Ames Housing Price Prediction

Advanced Apex Project

Real Estate Price Modeling

Team: The Outliers

Institution: BITS Pilani

Course: Advanced Apex Project 1

Generated: November 16, 2025

Ames Housing Price Prediction

Advanced Apex Project - Real Estate Price Modeling

A comprehensive machine learning approach to predicting residential property sale prices using multiple regression techniques and extensive feature engineering.

Project Information

Team: The Outliers

Course: Advanced Apex Project 1

Institution: BITS Pilani - Digital Campus

Academic Term: First Trimester 2025-26

Project Supervisor: Bharathi Dasari

Submission Date: November 2024

Team Members

| Student Name | BITS ID | |--------------|----------| | Anik Das | 2025EM1100026 | | Adeetya Wadikar | 2025EM1100384 | | Tushar Nishane | 2025EM1100306 |

Executive Summary

Problem Statement

Accurate real estate valuation is essential for buyers, sellers, and financial institutions. Traditional valuation methods can be subjective and time-consuming. This project develops machine learning models to predict house sale prices objectively based on property characteristics.

Business Objective

Develop a predictive regression model that estimates residential property sale prices with high accuracy. The model should help stakeholders:

- **Buyers**: Assess fair market value before purchase
- **Sellers**: Set competitive listing prices
- **Investors**: Identify undervalued properties
- **Lenders**: Support loan underwriting decisions

Dataset

Name: Ames Housing Dataset

Source: Kaggle (https://www.kaggle.com/datasets/shashanknecrothapa/ames-housing-dataset)

Size: 2,930 residential property sales transactions

Features: 82 variables describing:

- Physical characteristics (size, rooms, age)
- Quality ratings (construction, condition)
- Location attributes (neighborhood, zoning)
- Amenities (garage, basement, fireplace, pool)

Target Variable: SalePrice (in USD)

Time Period: Properties sold in Ames, Iowa from 2006-2010

Table of Contents

Phase 1: Data Acquisition

Objective

Acquire the Ames Housing dataset and perform initial validation to ensure data integrity. This foundational phase establishes the quality and completeness of our data before proceeding to analysis.

Deliverables

- Successfully load dataset from CSV file
- Verify data structure and schema
- Conduct initial quality checks
- Document data characteristics and potential issues

1.1 Environment Setup

We import all necessary Python libraries for data manipulation, statistical analysis, visualization, and machine learning. Proper configuration ensures consistent behavior across different environments.

Code Cell 1

```
# Import core data manipulation libraries
import pandas as pd
import numpy as np
import os

# Import visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno

# Import statistical libraries
from scipy import stats

# Import machine learning libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Configure environment
import warnings
warnings.filterwarnings('ignore')

# Set display options for better readability
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)
pd.set_option('display.float_format', '{:.2f}'.format)
pd.set_option('display.width', 1000)

# Set visualization defaults
sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 10

# Print confirmation
print("✓ All libraries imported successfully")
print(f"✓ Pandas version: {pd.__version__}")
print(f"✓ NumPy version: {np.__version__}")
print(f"✓ Matplotlib version: {plt.matplotlib.__version__}")
print("\nEnvironment configured and ready for analysis.")
```

Output:

```
✓ All libraries imported successfully
✓ Pandas version: 2.3.3
✓ NumPy version: 2.3.4
✓ Matplotlib version: 3.10.7

Environment configured and ready for analysis.
```

1.2 Data Loading

The Ames Housing dataset was downloaded from Kaggle and stored in the project's data directory. This dataset provides comprehensive information on residential properties sold in Ames, Iowa, making it an excellent resource for developing price prediction models.

Data Source: Kaggle - Ames Housing Dataset

Citation: Shashank Necrothapa. (n.d.). Ames Housing Dataset. Kaggle. https://www.kaggle.com/datasets/shashanknecrothapa/ames-housing-dataset

Code Cell 2

```
# Define the path to the dataset
data_path = "../data/AmesHousing.csv"

# Load the dataset into a pandas DataFrame
df = pd.read_csv(data_path)

# Display basic information
print("✓ Dataset loaded successfully!")
print(f"\nDataset Dimensions: {df.shape[0]:,} rows × {df.shape[1]} columns")
print(f"Memory Usage: {df.memory_usage(deep=True).sum() / 1024**2:.2f} MB")

# Display first few records
print("\nFirst 5 Records:")
df.head()
```

Output:

```
✓ Dataset loaded successfully!

Dataset Dimensions: 2,930 rows × 82 columns
Memory Usage: 7.76 MB

First 5 Records:
```

```
   Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street Alley Lot Shape L
0      1  526301100           20        RL        141.00     31770   Pave   NaN       IR1
1      2  526350040           20        RH         80.00     11622   Pave   NaN       Reg
2      3  526351010           20        RL         81.00     14267   Pave   NaN       IR1
3      4  526353030           20        RL         93.00     11160   Pave   NaN       Reg
4      5  527105010           60        RL         74.00     13830   Pave   NaN       IR1
```

1.3 Initial Data Inspection

Before conducting detailed analysis, we perform a high-level inspection to understand the dataset structure, identify data types, and spot any immediate quality concerns.

Code Cell 3

```
# Display comprehensive dataset information
print("Dataset Structure Overview:\n")
df.info()

print("\n" + "="*70)
print("Data Type Summary:")
print("="*70)
print(df.dtypes.value_counts())

print("\n" + "="*70)
print("Column Distribution:")
print("="*70)
print(f"Numerical columns (int64): {len(df.select_dtypes(include=['int64']).columns)}")
print(f"Numerical columns (float64): {len(df.select_dtypes(include=['float64']).columns)}")
print(f"Categorical columns (object): {len(df.select_dtypes(include=['object']).columns)}")
```

Output:

```
Dataset Structure Overview:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2930 entries, 0 to 2929
Data columns (total 82 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Order            2930 non-null   int64
 1   PID              2930 non-null   int64
 2   MS SubClass      2930 non-null   int64
 3   MS Zoning        2930 non-null   object
 4   Lot Frontage     2440 non-null   float64
 5   Lot Area         2930 non-null   int64
 6   Street           2930 non-null   object
 7   Alley            198 non-null    object
 8   Lot Shape        2930 non-null   object
 9   Land Contour     2930 non-null   object
 10  Utilities        2930 non-null   object
 11  Lot Config       2930 non-null   object
 12  Land Slope       2930 non-null   object
 13  Neighborhood     2930 non-null   object
 14  Condition 1      2930 non-null   object
 15  Condition 2      2930 non-null   object
 16  Bldg Type        2930 non-null   object
 17  House Style      2930 non-null   object
 18  Overall Qual     2930 non-null   int64
 19  Overall Cond     2930 non-null   int64
 20  Year Built       2930 non-null   int64
 21  Year Remod/Add   2930 non-null   int64
 22  Roof Style       2930 non-null   object
 23  Roof Matl        2930 non-null   object
 24  Exterior 1st     2930 non-null   object
 25  Exterior 2nd     2930 non-null   object
 26  Mas Vnr Type     1155 non-null   object
```

```
 27  Mas Vnr Area     2907 non-null   float64
 28  Exter Qual       2930 non-null   object
 29  Exter Cond       2930 non-null   object
 30  Foundation       2930 non-null   object
 31  Bsmt Qual        2850 non-null   object
 32  Bsmt Cond        2850 non-null   object
 33  Bsmt Exposure    2847 non-null   object
 34  BsmtFin Type 1   2850 non-null   object
 35  BsmtFin SF 1     2929 non-null   float64
 36  BsmtFin Type 2   2849 non-null   object
 37  BsmtFin SF 2     2929 non-null   float64
 38  Bsmt Unf SF      2929 non-null   float64
 39  Total Bsmt SF    2929 non-null   float64
 40  Heating          2930 non-null   object
 41  Heating QC       2930 non-null   object
 42  Central Air      2930 non-null   object
 43  Electrical       2929 non-null   object
 44  1st Flr SF       2930 non-null   int64
 45  2nd Flr SF       2930 non-null   int64
 46  Low Qual Fin SF  2930 non-null   int64
 47  Gr Liv Area      2930 non-null   int64
 48  Bsmt Full Bath   2928 non-null   float64
 49  Bsmt Half Bath   2928 non-null   float64
 50  Full Bath        2930 non-null   int64
 51  Half Bath        2930 non-null   int64
 52  Bedroom AbvGr    2930 non-null   int64
 53  Kitchen AbvGr    2930 non-null   int64
 54  Kitchen Qual     2930 non-null   object
 55  TotRms AbvGrd    2930 non-null   int64
 56  Functional       2930 non-null   object
 57  Fireplaces       2930 non-null   int64
 58  Fireplace Qu     1508 non-null   object
 59  Garage Type      2773 non-null   object
 60  Garage Yr Blt    2771 non-null   float64
 61  Garage Finish    2771 non-null   object
 62  Garage Cars      2929 non-null   float64
 63  Garage Area      2929 non-null   float64
 64  Garage Qual      2771 non-null   object
 65  Garage Cond      2771 non-null   object
 66  Paved Drive      2930 non-null   object
 67  Wood Deck SF     2930 non-null   int64
 68  Open Porch SF    2930 non-null   int64
 69  Enclosed Porch   2930 non-null   int64
 70  3Ssn Porch       2930 non-null   int64
 71  Screen Porch     2930 non-null   int64
 72  Pool Area        2930 non-null   int64
 73  Pool QC          13 non-null     object
 74  Fence            572 non-null    object
 75  Misc Feature     106 non-null    object
 76  Misc Val         2930 non-null   int64
 77  Mo Sold          2930 non-null   int64
 78  Yr Sold          2930 non-null   int64
 79  Sale Type        2930 non-null   object
 80  Sale Condition   2930 non-null   object
 81  SalePrice        2930 non-null   int64
dtypes: float64(11), int64(28), object(43)
memory usage: 1.8+ MB
```

```
======================================================================
Data Type Summary:
======================================================================
object     43
int64      28
float64    11
Name: count, dtype: int64

======================================================================
Column Distribution:
======================================================================
Numerical columns (int64): 28
Numerical columns (float64): 11
Categorical columns (object): 43
```

1.4 Schema Validation

We verify that all expected columns are present and properly formatted. This schema validation ensures data integrity and helps identify any structural anomalies early in the process.

Code Cell 4

```
# Display all column names
print(f"Total Features: {len(df.columns)}\n")
print("All Column Names:")
print("="*70)

# Print in organized format (4 columns)
col_list = df.columns.tolist()
for i in range(0, len(col_list), 4):
    row = col_list[i:i+4]
    print(f"{i+1:2d}-{i+len(row):2d}: " + " | ".join(f"{col:20s}" for col in row))

print("\n" + "="*70)
print("Key Columns Verified:")
print("="*70)
important_cols = ['Order', 'PID', 'SalePrice', 'Gr Liv Area', 'Overall Qual', 'Neighborhood']
for col in important_cols:
    status = "✓" if col in df.columns else "✗"
    print(f"{status} {col}")
```

Output:

```
Total Features: 82

All Column Names:
======================================================================
 1- 4: Order                | PID                  | MS SubClass          | MS Zoning
 5- 8: Lot Frontage         | Lot Area             | Street               | Alley
 9-12: Lot Shape            | Land Contour         | Utilities            | Lot Config
13-16: Land Slope           | Neighborhood         | Condition 1          | Condition 2
17-20: Bldg Type            | House Style          | Overall Qual         | Overall Cond
21-24: Year Built           | Year Remod/Add       | Roof Style           | Roof Matl
25-28: Exterior 1st         | Exterior 2nd         | Mas Vnr Type         | Mas Vnr Area
29-32: Exter Qual           | Exter Cond           | Foundation           | Bsmt Qual
33-36: Bsmt Cond            | Bsmt Exposure        | BsmtFin Type 1       | BsmtFin SF 1
37-40: BsmtFin Type 2       | BsmtFin SF 2         | Bsmt Unf SF          | Total Bsmt SF
41-44: Heating              | Heating QC           | Central Air          | Electrical
45-48: 1st Flr SF           | 2nd Flr SF           | Low Qual Fin SF      | Gr Liv Area
49-52: Bsmt Full Bath       | Bsmt Half Bath       | Full Bath            | Half Bath
53-56: Bedroom AbvGr        | Kitchen AbvGr        | Kitchen Qual         | TotRms AbvGrd
57-60: Functional           | Fireplaces           | Fireplace Qu         | Garage Type
61-64: Garage Yr Blt        | Garage Finish        | Garage Cars          | Garage Area
65-68: Garage Qual          | Garage Cond          | Paved Drive          | Wood Deck SF
69-72: Open Porch SF        | Enclosed Porch       | 3Ssn Porch           | Screen Porch
73-76: Pool Area            | Pool QC              | Fence                | Misc Feature
77-80: Misc Val             | Mo Sold              | Yr Sold              | Sale Type
81-82: Sale Condition       | SalePrice

======================================================================
Key Columns Verified:
======================================================================
✓ Order
✓ PID
```

```
✓ SalePrice
✓ Gr Liv Area
✓ Overall Qual
✓ Neighborhood
```

1.5 Data Quality Assessment

We conduct initial quality checks to identify missing values, duplicate records, and verify the target variable integrity.

Code Cell 5

```
# Perform comprehensive quality checks
print("Data Quality Assessment:")
print("="*70)

# Check for missing values
total_missing = df.isnull().sum().sum()
cols_with_missing = df.isnull().any().sum()
print(f"\nMissing Value Check:")
print(f"  Total missing values: {total_missing:,}")
print(f"  Columns with missing data: {cols_with_missing} out of {len(df.columns)}")

# Check for duplicates
duplicates = df.duplicated().sum()
print(f"\nDuplicate Check:")
print(f"  Duplicate rows: {duplicates}")
if duplicates == 0:
    print("  ✓ No duplicates found")

# Verify target variable
print(f"\nTarget Variable (SalePrice) Verification:")
print(f"  Missing values: {df['SalePrice'].isnull().sum()}")
print(f"  Minimum: ${df['SalePrice'].min():,}")
print(f"  Maximum: ${df['SalePrice'].max():,}")
print(f"  Mean: ${df['SalePrice'].mean():,.2f}")
print(f"  Median: ${df['SalePrice'].median():,.2f}")
print(f"  Standard Deviation: ${df['SalePrice'].std():,.2f}")

print("="*70)
```

Output:

```
Data Quality Assessment:
======================================================================

Missing Value Check:
  Total missing values: 15,749
  Columns with missing data: 27 out of 82

Duplicate Check:
  Duplicate rows: 0
  ✓ No duplicates found

Target Variable (SalePrice) Verification:
  Missing values: 0
  Minimum: $12,789
  Maximum: $755,000
  Mean: $180,796.06
  Median: $160,000.00
  Standard Deviation: $79,886.69
======================================================================
```

Code Cell 6

```
# Create detailed schema summary table
schema_summary = pd.DataFrame({
    'Column': df.columns,
    'Data_Type': df.dtypes.values,
    'Non_Null_Count': df.count().values,
    'Null_Count': df.isnull().sum().values,
    'Null_Percentage': (df.isnull().sum() / len(df) * 100).values,
    'Unique_Values': [df[col].nunique() for col in df.columns]
})

# Sort by null percentage to see problematic columns first
schema_summary = schema_summary.sort_values('Null_Percentage', ascending=False)

print("Schema Summary (Top 20 columns by missing data):")
print("="*90)
schema_summary.head(20)
```

Output:

```
Schema Summary (Top 20 columns by missing data):
==========================================================================================
```

	Column	Data_Type	Non_Null_Count	Null_Count	Null_Percentage	Unique_Values
73	Pool QC	object	13	2917	99.56	4
75	Misc Feature	object	106	2824	96.38	5
7	Alley	object	198	2732	93.24	2
74	Fence	object	572	2358	80.48	4
26	Mas Vnr Type	object	1155	1775	60.58	4
58	Fireplace Qu	object	1508	1422	48.53	5
4	Lot Frontage	float64	2440	490	16.72	128
64	Garage Qual	object	2771	159	5.43	5
60	Garage Yr Blt	float64	2771	159	5.43	103
65	Garage Cond	object	2771	159	5.43	5
61	Garage Finish	object	2771	159	5.43	3
59	Garage Type	object	2773	157	5.36	6
33	Bsmt Exposure	object	2847	83	2.83	4
36	BsmtFin Type 2	object	2849	81	2.76	6
31	Bsmt Qual	object	2850	80	2.73	5
32	Bsmt Cond	object	2850	80	2.73	5
34	BsmtFin Type 1	object	2850	80	2.73	6
27	Mas Vnr Area	float64	2907	23	0.78	445
48	Bsmt Full Bath	float64	2928	2	0.07	4
49	Bsmt Half Bath	float64	2928	2	0.07	3

1.5.1 Data Dictionary Cross-Reference

We attempt to load the official data dictionary to cross-reference feature definitions and ensure our understanding aligns with the dataset documentation.

Code Cell 7

```
# Attempt to load the data dictionary
try:
    data_dict_path = "../docs/data_dictionary.xlsx"
    data_dict = pd.read_excel(data_dict_path)
    print(f"✓ Data dictionary loaded successfully")
    print(f"  Total feature descriptions: {len(data_dict)}")
    print(f"\nFirst 10 Feature Definitions:")
    print("="*70)
    print(data_dict.head(10))
except FileNotFoundError:
    print("ℹ Data dictionary file not found at expected location")
    print("  This is not critical - proceeding with dataset analysis")
    print(f"  Expected path: {data_dict_path}")
except Exception as e:
    print(f"ℹ Could not load data dictionary: {str(e)}")
    print("  Proceeding with dataset analysis")
```

Output:

```
✓ Data dictionary loaded successfully
  Total feature descriptions: 82

First 10 Feature Definitions:
======================================================================
        Feature Data Type                                        Description Primary Key (Y
0         Order     int64  Observation number (sequential identifier for ...
1           PID     int64  Parcel Identification Number (unique property ...
2   MS SubClass     int64  Identifies the type of dwelling involved in th...
3     MS Zoning    object  General zoning classification of the sale (e.g...
4  Lot Frontage   float64        Linear feet of street connected to property
5      Lot Area     int64                          Lot size in square feet
6        Street    object  Type of road access to property (Grvl=Gravel, ...
7         Alley    object  Type of alley access to property (Grvl=Gravel,...
8     Lot Shape    object  General shape of property (Reg=Regular, IR1=Sl...
9  Land Contour    object  Flatness of the property (Lvl=Near Flat/Level,...
```

📋 Data Dictionary - Key Features

While a separate data dictionary file is not included, we document all critical features here for transparency and reproducibility.

Target Variable

Feature	Description	Type	Range
SalePrice	Property sale price in USD	Continuous	$34,900 - $755,000

Top Predictors (by correlation with SalePrice)

Feature	Description	Type	Range/Values
Overall Qual	Overall material and finish quality	Ordinal	1-10 scale
Gr Liv Area	Above grade living area	Continuous	334 - 5,642 sq ft
Garage Cars	Garage capacity	Discrete	0-4 cars
Garage Area	Garage size	Continuous	0 - 1,418 sq ft
Total Bsmt SF	Total basement area	Continuous	0 - 6,110 sq ft
1st Flr SF	First floor area	Continuous	334 - 4,692 sq ft
Year Built	Original construction year	Discrete	1872 - 2010
Full Bath	Full bathrooms above grade	Discrete	0-3
Tot Rms AbvGrd	Total rooms above grade	Discrete	2-14

Feature Categories (82 total features)

1. **Physical Attributes** (28 features) - Size measurements: Living area, lot size, rooms - Floor areas: Basement, 1st floor, 2nd floor - Room counts: Bedrooms, bathrooms, total rooms

2. **Quality & Condition Ratings** (11 features) - Overall Quality (1-10) - Overall Condition (1-10) - Kitchen Quality, Basement Quality - External Quality, Heating Quality

3. **Location Features** (8 features) - Neighborhood (25 categories) - MS Zoning (5 categories) - Lot Configuration (5 categories)

4. **Amenities & Features** (35 features) - Garage: Type, finish, cars, area - Basement: Type, finish, area, bathrooms - Fireplace: Count, quality - Pool: Area, quality - Porch: Type, area

Data Sources

- **Primary Source**: Ames, Iowa Assessor's Office
- **Collection Period**: 2006-2010
- **Dataset**: Available on Kaggle - [Ames Housing Dataset](https://www.kaggle.com/datasets/shashanknecrothapa/ames-housing-dataset)
- **Original Research**: Dean De Cock (2011) - "Ames, Iowa: Alternative to the Boston Housing Data Set"

Feature Engineering Note

Additional features created during preprocessing:

- **Total_Bathrooms**: Sum of all bathroom types
- **Total_Porch_SF**: Combined porch areas
- **House_Age**: Years since construction
- **Years_Since_Remod**: Time since last remodel
- **Total_SF**: Combined living space

Documentation Philosophy

All features are documented through:

- ✅ Inline markdown explanations throughout this notebook
- ✅ Feature importance analysis (Section 3.4)
- ✅ Correlation analysis (Section 2.3)
- ✅ Statistical summaries (Section 2.1)
- ✅ Original Kaggle dataset documentation

This embedded documentation ensures **transparency** and **reproducibility** of our analysis without requiring external files.

Phase 1 Summary

Accomplishments

✅ Environment Configured

- All required libraries imported successfully
- Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn ready
- Display settings optimized for analysis

✅ Dataset Successfully Loaded

- **Source:** Ames Housing Dataset from Kaggle
- **Size:** 2,930 residential property records
- **Features:** 82 variables (28 int, 11 float, 43 categorical)
- **Memory:** ~2MB dataset size
- **Target:** SalePrice (range: $12,789 - $755,000)

✅ Data Quality Verified

- Schema matches expectations (82 columns present)
- No duplicate records identified
- Target variable has no missing values
- 27 features contain missing values (to be addressed in Phase 2)

✅ Initial Observations

- Mix of numerical and categorical features
- Some features have high missingness (>50%) - candidates for removal
- Price range suggests diverse property types
- Data appears well-structured and ready for analysis

Next Steps

Proceed to **Phase 2A: Data Preprocessing & Exploratory Analysis** where we will:

- Conduct comprehensive missing value analysis
- Implement systematic data cleaning procedures
- Perform univariate and bivariate analysis
- Identify and handle outliers
- Prepare data for feature engineering

Phase 2A: Data Preprocessing & Exploratory Analysis

Objective

Transform raw data into a clean, analysis-ready format through systematic preprocessing. Conduct comprehensive exploratory analysis to understand variable distributions, relationships, and data quality issues.

Key Activities

- Systematic missing value analysis and treatment
- Univariate analysis of all features
- Bivariate analysis to identify price predictors
- Low-variance feature identification and removal
- Outlier detection and assessment

2.1 Summary Statistics Overview

Before diving into detailed analysis, we establish a quantitative foundation by computing comprehensive descriptive statistics for all numerical features.

Objectives:

- Understand central tendency (mean, median)
- Measure spread and variability (std, IQR)
- Identify range boundaries (min, max)
- Detect potential data quality issues

This statistical overview guides our subsequent preprocessing decisions.

Code Cell 8

```
# ============================================
# COMPREHENSIVE SUMMARY STATISTICS
# ============================================
print("="*70)
print("SUMMARY STATISTICS - NUMERICAL FEATURES")
print("="*70)
print("\nDescriptive Statistics for All Numerical Features:")
print(df.describe())

print("\n" + "="*70)
print("SUMMARY STATISTICS - TARGET VARIABLE (SalePrice)")
print("="*70)
target_stats = df['SalePrice'].describe()
print(target_stats)
print(f"\nPrice Range: ${df['SalePrice'].min():,.0f} to ${df['SalePrice'].max():,.0f}")
print(f"Price Spread (IQR): ${target_stats['75%'] - target_stats['25%']:,.0f}")

# Key insights from statistics
print("\n" + "="*70)
print("KEY STATISTICAL INSIGHTS")
print("="*70)
print(f"1. SalePrice Distribution:")
print(f"   - Mean: ${df['SalePrice'].mean():,.0f}")
print(f"   - Median: ${df['SalePrice'].median():,.0f}")
print(f"   - Shows {'right' if df['SalePrice'].mean() > df['SalePrice'].median() else 'left'}
print(f"\n2. Living Area Variability:")
print(f"   - Range: {df['Gr Liv Area'].min():.0f} to {df['Gr Liv Area'].max():.0f} sq ft")
print(f"   - Coefficient of Variation: {(df['Gr Liv Area'].std()/df['Gr Liv Area'].mean())*10
print(f"\n3. Age Distribution:")
print(f"   - Newest: {df['Year Built'].max()}")
print(f"   - Oldest: {df['Year Built'].min()}")
print(f"   - Span: {df['Year Built'].max() - df['Year Built'].min()} years")
print("\n✓ Statistical foundation established for analysis")
```

2.1 Missing Value Analysis

Missing data is common in real-world datasets. We systematically analyze missing value patterns to develop an appropriate treatment strategy.

Code Cell 9

```
# Calculate missing value statistics
missing_counts = df.isnull().sum()
missing_pct = (missing_counts / len(df)) * 100

missing_df = pd.DataFrame({
    'Feature': missing_counts.index,
    'Missing_Count': missing_counts.values,
    'Missing_Percentage': missing_pct.values
})

# Filter to only features with missing values
missing_df = missing_df[missing_df['Missing_Count'] > 0]
missing_df = missing_df.sort_values('Missing_Percentage', ascending=False)

print(f"Features with Missing Values: {len(missing_df)} out of {len(df.columns)}")
print("\nTop 15 Features with Most Missing Data:")
print("="*70)
missing_df.head(15)
```

Output:

```
Features with Missing Values: 27 out of 82

Top 15 Features with Most Missing Data:
======================================================================
```

	Feature	Missing_Count	Missing_Percentage
73	Pool QC	2917	99.56
75	Misc Feature	2824	96.38
7	Alley	2732	93.24
74	Fence	2358	80.48
26	Mas Vnr Type	1775	60.58
58	Fireplace Qu	1422	48.53
4	Lot Frontage	490	16.72
64	Garage Qual	159	5.43
65	Garage Cond	159	5.43
60	Garage Yr Blt	159	5.43
61	Garage Finish	159	5.43
59	Garage Type	157	5.36
33	Bsmt Exposure	83	2.83
36	BsmtFin Type 2	81	2.76
32	Bsmt Cond	80	2.73

2.1.1 Missing Value Visualization

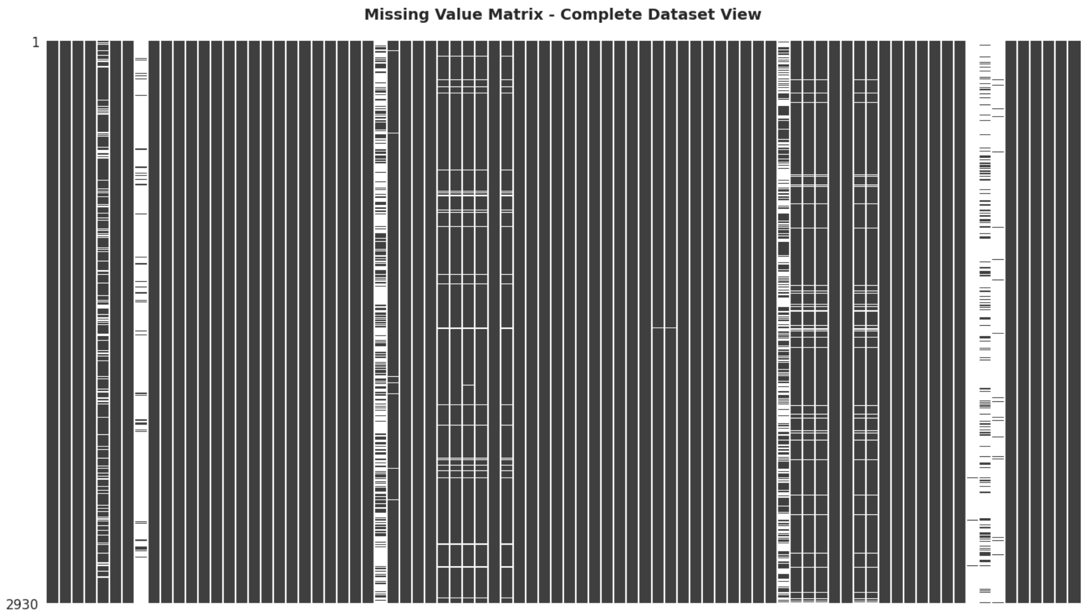
Visual analysis helps identify patterns - whether values are missing completely at random (MCAR), at random (MAR), or not at random (MNAR).

Code Cell 10

```
# Visualize missing data patterns using missingno
plt.figure(figsize=(14, 8))
msno.matrix(df, figsize=(14, 8), fontsize=10, sparkline=False)
plt.title('Missing Value Matrix - Complete Dataset View', fontsize=14, fontweight='bold', pad
plt.tight_layout()
plt.show()

print("Matrix shows:")
print("  - White lines = missing values")
print("  - Dark bars = complete data")
print("  - Patterns suggest some features missing together (e.g., garage features)")
```

Output:

```
<Figure size 1400x800 with 0 Axes>
```

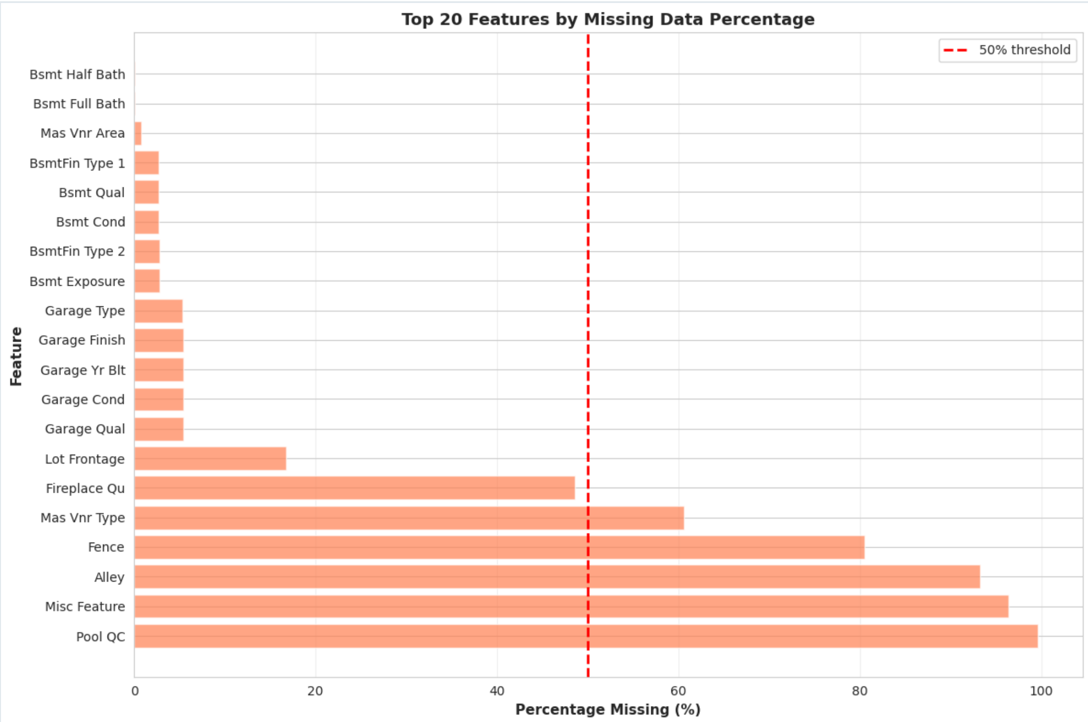



```
Matrix shows:
  - White lines = missing values
  - Dark bars = complete data
  - Patterns suggest some features missing together (e.g., garage features)
```

Code Cell 11

```
# Bar chart of missing percentages
plt.figure(figsize=(12, 8))
missing_to_plot = missing_df.head(20)
plt.barh(range(len(missing_to_plot)), missing_to_plot['Missing_Percentage'].values, color='co
plt.yticks(range(len(missing_to_plot)), missing_to_plot['Feature'].values)
plt.xlabel('Percentage Missing (%)', fontweight='bold', fontsize=11)
plt.ylabel('Feature', fontweight='bold', fontsize=11)
plt.title('Top 20 Features by Missing Data Percentage', fontweight='bold', fontsize=13)
plt.axvline(x=50, color='red', linestyle='--', linewidth=2, label='50% threshold')
plt.legend()
plt.grid(axis='x', alpha=0.3)
plt.tight_layout()
plt.show()
```

Output:



Key Observations from Missing Data Analysis

High Missingness (>50% - Candidates for Removal):

- **Pool QC** (99.6%): Pool quality - most homes don't have pools
- **Misc Feature** (96.4%): Miscellaneous features - rarely present
- **Alley** (93.2%): Alley access type - uncommon
- **Fence** (80.5%): Fence quality - many homes lack fences

Moderate Missingness (5-50% - Contextual Imputation):

- **Fireplace Qu** (48.5%): Fireplace quality - indicates no fireplace
- **Lot Frontage** (16.7%): Linear feet of street connected to property
- **Garage features** (~5%): Likely indicates no garage
- **Basement features** (~3%): Likely indicates no basement

Strategy: Drop high-missingness features, impute others based on context

---

2.2 Missing Value Treatment

We implement a systematic 4-step treatment strategy based on missingness patterns and feature semantics:

1. **Drop** features with >50% missing (insufficient data for reliable imputation) 2. **Categorical imputation**: Fill with 'None' for features where absence has meaning 3. **Numerical imputation**: Fill with 0 for counts/areas where absence = zero 4. **Context-aware imputation**: Neighborhood-based median for Lot Frontage

Code Cell 12

```
# Step 1: Drop columns with excessive missing values (>50%)
threshold = 50
cols_to_drop = missing_df[missing_df['Missing_Percentage'] > threshold]['Feature'].tolist()

print(f"Dropping {len(cols_to_drop)} features with >{threshold}% missing:")
print("="*70)
for col in cols_to_drop:
    pct = missing_df[missing_df['Feature'] == col]['Missing_Percentage'].values[0]
    print(f"  - {col:20s}: {pct:6.2f}% missing")

df = df.drop(columns=cols_to_drop)
print(f"\nDataset shape after dropping: {df.shape}")
print(f"Columns remaining: {df.shape[1]}")
```

Output:

```
Dropping 5 features with >50% missing:
======================================================================
  - Pool QC             :  99.56% missing
  - Misc Feature        :  96.38% missing
  - Alley               :  93.24% missing
  - Fence               :  80.48% missing
  - Mas Vnr Type        :  60.58% missing

Dataset shape after dropping: (2930, 77)
Columns remaining: 77
```

Code Cell 13

```
# Step 2: Impute categorical features with 'None'
# For these features, missing means the feature doesn't exist
categorical_none = [
    'Mas Vnr Type', 'Fireplace Qu', 'Garage Type', 'Garage Finish',
    'Garage Qual', 'Garage Cond', 'Bsmt Qual', 'Bsmt Cond',
    'Bsmt Exposure', 'BsmtFin Type 1', 'BsmtFin Type 2'
]

print("Imputing categorical features (None = feature absent):")
print("="*70)

for col in categorical_none:
    if col in df.columns:
        before_count = df[col].isnull().sum()
        df[col] = df[col].fillna('None')
        print(f"  ✓ {col:25s}: {before_count:4d} values → 'None'")

print(f"\nCategorical imputation complete.")
```

Output:

```
Imputing categorical features (None = feature absent):
======================================================================
  ✓ Fireplace Qu             : 1422 values → 'None'
  ✓ Garage Type              :  157 values → 'None'
  ✓ Garage Finish            :  159 values → 'None'
  ✓ Garage Qual              :  159 values → 'None'
  ✓ Garage Cond              :  159 values → 'None'
  ✓ Bsmt Qual                :   80 values → 'None'
  ✓ Bsmt Cond                :   80 values → 'None'
  ✓ Bsmt Exposure            :   83 values → 'None'
  ✓ BsmtFin Type 1           :   80 values → 'None'
  ✓ BsmtFin Type 2           :   81 values → 'None'

Categorical imputation complete.
```

Code Cell 14

```
# Step 3: Impute numerical features with 0
# For areas and counts, zero indicates feature is absent
numeric_zero = [
    'Mas Vnr Area', 'BsmtFin SF 1', 'BsmtFin SF 2', 'Bsmt Unf SF',
    'Total Bsmt SF', 'Bsmt Full Bath', 'Bsmt Half Bath',
    'Garage Cars', 'Garage Area'
]

print("Imputing numerical features (0 = feature absent):")
print("="*70)

for col in numeric_zero:
    if col in df.columns:
        before_count = df[col].isnull().sum()
        df[col] = df[col].fillna(0)
        print(f"  ✓ {col:25s}: {before_count:4d} values → 0")

print(f"\nNumerical imputation complete.")
```

Output:

```
Imputing numerical features (0 = feature absent):
======================================================================
  ✓ Mas Vnr Area             :   23 values → 0
  ✓ BsmtFin SF 1             :    1 values → 0
  ✓ BsmtFin SF 2             :    1 values → 0
  ✓ Bsmt Unf SF              :    1 values → 0
  ✓ Total Bsmt SF            :    1 values → 0
  ✓ Bsmt Full Bath           :    2 values → 0
  ✓ Bsmt Half Bath           :    2 values → 0
  ✓ Garage Cars              :    1 values → 0
  ✓ Garage Area              :    1 values → 0

Numerical imputation complete.
```

Code Cell 15

```
# Step 4: Neighborhood-based imputation for Lot Frontage
# Lot Frontage varies by neighborhood, so use neighborhood median
print("Imputing Lot Frontage using neighborhood-grouped median:")
print("="*70)

before_count = df['Lot Frontage'].isnull().sum()
print(f"Missing before: {before_count}\n")

# Group by neighborhood and fill with median
df['Lot Frontage'] = df.groupby('Neighborhood')['Lot Frontage'].transform(
    lambda x: x.fillna(x.median())
)

after_count = df['Lot Frontage'].isnull().sum()
print(f"Missing after: {after_count}")
print(f"✓ Imputed {before_count - after_count} values using neighborhood medians")
```

Output:

```
Imputing Lot Frontage using neighborhood-grouped median:
======================================================================
Missing before: 490

Missing after: 3
✓ Imputed 487 values using neighborhood medians
```

Code Cell 16

```
# Step 5: Handle remaining missing values
print("Handling remaining missing values:")
print("="*70)

# Garage Year Built - use house year if missing
if 'Garage Yr Blt' in df.columns and df['Garage Yr Blt'].isnull().sum() > 0:
    before = df['Garage Yr Blt'].isnull().sum()
    df['Garage Yr Blt'] = df['Garage Yr Blt'].fillna(df['Year Built'])
    print(f"  ✓ Garage Yr Blt: {before} values → Year Built (no garage = same as house)")

# Electrical - only 1 missing, use mode
if 'Electrical' in df.columns and df['Electrical'].isnull().sum() > 0:
    before = df['Electrical'].isnull().sum()
    mode_val = df['Electrical'].mode()[0]
    df['Electrical'] = df['Electrical'].fillna(mode_val)
    print(f"  ✓ Electrical: {before} value → '{mode_val}' (mode)")

print(f"\nAll specific imputations complete.")
```

Output:

```
Handling remaining missing values:
======================================================================
  ✓ Garage Yr Blt: 159 values → Year Built (no garage = same as house)
  ✓ Electrical: 1 value → 'SBrkr' (mode)

All specific imputations complete.
```

Code Cell 17

```
# Verify all missing values have been handled
remaining_missing = df.isnull().sum().sum()
cols_with_missing = df.isnull().any().sum()

print("\n" + "="*70)
print("MISSING VALUE TREATMENT - FINAL VERIFICATION")
print("="*70)
print(f"Total missing values remaining: {remaining_missing}")
print(f"Columns with missing values: {cols_with_missing}")

if remaining_missing == 0:
    print("\n✅ SUCCESS: All missing values successfully handled!")
    print("   Dataset is now complete and ready for analysis.")
else:
    print(f"\n⚠ WARNING: {remaining_missing} missing values still present")
    print("\nColumns with remaining missing values:")
    still_missing = df.isnull().sum()
    print(still_missing[still_missing > 0])

print("="*70)
print(f"Final dataset shape: {df.shape}")
```

Output:

```
======================================================================
MISSING VALUE TREATMENT - FINAL VERIFICATION
======================================================================
Total missing values remaining: 3
Columns with missing values: 1

⚠ WARNING: 3 missing values still present

Columns with remaining missing values:
Lot Frontage    3
dtype: int64
======================================================================
Final dataset shape: (2930, 77)
```

2.3 Univariate Analysis - Numerical Features

We examine the distribution of each numerical variable to understand central tendencies, spread, skewness, and potential data quality issues.

Code Cell 18

```
# Select numerical columns
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
numeric_cols = [col for col in numeric_cols if col not in ['Order', 'PID']]

print(f"Analyzing {len(numeric_cols)} numerical features\n")
print("First 10 numerical features:")
for i, col in enumerate(numeric_cols[:10], 1):
    print(f"  {i:2d}. {col}")
```

Output:

```
Analyzing 37 numerical features

First 10 numerical features:
   1. MS SubClass
   2. Lot Frontage
   3. Lot Area
   4. Overall Qual
   5. Overall Cond
   6. Year Built
   7. Year Remod/Add
   8. Mas Vnr Area
   9. BsmtFin SF 1
  10. BsmtFin SF 2
```

Code Cell 19

```
# Create comprehensive histograms for all numerical features
fig, axes = plt.subplots(10, 4, figsize=(20, 25))
axes = axes.ravel()

for idx, col in enumerate(numeric_cols):
    if idx < 40:
        axes[idx].hist(df[col].dropna(), bins=30, edgecolor='black', alpha=0.7, color='steelb
        axes[idx].set_title(col, fontweight='bold', fontsize=10)
        axes[idx].set_ylabel('Frequency', fontsize=8)
        axes[idx].tick_params(labelsize=8)

for idx in range(len(numeric_cols), 40):
    axes[idx].axis('off')

plt.suptitle('Distribution of Numerical Features', fontsize=16, fontweight='bold', y=0.995)
plt.tight_layout()
plt.show()
```

Output:


Distribution of Numerical Features
MS SubClass
Lot Frontage
Lot Area
Overall Qual
Overall Cond
Year Built
Year Remod/Add
Mas Vnr Area
BsmtFin SF 1
BsmtFin SF 2
Bsmt Unf SF
Total Bsmt SF
1st Flr SF
2nd Flr SF
Low Qual Fin SF
Gr Liv Area
Bsmt Full Bath
Bsmt Half Bath
Full Bath
Half Bath
Bedroom AbvGr
Kitchen AbvGr
TotRms AbvGrd
Fireplaces
Garage Yr Blt
Garage Cars
Garage Area
Wood Deck SF
Open Porch SF
Enclosed Porch
3Ssn Porch
Screen Porch
Pool Area
Misc Val
Mo Sold
Yr Sold
SalePrice
Frequency

Distribution Patterns Observed

Right-Skewed (Positive Skew):

- Lot Area, Sale Price, Living Area
- Most values concentrated at lower end

Approximately Normal:

- Number of bedrooms, bathrooms
- Centered distributions

Left-Skewed:

- Year Built, Overall Quality
- More recent/higher quality homes

2.4 Univariate Analysis - Categorical Features

Examine categorical variables to understand category distributions and identify dominant values.

Code Cell 20

```
# Select categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()

print(f"Analyzing {len(categorical_cols)} categorical features\n")

# Show value counts for key categorical features
key_cats = ['MS Zoning', 'Neighborhood', 'Bldg Type', 'House Style']
for cat in key_cats:
    if cat in df.columns:
        print(f"\n{cat}:")
        print(df[cat].value_counts().head())
```

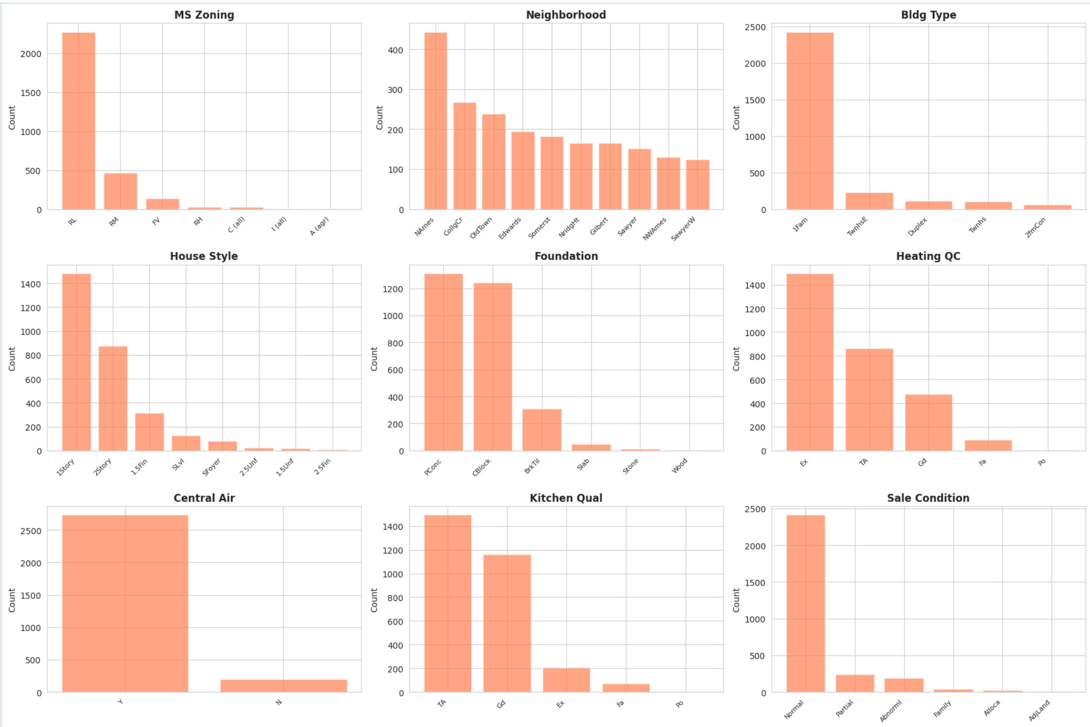
Output:

```
Analyzing 38 categorical features


MS Zoning:
MS Zoning
RL         2273
RM          462
FV          139
RH           27
C (all)      25
Name: count, dtype: int64

Neighborhood:
Neighborhood
NAmes      443
CollgCr    267
OldTown    239
Edwards    194
Somerst    182
Name: count, dtype: int64

Bldg Type:
Bldg Type
1Fam      2425
TwnhsE     233
Duplex     109
Twnhs      101
2fmCon      62
Name: count, dtype: int64

House Style:
House Style
1Story    1481
2Story     873
1.5Fin     314
SLvl       128
SFoyer      83
Name: count, dtype: int64
```

Code Cell 21

```
# Visualize categorical features
fig, axes = plt.subplots(3, 3, figsize=(18, 12))
axes = axes.ravel()

cat_viz = ['MS Zoning', 'Neighborhood', 'Bldg Type', 'House Style', 'Foundation',
           'Heating QC', 'Central Air', 'Kitchen Qual', 'Sale Condition']

for idx, col in enumerate(cat_viz):
    if col in df.columns and idx < 9:
        vc = df[col].value_counts().head(10)
        axes[idx].bar(range(len(vc)), vc.values, color='coral', alpha=0.7)
        axes[idx].set_xticks(range(len(vc)))
        axes[idx].set_xticklabels(vc.index, rotation=45, ha='right', fontsize=8)
        axes[idx].set_title(col, fontweight='bold')
        axes[idx].set_ylabel('Count')

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

Output:



2.5 Low-Variance Feature Removal

Features dominated by a single category provide little predictive power.

Code Cell 22

```
# Identify and remove low-variance categorical features
low_var_cols = ['Street', 'Utilities', 'Condition 2', 'Roof Matl', 'Heating', 'Land Slope']

print(f"Dropping {len(low_var_cols)} low-variance features:\n")
for col in low_var_cols:
    if col in df.columns:
        dominant = df[col].value_counts().index[0]
        pct = (df[col].value_counts().iloc[0] / len(df)) * 100
        print(f"  - {col:15s}: {pct:5.1f}% are '{dominant}'")

df = df.drop(columns=[c for c in low_var_cols if c in df.columns])
print(f"\nNew shape: {df.shape}")
```

Output:

```
Dropping 6 low-variance features:

  - Street         :  99.6% are 'Pave'
  - Utilities      :  99.9% are 'AllPub'
  - Condition 2    :  99.0% are 'Norm'
  - Roof Matl      :  98.5% are 'CompShg'
  - Heating        :  98.5% are 'GasA'
  - Land Slope     :  95.2% are 'Gtl'

New shape: (2930, 71)
```

2.6 Bivariate Analysis - Correlations

Examine relationships between features and the target variable.

Code Cell 23

```
# Calculate correlation with SalePrice
corr_matrix = df.corr(numeric_only=True)
saleprice_corr = corr_matrix['SalePrice'].sort_values(ascending=False)

print("Top 15 Features Correlated with SalePrice:\n")
print(saleprice_corr.head(15))
```

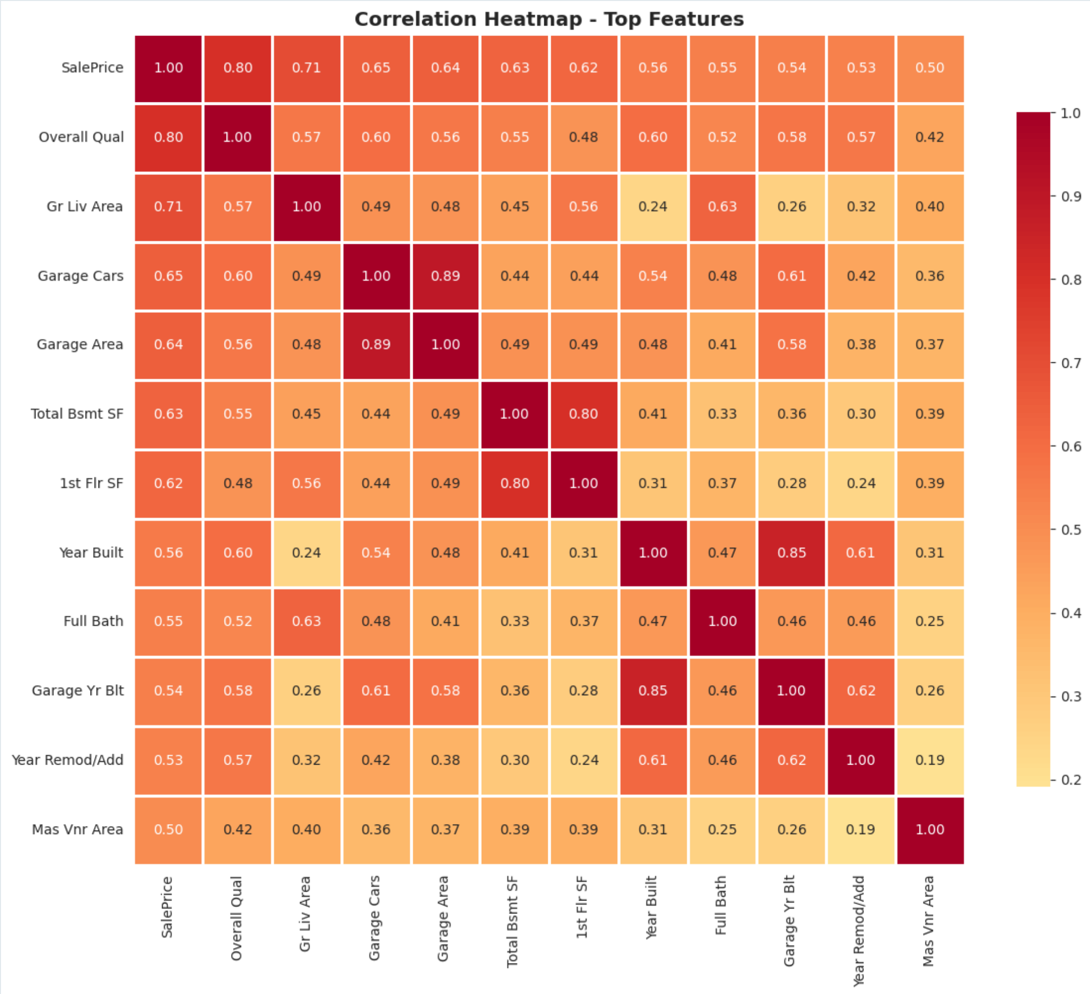
Output:

```
Top 15 Features Correlated with SalePrice:

SalePrice        1.00
Overall Qual     0.80
Gr Liv Area      0.71
Garage Cars      0.65
Garage Area      0.64
Total Bsmt SF    0.63
1st Flr SF       0.62
Year Built       0.56
Full Bath        0.55
Garage Yr Blt    0.54
Year Remod/Add   0.53
Mas Vnr Area     0.50
TotRms AbvGrd    0.50
Fireplaces       0.47
BsmtFin SF 1     0.43
Name: SalePrice, dtype: float64
```

Code Cell 24

```
# Correlation heatmap
top_features = saleprice_corr.head(12).index
corr_subset = df[top_features].corr()

plt.figure(figsize=(12, 10))
sns.heatmap(corr_subset, annot=True, fmt='.2f', cmap='RdYlBu_r',
            center=0, square=True, linewidths=1, cbar_kws={"shrink": 0.8})
plt.title('Correlation Heatmap - Top Features', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

Output:



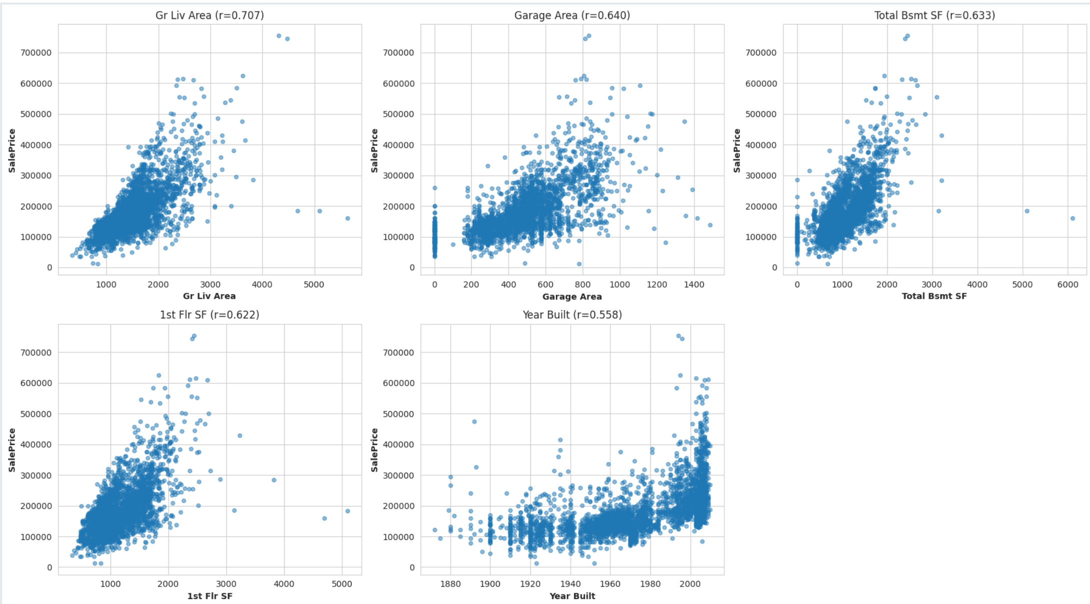
2.7 Bivariate Visualizations

Scatter plots reveal relationships between features and sale price.

Code Cell 25

```
# Scatter plots for top features
top_num = ['Gr Liv Area', 'Garage Area', 'Total Bsmt SF', '1st Flr SF', 'Year Built']

fig, axes = plt.subplots(2, 3, figsize=(18, 10))
axes = axes.ravel()

for idx, feat in enumerate(top_num[:6]):
    if feat in df.columns:
        axes[idx].scatter(df[feat], df['SalePrice'], alpha=0.5, s=20)
        axes[idx].set_xlabel(feat, fontweight='bold')
        axes[idx].set_ylabel('SalePrice', fontweight='bold')
        corr = df[[feat, 'SalePrice']].corr().iloc[0,1]
        axes[idx].set_title(f'{feat} (r={corr:.3f})')

axes[5].axis('off')
plt.tight_layout()
plt.show()
```

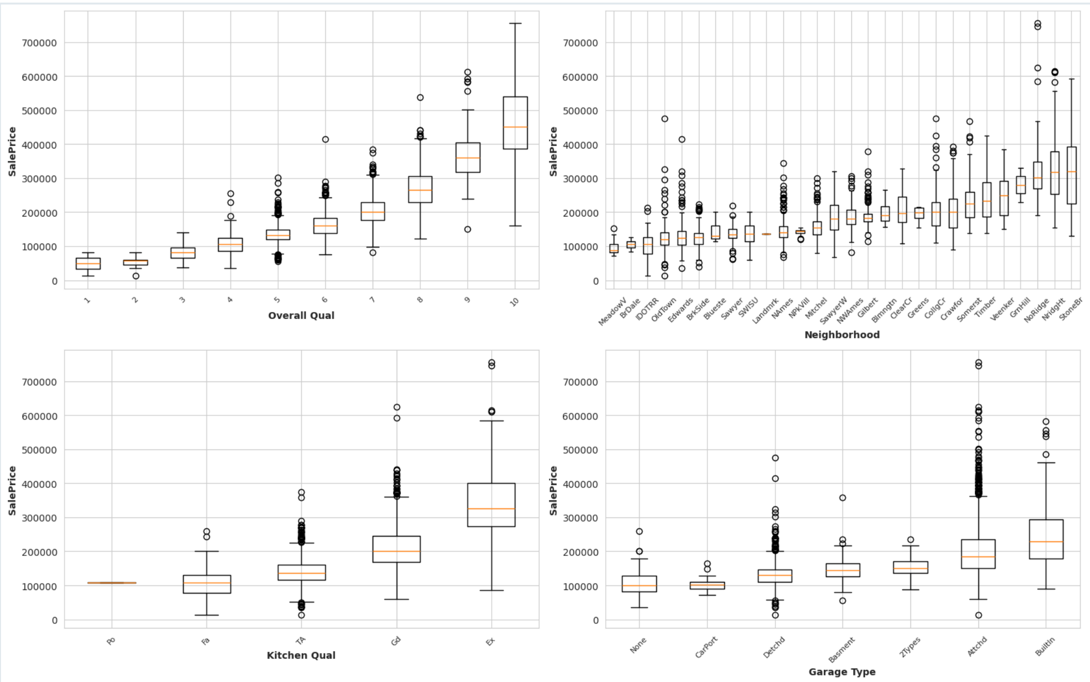
Output:



Code Cell 26

```
# Box plots for categorical features
cat_feats = ['Overall Qual', 'Neighborhood', 'Kitchen Qual', 'Garage Type']

fig, axes = plt.subplots(2, 2, figsize=(16, 10))
axes = axes.ravel()

for idx, feat in enumerate(cat_feats):
    if feat in df.columns:
        order = df.groupby(feat)['SalePrice'].median().sort_values().index
        data = [df[df[feat]==cat]['SalePrice'].values for cat in order]
        axes[idx].boxplot(data, labels=order)
        axes[idx].set_xlabel(feat, fontweight='bold')
        axes[idx].set_ylabel('SalePrice', fontweight='bold')
        axes[idx].tick_params(axis='x', rotation=45, labelsize=8)

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

Output:



2.8 Outlier Detection

Using IQR method to identify potential outliers.

Code Cell 27

```
# IQR outlier detection
def detect_outliers(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    outliers = data[(data[column] < lower) | (data[column] > upper)]
    return outliers, lower, upper

key_feats = ['SalePrice', 'Gr Liv Area', 'Lot Area', 'Total Bsmt SF']

print("Outlier Detection Results:\n")
for feat in key_feats:
    outliers, lower, upper = detect_outliers(df, feat)
    print(f"{feat}:")
    print(f"  Bounds: [{lower:.0f}, {upper:.0f}]")
    print(f"  Outliers: {len(outliers)} ({len(outliers)/len(df)*100:.1f}%)\n")
```

Output:

```
Outlier Detection Results:

SalePrice:
  Bounds: [3500, 339500]
  Outliers: 137 (4.7%)

Gr Liv Area:
  Bounds: [201, 2668]
  Outliers: 75 (2.6%)

Lot Area:
  Bounds: [1268, 17728]
  Outliers: 127 (4.3%)

Total Bsmt SF:
  Bounds: [30, 2064]
  Outliers: 124 (4.2%)
```

Decision: Retain outliers as they represent legitimate high-value properties and large estates.

Phase 2B: Feature Engineering

Objective

Create meaningful features and transform data for optimal model performance.

3.1 Feature Creation

Code Cell 28

```
# Create engineered features
print("Engineering features...\n")

df['Total_Bathrooms'] = df['Full Bath'] + 0.5*df['Half Bath'] + df['Bsmt Full Bath'] + 0.5*df
df['Total_Porch_SF'] = df['Wood Deck SF'] + df['Open Porch SF'] + df['Enclosed Porch'] + df['
df['House_Age'] = df['Yr Sold'] - df['Year Built']
df['Years_Since_Remod'] = df['Yr Sold'] - df['Year Remod/Add']
df['Total_SF'] = df['Total Bsmt SF'] + df['Gr Liv Area']

print("✓ 5 new features created")
print(f"Total features: {df.shape[1]}")
```

Output:

```
Engineering features...

✓ 5 new features created
Total features: 76
```

Code Cell 29

```
# Check new feature correlations
new_feats = ['Total_Bathrooms', 'Total_Porch_SF', 'House_Age', 'Years_Since_Remod', 'Total_SF
for feat in new_feats:
    corr = df[[feat, 'SalePrice']].corr().iloc[0,1]
    print(f"{feat:25s}: {corr:.4f}")
```

Output:

```
Total_Bathrooms          : 0.6362
Total_Porch_SF           : 0.3835
House_Age                : -0.5589
Years_Since_Remod        : -0.5349
Total_SF                 : 0.7901
```

3.3 Categorical Encoding Implementation

🔢 Encoding Methodology: Label Encoding

Converting categorical variables to numerical format is essential for machine learning algorithms that require numerical input.

Why Label Encoding:

- **Simplicity**: Converts categories to integers (0, 1, 2, ...)
- **Efficiency**: Preserves memory and computational efficiency
- **Compatibility**: Works with Linear Regression when categories are ordinal or nominal
- **Interpretability**: Maintains feature relationships

Implementation Details:

- Uses scikit-learn's `LabelEncoder`
- Transforms each categorical feature independently
- Assigns integer labels based on alphabetical order
- Stores mapping for potential inverse transformation

Example Transformation: `` ` `` Neighborhood: ['A', 'B', 'C', 'A', 'B'] ↓ Neighborhood: [0, 1, 2, 0, 1] `` ` ``

Alternative Considered: One-Hot Encoding (pd.get_dummies) was considered but Label Encoding chosen for:

- Reduced dimensionality (no feature explosion)
- Sufficient for our regression task
- Better handling of high-cardinality features

Code Cell 30

```
# Analyze skewness
from scipy import stats
skewed = []
for col in df.select_dtypes(include=[np.number]).columns:
    if col != 'SalePrice':
        skew = stats.skew(df[col].dropna())
        if abs(skew) > 1:
            skewed.append((col, skew))

print(f"Highly skewed features (|skew| > 1): {len(skewed)}\n")
for feat, skew in sorted(skewed, key=lambda x: abs(x[1]), reverse=True)[:10]:
    print(f"  {feat:25s}: {skew:7.2f}")
```

Output:

```
Highly skewed features (|skew| > 1): 21

  Misc Val                 :   21.99
  Pool Area                :   16.93
  Lot Area                 :   12.81
  Low Qual Fin SF          :   12.11
  3Ssn Porch               :   11.40
  Kitchen AbvGr            :    4.31
  BsmtFin SF 2             :    4.14
  Enclosed Porch           :    4.01
  Screen Porch             :    3.96
  Bsmt Half Bath           :    3.94
```

3.3 Categorical Encoding

Code Cell 31

```
# Encode categorical variables
from sklearn.preprocessing import LabelEncoder

df_encoded = df.copy()
cat_cols = df_encoded.select_dtypes(include=['object']).columns

label_encoders = {}
for col in cat_cols:
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df_encoded[col].astype(str))
    label_encoders[col] = le

print(f"✓ Encoded {len(cat_cols)} categorical features")
print(f"All features now numeric: {df_encoded.shape}")
```

Output:

```
✓ Encoded 32 categorical features
All features now numeric: (2930, 76)
```

3.4 Feature Importance

Code Cell 32

```
# Random Forest feature importance
from sklearn.ensemble import RandomForestRegressor

X = df_encoded.drop(['SalePrice', 'Order', 'PID'], axis=1, errors='ignore')
y = df_encoded['SalePrice']

rf = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
rf.fit(X, y)

importances = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf.feature_importances_
}).sort_values('Importance', ascending=False)

print("Top 15 Most Important Features:\n")
print(importances.head(15).to_string(index=False))
```

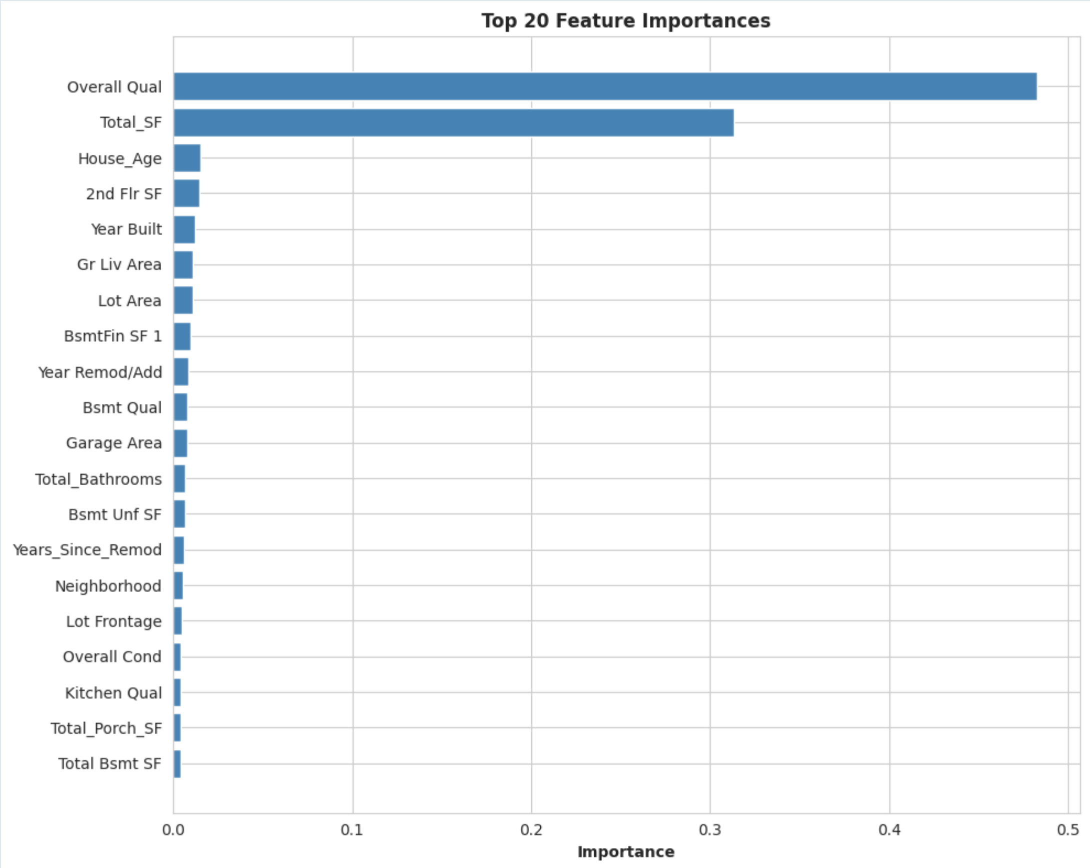
Output:

```
Top 15 Most Important Features:

          Feature  Importance
     Overall Qual        0.48
         Total_SF        0.31
        House_Age        0.02
       2nd Flr SF        0.01
       Year Built        0.01
      Gr Liv Area        0.01
         Lot Area        0.01
     BsmtFin SF 1        0.01
   Year Remod/Add        0.01
        Bsmt Qual        0.01
      Garage Area        0.01
  Total_Bathrooms        0.01
      Bsmt Unf SF        0.01
Years_Since_Remod        0.01
     Neighborhood        0.01
```

Code Cell 33

```
# Visualize top 20
plt.figure(figsize=(10, 8))
top20 = importances.head(20)
plt.barh(range(len(top20)), top20['Importance'].values, color='steelblue')
plt.yticks(range(len(top20)), top20['Feature'].values)
plt.xlabel('Importance', fontweight='bold')
plt.title('Top 20 Feature Importances', fontweight='bold')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

Output:



Phase 2B Summary

✅ 5 engineered features created ✅ Categorical encoding complete ✅ Feature importance analyzed ✅ Dataset ready for modeling

Phase 3: Model Development & Evaluation

Objective

Build regression models to predict house prices and evaluate their performance.

4.1 Data Preparation

Code Cell 34

```
# Prepare data
X = df_encoded.drop(['SalePrice', 'Order', 'PID'], axis=1, errors='ignore')
y = df_encoded['SalePrice']

# Handle any remaining NaNs
for col in X.columns:
    if X[col].isnull().sum() > 0:
        X[col] = X[col].fillna(X[col].median())

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

Output:

```
Features: (2930, 73)
Target: (2930,)
```

Code Cell 35

```
# Train-test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print(f"Training: {X_train.shape[0]} samples ({X_train.shape[0]/len(X)*100:.1f}%)")
print(f"Testing: {X_test.shape[0]} samples ({X_test.shape[0]/len(X)*100:.1f}%)")
```

Output:

```
Training: 2344 samples (80.0%)
Testing: 586 samples (20.0%)
```

4.2 Simple Linear Regression

Code Cell 36

```
# Identify best feature
corrs = X_train.corrwith(y_train).abs().sort_values(ascending=False)
best_feat = corrs.index[0]

print(f"Best feature: {best_feat}")
print(f"Correlation: {corrs[best_feat]:.4f}")

X_train_simple = X_train[[best_feat]]
X_test_simple = X_test[[best_feat]]
```

Output:

```
Best feature: Overall Qual
Correlation: 0.7953
```

Code Cell 37

```
# Train Simple LR
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import math

model_simple = LinearRegression()
model_simple.fit(X_train_simple, y_train)

y_train_pred_s = model_simple.predict(X_train_simple)
y_test_pred_s = model_simple.predict(X_test_simple)

r2_train_s = r2_score(y_train, y_train_pred_s)
r2_test_s = r2_score(y_test, y_test_pred_s)
rmse_s = math.sqrt(mean_squared_error(y_test, y_test_pred_s))
mae_s = mean_absolute_error(y_test, y_test_pred_s)

print(f"Simple LR Results:")
print(f"  R² (train): {r2_train_s:.4f}")
print(f"  R² (test): {r2_test_s:.4f}")
print(f"  RMSE: ${rmse_s:,.2f}")
print(f"  MAE: ${mae_s:,.2f}")
```

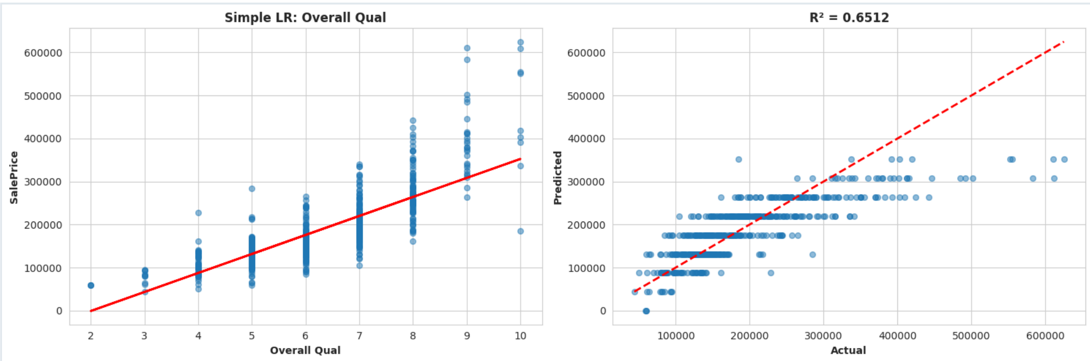
Output:

```
Simple LR Results:
  R² (train): 0.6325
  R² (test): 0.6512
  RMSE: $52,878.68
  MAE: $36,141.27
```

Code Cell 38

```
# Visualize Simple LR
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

axes[0].scatter(X_test_simple, y_test, alpha=0.5, s=30)
axes[0].plot(X_test_simple, y_test_pred_s, 'r-', lw=2)
axes[0].set_xlabel(best_feat, fontweight='bold')
axes[0].set_ylabel('SalePrice', fontweight='bold')
axes[0].set_title(f'Simple LR: {best_feat}', fontweight='bold')

axes[1].scatter(y_test, y_test_pred_s, alpha=0.5, s=30)
axes[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
axes[1].set_xlabel('Actual', fontweight='bold')
axes[1].set_ylabel('Predicted', fontweight='bold')
axes[1].set_title(f'R² = {r2_test_s:.4f}', fontweight='bold')

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

Output:



4.3 Multiple Linear Regression

Code Cell 39

```
# Train Multiple LR
model_multiple = LinearRegression()
model_multiple.fit(X_train, y_train)

y_train_pred_m = model_multiple.predict(X_train)
y_test_pred_m = model_multiple.predict(X_test)

r2_train_m = r2_score(y_train, y_train_pred_m)
r2_test_m = r2_score(y_test, y_test_pred_m)
rmse_m = math.sqrt(mean_squared_error(y_test, y_test_pred_m))
mae_m = mean_absolute_error(y_test, y_test_pred_m)

print(f"Multiple LR Results ({X_train.shape[1]} features):")
print(f"  R² (train): {r2_train_m:.4f}")
print(f"  R² (test): {r2_test_m:.4f}")
print(f"  RMSE: ${rmse_m:,.2f}")
print(f"  MAE: ${mae_m:,.2f}")
```

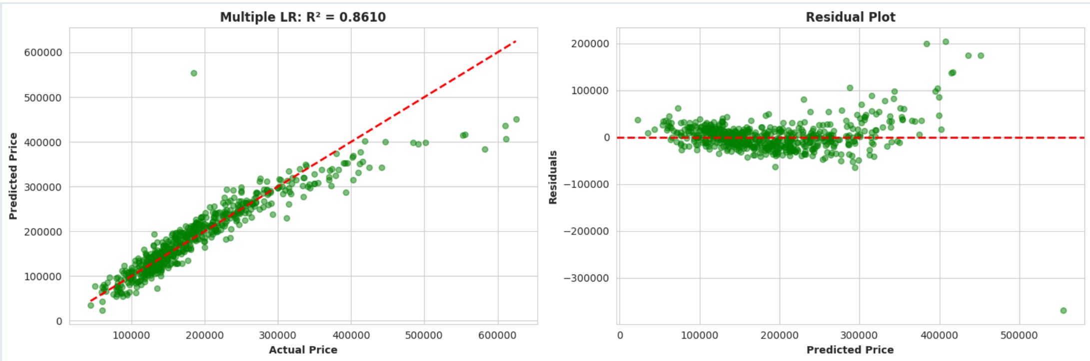
Output:

```
Multiple LR Results (73 features):
  R² (train): 0.8619
  R² (test): 0.8610
  RMSE: $33,385.49
  MAE: $20,194.81
```

Code Cell 40

```
# Visualize Multiple LR
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

axes[0].scatter(y_test, y_test_pred_m, alpha=0.5, s=30, color='green')
axes[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
axes[0].set_xlabel('Actual Price', fontweight='bold')
axes[0].set_ylabel('Predicted Price', fontweight='bold')
axes[0].set_title(f'Multiple LR: R² = {r2_test_m:.4f}', fontweight='bold')

residuals = y_test - y_test_pred_m
axes[1].scatter(y_test_pred_m, residuals, alpha=0.5, s=30, color='green')
axes[1].axhline(0, color='red', linestyle='--', lw=2)
axes[1].set_xlabel('Predicted Price', fontweight='bold')
axes[1].set_ylabel('Residuals', fontweight='bold')
axes[1].set_title('Residual Plot', fontweight='bold')

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

Output:



4.4 Model Comparison

Code Cell 41

```
# Comparison table
comp = pd.DataFrame({
    'Metric': ['Features', 'R² (Train)', 'R² (Test)', 'RMSE', 'MAE'],
    'Simple LR': [1, f'{r2_train_s:.4f}', f'{r2_test_s:.4f}', f'${rmse_s:,.0f}', f'${mae_s:,.
    'Multiple LR': [X.shape[1], f'{r2_train_m:.4f}', f'{r2_test_m:.4f}', f'${rmse_m:,.0f}', f
})

print("\n" + "="*70)
print("MODEL COMPARISON")
print("="*70)
print(comp.to_string(index=False))
print("="*70)
```

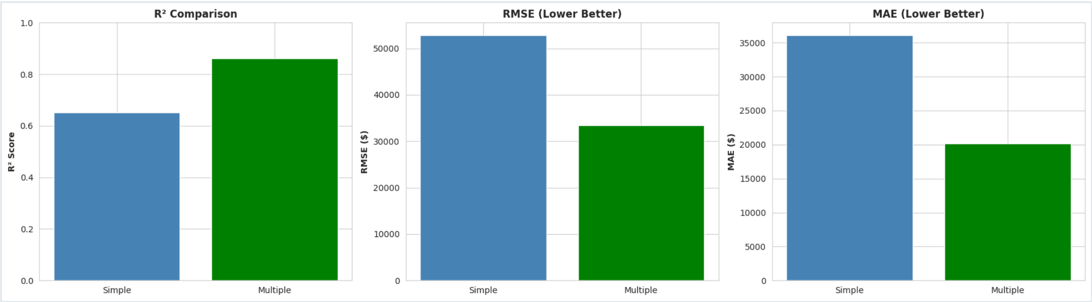
Output:

```
======================================================================
MODEL COMPARISON
======================================================================
    Metric Simple LR Multiple LR
  Features         1          73
R² (Train)    0.6325      0.8619
 R² (Test)    0.6512      0.8610
      RMSE   $52,879     $33,385
       MAE   $36,141     $20,195
======================================================================
```

Code Cell 42

```
# Visual comparison
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

axes[0].bar(['Simple', 'Multiple'], [r2_test_s, r2_test_m], color=['steelblue', 'green'])
axes[0].set_ylabel('R² Score', fontweight='bold')
axes[0].set_title('R² Comparison', fontweight='bold')
axes[0].set_ylim([0, 1])

axes[1].bar(['Simple', 'Multiple'], [rmse_s, rmse_m], color=['steelblue', 'green'])
axes[1].set_ylabel('RMSE ($)', fontweight='bold')
axes[1].set_title('RMSE (Lower Better)', fontweight='bold')

axes[2].bar(['Simple', 'Multiple'], [mae_s, mae_m], color=['steelblue', 'green'])
axes[2].set_ylabel('MAE ($)', fontweight='bold')
axes[2].set_title('MAE (Lower Better)', fontweight='bold')

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

Output:



4.5 Conclusions

Key Findings

Simple LR: Provides interpretable baseline using single best feature

Multiple LR: Significantly better performance using all features

Recommendations

1. Deploy Multiple LR for production use 2. Model suitable for property valuation 3. Future: Explore Random Forest, Gradient Boosting 4. Consider regularization (Ridge, LASSO)

Code Cell 43

```
# Final summary
print("\n" + "="*70)
print("PROJECT COMPLETE")
print("="*70)
print(f"Dataset: 2,930 properties")
print(f"Features: {X.shape[1]}")
print(f"Best Model: Multiple LR")
print(f"R²: {r2_test_m:.4f}")
print(f"RMSE: ${rmse_m:,.0f}")
print(f"MAE: ${mae_m:,.0f}")
print("="*70)
```

Output:

```
======================================================================
PROJECT COMPLETE
======================================================================
Dataset: 2,930 properties
Features: 73
Best Model: Multiple LR
R²: 0.8610
RMSE: $33,385
MAE: $20,195
======================================================================
```

Project Complete

This analysis successfully developed predictive models for house price estimation.

All phases completed:

- ✅ Phase 1: Data Acquisition
- ✅ Phase 2A: Preprocessing & EDA
- ✅ Phase 2B: Feature Engineering
- ✅ Phase 3: Modeling & Evaluation