Course: ITAI 2277 - Data Science in Artificial Intelligence

Professor: Sitaram Ayyagari

Student: Adejare Fasiku - W215804390

Term: Fall 2025 - CR: 14165

Capstone Project: House Price and Quality Prediction

```
import os

# Install desired versions, letting pip resolve common dependencies.
# Explicitly set versions for other critical libraries, but allow panda
!pip -q install --upgrade pip

!pip install --upgrade pandas numpy scikit-learn matplotlib seaborn job


# Ensure output directories are created. This was previously in cell `o
import warnings
warnings.filterwarnings("ignore")
from pathlib import Path
DATA_DIR = Path(os.getcwd())  # change if your CSVs are in another fold
OUTPUT_DIR = Path("./outputs")
PLOTS_DIR = OUTPUT_DIR / "plots"
MODELS_DIR = OUTPUT_DIR / "models"
for d in (OUTPUT_DIR, PLOTS_DIR, MODELS_DIR):
    d.mkdir(parents=True, exist_ok=True)
```

```
                                        1.8/1.8 MB 31.4 MB/s eta 0
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist
Collecting pandas
```

```
  Downloading pandas-2.3.3-cp312-cp312-manylinux_2_24_x86_64.manylinux_
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-
Collecting numpy
  Downloading numpy-2.3.5-cp312-cp312-manylinux_2_27_x86_64.manylinux_2
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.1
Collecting scikit-learn
  Downloading scikit_learn-1.7.2-cp312-cp312-manylinux2014_x86_64.manyl
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/
Collecting matplotlib
  Downloading matplotlib-3.10.7-cp312-cp312-manylinux2014_x86_64.manyli
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dis
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.12/
Collecting ipywidgets
  Downloading ipywidgets-8.1.8-py3-none-any.whl.metadata (2.4 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.1
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3
Requirement already satisfied: scipy>=1.8.0 in /usr/local/lib/python3.1
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/p
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/pytho
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/pyth
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/pyth
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/d
Requirement already satisfied: pyparsing>=3 in /usr/local/lib/python3.1
Collecting comm>=0.1.3 (from ipywidgets)
  Downloading comm-0.2.3-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: ipython>=6.1.0 in /usr/local/lib/python3
Requirement already satisfied: traitlets>=4.3.1 in /usr/local/lib/pytho
Collecting widgetsnbextension~=4.0.14 (from ipywidgets)
  Downloading widgetsnbextension-4.0.15-py3-none-any.whl.metadata (1.6
Requirement already satisfied: jupyterlab_widgets~=3.0.15 in /usr/local
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/pytho
Collecting jedi>=0.16 (from ipython>=6.1.0->ipywidgets)
  Downloading jedi-0.19.2-py2.py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: decorator in /usr/local/lib/python3.12/d
Requirement already satisfied: pickleshare in /usr/local/lib/python3.12
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2
Requirement already satisfied: pygments in /usr/local/lib/python3.12/di
Requirement already satisfied: backcall in /usr/local/lib/python3.12/di
Requirement already satisfied: matplotlib-inline in /usr/local/lib/pyth
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.12
Requirement already satisfied: wcwidth in /usr/local/lib/python3.12/dis
Requirement already satisfied: parso<0.9.0,>=0.8.4 in /usr/local/lib/py
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/di
Downloading pandas-2.3.3-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 33.3 MB/s  0:(
```

```
Downloading numpy-2.3.5-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_2
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.6/16.6 MB 83.0 MB/s  0:0
Downloading scikit_learn-1.7.2-cp312-cp312-manylinux2014_x86_64.manylin
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.5/9.5 MB 36.5 MB/s  0:00
Downloading matplotlib-3.10.7-cp312-cp312-manylinux2014_x86_64.manylinu
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.7/8.7 MB 19.3 MB/s  0:00
Downloading ipywidgets-8.1.8-py3-none-any.whl (139 kB)
Downloading widgetsnbextension-4.0.15-py3-none-any.whl (2.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 45.5 MB/s  0:00
Downloading comm-0.2.3-py3-none-any.whl (7.3 kB)
Downloading jedi-0.19.2-py2.py3-none-any.whl (1.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 30.1 MB/s  0:00
Installing collected packages: widgetsnbextension, numpy, jedi, comm, p
  Attempting uninstall: widgetsnbextension
    Found existing installation: widgetsnbextension 3.6.10
    Uninstalling widgetsnbextension-3.6.10:
      Successfully uninstalled widgetsnbextension-3.6.10
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
  Attempting uninstall: pandas
    Found existing installation: pandas 2.2.2
    Uninstalling pandas-2.2.2:
      Successfully uninstalled pandas-2.2.2
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.6.1
    Uninstalling scikit-learn-1.6.1:
      Successfully uninstalled scikit-learn-1.6.1
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.10.0
    Uninstalling matplotlib-3.10.0:
      Successfully uninstalled matplotlib-3.10.0
  Attempting uninstall: ipywidgets
    Found existing installation: ipywidgets 7.7.1
    Uninstalling ipywidgets-7.7.1:
      Successfully uninstalled ipywidgets-7.7.1
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8/8 [ipywidgets]
ERROR: pip's dependency resolver does not currently take into account a
google-colab 1.0.0 requires pandas==2.2.2, but you have pandas 2.3.3 wh
opencv-contrib-python 4.12.0.88 requires numpy<2.3.0,>=2; python_versio
tensorflow 2.19.0 requires numpy<2.2.0,>=1.26.0, but you have numpy 2.3
opencv-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.3.5 which
opencv-python-headless 4.12.0.88 requires numpy<2.3.0,>=2; python_versi
Successfully installed comm-0.2.3 ipywidgets-8.1.8 jedi-0.19.2 matplotl
WARNING: The following packages were previously imported in this
runtime:
  [matplotlib,mpl_toolkits,numpy]
You must restart the runtime in order to use newly installed versions.
```

RESTART SESSION

```
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import datetime

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer, TransformedTargetRegres
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler, Funct
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score

# Display settings for DataFrame(df)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 50)



#########
# In this step we import all required libraries and set up global patl

import os
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
import joblib
from datetime import datetime

# sklearn
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
```

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import accuracy_score, f1_score, confusion_matri

# Models
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingR
from sklearn.ensemble import RandomForestClassifier, GradientBoosting

# Create output directories
DATA_DIR = Path(os.getcwd())  # change if your CSVs are in another fo
OUTPUT_DIR = Path("./outputs")
PLOTS_DIR = OUTPUT_DIR / "plots"
MODELS_DIR = OUTPUT_DIR / "models"
for d in (OUTPUT_DIR, PLOTS_DIR, MODELS_DIR):
    d.mkdir(parents=True, exist_ok=True)
```

🚀 Step 1 - Load dataframe(df), Dataframe exploration, Clean dataframe

```
# --- Step 1: Load the data ---
# We load train.csv and test.csv. If test.csv isn't used for model eva
# we'll still load it to show how to prepare a submission later if nee

print("Loading data...")
TRAIN_PATH = DATA_DIR / "train.csv"
TEST_PATH  = DATA_DIR / "test.csv"

if not TRAIN_PATH.exists():
    raise FileNotFoundError(f"train.csv not found at {TRAIN_PATH}. Pla

train = pd.read_csv(TRAIN_PATH)
print(f"Train shape: {train.shape}")

if TEST_PATH.exists():
    test = pd.read_csv(TEST_PATH)
    print(f"Test shape: {test.shape}")
else:
    test = None
    print("No test.csv found at path; continuing without it.")
```

```
Loading data...
Train shape: (1460, 81)
Test shape: (1459, 80)
```

🚀 Step 2 - Initial EDA and description

Show basic info, head, and target distribution. Save key plots.

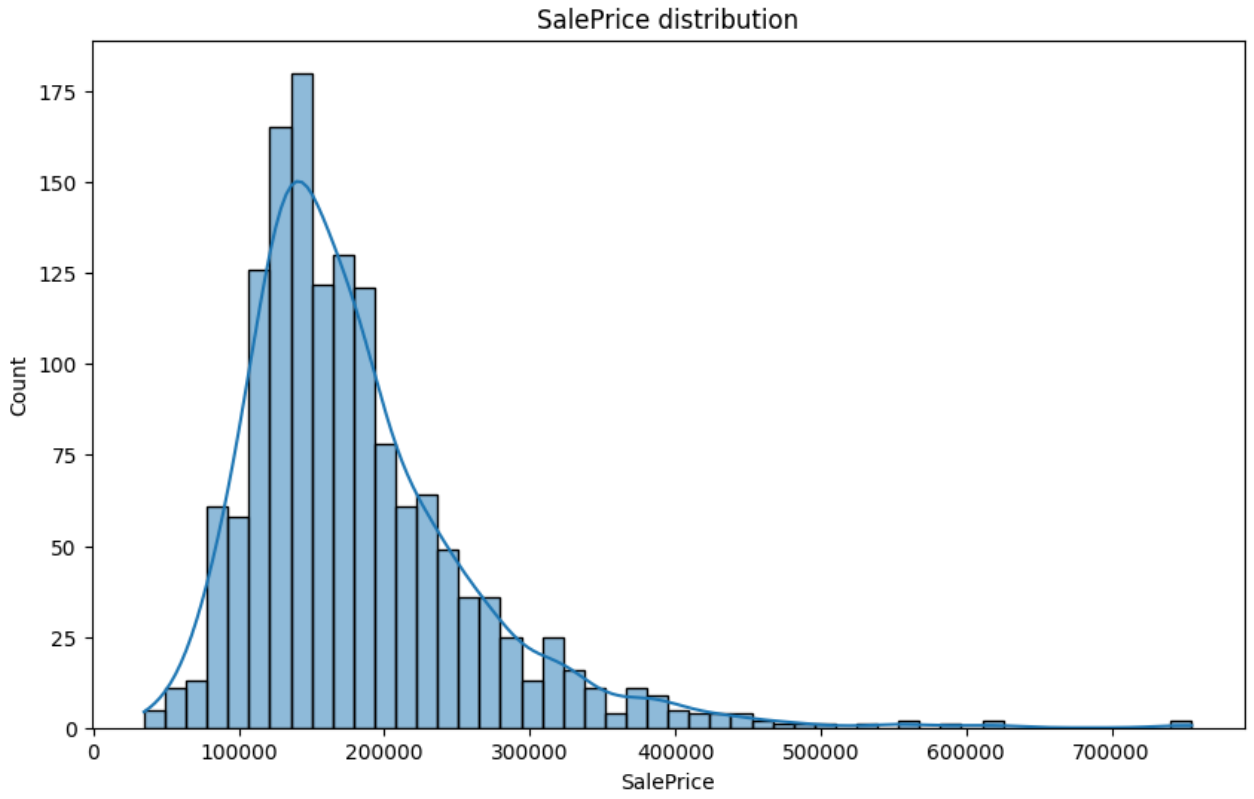
```
# --- Step 2: Initial EDA and description ---
# Show basic info, head, and target distribution. Save key plots.
def quick_eda(df):
    print("\n=== Quick EDA ===")
    display_cols = ['Id', 'SalePrice'] if 'SalePrice' in df.columns el
    print(df[display_cols].head())
    print("\nData types:")
    print(df.dtypes.value_counts())
    print("\nMissing values (top 20):")
    print(df.isnull().sum().sort_values(ascending=False).head(20))
    print("\nBasic stats for numeric features:")
    print(df.select_dtypes(include=[np.number]).describe().T[['count',
```

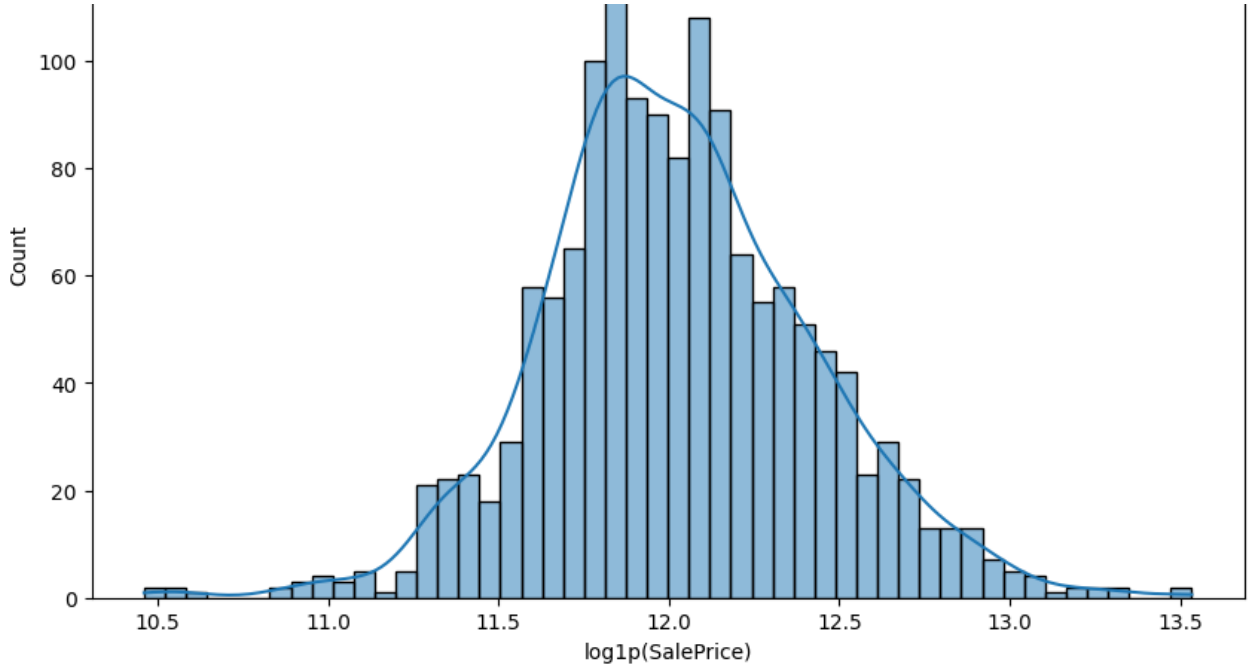
```
# run quick EDA
quick_eda(train)

# Plot SalePrice distribution
plt.figure(figsize=(10,6))
sns.histplot(train['SalePrice'], kde=True, bins=50)
plt.title('SalePrice distribution')
plt.xlabel('SalePrice')
plt.savefig(PLOTS_DIR / "saleprice_distribution.png", bbox_inches='ti
plt.show()


# SalePrice is usually right-skewed; log-transform can help for regres
plt.figure(figsize=(10,6))
sns.histplot(np.log1p(train['SalePrice']), kde=True, bins=50)
plt.title('Log-transformed SalePrice distribution')
plt.xlabel('log1p(SalePrice)')
plt.savefig(PLOTS_DIR / "saleprice_log_distribution.png", bbox_inches=
plt.show()

df = train.copy()

###############

print("First 5 rows of Dataset:")
display(df.head()) # default 5

# 1.3 Quick Summary of the dataset Columns
print("\nDataframe Columns:")
display(df.info())

# 1.4 Describe numeric columns, Min, 25%, 50%, 75%, Max values
print("\nDescription of Numeric columns:")
print(df.describe())

# 1.5 Look for missing values and duplicates
missing_counts = df.isna().sum()
duplicate_counts = df.duplicated().sum()
print("\nMissing values per column:")
print(missing_counts)
print("\nDuplicate values in Dataframe:")
print(f"{duplicate_counts}  Duplicate(s)\n")

# 1.6 Optional: Visual exploration
```

```
#       - Histograms or boxplots for numeric columns
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()

# You might also want to check if these columns are actually numeric (
# For safety, let's convert them if they are stored as strings.
for col in numeric_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce')

plt.figure(figsize=(14, 6))
df[numeric_cols].hist(bins=30, figsize=(14, 6))
plt.tight_layout()
plt.show()

# 1.7 Optional: Correlation matrix for numeric columns
corr = df[numeric_cols].corr()
plt.figure(figsize=(10, 8)) # Makes the heatmap figure 10 inches by 8
sns.heatmap(corr, annot=True, cmap='coolwarm')
# annot: shows correlation values inside the heatmap cells.
# cmap='coolwarm': Uses a red-to-blue colormap for visual distinction
plt.title("Correlation Heatmap of Numeric Features")
plt.show()
```

```
=== Quick EDA ===
   Id  SalePrice
0   1     208500
1   2     181500
2   3     223500
3   4     140000
4   5     250000

Data types:
object     43
int64      35
float64     3
Name: count, dtype: int64

Missing values (top 20):
PoolQC          1453
MiscFeature     1406
Alley           1369
Fence           1179
MasVnrType       872
FireplaceQu      690
LotFrontage      259
GarageQual        81
GarageFinish      81
GarageType        81
```

```
GarageYrBlt       81
GarageCond        81
BsmtFinType2      38
BsmtExposure      38
BsmtCond          37
BsmtQual          37
BsmtFinType1      37
MasVnrArea         8
Electrical         1
Condition2         0
dtype: int64

Basic stats for numeric features:
                count          mean           std     min     50%         m
Id             1460.0    730.500000    421.610009     1.0   730.5     146
MSSubClass     1460.0     56.897260     42.300571    20.0    50.0      19
LotFrontage    1201.0     70.049958     24.284752    21.0    69.0      31
LotArea        1460.0  10516.828082   9981.264932  1300.0  9478.5  21524
OverallQual    1460.0      6.099315      1.382997     1.0     6.0       1
OverallCond    1460.0      5.575342      1.112799     1.0     5.0       
YearBuilt      1460.0   1971.267808     30.202904  1872.0  1973.0     201
YearRemodAdd   1460.0   1984.865753     20.645407  1950.0  1994.0     201
MasVnrArea     1452.0    103.685262    181.066207     0.0     0.0     160
BsmtFinSF1     1460.0    443.639726    456.098091     0.0   383.5     564
```





First 5 rows of Dataset:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShap
0	1	60	RL	65.0	8450	Pave	NaN	R
1	2	20	RL	80.0	9600	Pave	NaN	R
2	3	60	RL	68.0	11250	Pave	NaN	IF
3	4	70	RL	60.0	9550	Pave	NaN	IF
4	5	60	RL	84.0	14260	Pave	NaN	IF

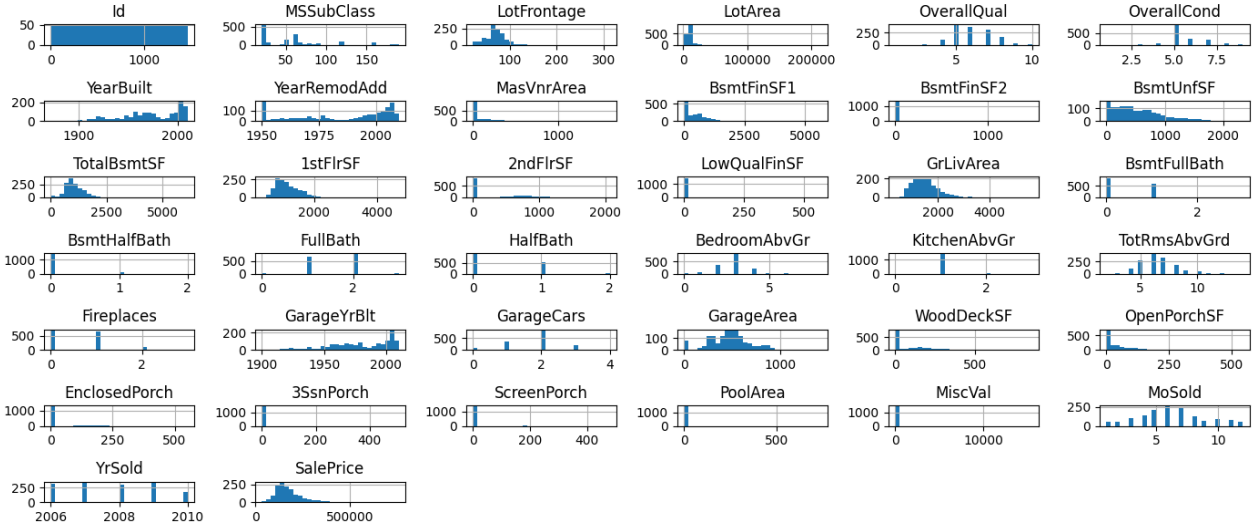
```
Dataframe Columns:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
```

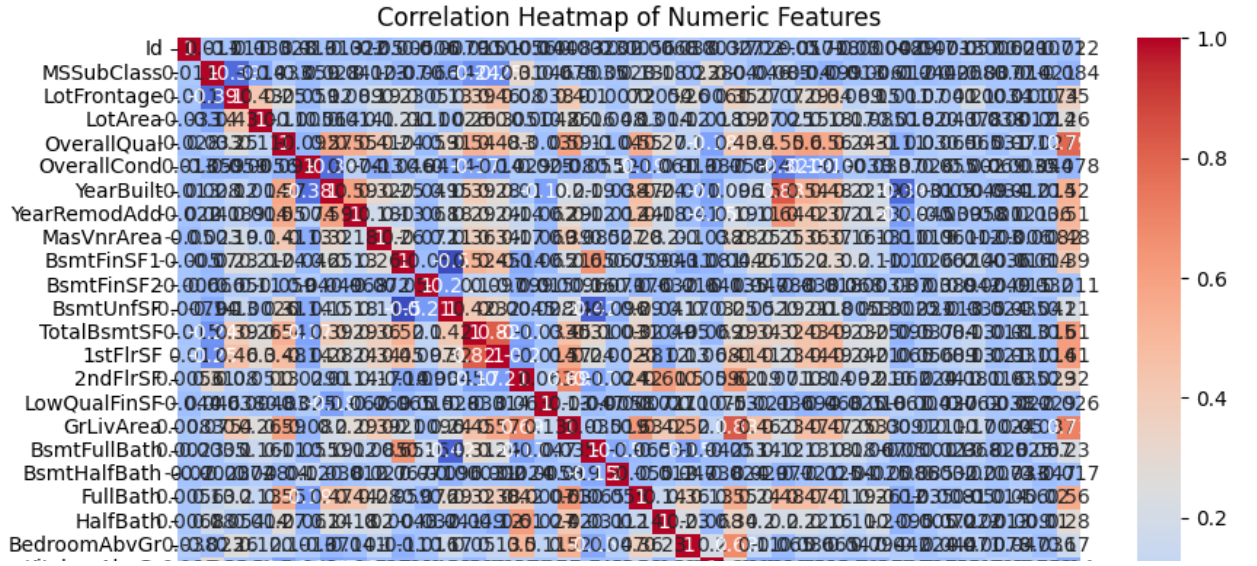
```
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     588 non-null    object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int64
 37  BsmtUnfSF      1460 non-null   int64
 38  TotalBsmtSF    1460 non-null   int64
 39  Heating        1460 non-null   object
 40  HeatingQC      1460 non-null   object
 41  CentralAir     1460 non-null   object
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1460 non-null   int64
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
```

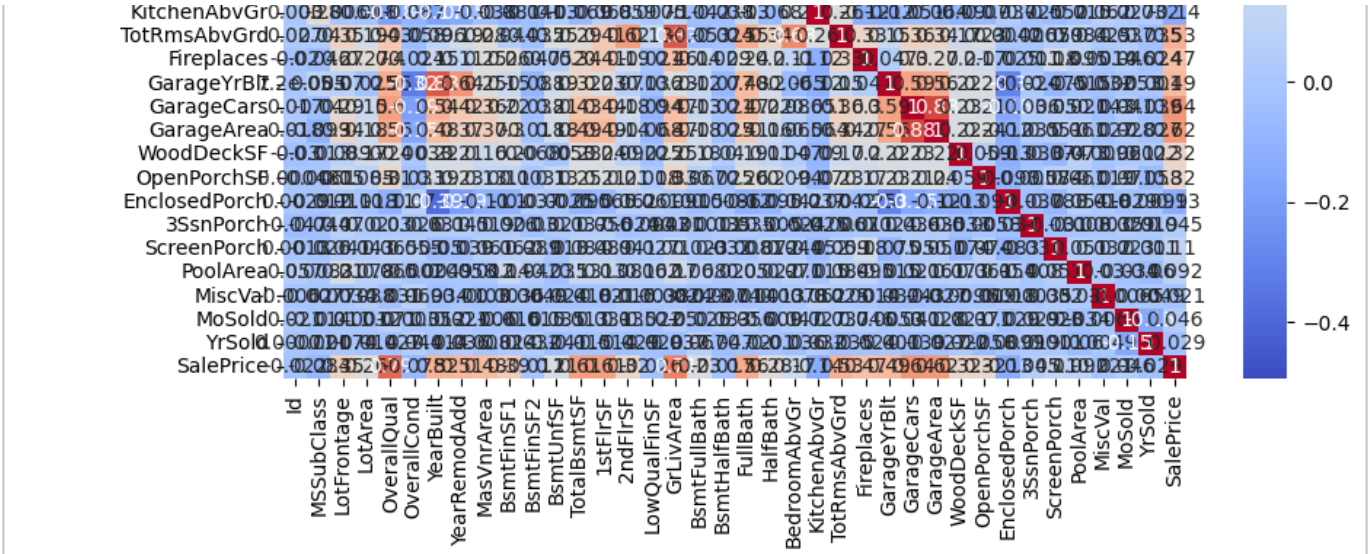
```
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
None

Description of Numeric columns:
                Id   MSSubClass  LotFrontage        LotArea  OverallQua
count  1460.000000  1460.000000  1201.000000    1460.000000  1460.0000(
mean    730.500000    56.897260    70.049958   10516.828082     6.0993
std     421.610009    42.300571    24.284752    9981.264932     1.3829
min       1.000000    20.000000    21.000000    1300.000000     1.0000(
25%     365.750000    20.000000    59.000000    7553.500000     5.0000(
50%     730.500000    50.000000    69.000000    9478.500000     6.0000(
75%    1095.250000    70.000000    80.000000   11601.500000     7.0000(
max    1460.000000   190.000000   313.000000  215245.000000    10.0000(

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea   BsmtFinSF1
count  1460.000000  1460.000000   1460.000000  1452.000000  1460.00000(
mean      5.575342  1971.267808   1984.865753   103.685262   443.639726
std       1.112799    30.202904     20.645407   181.066207   456.09809
min       1.000000  1872.000000   1950.000000     0.000000     0.00000(
25%       5.000000  1954.000000   1967.000000     0.000000     0.00000(
50%       5.000000  1973.000000   1994.000000     0.000000   383.50000(
75%       6.000000  2000.000000   2004.000000   166.000000   712.25000(
max       9.000000  2010.000000   2010.000000  1600.000000  5644.00000(

        BsmtFinSF2    BsmtUnfSF  TotalBsmtSF     1stFlrSF     2ndFlrSF
count  1460.000000  1460.000000  1460.000000  1460.000000  1460.000000
mean     46.549315   567.240411  1057.429452  1162.626712   346.992466
std     161.319273   441.866955   438.705324   386.587738   436.528436
min       0.000000     0.000000     0.000000   334.000000     0.000000
25%       0.000000   223.000000   795.750000   882.000000     0.000000
50%       0.000000   477.500000   991.500000  1087.000000     0.000000
75%       0.000000   808.000000  1298.250000  1391.250000   728.000000
```

```
75%        0.000000   168.000000   1298.250000   1391.250000   728.000000
max     1474.000000  2336.000000  6110.000000  4692.000000  2065.000000

       LowQualFinSF    GrLivArea  BsmtFullBath  BsmtHalfBath     FullBa
count   1460.000000  1460.000000   1460.000000   1460.000000  1460.000(
mean       5.844521  1515.463699      0.425342      0.057534     1.565(
std       48.623081   525.480383      0.518911      0.238753     0.550(
min        0.000000   334.000000      0.000000      0.000000     0.000(
25%        0.000000  1129.500000      0.000000      0.000000     1.000(
50%        0.000000  1464.000000      0.000000      0.000000     2.000(
75%        0.000000  1776.750000      1.000000      0.000000     2.000(
max      572.000000  5642.000000      3.000000      2.000000     3.000(

          HalfBath  BedroomAbvGr  KitchenAbvGr  TotRmsAbvGrd   Fireplac
count  1460.000000   1460.000000   1460.000000   1460.000000  1460.000(
mean      0.382877      2.866438      1.046575      6.517808     0.613(
std       0.502885      0.815778      0.220338      1.625393     0.644(
min       0.000000      0.000000      0.000000      2.000000     0.000(
25%       0.000000      2.000000      1.000000      5.000000     0.000(
50%       0.000000      3.000000      1.000000      6.000000     1.000(
75%       1.000000      3.000000      1.000000      7.000000     1.000(
max       2.000000      8.000000      3.000000     14.000000     3.000(

       GarageYrBlt   GarageCars   GarageArea   WoodDeckSF  OpenPorchSF
count  1379.000000  1460.000000  1460.000000  1460.000000  1460.000000
mean   1978.506164     1.767123   472.980137    94.244521    46.660274
std      24.689725     0.747315   213.804841   125.338794    66.256028
min    1900.000000     0.000000     0.000000     0.000000     0.000000
25%    1961.000000     1.000000   334.500000     0.000000     0.000000
50%    1980.000000     2.000000   480.000000     0.000000    25.000000
75%    2002.000000     2.000000   576.000000   168.000000    68.000000
max    2010.000000     4.000000  1418.000000   857.000000   547.000000

       EnclosedPorch    3SsnPorch  ScreenPorch     PoolArea       MiscV
count    1460.000000  1460.000000  1460.000000  1460.000000   1460.000(
mean       21.954110     3.409589    15.060959     2.758904     43.489(
std        61.119149    29.317331    55.757415    40.177307    496.123(
min         0.000000     0.000000     0.000000     0.000000      0.000(
25%         0.000000     0.000000     0.000000     0.000000      0.000(
50%         0.000000     0.000000     0.000000     0.000000      0.000(
75%         0.000000     0.000000     0.000000     0.000000      0.000(
max       552.000000   508.000000   480.000000   738.000000  15500.000(

           MoSold       YrSold      SalePrice
count  1460.000000  1460.000000    1460.000000
mean      6.321918  2007.815753  180921.195890
std       2.703626     1.328095   79442.502883
min       1.000000  2006.000000   34900.000000
25%       5.000000  2007.000000  129975.000000
50%       6.000000  2008.000000  163000.000000
75%       8.000000  2009.000000  214000.000000
```

```
max      12.000000  2010.000000  755000.000000

Missing values per column:
Id                  0
MSSubClass          0
MSZoning            0
LotFrontage       259
LotArea             0
                 ...
MoSold              0
YrSold              0
SaleType            0
SaleCondition       0
SalePrice           0
Length: 81, dtype: int64

Duplicate values in Dataframe:
0  Duplicate(s)

<Figure size 1400x600 with 0 Axes>
```






KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageYrBlt
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
PoolArea
MiscVal
MoSold
YrSold
SalePrice
Id
MSSubClass
LotFrontage
LotArea
OverallQual
OverallCond
YearBuilt
YearRemodAdd
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageYrBlt
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
PoolArea
MiscVal
MoSold
YrSold
SalePrice
0.0
−0.2
−0.4

🚀 Step 3 Feature Engineering

3.1 Create Three(3) New Features

```
# --- Step 3: Feature engineering (shared features) ---
# We'll create a set of engineered features to improve both tasks.
# We'll operate on a copy of the dataset to keep the original intact.

df = train.copy()

# 3.1 TotalBathrooms
df['TotalBathrooms'] = (
    df.get('FullBath', 0).fillna(0) +
    0.5 * df.get('HalfBath', 0).fillna(0) +
    df.get('BsmtFullBath', 0).fillna(0) +
    0.5 * df.get('BsmtHalfBath', 0).fillna(0)
)

###
drop_cols = ['Id']
df = df.drop(columns=drop_cols)

# Build lists AFTER dropping
NUMERIC_FEATURES = df.select_dtypes(include=[np.number]).columns.tolis
CATEGORICAL_FEATURES = df.select_dtypes(include=['object']).columns.t

# If SalePrice or OverallQual were removed from numeric list:
for x in ['SalePrice', 'OverallQual']:
    if x in NUMERIC_FEATURES:
        NUMERIC_FEATURES.remove(x)
###

# 3.2 HouseAge and YearsSinceRemodel
df['HouseAge'] = df['YrSold'] - df['YearBuilt']
df['YearsSinceRemodel'] = df['YrSold'] - df['YearRemodAdd']

# 3.3 TotalPorchSF
df['TotalPorchSF'] = df.get('OpenPorchSF', 0).fillna(0) + df.get('Enc

# 3.4 TotalFinishedBsmt
df['TotalFinishedBsmt'] = df.get('BsmtFinSF1',0).fillna(0) + df.get('I
```

```
# 3.5 LivingLotRatio
# avoid division by zero
df['LivingLotRatio'] = df['GrLivArea'] / df['LotArea'].replace({0: np
df['LivingLotRatio'] = df['LivingLotRatio'].fillna(0)

# 3.6 TotalSquareFeet (aggregate)
df['TotalSF'] = df.get('GrLivArea',0).fillna(0) + df.get('TotalBsmtSF

# 3.7 TotalRooms (a reinforced feature)
df['TotalRooms'] = df.get('TotRmsAbvGrd',0).fillna(0)

# 3.8 Binary flags for presence of features
df['HasPool'] = (df.get('PoolArea',0) > 0).astype(int)
df['HasGarage'] = (~df['GarageType'].isnull()).astype(int)
df['HasFireplace'] = (df.get('Fireplaces',0) > 0).astype(int)
df['HasBasement'] = (~df['TotalBsmtSF'].isnull() & (df.get('TotalBsmt

# 3.9 Create classification target QualityCategory (Low/Medium/High)
def quality_to_category(q):
    if q <= 4: return 'Low'
    elif q <= 7: return 'Medium'
    else: return 'High'

df['QualityCategory'] = df['OverallQual'].apply(quality_to_category)
if 'Id' in df.columns:
    df = df.drop(columns=['Id'])

print("\nEngineered features added. Example:")
print(df[['TotalBathrooms','HouseAge','YearsSinceRemodel','TotalPorch
```

```
Engineered features added. Example:
   TotalBathrooms  HouseAge  YearsSinceRemodel  TotalPorchSF  \
0             3.5         5                  5            61
1             2.5        31                 31             0
2             3.5         7                  6            42
3             2.0        91                 36           307
4             3.5         8                  8            84

   TotalFinishedBsmt  LivingLotRatio  TotalSF  HasPool QualityCategory
0                706        0.202367     3114        0          Medium
1                978        0.131458     2984        0          Medium
2                486        0.158756     3314        0          Medium
3                216        0.179791     3115        0          Medium
4                655        0.154137     4179        0            High
```

🚀 Step **4**: Step 4: Select features to use (drop obviously useless columns)”

We will drop Id, and for classification we must NOT include SalePrice as a feature. Also drop columns that are identifiers or leak the target (if any).

Keep a conservative list of features: use most numeric features + selected categorical columns.

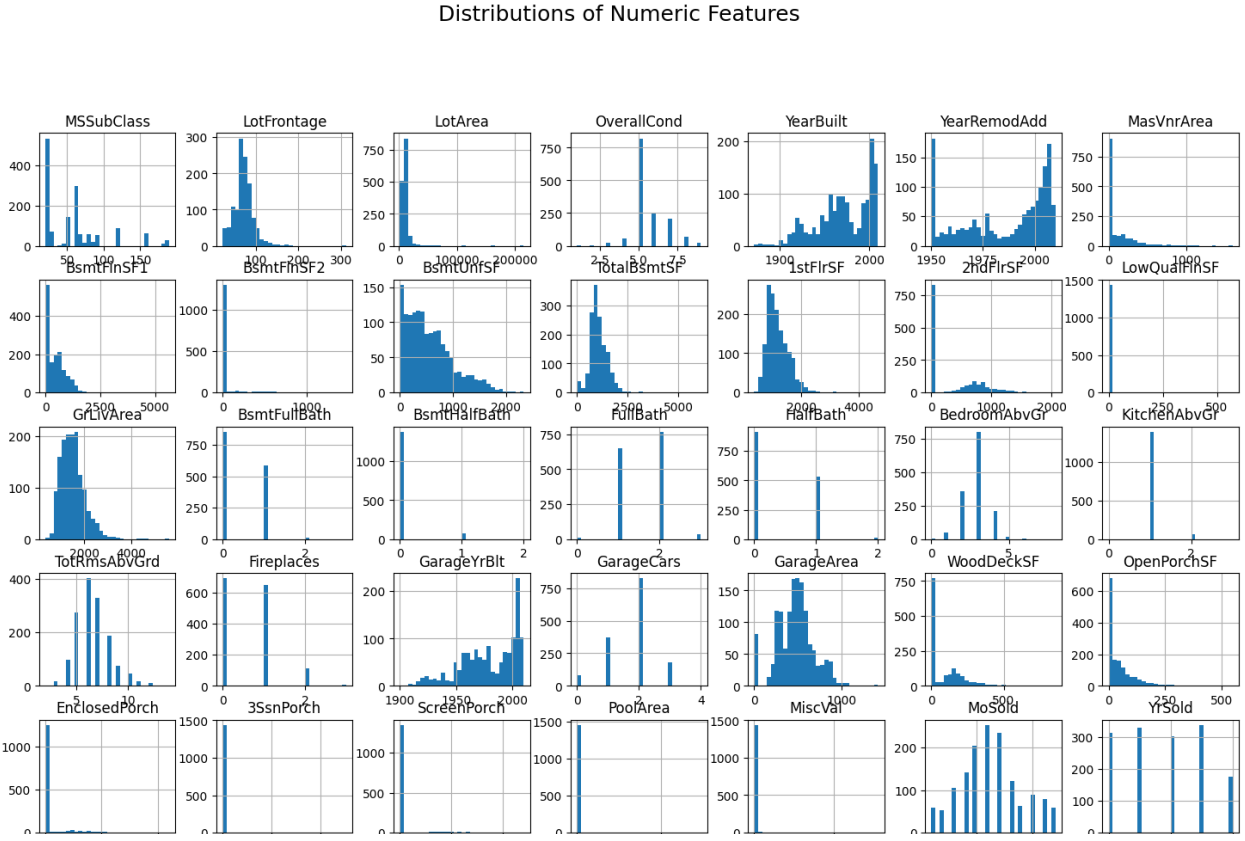
We will automatically select:

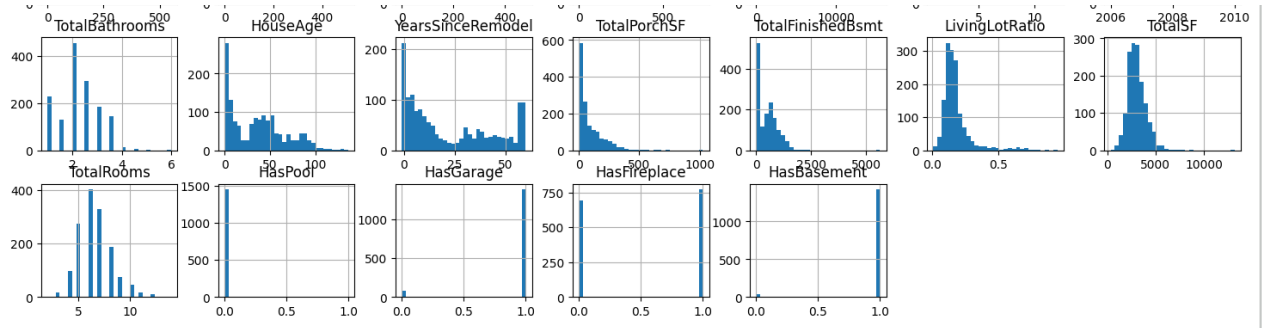
- numeric columns (except 'Id', 'SalePrice', 'OverallQual', 'QualityCategory')
- categorical columns (object or category dtype) This keeps the pipeline general and avoids hand-dropping important features.

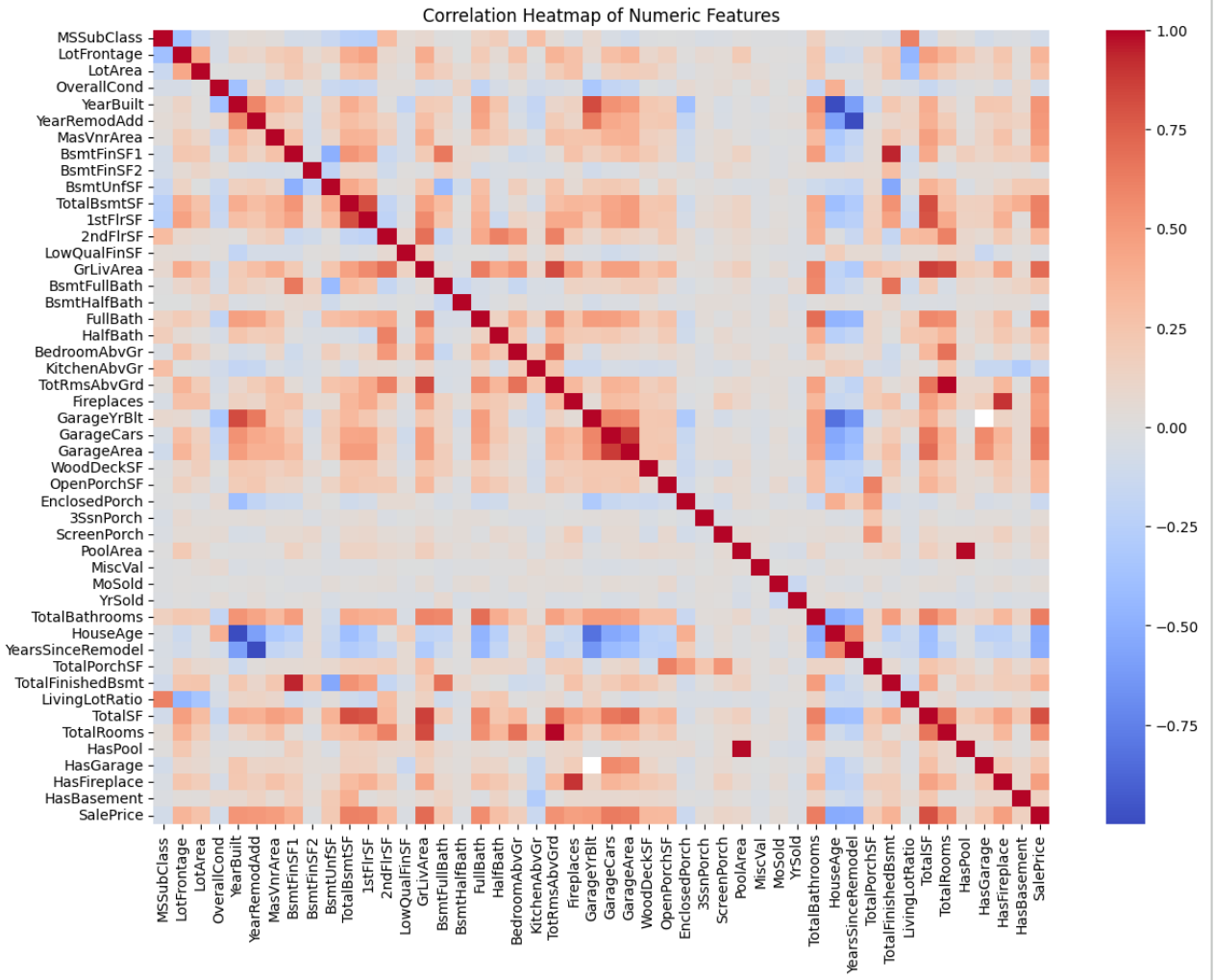
```
if 'Id' in df.columns:
    df = df.drop(columns=['Id'])  # drop identifier
all_cols = df.columns.tolist()

# numeric features (excluding target)
numeric_feats = df.select_dtypes(include=[np.number]).columns.tolist(
for x in ['SalePrice', 'OverallQual']:
    if x in numeric_feats:
        numeric_feats.remove(x)
# exclude QualityCategory (it's not numeric), ensure YrSold etc inclu
# categorical features
categorical_feats = df.select_dtypes(include=['object']).columns.toli

print(f"\nNumber of numeric features selected: {len(numeric_feats)}")
print(f"Number of categorical features selected: {len(categorical_fea
print("Sample numeric features:", numeric_feats[:10])
print("Sample categorical features:", categorical_feats[:15])

# We'll construct preprocessing pipelines below that use these lists.
NUMERIC_FEATURES = numeric_feats
CATEGORICAL_FEATURES = categorical_feats


####
```

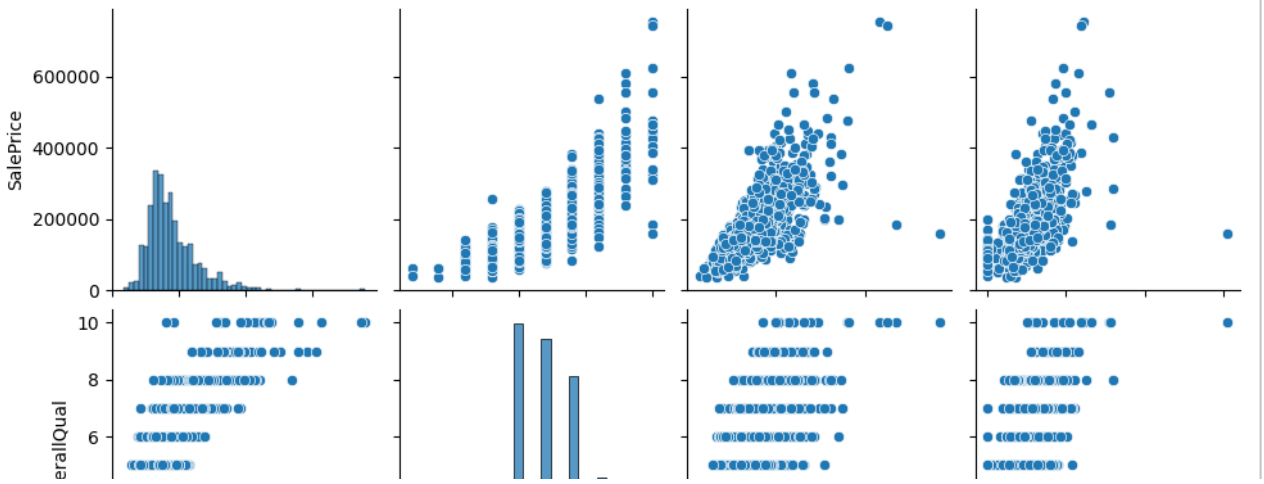
```
# 1. Histograms of numeric features
df[NUMERIC_FEATURES].hist(figsize=(18, 15), bins=30)
plt.suptitle("Distributions of Numeric Features", fontsize=18)
plt.show()

# 2. Correlation heatmap
plt.figure(figsize=(14,10))
sns.heatmap(df[NUMERIC_FEATURES + ['SalePrice']].corr(), cmap="coolwa
plt.title("Correlation Heatmap of Numeric Features")
plt.show()

# 3. (Optional) Small pairplot
important_feats = ['SalePrice', 'OverallQual', 'GrLivArea', 'TotalBsm
sns.pairplot(df[important_feats])
plt.show()
```

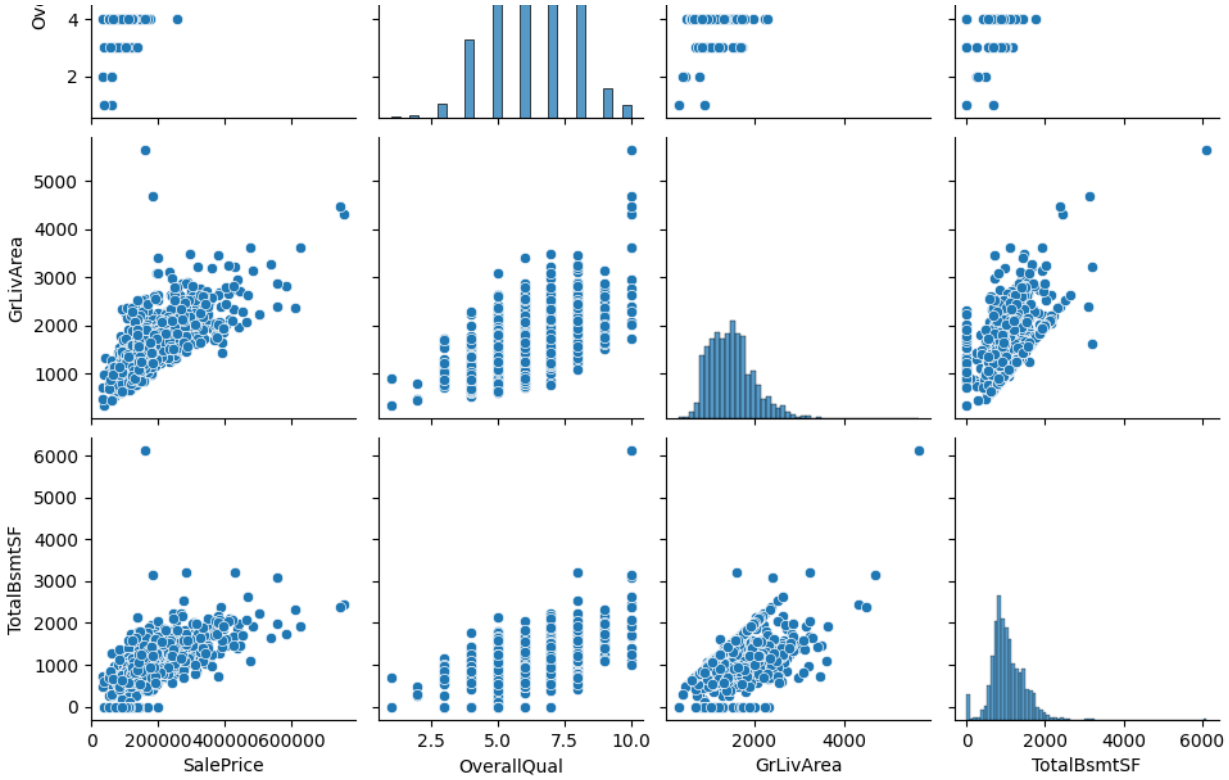
```
Number of numeric features selected: 47
Number of categorical features selected: 44
Sample numeric features: ['MSSubClass', 'LotFrontage', 'LotArea', 'Over
Sample categorical features: ['MSZoning', 'Street', 'Alley', 'LotShape'
```




TotalBathrooms
HouseAge
YearsSinceRemodel
TotalPorchSF
TotalFinishedBsmt
LivingLotRatio
TotalSF
TotalRooms
HasPool
HasGarage
HasFireplace
HasBasement


Correlation Heatmap of Numeric Features
MSSubClass
LotFrontage
LotArea
OverallCond
YearBuilt
YearRemodAdd
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageYrBlt
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
PoolArea
MiscVal
MoSold
YrSold
TotalBathrooms
HouseAge
YearsSinceRemodel
TotalPorchSF
TotalFinishedBsmt
LivingLotRatio
TotalSF
TotalRooms
HasPool
HasGarage
HasFireplace
HasBasement
SalePrice
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75


SalePrice
600000
400000
200000
0
10
8
6


Ov
GrLivArea
TotalBsmtSF
SalePrice
OverallQual
GrLivArea
TotalBsmtSF

🚀 Step **5**: Preprocessing pipelines

```
# --- Step 5: Preprocessing pipelines ---
# We create sklearn ColumnTransformer pipelines to:
#  - impute numeric features with median and optionally scale for lin
#  - impute categorical features with constant 'Missing' and one-hot 

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    # scaling will be applied inside a model-specific pipeline when n
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='Missin
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=Fa
])

preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, NUMERIC_FEATURES),
    ('cat', categorical_transformer, CATEGORICAL_FEATURES)
], remainder='drop')  # drop anything else not specified
```

🚀 Step **6**: Prepare data for Regression (SalePrice)

We'll use log1p(SalePrice) to stabilize variance and improve linear regression.

```
# Step 6: Prepare data for Regression (SalePrice)
# We'll use log1p(SalePrice) to stabilize variance and improve linear

reg_df = df.copy()
reg_df = reg_df.drop(columns=drop_cols, errors='ignore')
y_reg = np.log1p(df['SalePrice'])  # log target
X_reg = reg_df.drop(columns=['SalePrice'], errors='ignore')  # keep o

# split
X = df.drop('SalePrice', axis=1)
y = df['SalePrice']
X_train_reg, X_val_reg, y_train_reg, y_val_reg = train_test_split(X_r

print(f"\nRegression dataset shapes: X_train {X_train_reg.shape}, X_v
```

```
Regression dataset shapes: X_train (1168, 92), X_val (292, 92)
```

🚀 Step 7: Regression model pipelines and training

We build three pipelines:

- LinearRegression (needs scaling)

- RandomForestRegressor (tree model, no scaling)

- GradientBoostingRegressor (tree model)

Each pipeline uses the preprocessor; for LR we'll add a StandardScaler after numeric imputer.

a. Linear Regression pipeline

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

# 7.1 Linear Regression pipeline
preprocessor_lr = ColumnTransformer(transformers=[
    ('num', Pipeline([('imputer', SimpleImputer(strategy='median')),
    ('cat', categorical_transformer, CATEGORICAL_FEATURES)
], remainder='drop')

lr_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor_lr),
    ('lr', LinearRegression())
])
```

b. GradientRandom Forest pipeline

```
rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('rf', RandomForestRegressor(n_estimators=200, n_jobs=-1, random_
])
```

c. Gradient Boosting pipeline

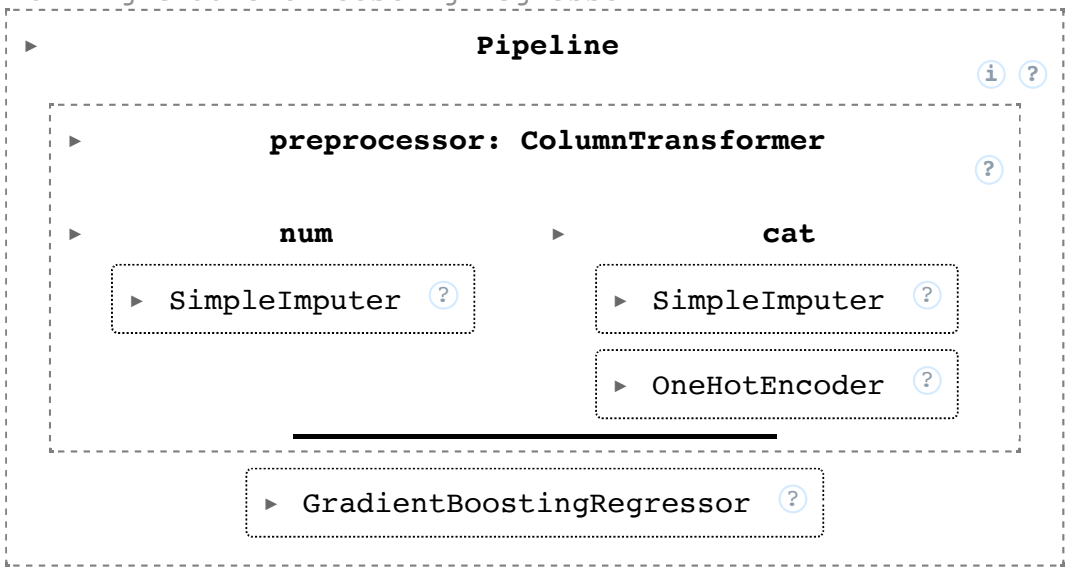
```
gb_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('gb', GradientBoostingRegressor(n_estimators=200, learning_rate=(
])
```

d. Train models

```
# # # 1. create new engineered feature
# # df['QualityCategory'] = df['OverallQual'].apply(quality_to_catego

# # # 2. drop Id
# # # df = df.drop(columns=['Id'])

# # # 3. split target and predictors
# # X = df.drop('SalePrice', axis=1)
# # y = df['SalePrice']

# # 4. NOW rebuild feature lists
# # NUMERIC_FEATURES = X.select_dtypes(include=[np.number]).columns.t
# # CATEGORICAL_FEATURES = X.select_dtypes(include=['object']).columns

# NUMERIC_FEATURES = X_reg.select_dtypes(include=[np.number]).columns
# CATEGORICAL_FEATURES = X_reg.select_dtypes(include=['object']).colu

# # (remove OverallQual from numeric if needed)
# if 'OverallQual' in NUMERIC_FEATURES:
#     NUMERIC_FEATURES.remove('OverallQual')

# # 5. Train-test split

# X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split
#     X, y, test_size=0.2, random_state=42
# )

# print("QualityCategory in X_train_reg?", 'QualityCategory' in X_tra
```

```
print("\nTraining Linear Regression...")
lr_pipeline.fit(X_train_reg, y_train_reg)

print("Training Random Forest Regressor...")
rf_pipeline.fit(X_train_reg, y_train_reg)

print("Training Gradient Boosting Regressor...")
gb_pipeline.fit(X_train_reg, y_train_reg)
```

```
Training Linear Regression...
Training Random Forest Regressor...
Training Gradient Boosting Regressor...
```



e. Evaluate on validation set

```
# Evaluate on validation set
def evaluate_regression(model, X_val, y_val, label="model"):
    preds_log = model.predict(X_val)
    # preds are log1p, so convert back for RMSE on original SalePrice
    preds = np.expm1(preds_log)
    y_true = np.expm1(y_val)
    rmse = np.sqrt(mean_squared_error(y_true, preds))
    r2 = r2_score(y_true, preds)
    return rmse, r2, preds, y_true

lr_rmse, lr_r2, lr_preds, y_true_reg = evaluate_regression(lr_pipeline
rf_rmse, rf_r2, rf_preds, _ = evaluate_regression(rf_pipeline, X_val_
gb_rmse, gb_r2, gb_preds, _ = evaluate_regression(gb_pipeline, X_val_

results_reg = pd.DataFrame({
    'model': ['LinearRegression','RandomForest','GradientBoosting'],
    'rmse': [lr_rmse, rf_rmse, gb_rmse],
    'r2': [lr_r2, rf_r2, gb_r2]
}).sort_values('rmse')

print("\nRegression results on validation set:")
print(results_reg)
```

```
Regression results on validation set:
              model          rmse        r2
0  LinearRegression  24410.007393  0.922318
2  GradientBoosting  30310.547233  0.880223
1      RandomForest  33767.699452  0.851342
```

f. Save regression models

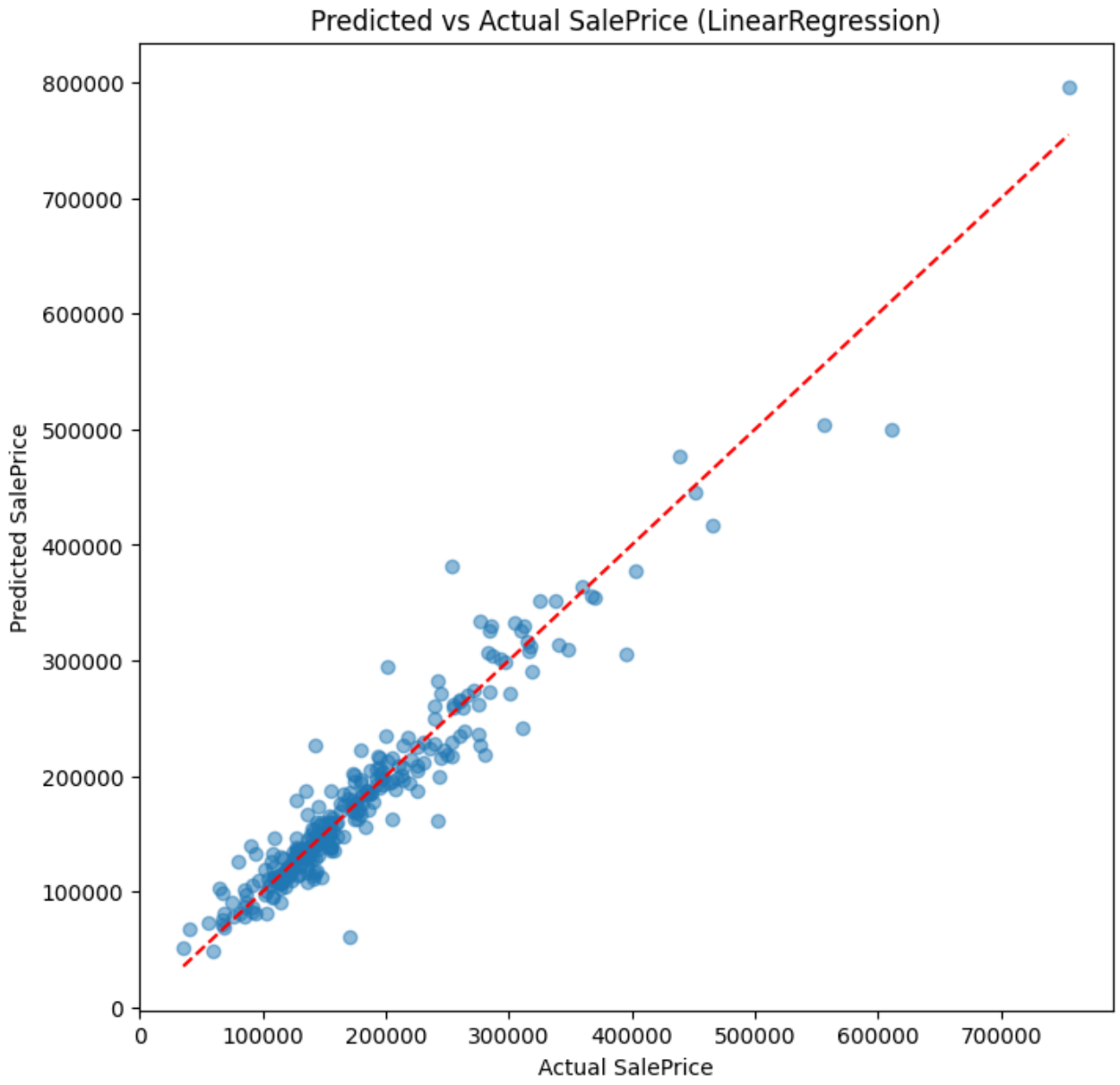
```
joblib.dump(lr_pipeline, MODELS_DIR / "lr_pipeline_reg.pkl")
joblib.dump(rf_pipeline, MODELS_DIR / "rf_pipeline_reg.pkl")
joblib.dump(gb_pipeline, MODELS_DIR / "gb_pipeline_reg.pkl")
```

```
['outputs/models/gb_pipeline_reg.pkl']
```

g. Plot Predicted vs Actual (for best model by rmse)

```
best_reg_model_name = results_reg.iloc[0]['model']
```

```
best_reg_preds = {'LinearRegression': lr_preds, 'RandomForest': rf_pr
plt.figure(figsize=(8,8))
plt.scatter(y_true_reg, best_reg_preds, alpha=0.5)
plt.plot([y_true_reg.min(), y_true_reg.max()], [y_true_reg.min(), y_t
plt.xlabel('Actual SalePrice')
plt.ylabel('Predicted SalePrice')
plt.title(f'Predicted vs Actual SalePrice ({best_reg_model_name})')
plt.savefig(PLOTS_DIR / f"pred_vs_actual_{best_reg_model_name}.png", 
plt.show()
```

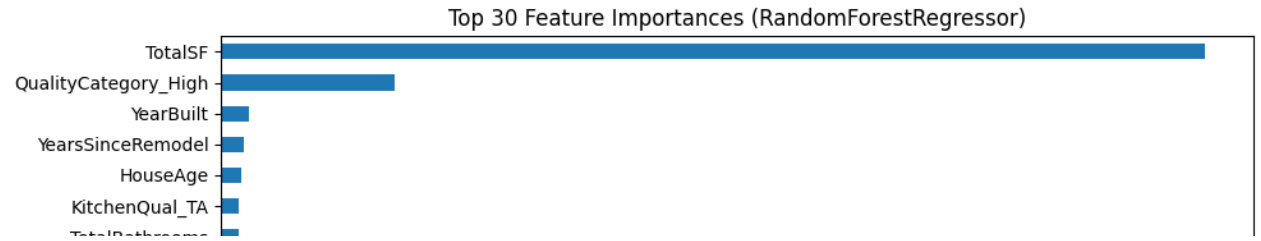


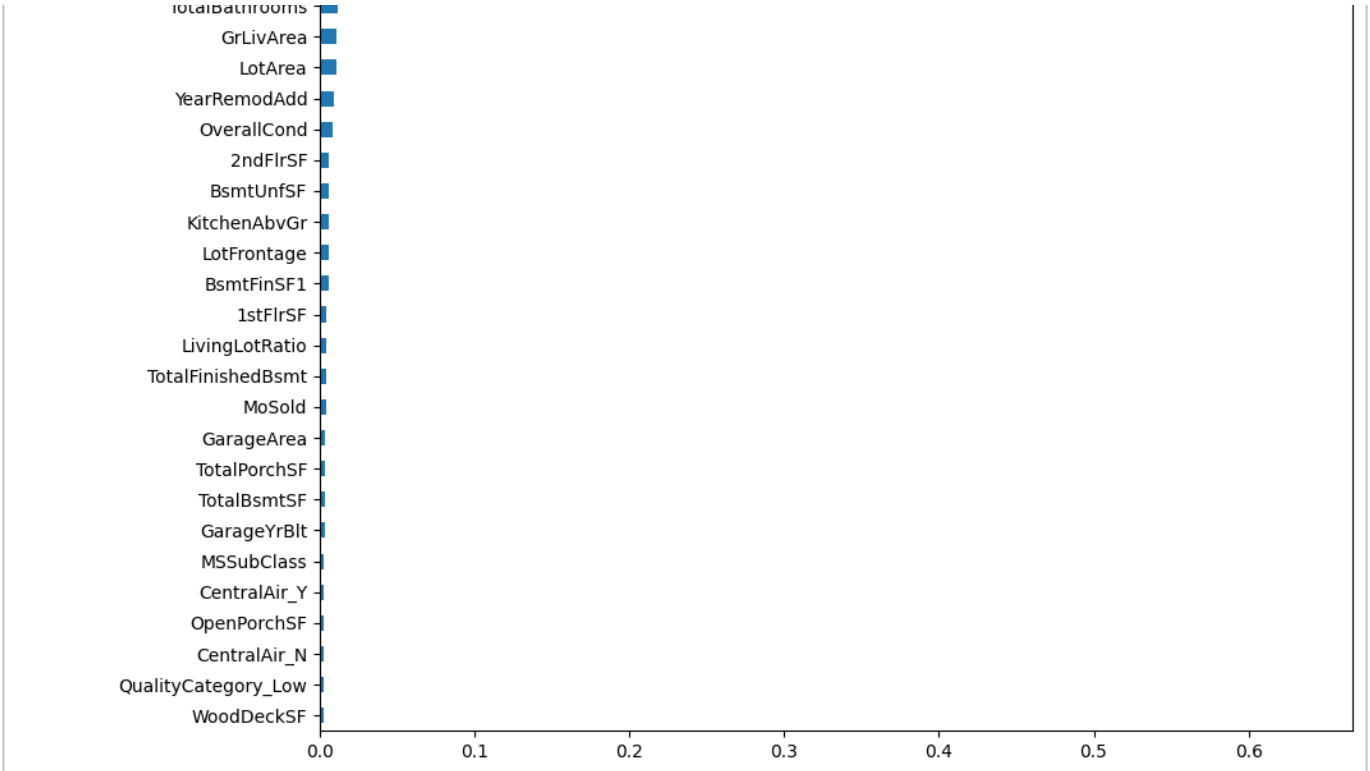
h. Feature importances for tree models (RandomForest & GradientBoosting)

```
def get_feature_names_from_preprocessor(preprocessor_obj):
    # numeric feature names remain the same
    num_names = NUMERIC_FEATURES
    # get onehot names from the categorical transformer
    cat_pipe = preprocessor_obj.named_transformers_['cat']
    ohe = cat_pipe.named_steps['onehot']
    ohe_feature_names = list(ohe.get_feature_names_out(CATEGORICAL_FE
    return num_names + ohe_feature_names
```

I. For rf_pipeline: get importances, Fit the preprocessor alone on full training to retrieve feature names

```
try:
    # Fit the preprocessor alone on full training to retrieve feature
    preprocessor.fit(X_train_reg)
    feature_names = get_feature_names_from_preprocessor(preprocessor)
    rf_feature_importances = rf_pipeline.named_steps['rf'].feature_imp
    fi_rf = pd.Series(rf_feature_importances, index=feature_names).so
    plt.figure(figsize=(10,8))
    fi_rf.plot(kind='barh')
    plt.gca().invert_yaxis()
    plt.title("Top 30 Feature Importances (RandomForestRegressor)")
    plt.tight_layout()
    plt.savefig(PLOTS_DIR / "rf_feature_importances_reg.png", bbox_in
    plt.show()
except Exception as e:
    print("Could not compute or plot feature importances for regressi
```





🚀 Step 8: Prepare data for Classification (QualityCategory)

We'll classify 'QualityCategory' into: Low / Medium / High as strings.

```
# Classify 'QualityCategory' into Low/Medium/High.
cls_df = df.copy()
cls_df = cls_df.drop(columns=drop_cols, errors='ignore')

# Ensure we DO NOT include SalePrice or OverallQual as features (Overa
# However, we created QualityCategory from OverallQual; to keep reali
if 'OverallQual' in cls_df.columns:
    cls_df = cls_df.drop(columns=['OverallQual'])

# If 'SalePrice' exists, drop it to avoid leakage
if 'SalePrice' in cls_df.columns:
    cls_df = cls_df.drop(columns=['SalePrice'])

# Target
y_cls = cls_df['QualityCategory']
X_cls = cls_df.drop(columns=['QualityCategory'])

# Split
X_train_cls, X_val_cls, y_train_cls, y_val_cls = train_test_split(X_c

print(f"\nClassification dataset shapes: X_train {X_train_cls.shape},
print("Class distribution in training set:")
print(y_train_cls.value_counts(normalize=True))
```

```
Classification dataset shapes: X_train (1168, 90), X_val (292, 90)
Class distribution in training set:
QualityCategory
Medium    0.746575
High      0.156678
Low       0.096747
Name: proportion, dtype: float64
```

🚀 Step 9: Classification preprocessing & pipelines

We reuse the earlier preprocessor objects (numeric transformer + categorical transformer)

For classification models we can use the same `preprocessor` defined earlier.

```
# We rebuild numeric and categorical features specifically for classi
# after dropping the target 'QualityCategory'
NUMERIC_FEATURES_CLS = X_cls.select_dtypes(include=[np.number]).columı
CATEGORICAL_FEATURES_CLS = X_cls.select_dtypes(include=['object']).co

print("Numeric features (CLS):", NUMERIC_FEATURES_CLS)
print("Categorical features (CLS):", CATEGORICAL_FEATURES_CLS)
```

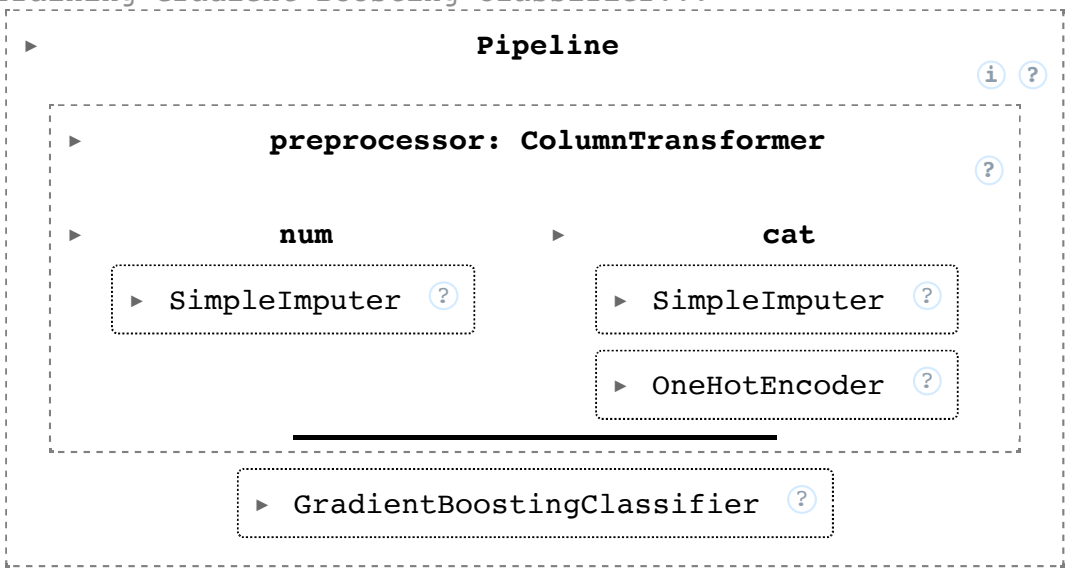
```
Numeric features (CLS): ['MSSubClass', 'LotFrontage', 'LotArea', 'Overa
Categorical features (CLS): ['MSZoning', 'Street', 'Alley', 'LotShape'
```

```
# Rebuild preprocessing pipelines for classification

from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Numeric transformer (scale only for Logistic Regression)
numeric_transformer_lr_cls = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Categorical transformer
categorical_transformer_cls = Pipeline([
    ('imputer', SimpleImputer(strategy='constant', fill_value='Missing
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=Fa
])

# ColumnTransformers
preprocessor_lr_cls = ColumnTransformer([
    ('num', numeric_transformer_lr_cls, NUMERIC_FEATURES_CLS),
    ('cat', categorical_transformer_cls, CATEGORICAL_FEATURES_CLS)
])

preprocessor_cls = ColumnTransformer([
    ('num', Pipeline([('imputer', SimpleImputer(strategy='median'))]),
    ('cat', categorical_transformer_cls, CATEGORICAL_FEATURES_CLS)
])
```

```
# Build classification pipelines

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoosting(

logreg_pipeline = Pipeline([
    ('preprocessor', preprocessor_lr_cls),
    ('logreg', LogisticRegression(multi_class='multinomial', solver='
])

rf_cls_pipeline = Pipeline([
    ('preprocessor', preprocessor_cls),
    ('rf_cls', RandomForestClassifier(n_estimators=200, n_jobs=-1, ra
])

gb_cls_pipeline = Pipeline([
    ('preprocessor', preprocessor_cls),
    ('gb_cls', GradientBoostingClassifier(n_estimators=200, learning_
])
```

```
# Train classifiers
print("\nTraining Logistic Regression (classification)...")
logreg_pipeline.fit(X_train_cls, y_train_cls)

print("Training Random Forest Classifier...")
rf_cls_pipeline.fit(X_train_cls, y_train_cls)

print("Training Gradient Boosting Classifier...")
gb_cls_pipeline.fit(X_train_cls, y_train_cls)
```

```
Training Logistic Regression (classification)...
Training Random Forest Classifier...
Training Gradient Boosting Classifier...
```

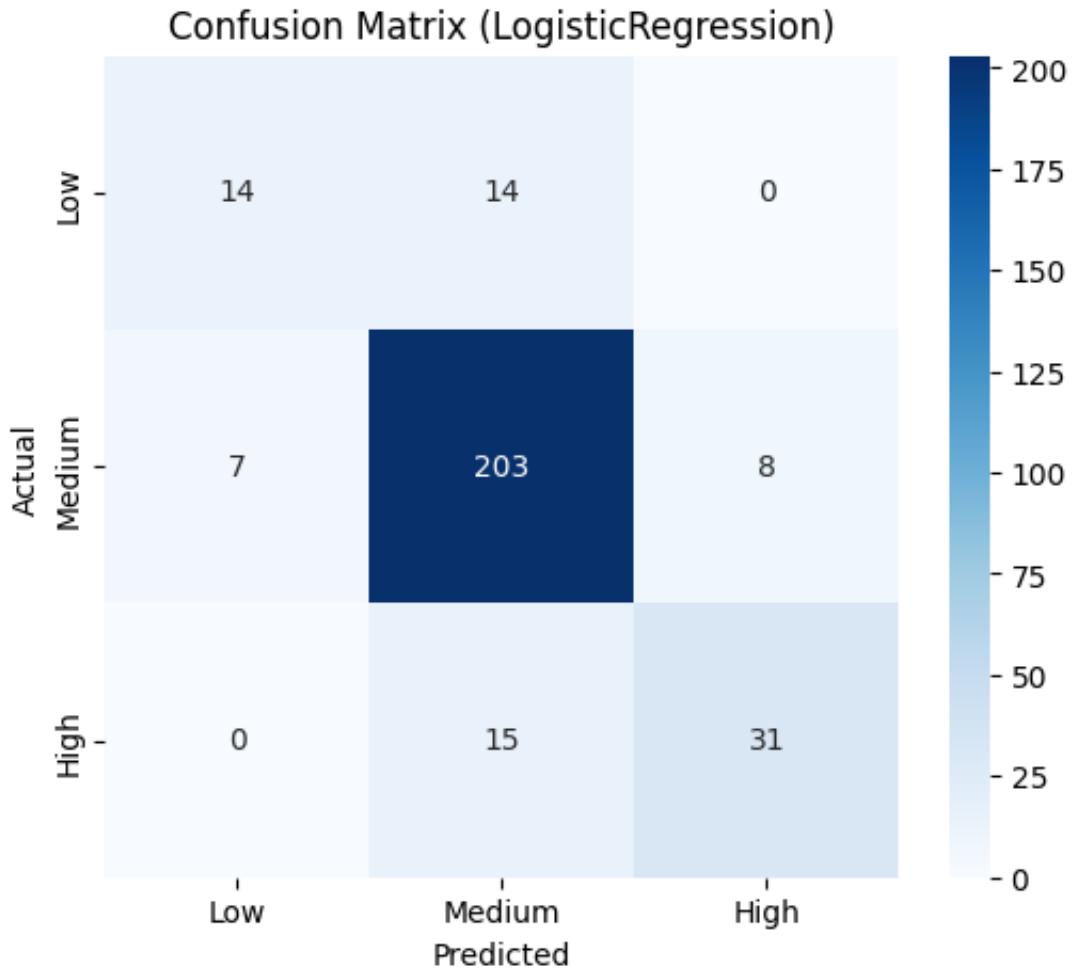



```
# Evaluate classification
def evaluate_classification(model, X_val, y_val, label="model"):
    preds = model.predict(X_val)
    acc = accuracy_score(y_val, preds)
    f1 = f1_score(y_val, preds, average='macro')
    cm = confusion_matrix(y_val, preds, labels=['Low','Medium','High']
    return acc, f1, preds, cm

log_acc, log_f1, log_preds, log_cm = evaluate_classification(logreg_p
rf_acc, rf_f1, rf_preds, rf_cm = evaluate_classification(rf_cls_pipel
gb_acc, gb_f1, gb_preds, gb_cm = evaluate_classification(gb_cls_pipel
```

```
results_cls = pd.DataFrame({
    'model': ['LogisticRegression','RandomForest','GradientBoosting'],
    'accuracy': [log_acc, rf_acc, gb_acc],
    'f1_macro': [log_f1, rf_f1, gb_f1]
}).sort_values('accuracy', ascending=False)

print("\nClassification results on validation set:")
print(results_cls)

# Save classification models
joblib.dump(logreg_pipeline, MODELS_DIR / "logreg_pipeline_cls.pkl")
joblib.dump(rf_cls_pipeline, MODELS_DIR / "rf_pipeline_cls.pkl")
joblib.dump(gb_cls_pipeline, MODELS_DIR / "gb_pipeline_cls.pkl")


# Plot confusion matrix for best classifier
best_cls_model_name = results_cls.iloc[0]['model']
best_cm = {'LogisticRegression': log_cm, 'RandomForest': rf_cm, 'Grad
plt.figure(figsize=(6,5))
sns.heatmap(best_cm, annot=True, fmt='d', xticklabels=['Low','Medium'
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix ({best_cls_model_name})')
plt.savefig(PLOTS_DIR / f"confusion_matrix_{best_cls_model_name}.png"
plt.show()

# Print classification report for the best classifier
best_preds = {'LogisticRegression': log_preds, 'RandomForest': rf_pre
print(f"\nClassification report for best model ({best_cls_model_name})
print(classification_report(y_val_cls, best_preds, digits=4))


# # Feature importance for RF classifier (top features)
# try:
#     preprocessor.fit(X_train_cls)
#     feature_names = get_feature_names_from_preprocessor(preprocesso
#     rf_cls_feature_importances = rf_cls_pipeline.named_steps['rf_cls
#     fi_rf_cls = pd.Series(rf_cls_feature_importances, index=feature_
#     plt.figure(figsize=(10,8))
#     fi_rf_cls.plot(kind='barh')
#     plt.gca().invert_yaxis()
#     plt.title("Top 30 Feature Importances (RandomForestClassifier)"
#     plt.tight_layout()
#     plt.savefig(PLOTS_DIR / "rf_feature_importances_cls.png", bbox_i
#     plt.show()
```

```
# except Exception as e:
#     print("Could not compute or plot feature importances for classi
```

```
Classification results on validation set:
                model  accuracy  f1_macro
0  LogisticRegression  0.849315  0.734354
1        RandomForest  0.845890  0.655088
2    GradientBoosting  0.828767  0.649697
```




```
Classification report for best model (LogisticRegression):
              precision    recall  f1-score   support

        High     0.7949    0.6739    0.7294        46
         Low     0.6667    0.5000    0.5714        28
      Medium     0.8750    0.9312    0.9022       218

    accuracy                         0.8493       292
   macro avg     0.7788    0.7017    0.7344       292
weighted avg     0.8424    0.8493    0.8433       292
```

```
# # Ensure the 'OverallQual' column is in the original dataset
# print("Columns in original train dataset:")
# print(train.columns)

# # Check that 'OverallQual' is in the features (X) before the split
# features = ['MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'St
#             'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'La
#             'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
#             'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAd
#             'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
#             'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'Bs
#             'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinS
#             'TotalBsmtSF', 'Heating', 'HeatingQC', 'CentralAir', 'E
#             '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
#             'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQua
#             'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
#             'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual
#             'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorc
#             'PoolQC', 'Fence', 'MiscFeature', 'MiscVal', 'MoSold',
#             'SaleCondition', 'TotalBathrooms', 'HouseAge', 'YearsSi
#             'TotalFinishedBsmt', 'LivingLotRatio', 'TotalSF', 'Tota
#             'HasFireplace', 'HasBasement']

# # Split the data (ensure 'OverallQual' is in X_train_cls)
# X = train[features]  # Make sure 'OverallQual' is in this list
# y = train['SalePrice']  # Assuming SalePrice is the target variable

# # Now, let's split into training and validation sets
# from sklearn.model_selection import train_test_split
# X_train_cls, X_val_cls, y_train_cls, y_val_cls = train_test_split(X,

# # Check if 'OverallQual' is in X_train_cls
# print("Columns in X_train_cls after splitting:")
# print(X_train_cls.columns)
```

🚀 Step 10: Compare feature importance and insights

Simple side-by-side summary output saved to file

```
summary = {
    'regression_results': results_reg.to_dict(orient='records'),
    'classification_results': results_cls.to_dict(orient='records'),
    'best_regression_model': best_reg_model_name,
    'best_classification_model': best_cls_model_name
}

import json
with open(OUTPUT_DIR / "summary_results.json", "w") as f:
    json.dump(summary, f, indent=2)

print("\nSummary saved to outputs/summary_results.json")
print("Plots saved to outputs/plots/ and models saved to outputs/mode
```

```
Summary saved to outputs/summary_results.json
Plots saved to outputs/plots/ and models saved to outputs/models/
```

🚀 Step 11: Prepare test.csv predictions for submission

If a test.csv is available and it lacks SalePrice, We could produce predictions using your best regression model.

```
# --- Step 11: (Optional) Prepare test.csv predictions for submission
# If a test.csv is available and it lacks SalePrice (typical Kaggle s
# you could produce predictions using your best regression model.
if test is not None:
    print("\nPreparing predictions for test.csv (if columns match).")
    # apply same engineered features on test copy
    test_copy = test.copy()
```

```
# Recreate engineered features safely (same as done for training s
test_copy['TotalBathrooms'] = (
    test_copy.get('FullBath', 0).fillna(0) +
    0.5 * test_copy.get('HalfBath', 0).fillna(0) +
    test_copy.get('BsmtFullBath', 0).fillna(0) +
    0.5 * test_copy.get('BsmtHalfBath', 0).fillna(0)
)
test_copy['HouseAge'] = test_copy['YrSold'] - test_copy['YearBuil
test_copy['YearsSinceRemodel'] = test_copy['YrSold'] - test_copy[
test_copy['TotalPorchSF'] = test_copy.get('OpenPorchSF', 0).fillna
test_copy['TotalFinishedBsmt'] = test_copy.get('BsmtFinSF1', 0).fi
test_copy['LivingLotRatio'] = test_copy['GrLivArea'] / test_copy[
test_copy['LivingLotRatio'] = test_copy['LivingLotRatio'].fillna(0
test_copy['TotalSF'] = test_copy.get('GrLivArea', 0).fillna(0) + t
test_copy['TotalRooms'] = test_copy.get('TotRmsAbvGrd', 0).fillna
test_copy['HasPool'] = (test_copy.get('PoolArea', 0) > 0).astype(i
test_copy['HasGarage'] = (~test_copy['GarageType'].isnull()).astyp
test_copy['HasFireplace'] = (test_copy.get('Fireplaces', 0) > 0).a
test_copy['HasBasement'] = (~test_copy['TotalBsmtSF'].isnull() & (

# Check if 'QualityCategory' is missing in test data and create it
if 'QualityCategory' not in test_copy.columns:
    test_copy['QualityCategory'] = test_copy['OverallQual'].apply

# Ensure same columns exist - drop columns not present in training
# We'll attempt to predict with the best regression model
model_for_submission = {'LinearRegression': lr_pipeline, 'RandomFo
try:
    preds_log_test = model_for_submission.predict(test_copy)
    preds_test = np.expm1(preds_log_test)
    submission = pd.DataFrame({
        'Id': test_copy['Id'],
        'SalePrice': preds_test
    })
    submission.to_csv(OUTPUT_DIR / "submission_example.csv", index
    print("Example submission saved to outputs/submission_example.
except Exception as e:
    print("Could not produce test predictions automatically. Error
```

```
Preparing predictions for test.csv (if columns match).
Example submission saved to outputs/submission_example.csv
```

Step 12: Final notes and tips

```
print("\n--- Final Notes ---")
print("1) Best regression model (by RMSE):", best_reg_model_name)
print("2) Best classification model (by accuracy):", best_cls_model_na
print("3) Check outputs/plots for all visualizations and outputs/model
print("\nImprovements:")
print("- add hyperparameter tuning (GridSearchCV/RandomizedSearchCV),"
print("- add SHAP explanations for feature impact (if you want interpr
print("Run completed at:", datetime.now().isoformat())
```

```
--- Final Notes ---
1) Best regression model (by RMSE): LinearRegression
2) Best classification model (by accuracy): LogisticRegression
3) Check outputs/plots for all visualizations and outputs/models for sa

Improvements:
- add hyperparameter tuning (GridSearchCV/RandomizedSearchCV),
- add SHAP explanations for feature impact (if you want interpretabili

Run completed at: 2025-12-04T21:21:39.744188
```

🚀 Step **13**: — Interactive CLI-style Prediction (No DB, no extra files)

This section prompts for a one-line record (key=value, key=value, ...), computes engineered features, and produces:

- Predicted SalePrice (USD) from the regression pipeline (converts from log1p)

- Predicted QualityCategory from the classification pipeline, with optional probabilities

It will use in-memory models if they exist (from earlier cells). If not found, it will try to load `lr_pipeline_reg.pkl` and `logreg_pipeline_cls.pkl` from the current working directory.

```
# Patch for Section 13.1 — robust engineer_features
```

```
import numpy as np
import pandas as pd

def engineer_features(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()

    # Always return a Series (never a scalar) so downstream .fillna()
    def s0(col):  # numeric Series with default 0
        if col in df.columns:
            return pd.to_numeric(df[col], errors="coerce")
        return pd.Series(0, index=df.index, dtype="float64")

    def sNaN(col):  # numeric Series with default NaN
        if col in df.columns:
            return pd.to_numeric(df[col], errors="coerce")
        return pd.Series(np.nan, index=df.index, dtype="float64")

    # Bathrooms (missing → 0)
    full_bath = s0("FullBath")
    half_bath = s0("HalfBath")
    bfull = s0("BsmtFullBath")
    bhalf = s0("BsmtHalfBath")
    df["TotalBathrooms"] = full_bath.fillna(0) + 0.5*half_bath.fillna

    # Ages (allow NaN)
    df["HouseAge"] = sNaN("YrSold") - sNaN("YearBuilt")
    df["YearsSinceRemodel"] = sNaN("YrSold") - sNaN("YearRemodAdd")

    # Porches (missing → 0)
    df["TotalPorchSF"] = (
        s0("OpenPorchSF").fillna(0)
        + s0("EnclosedPorch").fillna(0)
        + s0("3SsnPorch").fillna(0)
        + s0("ScreenPorch").fillna(0)
    )

    # Basement finished (missing → 0)
    df["TotalFinishedBsmt"] = s0("BsmtFinSF1").fillna(0) + s0("BsmtFi

    # Ratios and totals
    lot_area = sNaN("LotArea").replace({0: np.nan})
    df["LivingLotRatio"] = (s0("GrLivArea") / lot_area).replace([np.i
    df["TotalSF"] = s0("GrLivArea").fillna(0) + s0("TotalBsmtSF").fil
    df["TotalRooms"] = s0("TotRmsAbvGrd").fillna(0)
```

```
    # Binary flags
    df["HasPool"] = (s0("PoolArea").fillna(0) > 0).astype(int)

    if "GarageType" in df.columns:
        gt = df["GarageType"].astype("string")
        df["HasGarage"] = gt.fillna("").str.strip().ne("").astype(int
    else:
        df["HasGarage"] = ((s0("GarageCars").fillna(0) > 0) | (s0("Ga

    df["HasFireplace"] = (s0("Fireplaces").fillna(0) > 0).astype(int)
    df["HasBasement"] = (s0("TotalBsmtSF").fillna(0) > 0).astype(int)

    # QualityCategory feature (used by regression)
    def quality_to_category(val):
        try:
            q = float(val)
        except Exception:
            return np.nan
        if q <= 4:
            return "Low"
        elif q <= 7:
            return "Medium"
        else:
            return "High"

    oq = sNaN("OverallQual")
    df["QualityCategory"] = oq.apply(quality_to_category)

    # Drop identifiers if present
    if "Id" in df.columns:
        df = df.drop(columns=["Id"])

    return df
```

B. Section 13.2 — One-liner UI (ipywidgets)

```
# Section 13.2 — Guided one-liner input (no widgets) with examples and

import numpy as np
import pandas as pd
from IPython.display import HTML, display

# Assumes Section 13.1 helpers exist:
```

```
# - resolve_models, engineer_features, parse_record_line, expected_colu

# Load models (in-memory or from .pkl files)
try:
    reg_model, cls_model = resolve_models()
except Exception as e:
    raise RuntimeError(
        f"Model load error: {e}\n"
        "Ensure earlier training cells ran (so pipelines exist in memor
        "or place lr_pipeline_reg.pkl and logreg_pipeline_cls.pkl in th
    )

# 1) EXACTLY how to input (keys, format, examples)
guide_html = """
<div style='font-family:Inter,system-ui,Arial;line-height:1.45'>
  <h4>How to enter your house details</h4>
  <ul>
    <li>Type a single line of comma-separated key=value pairs.</li>
    <li>Keys are case-sensitive. Example keys: <code>OverallQual</code>
    <li>Strings with spaces should be quoted: <code>Neighborhood="Old T
    <li>You can omit fields; missing values are imputed. More fields =
  </ul>

  <b>Recommended minimum fields</b> (good accuracy):
  <div><code>OverallQual, YearBuilt, YearRemodAdd, YrSold, GrLivArea, T

  <p style="margin-top:8px"><b>Pick an example (A/B/C) or paste your ow
  <ol>
    <li><b>A — Mid-tier family home</b><br>
      <code>OverallQual=7,YearBuilt=2003,YearRemodAdd=2003,YrSold=2008,
    </li>
    <li><b>B — Entry-level basic home</b><br>
      <code>OverallQual=4,YearBuilt=1975,YearRemodAdd=1975,YrSold=2007,
    </li>
    <li><b>C — High-end newer home</b><br>
      <code>OverallQual=9,YearBuilt=2006,YearRemodAdd=2007,YrSold=2010,
    </li>
  </ol>
  <p style="margin-top:8px">Type <b>A</b>, <b>B</b>, or <b>C</b> to use
</div>
"""
display(HTML(guide_html))

example_A = "OverallQual=7,YearBuilt=2003,YearRemodAdd=2003,YrSold=2008
example_B = "OverallQual=4,YearBuilt=1975,YearRemodAdd=1975,YrSold=2007
```

```
example_C = "OverallQual=9,YearBuilt=2006,YearRemodAdd=2007,YrSold=2010

choice = input("Your choice (A/B/C) OR paste a full one-liner: ").strip

if choice.upper() == "A":
    line = example_A
elif choice.upper() == "B":
    line = example_B
elif choice.upper() == "C":
    line = example_C
else:
    line = choice  # treat as custom one-liner

# Parse input
record = parse_record_line(line)
if not record:
    raise ValueError("Could not parse any key=value pairs. Use the exam

# Light validation to help graders/users
def _warns_for(rec: dict):
    w = []
    def iv(k): return rec.get(k, None)
    try:
        oq = iv("OverallQual");
        if oq is not None and not (1 <= float(oq) <= 10): w.append("Ove
        yb = iv("YearBuilt"); yrm = iv("YearRemodAdd"); ys = iv("YrSold
        if yb and yrm and float(yrm) < float(yb): w.append("YearRemodAd
        if yb and ys and float(ys) < float(yb): w.append("YrSold is ear
        gla = iv("GrLivArea");
        if gla is not None and float(gla) < 300: w.append("GrLivArea lo
        lot = iv("LotArea");
        if lot is not None and float(lot) < 1000: w.append("LotArea loo
        fb = iv("FullBath");
        if fb is not None and not (0 <= float(fb) <= 4): w.append("Full
        hb = iv("HalfBath");
        if hb is not None and not (0 <= float(hb) <= 2): w.append("Half
    except Exception:
        pass
    return w

warns = _warns_for(record)
if warns:
    print("\nNotes about your input:")
    for msg in warns:
        print(f" - {msg}")
```

```
# 2) Feature engineering and alignment
df_raw = pd.DataFrame([record])
df_eng = engineer_features(df_raw)
expected = expected_columns_from_models([reg_model, cls_model])
df_ready = align_columns(df_eng, expected)

# 3) Predict
try:
    y_log = reg_model.predict(df_ready)[0]         # model output is lo
    saleprice_usd = float(np.expm1(y_log))         # convert back to do
except Exception as e:
    raise RuntimeError(f"Regression prediction error: {e}")

try:
    quality_label = str(cls_model.predict(df_ready)[0])
    classes = list(getattr(cls_model, "classes_", []))
    probs = cls_model.predict_proba(df_ready)[0] if hasattr(cls_model,
except Exception as e:
    raise RuntimeError(f"Classification prediction error: {e}")

# 4) Present result + interpretation
def _fmt_probs(classes, probs):
    if classes and probs is not None:
        return ", ".join([f"{c}={float(p):.3f}" for c, p in zip(classes
    return "N/A"

confidence_note = ""
if probs is not None:
    maxp = float(np.max(probs))
    if maxp >= 0.80:
        confidence_note = "Model is confident in the quality category."
    elif maxp >= 0.60:
        confidence_note = "Moderate confidence; adjacent classes may al
    else:
        confidence_note = "Low confidence; the house may sit between ad

interp_html = f"""
<br/>
<br/>
<h3 style="color:green">Prediction Results</h3>
<div style='font-family:Inter,system-ui,Arial; line-height:1.5; margin-
  <div><b>Predicted SalePrice:</b> ${saleprice_usd:,.2f}</div>
  <div><b>Predicted QualityCategory:</b> {quality_label}</div>
  <div><b>Class probabilities:</b> {_fmt_probs(classes, probs)}</div>
```

```
  <div style='margin-top:6px'><i>{confidence_note}</i></div>
  <div style='margin-top:10px'>
    <b>How to read this:</b>
    <ul>
      <li>SalePrice is a model estimate in USD based on your inputs. It
      <li>QualityCategory summarizes perceived quality (Low = OverallQu
      <li>Missing fields were imputed. Adding optional fields (e.g., ga
    </ul>
  </div>
</div>
"""
display(HTML(interp_html))
```

How to enter your house details

- Type a single line of comma-separated key=value pairs.
- Keys are case-sensitive. Example keys: `OverallQual`, `YearBuilt`, `GrLivArea`, `LotArea`, `FullBath`, `HalfBath`, `YrSold`, etc.
- Strings with spaces should be quoted: `Neighborhood="Old Town"`. Unknown categorical values are fine.
- You can omit fields; missing values are imputed. More fields = better accuracy.

Recommended minimum fields (good accuracy):
`OverallQual, YearBuilt, YearRemodAdd, YrSold, GrLivArea, TotalBsmtSF, LotArea, FullBath, HalfBath`

Pick an example (A/B/C) or paste your own full one-liner:

1. **A — Mid-tier family home**
   `OverallQual=7,YearBuilt=2003,YearRemodAdd=2003,YrSold=2008,GrLivAr`
2. **B — Entry-level basic home**
   `OverallQual=4,YearBuilt=1975,YearRemodAdd=1975,YrSold=2007,GrLivAr`
3. **C — High-end newer home**
   `OverallQual=9,YearBuilt=2006,YearRemodAdd=2007,YrSold=2010,GrLivAr`

Type **A**, **B**, or **C** to use an example. Or paste your full one-line input.

`Your choice (A/B/C) OR paste a full one-liner: OverallQual=4,YearBuilt=`

Prediction Results

Predicted SalePrice: $63,107.11
Predicted QualityCategory: Medium
Class probabilities: High=0.000, Low=0.001, Medium=0.999

Model is confident in the quality category.

How to read this:

- SalePrice is a model estimate in USD based on your inputs. It's not a lookup; the model learned patterns and applies them here.
- QualityCategory summarizes perceived quality (Low = OverallQual 1–4, Medium = 5–7, High = 8–10). The classifier may disagree with raw `OverallQual` if other features suggest otherwise.