,.0f} m²")
```



```
Median Property Area: 86.9 m²
99th Percentile Property Area: 1,154 m²
```

In [13]:

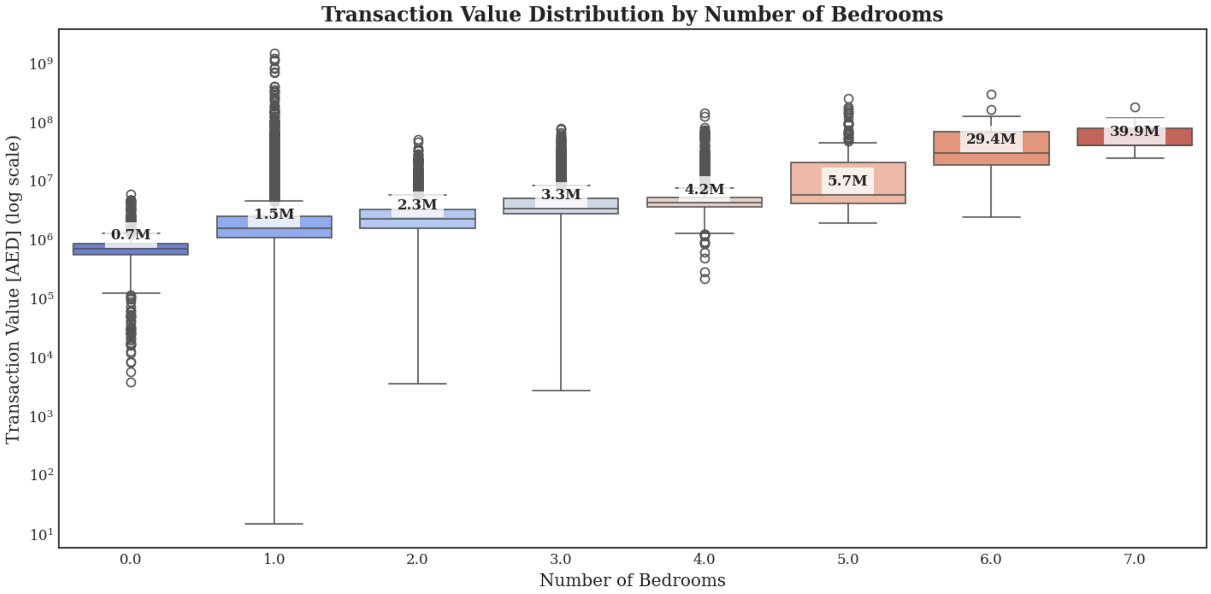
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Log transform transaction values (if not already done)
df_cleaned['LOG_TRANS_VALUE'] = np.log10(df_cleaned['TRANS_VALUE'].clip(lower=1))

prop_order = ['Unit', 'Building', 'Land']  # Adjust as needed
palette = {
    'Unit': '#2E86C1',      # Blue
    'Building': '#E67E22',  # Orange
    'Land': '#239B56'       # Green
}

plt.figure(figsize=(10, 6))
ax = sns.boxplot(
    data=df_cleaned,
    x='PROP_TYPE_EN',
    y='LOG_TRANS_VALUE',
    hue='PROP_TYPE_EN',
    order=prop_order,
    hue_order=prop_order,
    dodge=False,
    palette=palette
)

# Remove legend safely if it exists
if ax.get_legend() is not None:
    ax.get_legend().remove()

plt.title("Transaction Value by Property Type (log10 AED)", fontsize=15, fontweight
plt.xlabel("Property Type")
plt.ylabel("log10(Transaction Value)")
plt.grid(axis='y', linestyle=':', alpha=0.6)
plt.tight_layout()
plt.show()
```



In [14]:

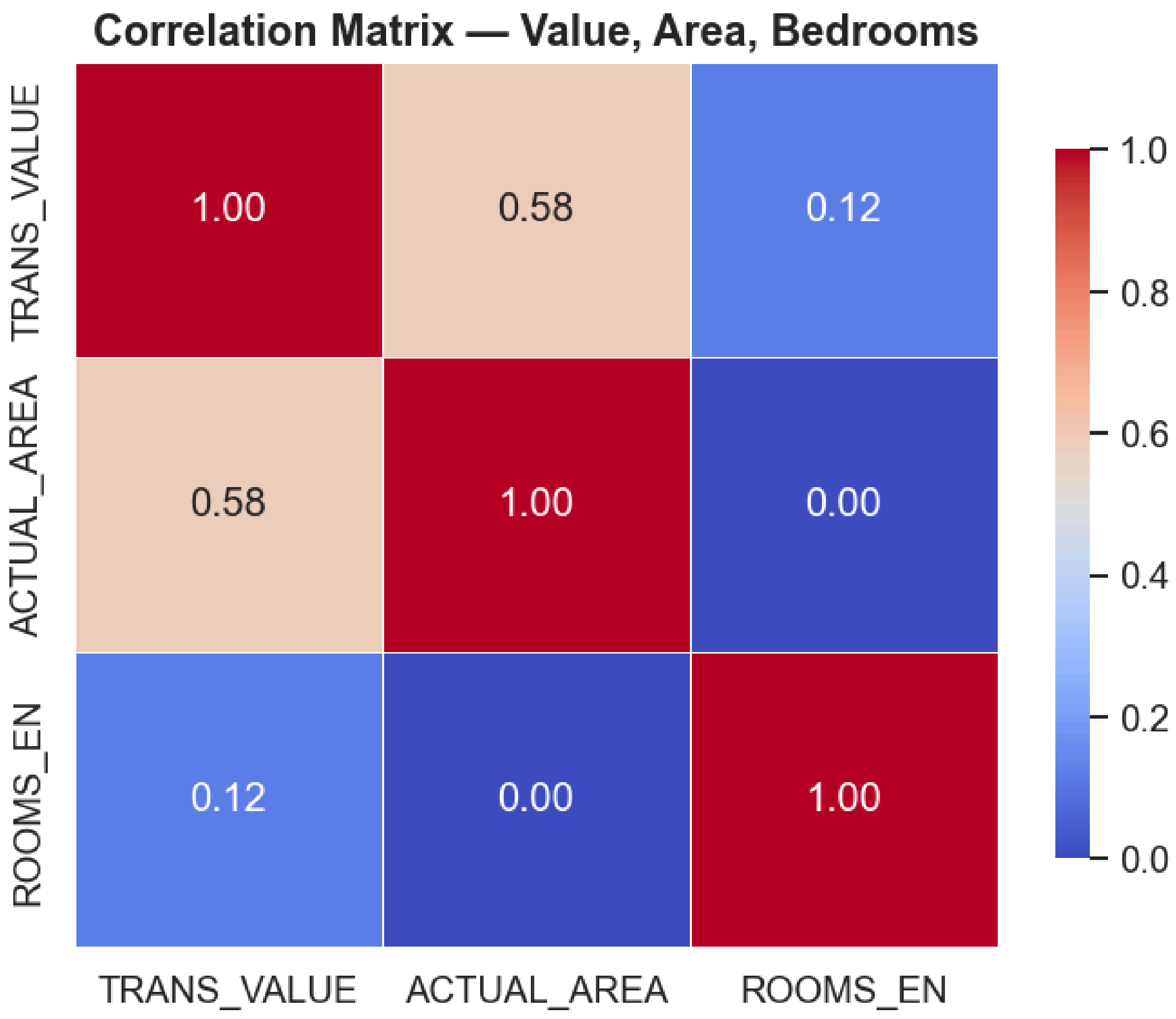
```
import re
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# ---------------------------
# Prepare data
# ---------------------------
df_plot = df_cleaned.copy()

def to_numeric_bedrooms(x):
    if pd.isna(x): return np.nan
    s = str(x).lower().strip()
    if 'studio' in s: return 0
    digits = re.sub(r'[^0-9]', '', s)
    return float(digits) if digits else np.nan

df_plot['ROOMS_EN'] = df_plot['ROOMS_EN'].apply(to_numeric_bedrooms)
df_plot = df_plot.dropna(subset=['ROOMS_EN', 'TRANS_VALUE'])
df_plot = df_plot[df_plot['TRANS_VALUE'] > 0]
df_plot = df_plot[df_plot['ROOMS_EN'].between(0, 7)]

medians = df_plot.groupby('ROOMS_EN')['TRANS_VALUE'].median()
bedroom_order = sorted(df_plot['ROOMS_EN'].unique())
```

```python
# ---------------------------
# Plot
# ---------------------------
plt.figure(figsize=(12, 6))
palette = sns.color_palette("coolwarm", len(bedroom_order))

ax = sns.boxplot(
    data=df_plot,
    x='ROOMS_EN',
    y='TRANS_VALUE',
    order=bedroom_order,
    hue='ROOMS_EN',
    dodge=False,
    legend=False,
    palette=palette
)

ax.set_yscale('log')

# Place median labels closer to the box, not on top of chart
for i, room in enumerate(bedroom_order):
    m = medians.loc[room]
    ax.text(
        i, m * 1.3,  # Adjust multiplier to control height
        f"{m/1e6:.1f}M",
        ha='center', va='bottom',
        fontsize=10, fontweight='bold',
        bbox=dict(facecolor='white', edgecolor='none', alpha=0.8)
    )

ax.set_title("Transaction Value Distribution by Number of Bedrooms", fontsize=14, f
ax.set_xlabel("Number of Bedrooms")
ax.set_ylabel("Transaction Value [AED] (log scale)")
plt.tight_layout()
plt.show()
```



In [15]:

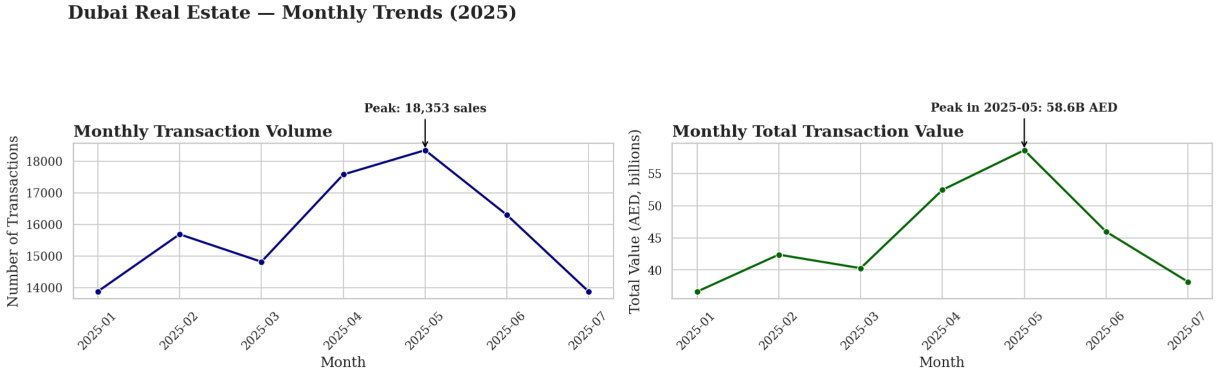
```
import seaborn as sns
import matplotlib.pyplot as plt

# Select key numeric columns
df_corr = df_cleaned[['TRANS_VALUE', 'ACTUAL_AREA', 'ROOMS_EN']].copy()

# Convert ROOMS_EN to numeric if needed
def to_numeric_bedrooms(x):
    if pd.isna(x): return np.nan
    s = str(x).lower().strip()
    if 'studio' in s: return 0
    digits = ''.join([c for c in s if c.isdigit()])
    return float(digits) if digits else np.nan

df_corr['ROOMS_EN'] = df_corr['ROOMS_EN'].apply(to_numeric_bedrooms)

# Drop rows with missing values
df_corr = df_corr.dropna()

# Compute correlation matrix
corr_matrix = df_corr.corr()

# Plot heatmap
plt.figure(figsize=(6, 5))
sns.set_theme(style="white", font_scale=1.1)

ax = sns.heatmap(
    corr_matrix,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    vmin=0, vmax=1,
    linewidths=0.5,
    linecolor='white',
    cbar_kws={"shrink": 0.8}
)

plt.title("Correlation Matrix – Value, Area, Bedrooms", fontsize=14, fontweight='bo
plt.tight_layout()
plt.show()
```



In [16]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# --- Monthly Data Preparation (same as before) ---
df_trends = df_cleaned.copy()
df_trends['INSTANCE_DATE'] = pd.to_datetime(df_trends['INSTANCE_DATE'])
df_2025 = df_trends[df_trends['INSTANCE_DATE'].dt.year == 2025].copy()
df_2025['Month'] = df_2025['INSTANCE_DATE'].dt.to_period('M').astype(str)

monthly_summary = df_2025.groupby('Month').agg(
    transactions=('TRANSACTION_NUMBER', 'count'),
    total_value=('TRANS_VALUE', 'sum')
).reset_index()
monthly_summary['total_value_billion'] = monthly_summary['total_value'] / 1e9

# Peak values
peak_idx = monthly_summary['transactions'].idxmax()
peak_month = monthly_summary.loc[peak_idx, 'Month']
peak_trans = monthly_summary.loc[peak_idx, 'transactions']
peak_value = monthly_summary.loc[peak_idx, 'total_value_billion']

# --- Plot Settings ---
sns.set_theme(style="whitegrid")
```

```
plt.rcParams.update({
    "font.family": "serif",
    "axes.titlesize": 15,
    "axes.labelsize": 13,
    "xtick.labelsize": 11,
    "ytick.labelsize": 11,
    "figure.dpi": 120,
})

# --- Larger Figure with Adjusted Layout ---
fig, axes = plt.subplots(1, 2, figsize=(16, 5))

# --- Monthly Transaction Volume ---
sns.lineplot(
    data=monthly_summary,
    x='Month',
    y='transactions',
    marker='o',
    color='navy',
    linewidth=2,
    ax=axes[0]
)
axes[0].set_title("Monthly Transaction Volume", fontsize=15, fontweight='bold', loc
axes[0].set_xlabel("Month")
axes[0].set_ylabel("Number of Transactions")
axes[0].tick_params(axis='x', rotation=45)
axes[0].annotate(
    f"Peak: {peak_trans:,} sales",
    xy=(peak_idx, peak_trans),
    xytext=(peak_idx, peak_trans + 1200),
    fontsize=11, fontweight='bold',
    ha='center',
    arrowprops=dict(arrowstyle="->", color='black', linewidth=1.2)
)

# --- Monthly Total Transaction Value ---
sns.lineplot(
    data=monthly_summary,
    x='Month',
    y='total_value_billion',
    marker='o',
    color='darkgreen',
    linewidth=2,
    ax=axes[1]
)
axes[1].set_title("Monthly Total Transaction Value", fontsize=15, fontweight='bold'
axes[1].set_xlabel("Month")
axes[1].set_ylabel("Total Value (AED, billions)")
axes[1].tick_params(axis='x', rotation=45)
axes[1].annotate(
    f"Peak in {peak_month}: {peak_value:.1f}B AED",
    xy=(peak_idx, peak_value),
    xytext=(peak_idx, peak_value + 6),
    fontsize=11, fontweight='bold',
    ha='center',
    arrowprops=dict(arrowstyle="->", color='black', linewidth=1.2)
```

```
)

# --- Reduce Top Gap ---
plt.suptitle("Dubai Real Estate — Monthly Trends (2025)", fontsize=17, fontweight='
plt.tight_layout(rect=[0, 0, 1, 0.92])  # Less top padding
plt.show()
```

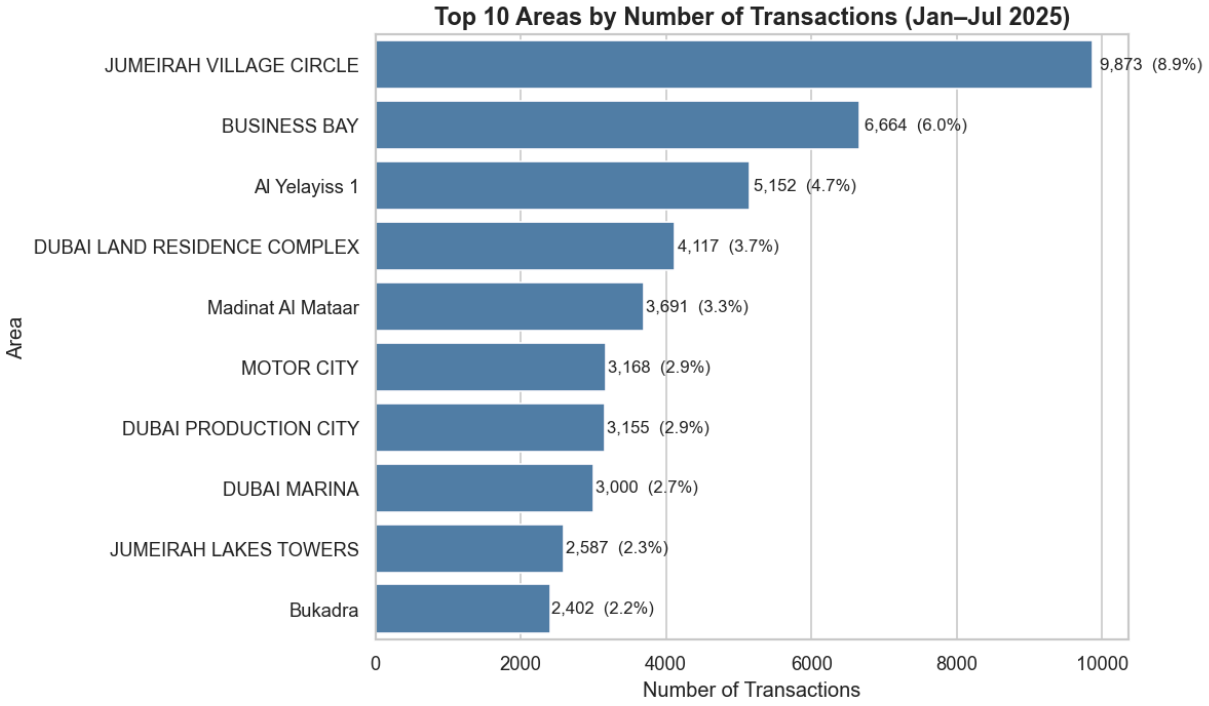


In [17]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# ----------------------------
# 1) Filter Jan–Jul 2025
# ----------------------------
df_geo = df_cleaned.copy()
df_geo['INSTANCE_DATE'] = pd.to_datetime(df_geo['INSTANCE_DATE'])

start, end = '2025-01-01', '2025-07-31'
mask = (df_geo['INSTANCE_DATE'] >= start) & (df_geo['INSTANCE_DATE'] <= end)
df_geo = df_geo.loc[mask]

# ----------------------------
# 2) Aggregate by AREA_EN
# ----------------------------
g = (df_geo
     .groupby('AREA_EN', dropna=False)
     .agg(
         transactions=('TRANSACTION_NUMBER', 'count'),
         total_value=('TRANS_VALUE', 'sum')
     )
     .sort_values('transactions', ascending=False))

# Calculate shares
total_txn = g['transactions'].sum()
g['txn_share_%'] = 100 * g['transactions'] / total_txn

total_val = g['total_value'].sum()
g['val_share_%'] = 100 * g['total_value'] / total_val

# ----------------------------
# 3) Top 10 by transactions
# ----------------------------
top10_txn = g.nlargest(10, 'transactions').reset_index()
```

```python
print("Top 10 areas by number of transactions (Jan–Jul 2025)")
print(top10_txn[['AREA_EN', 'transactions', 'txn_share_%']].round({'txn_share_%': 2

# ----------------------------
# 4) Plot – Top 10 by transactions
# ----------------------------
sns.set_theme(style="whitegrid")
plt.figure(figsize=(10, 6))

# FIX: Use 'color' instead of 'palette' (no hue required)
ax = sns.barplot(
    data=top10_txn,
    y='AREA_EN',
    x='transactions',
    color='steelblue'  # Fixed to avoid FutureWarning
)

# Annotate bars with counts & share
for i, r in top10_txn.iterrows():
    ax.text(r['transactions'] * 1.01, i,
            f"{r['transactions']:,}  ({r['txn_share_%']:.1f}%)",
            va='center', fontsize=10)

plt.title("Top 10 Areas by Number of Transactions (Jan–Jul 2025)",
          fontsize=14, fontweight='bold')
plt.xlabel("Number of Transactions")
plt.ylabel("Area")
plt.tight_layout()
plt.show()
```

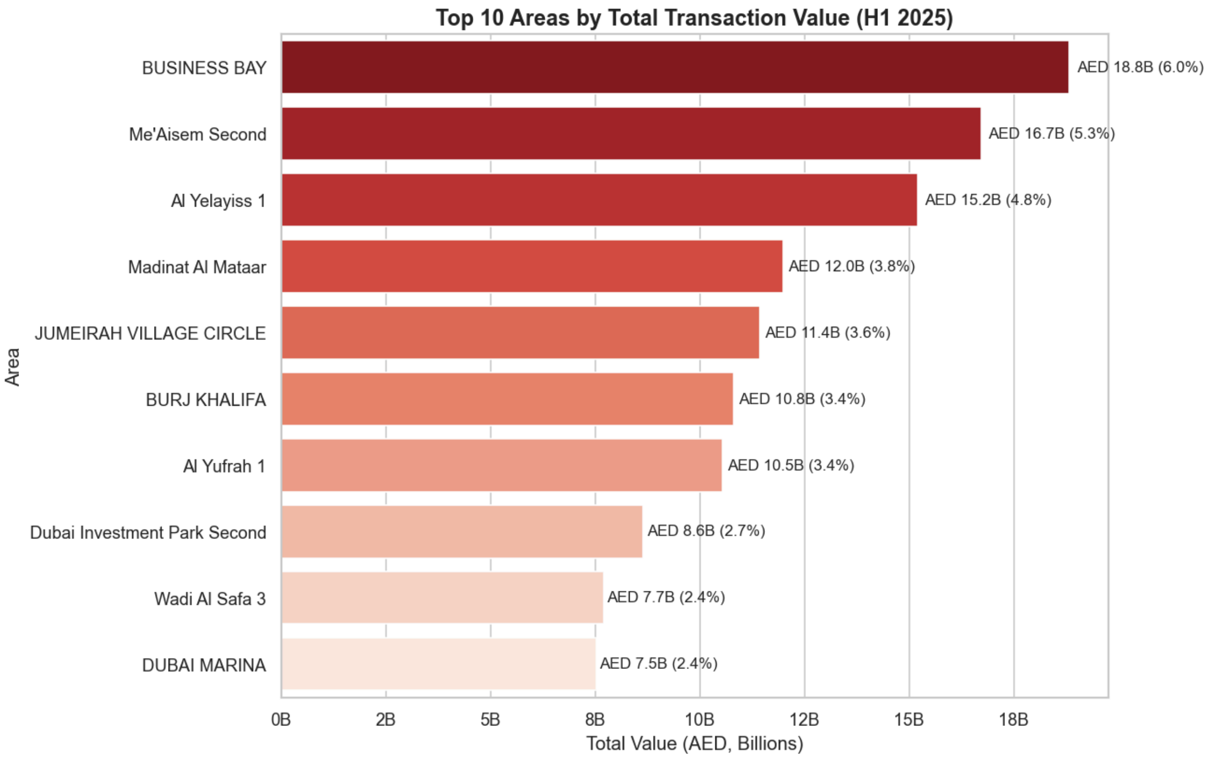
```
Top 10 areas by number of transactions (Jan–Jul 2025)
                        AREA_EN  transactions  txn_share_%
0       JUMEIRAH VILLAGE CIRCLE          9873         8.94
1                  BUSINESS BAY          6664         6.03
2                 Al Yelayiss 1          5152         4.66
3  DUBAI LAND RESIDENCE COMPLEX          4117         3.73
4             Madinat Al Mataar          3691         3.34
5                    MOTOR CITY          3168         2.87
6         DUBAI PRODUCTION CITY          3155         2.86
7                  DUBAI MARINA          3000         2.72
8         JUMEIRAH LAKES TOWERS          2587         2.34
9                       Bukadra          2402         2.17
```



In [18]:

```
import matplotlib.ticker as ticker

# Top 10 by transaction value
top10_val = g.nlargest(10, 'total_value').reset_index()

# Plot 2: Transaction Value
plt.figure(figsize=(11, 7))
ax = sns.barplot(
    data=top10_val,
    y='AREA_EN',
    x='total_value',
    hue='AREA_EN',
    legend=False,
    palette='Reds_r'
)

def billions(x, pos):
    return f'{x/1e9:.0f}B'
ax.xaxis.set_major_formatter(ticker.FuncFormatter(billions))

# Annotate each bar
for i, r in top10_val.iterrows():
    ax.text(
        r['total_value'] * 1.01,
        i,
        f"AED {r['total_value']/1e9:.1f}B ({r['val_share_%']:.1f}%)",
        va='center',
        fontsize=10
    )

ax.set_title("Top 10 Areas by Total Transaction Value (H1 2025)", fontsize=14, font
ax.set_xlabel("Total Value (AED, Billions)")
ax.set_ylabel("Area")
```

```
plt.tight_layout()
plt.show()
```



In [19]:

```
# 0) Imports

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np # Used for checking numerical types
import sklearn # Import sklearn to check version if needed

# 1) Configuration
#    - Centralize parameters for easy modification

FILE_PATH = "C:/Users/rohit/Downloads/transactions-2025-07-23.csv.xlsx"
TARGET_COLUMN = 'TRANS_VALUE'

# Columns to potentially drop if they have too many missing values
COLUMNS_TO_CHECK_MISSING = [
    'MASTER_PROJECT_EN', 'NEAREST_METRO_EN', 'NEAREST_MALL_EN',
    'NEAREST_LANDMARK_EN'
]
MISSING_THRESHOLD_PERCENT = 30 # Drop columns with over 30% missing values

# Specific columns for numerical imputation
NUMERICAL_IMPUTE_COLUMNS = ['PROCEDURE_AREA'] # Example if you have others

# Specific columns for categorical imputation
CATEGORICAL_IMPUTE_COLUMNS = ['PARKING', 'ROOMS_EN', 'PROJECT_EN', 'PROP_SB_TYPE_EN
```

```
# Features to be used in the model
# Ensure these align with your imputation and data types
MODEL_FEATURES = ['PROCEDURE_AREA', 'PARKING', 'ROOMS_EN', 'PROJECT_EN', 'PROP_SB_T

RANDOM_STATE = 42
TEST_SIZE = 0.2
N_ESTIMATORS = 100


# 2) Load & Initial Data Inspection

print("Attempting to load data...")
try:
    # Check for the correct file type and use appropriate reader
    if FILE_PATH.lower().endswith('.csv.xlsx') or FILE_PATH.lower().endswith('.xlsx
        df = pd.read_excel(FILE_PATH)
    elif FILE_PATH.lower().endswith('.csv'):
        df = pd.read_csv(FILE_PATH)
    else:
        raise ValueError("Unsupported file format. Please provide a .csv or .xlsx f
    print("✅ Data loaded successfully!")
    print(f"Initial DataFrame shape: {df.shape}")
    print("Initial DataFrame head:\n", df.head())
    print("Initial DataFrame info:")
    df.info() # Provides non-null counts and dtypes
    print("-" * 50)

except FileNotFoundError:
    print(f"❌ Error: File not found at '{FILE_PATH}'. Please check the path and fi
    exit()
except Exception as e:
    print(f"❌ An error occurred during data loading: {e}")
    exit()

# Make a copy to avoid SettingWithCopyWarning
df_cleaned = df.copy()


# 3) Data Cleaning

print("\nStarting data cleaning...")

# Drop columns with over MISSING_THRESHOLD_PERCENT missing values
print(f"Checking columns to drop with > {MISSING_THRESHOLD_PERCENT}% missing values
cols_to_drop_actual = []
for col in COLUMNS_TO_CHECK_MISSING:
    if col in df_cleaned.columns:
        missing_percent = df_cleaned[col].isnull().sum() / len(df_cleaned) * 100
        if missing_percent > MISSING_THRESHOLD_PERCENT:
            cols_to_drop_actual.append(col)
            print(f"  - Dropping '{col}' (missing: {missing_percent:.2f}%)")
    else:
        print(f"  - Warning: Column '{col}' not found in DataFrame for missing chec

if cols_to_drop_actual:
    df_cleaned = df_cleaned.drop(columns=cols_to_drop_actual)
    print(f"✅ Dropped {len(cols_to_drop_actual)} columns due to high missing value
else:
    print("✅ No columns dropped due to high missing values based on criteria.")

print(f"DataFrame shape after dropping columns: {df_cleaned.shape}")

# Impute missing numerical values
print("Imputing missing numerical values (median)...")
for col in NUMERICAL_IMPUTE_COLUMNS:
    if col in df_cleaned.columns and df_cleaned[col].isnull().any():
        if pd.api.types.is_numeric_dtype(df_cleaned[col]):
            median_val = df_cleaned[col].median()
            df_cleaned[col] = df_cleaned[col].fillna(median_val)
            print(f"  - Imputed '{col}' with median: {median_val}")
        else:
            print(f"  - Warning: '{col}' is not numeric, skipping median imputation
    elif col in df_cleaned.columns:
        print(f"  - '{col}' has no missing values, skipping imputation.")
    else:
        print(f"  - Warning: Numerical imputation column '{col}' not found in DataF

# Impute missing categorical values
print("Imputing missing categorical values (mode)...")
for col in CATEGORICAL_IMPUTE_COLUMNS:
    if col in df_cleaned.columns and df_cleaned[col].isnull().any():
        # Check if mode is empty
        if not df_cleaned[col].mode().empty:
            mode_val = df_cleaned[col].mode()[0]
            df_cleaned[col] = df_cleaned[col].fillna(mode_val)
            print(f"  - Imputed '{col}' with mode: '{mode_val}'")
        else:
            print(f"  - Warning: Cannot find mode for '{col}', column might be enti
    elif col in df_cleaned.columns:
        print(f"  - '{col}' has no missing values, skipping imputation.")
    else:
        print(f"  - Warning: Categorical imputation column '{col}' not found in Dat

print("✅ Data cleaning complete.")
print(f"DataFrame shape after cleaning: {df_cleaned.shape}")
print("Missing values after cleaning:\n", df_cleaned.isnull().sum())
print("-" * 50)


# 4) Features & Target Preparation

print("\nPreparing features and target...")

# Verify all MODEL_FEATURES exist in the cleaned DataFrame
missing_model_features = [col for col in MODEL_FEATURES if col not in df_cleaned.co
if missing_model_features:
    print(f"❌ Error: The following model features are missing from the cleaned Dat
    exit()

if TARGET_COLUMN not in df_cleaned.columns:
    print(f"❌ Error: Target column '{TARGET_COLUMN}' not found in the DataFrame.")
    exit()

X = df_cleaned[MODEL_FEATURES].copy()
y = df_cleaned[TARGET_COLUMN].copy()

# Handle potential NaN values in the target variable
if y.isnull().any():
    print(f"Warning: Target column '{TARGET_COLUMN}' contains missing values. Rows
    # Drop rows where the target is NaN
    initial_rows = X.shape[0]
    valid_indices = y.dropna().index
    X = X.loc[valid_indices]
    y = y.loc[valid_indices]
    print(f"Dropped {initial_rows - X.shape[0]} rows due to missing target values."

print(f"Features (X) shape: {X.shape}")
print(f"Target (y) shape: {y.shape}")

#categorical features are string type for OneHotEncoder
print("Ensuring categorical features are string type...")
# Dynamically determine which of MODEL_FEATURES are categorical based on their incl
# cat_features for the ColumnTransformer are derived from MODEL_FEATURES that are m
cat_features_for_ohe = [col for col in MODEL_FEATURES if col in CATEGORICAL_IMPUTE_

for col in cat_features_for_ohe:
    if col in X.columns:
        X[col] = X[col].astype(str)
        print(f"  - Converted '{col}' to string type.")
    else:
        print(f"  - Warning: Categorical feature '{col}' not found in X, skipping t

# Identify numerical features for the preprocessor
num_features_for_scaler = [col for col in MODEL_FEATURES if col not in cat_features
# A more robust way:
# num_features_for_scaler = X.select_dtypes(include=np.number).columns.tolist()


print("✅ Features and target prepared.")
print("-" * 50)


# 5) Train-Test Split

print("\nSplitting data into training and testing sets...")
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=TEST_SIZE, random_state=RANDOM_STATE
)
print(f"✅ Data split complete: Training ({len(X_train)} samples), Testing ({len(X_
print("-" * 50)


# 6) Preprocessing (ColumnTransformer)

print("\nSetting up preprocessing pipeline (ColumnTransformer)...")
preprocessor = ColumnTransformer(transformers=[
    ('num', StandardScaler(), num_features_for_scaler),
    ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), cat_featur
])
print("✅ Preprocessor configured.")
print("-" * 50)


# 7) Model Pipeline

print("\nBuilding the full model pipeline...")
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=N_ESTIMATORS, random_state=RAN
])
print("✅ Model pipeline created.")
print("-" * 50)


# 8) Train Model

print("\nStarting model training...")
try:
    model.fit(X_train, y_train)
    print("✅ Model training complete!")
except Exception as e:
    print(f"❌ An error occurred during model training: {e}")
    exit()
print("-" * 50)


# 9) Evaluate Model

print("\n🎯 Evaluating Model Performance...")
try:
    y_pred = model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)

    # Calculate Mean Squared Error first
    mse = mean_squared_error(y_test, y_pred)
    # Then calculate Root Mean Squared Error by taking the square root of MSE
    rmse = np.sqrt(mse)

    r2 = r2_score(y_test, y_pred)

    print("\n✅ Model Evaluation Results:")
    print(f"    MAE:  {mae:,.2f}")
    print(f"    RMSE: {rmse:,.2f}")
    print(f"    R²:   {r2:.4f}")
    print("-" * 50)

except Exception as e:
    print(f"❌ An error occurred during model evaluation: {e}")
    exit()

print("\nScript execution finished.")
```

```
Attempting to load data...
✅ Data loaded successfully!
Initial DataFrame shape: (110490, 22)
Initial DataFrame head:
   TRANSACTION_NUMBER       INSTANCE_DATE GROUP_EN             PROCEDURE_EN  \
0         102-1-2025 2025-01-07 15:57:40    Sales  Sell - Pre registration
1        102-10-2025 2025-01-02 08:23:00    Sales  Sell - Pre registration
2       102-100-2025 2025-01-02 15:18:51    Sales  Sell - Pre registration
3      102-1000-2025 2025-01-08 11:02:57    Sales  Sell - Pre registration
4     102-10000-2025 2025-02-07 11:15:01    Sales  Sell - Pre registration

  IS_OFFPLAN_EN IS_FREE_HOLD_EN     USAGE_EN             AREA_EN PROP_TYPE_EN  \
0      Off-Plan       Free Hold  Residential      Wadi Al Safa 4         Unit
1      Off-Plan       Free Hold  Residential  DUBAI SCIENCE PARK         Unit
2      Off-Plan       Free Hold  Residential           AL FURJAN         Unit
3      Off-Plan       Free Hold  Residential      Wadi Al Safa 5         Unit
4      Off-Plan       Free Hold  Residential       Al Yelayiss 2         Unit

   PROP_SB_TYPE_EN  ...  ACTUAL_AREA  ROOMS_EN  PARKING  \
0             Flat  ...        68.67     1 B/R        1
1             Flat  ...        47.25    Studio        1
2  Hotel Apartment  ...        32.35    Studio        1
3             Flat  ...       259.03     2 B/R        2
4             Flat  ...        66.08     1 B/R        1

                     NEAREST_METRO_EN       NEAREST_MALL_EN  \
0                                 NaN                   NaN
1  First Abu Dhabi Bank Metro Station  Mall of the Emirates
2                ENERGY Metro Station    Ibn-e-Battuta Mall
3                                 NaN                   NaN
4                                 NaN                   NaN

  NEAREST_LANDMARK_EN TOTAL_BUYER TOTAL_SELLER  MASTER_PROJECT_EN  \
0                 NaN           0            0                NaN
1          Motor City           0            0                NaN
2      Expo 2020 Site           0            0                NaN
3                 NaN           0            0                NaN
4                 NaN           0            0                NaN

          PROJECT_EN
0             Lacina
1   Binghatti Hills
2       AZIZI JEWEL
3  Verdes by Haven 1
4      The Baltimore

[5 rows x 22 columns]
Initial DataFrame info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110490 entries, 0 to 110489
Data columns (total 22 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   TRANSACTION_NUMBER   110490 non-null  object
 1   INSTANCE_DATE        110490 non-null  datetime64[ns]
 2   GROUP_EN             110490 non-null  object
 3   PROCEDURE_EN         110490 non-null  object
 4   IS_OFFPLAN_EN        110490 non-null  object
 5   IS_FREE_HOLD_EN      110490 non-null  object
 6   USAGE_EN             110490 non-null  object
 7   AREA_EN              110490 non-null  object
 8   PROP_TYPE_EN         110490 non-null  object
 9   PROP_SB_TYPE_EN      104394 non-null  object
 10  TRANS_VALUE          110490 non-null  float64
 11  PROCEDURE_AREA       110489 non-null  float64
 12  ACTUAL_AREA          110490 non-null  float64
 13  ROOMS_EN             94277 non-null   object
 14  PARKING              85646 non-null   object
 15  NEAREST_METRO_EN     56461 non-null   object
 16  NEAREST_MALL_EN      55007 non-null   object
 17  NEAREST_LANDMARK_EN  68028 non-null   object
 18  TOTAL_BUYER          110490 non-null  int64
 19  TOTAL_SELLER         110490 non-null  int64
 20  MASTER_PROJECT_EN    277 non-null     object
 21  PROJECT_EN           100900 non-null  object
dtypes: datetime64[ns](1), float64(3), int64(2), object(16)
memory usage: 18.5+ MB
--------------------------------------------------

Starting data cleaning...
Checking columns to drop with > 30% missing values...
  - Dropping 'MASTER_PROJECT_EN' (missing: 99.75%)
  - Dropping 'NEAREST_METRO_EN' (missing: 48.90%)
  - Dropping 'NEAREST_MALL_EN' (missing: 50.22%)
  - Dropping 'NEAREST_LANDMARK_EN' (missing: 38.43%)
✅ Dropped 4 columns due to high missing values.
DataFrame shape after dropping columns: (110490, 18)
Imputing missing numerical values (median)...
  - Imputed 'PROCEDURE_AREA' with median: 86.6
Imputing missing categorical values (mode)...
  - Imputed 'PARKING' with mode: '1'
  - Imputed 'ROOMS_EN' with mode: '1 B/R'
  - Imputed 'PROJECT_EN' with mode: 'Binghatti Skyrise'
  - Imputed 'PROP_SB_TYPE_EN' with mode: 'Flat'
✅ Data cleaning complete.
DataFrame shape after cleaning: (110490, 18)
Missing values after cleaning:
 TRANSACTION_NUMBER    0
INSTANCE_DATE         0
GROUP_EN              0
PROCEDURE_EN          0
IS_OFFPLAN_EN         0
IS_FREE_HOLD_EN       0
USAGE_EN              0
AREA_EN               0
PROP_TYPE_EN          0
PROP_SB_TYPE_EN       0
TRANS_VALUE           0
PROCEDURE_AREA        0
ACTUAL_AREA           0
ROOMS_EN              0
PARKING               0
```

```
TOTAL_BUYER           0
TOTAL_SELLER          0
PROJECT_EN            0
dtype: int64
--------------------------------------------------

Preparing features and target...
Features (X) shape: (110490, 5)
Target (y) shape: (110490,)
Ensuring categorical features are string type...
  - Converted 'PARKING' to string type.
  - Converted 'ROOMS_EN' to string type.
  - Converted 'PROJECT_EN' to string type.
  - Converted 'PROP_SB_TYPE_EN' to string type.
✅ Features and target prepared.
--------------------------------------------------

Splitting data into training and testing sets...
✅ Data split complete: Training (88392 samples), Testing (22098 samples).
--------------------------------------------------

Setting up preprocessing pipeline (ColumnTransformer)...
✅ Preprocessor configured.
--------------------------------------------------

Building the full model pipeline...
✅ Model pipeline created.
--------------------------------------------------

Starting model training...
✅ Model training complete!
--------------------------------------------------

🎯 Evaluating Model Performance...

✅ Model Evaluation Results:
    MAE:  493,806.29
    RMSE: 6,586,172.42
    R²:   0.6660
--------------------------------------------------

Script execution finished.
```

```
In [ ]:
```