Final Project - Neural Networks and Deep Learning

Course Code : 2024S-T3 AML 3104

Group Members

- **Mahmood Hossain**
- ID : c0896079

- **Nilesh Khurana**

- ID : c0894394

- **Chanpreet Kaur**

- ID : c0907021

- **Rajia Bano**

- ID : c0907016

Introduction

Predicting real estate prices is a complex task influenced by a myriad of factors. In this project, we aim to leverage deep learning techniques to predict the sale price of properties. By analyzing a diverse set of features ranging from physical characteristics of the property to its location and the quality of various aspects of the building, we can develop a robust model that captures the intricate relationships within the data. The goal is to provide accurate predictions that can aid in decision-making processes for buyers, sellers, and real estate professionals.

Abstract

This project explores the application of deep learning algorithms to predict real estate prices using a comprehensive dataset. The dataset includes 81 variables detailing various attributes of properties, such as building class, zoning classification, lot size, utility types, neighborhood characteristics, and overall quality and condition of the buildings. By employing advanced neural network architectures, we aim to create a predictive model that accurately forecasts the sale price of properties. The model's performance will be evaluated using appropriate metrics, and the results will be analyzed to understand the influence of different features on property prices.

Project Objective

The primary objective of this project is to develop a deep learning model capable of predicting the sale price of properties based on a diverse set of features. The specific goals are:

1. **Data Preprocessing**: Clean and preprocess the dataset to handle missing values, encode categorical variables, and scale numerical features.
2. **Feature Engineering**: Explore and engineer features to enhance the model's predictive power.

3. **Model Development**: Design and implement various deep learning architectures to find the most effective model for price prediction.
4. **Model Evaluation**: Assess the performance of the models using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared.
5. **Feature Analysis**: Investigate the impact of different features on the prediction outcomes to gain insights into the key drivers of property prices.
6. **Optimization and Tuning**: Optimize the model through hyperparameter tuning to achieve the best possible performance.
7. **Deployment**: Create a deployment strategy for the model to be used in real-world applications, providing valuable predictions for stakeholders.

By accomplishing these objectives, the project aims to provide a reliable tool for predicting real estate prices, contributing to more informed decision-making in the real estate market.

Project Roles

- MAHMOOD HOSSAIN : DATA ANALYSIS, MODELLING AND DEPLOYMENT
- CHANPREET KAUR : DATA ANALYSIS AND EXPLORATION
- RAJIA BANO : FEATURE ENGINEERING AND IMPORTANCE
- NILESH KHURANA : DATA CLEANING, FEATURE IMPORTANCE

Resoure Links

- Dataset Link : https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data
- Repository Link : https://github.com/farsim-hossain/house_price_prediction_ames_iowa.git

In [2]:
```
# loading the libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from geopy.geocoders import Nominatim
import folium
from IPython.display import display
import time
import re
import os
from folium.plugins import MarkerCluster
from folium.plugins import HeatMap
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
from sklearn.datasets import make_regression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.inspection import permutation_importance
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import f_regression
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
import tensorflow as tf
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.base import BaseEstimator, RegressorMixin
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from scipy.stats import skew
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
import time
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from scipy.stats import uniform, randint
import pickle
from tabulate import tabulate
```

In [ ]:
```
# Set options to display all columns
pd.set_option('display.max_columns', None)

# Set options to display all rows
pd.set_option('display.max_rows', None)
```

In [ ]:
```
# reading the datasets and combining the train and test sets

df_train = pd.read_csv("/kaggle/input/house-prices-advanced-regression-techniques/train.
df_test = pd.read_csv("/kaggle/input/house-prices-advanced-regression-techniques/test.cs

df = pd.concat([df_train, df_test], ignore_index=True).reset_index(drop=True)
```

In [ ]:
```
# getting comprehensive information about the dataset

def check_df(dataframe, head=5):
    print("SHAPE".center(70,"-"))
    print(dataframe.shape)
    print("INFO".center(70,"-"))
    print(dataframe.info())
    print("NUNIQUE".center(70,"-"))
    print(dataframe.nunique())
    print("MISSING VALUES".center(70,"-"))
    print(dataframe.isnull().sum())
    print("DUPLICATED VALUES".center(70,"-"))
    print(dataframe.duplicated().sum())


check_df(df)
```

```
--------------------------------SHAPE---------------------------------
(2919, 81)
---------------------------------INFO---------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2919 entries, 0 to 2918
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             2919 non-null   int64  
 1   MSSubClass     2919 non-null   int64  
 2   MSZoning       2915 non-null   object 
 3   LotFrontage    2433 non-null   float64
 4   LotArea        2919 non-null   int64  
 5   Street         2919 non-null   object 
 6   Alley          198 non-null    object 
 7   LotShape       2919 non-null   object 
 8   LandContour    2919 non-null   object 
 9   Utilities      2917 non-null   object 
 10  LotConfig      2919 non-null   object
 11  LandSlope      2919 non-null   object
 12  Neighborhood   2919 non-null   object
 13  Condition1     2919 non-null   object
 14  Condition2     2919 non-null   object
 15  BldgType       2919 non-null   object
 16  HouseStyle     2919 non-null   object
 17  OverallQual    2919 non-null   int64
 18  OverallCond    2919 non-null   int64
 19  YearBuilt      2919 non-null   int64
 20  YearRemodAdd   2919 non-null   int64
 21  RoofStyle      2919 non-null   object
 22  RoofMatl       2919 non-null   object
 23  Exterior1st    2918 non-null   object
 24  Exterior2nd    2918 non-null   object
 25  MasVnrType     1153 non-null   object
 26  MasVnrArea     2896 non-null   float64
 27  ExterQual      2919 non-null   object
 28  ExterCond      2919 non-null   object
 29  Foundation     2919 non-null   object
 30  BsmtQual       2838 non-null   object
 31  BsmtCond       2837 non-null   object
 32  BsmtExposure   2837 non-null   object
 33  BsmtFinType1   2840 non-null   object
 34  BsmtFinSF1     2918 non-null   float64
 35  BsmtFinType2   2839 non-null   object
 36  BsmtFinSF2     2918 non-null   float64
 37  BsmtUnfSF      2918 non-null   float64
 38  TotalBsmtSF    2918 non-null   float64
 39  Heating        2919 non-null   object
 40  HeatingQC      2919 non-null   object
 41  CentralAir     2919 non-null   object
 42  Electrical     2918 non-null   object
 43  1stFlrSF       2919 non-null   int64
 44  2ndFlrSF       2919 non-null   int64
 45  LowQualFinSF   2919 non-null   int64
 46  GrLivArea      2919 non-null   int64
 47  BsmtFullBath   2917 non-null   float64
 48  BsmtHalfBath   2917 non-null   float64
 49  FullBath       2919 non-null   int64
 50  HalfBath       2919 non-null   int64
 51  BedroomAbvGr   2919 non-null   int64
 52  KitchenAbvGr   2919 non-null   int64
 53  KitchenQual    2918 non-null   object
 54  TotRmsAbvGrd   2919 non-null   int64
 55  Functional     2917 non-null   object
 56  Fireplaces     2919 non-null   int64
 57  FireplaceQu    1499 non-null   object
 58  GarageType     2762 non-null   object
 59  GarageYrBlt    2760 non-null   float64
 60  GarageFinish   2760 non-null   object
 61  GarageCars     2918 non-null   float64
 62  GarageArea     2918 non-null   float64
 63  GarageQual     2760 non-null   object
 64  GarageCond     2760 non-null   object
 65  PavedDrive     2919 non-null   object
 66  WoodDeckSF     2919 non-null   int64
 67  OpenPorchSF    2919 non-null   int64
 68  EnclosedPorch  2919 non-null   int64
 69  3SsnPorch      2919 non-null   int64
 70  ScreenPorch    2919 non-null   int64
 71  PoolArea       2919 non-null   int64
 72  PoolQC         10 non-null     object
 73  Fence          571 non-null    object
 74  MiscFeature    105 non-null    object
 75  MiscVal        2919 non-null   int64
 76  MoSold         2919 non-null   int64
 77  YrSold         2919 non-null   int64
 78  SaleType       2918 non-null   object
 79  SaleCondition  2919 non-null   object
 80  SalePrice      1460 non-null   float64
dtypes: float64(12), int64(26), object(43)
memory usage: 1.8+ MB
None
-------------------------------NUNIQUE--------------------------------
Id               2919
MSSubClass         16
MSZoning            5
LotFrontage       128
LotArea          1951
Street              2
Alley               2
LotShape            4
LandContour         4
Utilities           2
LotConfig           5
LandSlope           3
Neighborhood       25
Condition1          9
Condition2          8
BldgType            5
HouseStyle          8
OverallQual        10
OverallCond         9
YearBuilt         118
YearRemodAdd       61
RoofStyle           6
RoofMatl            8
Exterior1st        15
Exterior2nd        16
MasVnrType          3
MasVnrArea        444
ExterQual           4
ExterCond           5
Foundation          6
BsmtQual            4
BsmtCond            4
BsmtExposure        4
BsmtFinType1        6
BsmtFinSF1        991
BsmtFinType2        6
BsmtFinSF2        272
BsmtUnfSF        1135
TotalBsmtSF      1058
Heating             6
HeatingQC           5
CentralAir          2
Electrical          5
1stFlrSF         1083
2ndFlrSF          635
LowQualFinSF       36
GrLivArea        1292
BsmtFullBath        4
BsmtHalfBath        3
FullBath            5
HalfBath            3
BedroomAbvGr        8
KitchenAbvGr        4
KitchenQual         4
TotRmsAbvGrd       14
Functional          7
Fireplaces          5
```

```
FireplaceQu          5
GarageType           6
GarageYrBlt        103
GarageFinish         3
GarageCars           6
GarageArea         603
GarageQual           5
GarageCond           5
PavedDrive           3
WoodDeckSF         379
OpenPorchSF        252
EnclosedPorch      183
3SsnPorch           31
ScreenPorch        121
PoolArea            14
PoolQC               3
Fence                4
MiscFeature          4
MiscVal             38
MoSold              12
YrSold               5
SaleType             9
SaleCondition        6
SalePrice          663
dtype: int64
----------------------------MISSING VALUES----------------------------
Id                   0
MSSubClass           0
MSZoning             4
LotFrontage        486
LotArea              0
Street               0
Alley             2721
LotShape             0
LandContour          0
Utilities            2
LotConfig            0
LandSlope            0
Neighborhood         0
Condition1           0
Condition2           0
BldgType             0
HouseStyle           0
OverallQual          0
OverallCond          0
YearBuilt            0
YearRemodAdd         0
RoofStyle            0
RoofMatl             0
Exterior1st          1
Exterior2nd          1
MasVnrType        1766
MasVnrArea          23
ExterQual            0
ExterCond            0
Foundation           0
BsmtQual            81
BsmtCond            82
BsmtExposure        82
BsmtFinType1        79
BsmtFinSF1           1
BsmtFinType2        80
BsmtFinSF2           1
BsmtUnfSF            1
TotalBsmtSF          1
Heating              0
```

```
HeatingQC           0
CentralAir          0
Electrical          1
1stFlrSF            0
2ndFlrSF            0
LowQualFinSF        0
GrLivArea           0
BsmtFullBath        2
BsmtHalfBath        2
FullBath            0
HalfBath            0
BedroomAbvGr        0
KitchenAbvGr        0
KitchenQual         1
TotRmsAbvGrd        0
Functional          2
Fireplaces          0
FireplaceQu      1420
GarageType        157
GarageYrBlt       159
GarageFinish      159
GarageCars          1
GarageArea          1
GarageQual        159
GarageCond        159
PavedDrive          0
WoodDeckSF          0
OpenPorchSF         0
EnclosedPorch       0
3SsnPorch           0
ScreenPorch         0
PoolArea            0
PoolQC           2909
Fence            2348
MiscFeature      2814
MiscVal             0
MoSold              0
YrSold              0
SaleType            1
SaleCondition       0
SalePrice        1459
dtype: int64
--------------------------DUPLICATED VALUES--------------------------
0
```

As we can see, there are few columns which have a good number of data missing. We have to find out a way to deal with these missing values.

In [ ]:
```
# what is the percentage of data missing
missing_percentage = df.isnull().mean() * 100
missing_percentage
```

Out[ ]:
```
Id                0.000000
MSSubClass        0.000000
MSZoning          0.137033
LotFrontage      16.649538
LotArea           0.000000
Street            0.000000
Alley            93.216855
LotShape          0.000000
LandContour       0.000000
Utilities         0.068517
LotConfig         0.000000
LandSlope         0.000000
Neighborhood      0.000000
```

```
Condition1        0.000000
Condition2        0.000000
BldgType          0.000000
HouseStyle        0.000000
OverallQual       0.000000
OverallCond       0.000000
YearBuilt         0.000000
YearRemodAdd      0.000000
RoofStyle         0.000000
RoofMatl          0.000000
Exterior1st       0.034258
Exterior2nd       0.034258
MasVnrType       60.500171
MasVnrArea        0.787941
ExterQual         0.000000
ExterCond         0.000000
Foundation        0.000000
BsmtQual          2.774923
BsmtCond          2.809181
BsmtExposure      2.809181
BsmtFinType1      2.706406
BsmtFinSF1        0.034258
BsmtFinType2      2.740665
BsmtFinSF2        0.034258
BsmtUnfSF         0.034258
TotalBsmtSF       0.034258
Heating           0.000000
HeatingQC         0.000000
CentralAir        0.000000
Electrical        0.034258
1stFlrSF          0.000000
2ndFlrSF          0.000000
LowQualFinSF      0.000000
GrLivArea         0.000000
BsmtFullBath      0.068517
BsmtHalfBath      0.068517
FullBath          0.000000
HalfBath          0.000000
BedroomAbvGr      0.000000
KitchenAbvGr      0.000000
KitchenQual       0.034258
TotRmsAbvGrd      0.000000
Functional        0.068517
Fireplaces        0.000000
FireplaceQu      48.646797
GarageType        5.378554
GarageYrBlt       5.447071
GarageFinish      5.447071
GarageCars        0.034258
GarageArea        0.034258
GarageQual        5.447071
GarageCond        5.447071
PavedDrive        0.000000
WoodDeckSF        0.000000
OpenPorchSF       0.000000
EnclosedPorch     0.000000
3SsnPorch         0.000000
ScreenPorch       0.000000
PoolArea          0.000000
PoolQC           99.657417
Fence            80.438506
MiscFeature      96.402878
MiscVal           0.000000
MoSold            0.000000
YrSold            0.000000
SaleType          0.034258
```

```
SaleCondition     0.000000
SalePrice        49.982871
dtype: float64
```

Columns like **Alley, PoolQC, Fence and MiscFeature** have almost **80-90%** data missing. We can think of removing these features.

In [ ]:
```
# looking at the dataset
df.head()
```

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfi
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Insid
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Insid
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corne
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR

We need to know what each of these features actually mean. We have a data description file to which we can look at

In [ ]:
```
# data description

with open('/kaggle/input/house-prices-advanced-regression-techniques/data_description.tx
    description = file.read()

print(description)
```

```
MSSubClass: Identifies the type of dwelling involved in the sale.

        20      1-STORY 1946 & NEWER ALL STYLES
        30      1-STORY 1945 & OLDER
        40      1-STORY W/FINISHED ATTIC ALL AGES
        45      1-1/2 STORY - UNFINISHED ALL AGES
        50      1-1/2 STORY FINISHED ALL AGES
        60      2-STORY 1946 & NEWER
        70      2-STORY 1945 & OLDER
        75      2-1/2 STORY ALL AGES
        80      SPLIT OR MULTI-LEVEL
        85      SPLIT FOYER
        90      DUPLEX - ALL STYLES AND AGES
       120      1-STORY PUD (Planned Unit Development) - 1946 & NEWER
       150      1-1/2 STORY PUD - ALL AGES
       160      2-STORY PUD - 1946 & NEWER
       180      PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
       190      2 FAMILY CONVERSION - ALL STYLES AND AGES

MSZoning: Identifies the general zoning classification of the sale.

       A        Agriculture
       C        Commercial
       FV       Floating Village Residential
       I        Industrial
       RH       Residential High Density
       RL       Residential Low Density
       RP       Residential Low Density Park
       RM       Residential Medium Density

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet
```

Street: Type of road access to property

Grvl	Gravel
Pave	Paved

Alley: Type of alley access to property

Grvl	Gravel
Pave	Paved
NA	No alley access

LotShape: General shape of property

Reg	Regular
IR1	Slightly irregular
IR2	Moderately Irregular
IR3	Irregular

LandContour: Flatness of the property

Lvl	Near Flat/Level
Bnk	Banked - Quick and significant rise from street grade to building
HLS	Hillside - Significant slope from side to side
Low	Depression

Utilities: Type of utilities available

AllPub	All public Utilities (E,G,W,& S)
NoSewr	Electricity, Gas, and Water (Septic Tank)
NoSeWa	Electricity and Gas Only
ELO	Electricity only

LotConfig: Lot configuration

Inside	Inside lot
Corner	Corner lot
CulDSac	Cul-de-sac
FR2	Frontage on 2 sides of property
FR3	Frontage on 3 sides of property

LandSlope: Slope of property

Gtl	Gentle slope
Mod	Moderate Slope
Sev	Severe Slope

Neighborhood: Physical locations within Ames city limits

Blmngtn	Bloomington Heights
Blueste	Bluestem
BrDale	Briardale
BrkSide	Brookside
ClearCr	Clear Creek
CollgCr	College Creek
Crawfor	Crawford
Edwards	Edwards
Gilbert	Gilbert
IDOTRR	Iowa DOT and Rail Road
MeadowV	Meadow Village
Mitchel	Mitchell
Names	North Ames
NoRidge	Northridge
NPkVill	Northpark Villa
NridgHt	Northridge Heights
NWAmes	Northwest Ames

OldTown	Old Town
SWISU	South & West of Iowa State University
Sawyer	Sawyer
SawyerW	Sawyer West
Somerst	Somerset
StoneBr	Stone Brook
Timber	Timberland
Veenker	Veenker

Condition1: Proximity to various conditions

Artery	Adjacent to arterial street
Feedr	Adjacent to feeder street
Norm	Normal
RRNn	Within 200' of North-South Railroad
RRAn	Adjacent to North-South Railroad
PosN	Near positive off-site feature--park, greenbelt, etc.
PosA	Adjacent to postive off-site feature
RRNe	Within 200' of East-West Railroad
RRAe	Adjacent to East-West Railroad

Condition2: Proximity to various conditions (if more than one is present)

Artery	Adjacent to arterial street
Feedr	Adjacent to feeder street
Norm	Normal
RRNn	Within 200' of North-South Railroad
RRAn	Adjacent to North-South Railroad
PosN	Near positive off-site feature--park, greenbelt, etc.
PosA	Adjacent to postive off-site feature
RRNe	Within 200' of East-West Railroad
RRAe	Adjacent to East-West Railroad

BldgType: Type of dwelling

1Fam	Single-family Detached
2FmCon	Two-family Conversion; originally built as one-family dwelling
Duplx	Duplex
TwnhsE	Townhouse End Unit
TwnhsI	Townhouse Inside Unit

HouseStyle: Style of dwelling

1Story	One story
1.5Fin	One and one-half story: 2nd level finished
1.5Unf	One and one-half story: 2nd level unfinished
2Story	Two story
2.5Fin	Two and one-half story: 2nd level finished
2.5Unf	Two and one-half story: 2nd level unfinished
SFoyer	Split Foyer
SLvl	Split Level

OverallQual: Rates the overall material and finish of the house

10	Very Excellent
9	Excellent
8	Very Good
7	Good
6	Above Average
5	Average
4	Below Average
3	Fair
2	Poor
1	Very Poor

OverallCond: Rates the overall condition of the house

10	Very Excellent
9	Excellent
8	Very Good
7	Good
6	Above Average
5	Average
4	Below Average
3	Fair
2	Poor
1	Very Poor

YearBuilt: Original construction date

YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)

RoofStyle: Type of roof

Flat	Flat
Gable	Gable
Gambrel	Gabrel (Barn)
Hip	Hip
Mansard	Mansard
Shed	Shed

RoofMatl: Roof material

ClyTile	Clay or Tile
CompShg	Standard (Composite) Shingle
Membran	Membrane
Metal	Metal
Roll	Roll
Tar&Grv	Gravel & Tar
WdShake	Wood Shakes
WdShngl	Wood Shingles

Exterior1st: Exterior covering on house

AsbShng	Asbestos Shingles
AsphShn	Asphalt Shingles
BrkComm	Brick Common
BrkFace	Brick Face
CBlock	Cinder Block
CemntBd	Cement Board
HdBoard	Hard Board
ImStucc	Imitation Stucco
MetalSd	Metal Siding
Other	Other
Plywood	Plywood
PreCast	PreCast
Stone	Stone
Stucco	Stucco
VinylSd	Vinyl Siding
Wd Sdng	Wood Siding
WdShing	Wood Shingles

Exterior2nd: Exterior covering on house (if more than one material)

AsbShng	Asbestos Shingles
AsphShn	Asphalt Shingles
BrkComm	Brick Common
BrkFace	Brick Face
CBlock	Cinder Block
CemntBd	Cement Board
HdBoard	Hard Board
ImStucc	Imitation Stucco

MetalSd	Metal Siding
Other	Other
Plywood	Plywood
PreCast	PreCast
Stone	Stone
Stucco	Stucco
VinylSd	Vinyl Siding
Wd Sdng	Wood Siding
WdShing	Wood Shingles

MasVnrType: Masonry veneer type

BrkCmn	Brick Common
BrkFace	Brick Face
CBlock	Cinder Block
None	None
Stone	Stone

MasVnrArea: Masonry veneer area in square feet

ExterQual: Evaluates the quality of the material on the exterior

Ex	Excellent
Gd	Good
TA	Average/Typical
Fa	Fair
Po	Poor

ExterCond: Evaluates the present condition of the material on the exterior

Ex	Excellent
Gd	Good
TA	Average/Typical
Fa	Fair
Po	Poor

Foundation: Type of foundation

BrkTil	Brick & Tile
CBlock	Cinder Block
PConc	Poured Contrete
Slab	Slab
Stone	Stone
Wood	Wood

BsmtQual: Evaluates the height of the basement

Ex	Excellent (100+ inches)
Gd	Good (90-99 inches)
TA	Typical (80-89 inches)
Fa	Fair (70-79 inches)
Po	Poor (<70 inches
NA	No Basement

BsmtCond: Evaluates the general condition of the basement

Ex	Excellent
Gd	Good
TA	Typical - slight dampness allowed
Fa	Fair - dampness or some cracking or settling
Po	Poor - Severe cracking, settling, or wetness
NA	No Basement

BsmtExposure: Refers to walkout or garden level walls

Gd	Good Exposure

Av	Average Exposure (split levels or foyers typically score average or above)
Mn	Mimimum Exposure
No	No Exposure
NA	No Basement

BsmtFinType1: Rating of basement finished area

GLQ	Good Living Quarters
ALQ	Average Living Quarters
BLQ	Below Average Living Quarters
Rec	Average Rec Room
LwQ	Low Quality
Unf	Unfinshed
NA	No Basement

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types)

GLQ	Good Living Quarters
ALQ	Average Living Quarters
BLQ	Below Average Living Quarters
Rec	Average Rec Room
LwQ	Low Quality
Unf	Unfinshed
NA	No Basement

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

Floor	Floor Furnace
GasA	Gas forced warm air furnace
GasW	Gas hot water or steam heat
Grav	Gravity furnace
OthW	Hot water or steam heat other than gas
Wall	Wall furnace

HeatingQC: Heating quality and condition

Ex	Excellent
Gd	Good
TA	Average/Typical
Fa	Fair
Po	Poor

CentralAir: Central air conditioning

N	No
Y	Yes

Electrical: Electrical system

SBrkr	Standard Circuit Breakers & Romex
FuseA	Fuse Box over 60 AMP and all Romex wiring (Average)
FuseF	60 AMP Fuse Box and mostly Romex wiring (Fair)
FuseP	60 AMP Fuse Box and mostly knob & tube wiring (poor)
Mix	Mixed

1stFlrSF: First Floor square feet

```
2ndFlrSF: Second floor square feet

LowQualFinSF: Low quality finished square feet (all floors)

GrLivArea: Above grade (ground) living area square feet

BsmtFullBath: Basement full bathrooms

BsmtHalfBath: Basement half bathrooms

FullBath: Full bathrooms above grade

HalfBath: Half baths above grade

Bedroom: Bedrooms above grade (does NOT include basement bedrooms)

Kitchen: Kitchens above grade

KitchenQual: Kitchen quality

       Ex       Excellent
       Gd       Good
       TA       Typical/Average
       Fa       Fair
       Po       Poor

TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

Functional: Home functionality (Assume typical unless deductions are warranted)

       Typ      Typical Functionality
       Min1     Minor Deductions 1
       Min2     Minor Deductions 2
       Mod      Moderate Deductions
       Maj1     Major Deductions 1
       Maj2     Major Deductions 2
       Sev      Severely Damaged
       Sal      Salvage only

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

       Ex       Excellent - Exceptional Masonry Fireplace
       Gd       Good - Masonry Fireplace in main level
       TA       Average - Prefabricated Fireplace in main living area or Masonry Firepla
ce in basement
       Fa       Fair - Prefabricated Fireplace in basement
       Po       Poor - Ben Franklin Stove
       NA       No Fireplace

GarageType: Garage location

       2Types   More than one type of garage
       Attchd   Attached to home
       Basment  Basement Garage
       BuiltIn  Built-In (Garage part of house - typically has room above garage)
       CarPort  Car Port
       Detchd   Detached from home
       NA       No Garage

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

       Fin      Finished
       RFn      Rough Finished
       Unf      Unfinished
       NA       No Garage

GarageCars: Size of garage in car capacity

GarageArea: Size of garage in square feet

GarageQual: Garage quality

       Ex       Excellent
       Gd       Good
       TA       Typical/Average
       Fa       Fair
       Po       Poor
       NA       No Garage

GarageCond: Garage condition

       Ex       Excellent
       Gd       Good
       TA       Typical/Average
       Fa       Fair
       Po       Poor
       NA       No Garage

PavedDrive: Paved driveway

       Y        Paved
       P        Partial Pavement
       N        Dirt/Gravel

WoodDeckSF: Wood deck area in square feet

OpenPorchSF: Open porch area in square feet

EnclosedPorch: Enclosed porch area in square feet

3SsnPorch: Three season porch area in square feet

ScreenPorch: Screen porch area in square feet

PoolArea: Pool area in square feet

PoolQC: Pool quality

       Ex       Excellent
       Gd       Good
       TA       Average/Typical
       Fa       Fair
       NA       No Pool

Fence: Fence quality

       GdPrv    Good Privacy
       MnPrv    Minimum Privacy
       GdWo     Good Wood
       MnWw     Minimum Wood/Wire
       NA       No Fence

MiscFeature: Miscellaneous feature not covered in other categories

       Elev     Elevator
       Gar2     2nd Garage (if not described in garage section)
       Othr     Other
       Shed     Shed (over 100 SF)
       TenC     Tennis Court
       NA       None

MiscVal: $Value of miscellaneous feature

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

       WD       Warranty Deed - Conventional
       CWD      Warranty Deed - Cash
       VWD      Warranty Deed - VA Loan
       New      Home just constructed and sold
       COD      Court Officer Deed/Estate
       Con      Contract 15% Down payment regular terms
       ConLw    Contract Low Down payment and low interest
       ConLI    Contract Low Interest
       ConLD    Contract Low Down
       Oth      Other

SaleCondition: Condition of sale

       Normal   Normal Sale
       Abnorml  Abnormal Sale -  trade, foreclosure, short sale
       AdjLand  Adjoining Land Purchase
       Alloca   Allocation - two linked properties with separate deeds, typically condo
with a garage unit
       Family   Sale between family members
       Partial  Home was not completed when last assessed (associated with New Homes)
```

Data Cleaning

There are variety of features in the dataset. We will take a look at the features but before that we need to deeal with the missing values. As we have mentioned already, few features will not be able to help our model which have more than 80% missing values. We will remove them now.

In [ ]:
```
# remove features with a high number of missing values

rem_cols = ['Alley', 'PoolQC', 'Fence', 'MiscFeature']
df.drop(columns=rem_cols, inplace=True)
```

FireplaceQu and MasVnrType columns have 40 - 60% data missing. We need to deal with these columns but before that lets take a look at what they mean. MasVnrType means Masonry veneer type and FireplaceQu means Fireplace Quality. For MasVnrType, we will use 'mode' and for FireplaceQu we will use 'mean' to fill the null values. But before that, we need to encode the categorical columns to number.

In [ ]:
```
# numeric columns
num_cols = df.select_dtypes(include=['number']).columns.tolist()

# non-numeric columns
non_num_cols = df.select_dtypes(exclude=['number']).columns.tolist()
```

In [ ]:
```
# ensuring the data types of both types of columns
num_cols_dtypes = df[num_cols].dtypes
non_num_cols_dtypes = df[non_num_cols].dtypes
```

In [ ]:
```
num_cols_dtypes
```

Out[ ]:
```
Id                 int64
MSSubClass         int64
LotFrontage      float64
LotArea            int64
OverallQual        int64
OverallCond        int64
YearBuilt          int64
YearRemodAdd       int64
MasVnrArea       float64
BsmtFinSF1       float64
BsmtFinSF2       float64
BsmtUnfSF        float64
TotalBsmtSF      float64
1stFlrSF           int64
2ndFlrSF           int64
LowQualFinSF       int64
GrLivArea          int64
BsmtFullBath     float64
BsmtHalfBath     float64
FullBath           int64
HalfBath           int64
BedroomAbvGr       int64
KitchenAbvGr       int64
TotRmsAbvGrd       int64
Fireplaces         int64
GarageYrBlt      float64
GarageCars       float64
GarageArea       float64
WoodDeckSF         int64
OpenPorchSF        int64
EnclosedPorch      int64
3SsnPorch          int64
ScreenPorch        int64
PoolArea           int64
MiscVal            int64
MoSold             int64
YrSold             int64
SalePrice        float64
dtype: object
```

In [ ]:
```
non_num_cols_dtypes
```

Out[ ]:
```
MSZoning         object
Street           object
LotShape         object
LandContour      object
Utilities        object
LotConfig        object
LandSlope        object
Neighborhood     object
Condition1       object
Condition2       object
BldgType         object
HouseStyle       object
RoofStyle        object
RoofMatl         object
Exterior1st      object
Exterior2nd      object
MasVnrType       object
ExterQual        object
ExterCond        object
Foundation       object
BsmtQual         object
BsmtCond         object
BsmtExposure     object
```

```
BsmtFinType1     object
BsmtFinType2     object
Heating          object
HeatingQC        object
CentralAir       object
Electrical       object
KitchenQual      object
Functional       object
FireplaceQu      object
GarageType       object
GarageFinish     object
GarageQual       object
GarageCond       object
PavedDrive       object
SaleType         object
SaleCondition    object
dtype: object
```

In [ ]:

```python
# label encoding
encoders = {}

# Label encode each categorical column
for column in non_num_cols:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    encoders[column] = le

# Print the DataFrame with encoded categorical columns
print("DataFrame with Label Encoded Categorical Columns:")
df.head()
```

```
DataFrame with Label Encoded Categorical Columns:
```

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	Lan
0	1	60	3	65.0	8450	1	3	3	0	4	
1	2	20	3	80.0	9600	1	3	3	0	2	
2	3	60	3	68.0	11250	1	0	3	0	4	
3	4	70	3	60.0	9550	1	0	3	0	0	
4	5	60	3	84.0	14260	1	0	3	0	2	

In [ ]:

```python
# We can extract the labels of the encoded values if we need to see
print("\nLabel Encoders:")
for column, le in encoders.items():
    print(f"Column: {column}")
    print(dict(zip(le.classes_, le.transform(le.classes_))))
```

```
Label Encoders:
Column: MSZoning
{'C (all)': 0, 'FV': 1, 'RH': 2, 'RL': 3, 'RM': 4, nan: 5}
Column: Street
{'Grvl': 0, 'Pave': 1}
Column: LotShape
{'IR1': 0, 'IR2': 1, 'IR3': 2, 'Reg': 3}
Column: LandContour
{'Bnk': 0, 'HLS': 1, 'Low': 2, 'Lvl': 3}
Column: Utilities
{'AllPub': 0, 'NoSeWa': 1, nan: 2}
Column: LotConfig
{'Corner': 0, 'CulDSac': 1, 'FR2': 2, 'FR3': 3, 'Inside': 4}
Column: LandSlope
{'Gtl': 0, 'Mod': 1, 'Sev': 2}
Column: Neighborhood
```

```
{'Blmngtn': 0, 'Blueste': 1, 'BrDale': 2, 'BrkSide': 3, 'ClearCr': 4, 'CollgCr': 5, 'Cra
wfor': 6, 'Edwards': 7, 'Gilbert': 8, 'IDOTRR': 9, 'MeadowV': 10, 'Mitchel': 11, 'NAme
s': 12, 'NPkVill': 13, 'NWAmes': 14, 'NoRidge': 15, 'NridgHt': 16, 'OldTown': 17, 'SWIS
U': 18, 'Sawyer': 19, 'SawyerW': 20, 'Somerst': 21, 'StoneBr': 22, 'Timber': 23, 'Veenke
r': 24}
Column: Condition1
{'Artery': 0, 'Feedr': 1, 'Norm': 2, 'PosA': 3, 'PosN': 4, 'RRAe': 5, 'RRAn': 6, 'RRNe':
7, 'RRNn': 8}
Column: Condition2
{'Artery': 0, 'Feedr': 1, 'Norm': 2, 'PosA': 3, 'PosN': 4, 'RRAe': 5, 'RRAn': 6, 'RRNn':
7}
Column: BldgType
{'1Fam': 0, '2fmCon': 1, 'Duplex': 2, 'Twnhs': 3, 'TwnhsE': 4}
Column: HouseStyle
{'1.5Fin': 0, '1.5Unf': 1, '1Story': 2, '2.5Fin': 3, '2.5Unf': 4, '2Story': 5, 'SFoyer':
6, 'SLvl': 7}
Column: RoofStyle
{'Flat': 0, 'Gable': 1, 'Gambrel': 2, 'Hip': 3, 'Mansard': 4, 'Shed': 5}
Column: RoofMatl
{'ClyTile': 0, 'CompShg': 1, 'Membran': 2, 'Metal': 3, 'Roll': 4, 'Tar&Grv': 5, 'WdShak
e': 6, 'WdShngl': 7}
Column: Exterior1st
{'AsbShng': 0, 'AsphShn': 1, 'BrkComm': 2, 'BrkFace': 3, 'CBlock': 4, 'CemntBd': 5, 'HdB
oard': 6, 'ImStucc': 7, 'MetalSd': 8, 'Plywood': 9, 'Stone': 10, 'Stucco': 11, 'VinylS
d': 12, 'Wd Sdng': 13, 'WdShing': 14, nan: 15}
Column: Exterior2nd
{'AsbShng': 0, 'AsphShn': 1, 'Brk Cmn': 2, 'BrkFace': 3, 'CBlock': 4, 'CmentBd': 5, 'HdB
oard': 6, 'ImStucc': 7, 'MetalSd': 8, 'Other': 9, 'Plywood': 10, 'Stone': 11, 'Stucco':
12, 'VinylSd': 13, 'Wd Sdng': 14, 'Wd Shng': 15, nan: 16}
Column: MasVnrType
{'BrkCmn': 0, 'BrkFace': 1, 'Stone': 2, nan: 3}
Column: ExterQual
{'Ex': 0, 'Fa': 1, 'Gd': 2, 'TA': 3}
Column: ExterCond
{'Ex': 0, 'Fa': 1, 'Gd': 2, 'Po': 3, 'TA': 4}
Column: Foundation
{'BrkTil': 0, 'CBlock': 1, 'PConc': 2, 'Slab': 3, 'Stone': 4, 'Wood': 5}
Column: BsmtQual
{'Ex': 0, 'Fa': 1, 'Gd': 2, 'TA': 3, nan: 4}
Column: BsmtCond
{'Fa': 0, 'Gd': 1, 'Po': 2, 'TA': 3, nan: 4}
Column: BsmtExposure
{'Av': 0, 'Gd': 1, 'Mn': 2, 'No': 3, nan: 4}
Column: BsmtFinType1
{'ALQ': 0, 'BLQ': 1, 'GLQ': 2, 'LwQ': 3, 'Rec': 4, 'Unf': 5, nan: 6}
Column: BsmtFinType2
{'ALQ': 0, 'BLQ': 1, 'GLQ': 2, 'LwQ': 3, 'Rec': 4, 'Unf': 5, nan: 6}
Column: Heating
{'Floor': 0, 'GasA': 1, 'GasW': 2, 'Grav': 3, 'OthW': 4, 'Wall': 5}
Column: HeatingQC
{'Ex': 0, 'Fa': 1, 'Gd': 2, 'Po': 3, 'TA': 4}
Column: CentralAir
{'N': 0, 'Y': 1}
Column: Electrical
{'FuseA': 0, 'FuseF': 1, 'FuseP': 2, 'Mix': 3, 'SBrkr': 4, nan: 5}
Column: KitchenQual
{'Ex': 0, 'Fa': 1, 'Gd': 2, 'TA': 3, nan: 4}
Column: Functional
{'Maj1': 0, 'Maj2': 1, 'Min1': 2, 'Min2': 3, 'Mod': 4, 'Sev': 5, 'Typ': 6, nan: 7}
Column: FireplaceQu
{'Ex': 0, 'Fa': 1, 'Gd': 2, 'Po': 3, 'TA': 4, nan: 5}
Column: GarageType
{'2Types': 0, 'Attchd': 1, 'Basment': 2, 'BuiltIn': 3, 'CarPort': 4, 'Detchd': 5, nan:
6}
Column: GarageFinish
{'Fin': 0, 'RFn': 1, 'Unf': 2, nan: 3}
```

```
Column: GarageQual
{'Ex': 0, 'Fa': 1, 'Gd': 2, 'Po': 3, 'TA': 4, nan: 5}
Column: GarageCond
{'Ex': 0, 'Fa': 1, 'Gd': 2, 'Po': 3, 'TA': 4, nan: 5}
Column: PavedDrive
{'N': 0, 'P': 1, 'Y': 2}
Column: SaleType
{'COD': 0, 'CWD': 1, 'Con': 2, 'ConLD': 3, 'ConLI': 4, 'ConLw': 5, 'New': 6, 'Oth': 7,
'WD': 8, nan: 9}
Column: SaleCondition
{'Abnorml': 0, 'AdjLand': 1, 'Alloca': 2, 'Family': 3, 'Normal': 4, 'Partial': 5}
```

Now lets first deal with the large null columns mentioned earlier

In [ ]:
```
# Fill missing values in 'MasVnrType' with the mode
mas_vnr_type_mode = df['MasVnrType'].mode()[0]
df['MasVnrType'].fillna(mas_vnr_type_mode)

# Fill missing values in 'FireplaceQu' with the mean
fireplacequ_mean = df['FireplaceQu'].mean()
df['FireplaceQu'].fillna(fireplacequ_mean)
```

Out[ ]:
```
0       5
1       4
2       4
3       2
4       4
5       5
6       2
7       4
8       4
9       4
10      5
11      2
12      5
13      2
14      1
15      5
16      4
17      5
18      5
19      5
20      2
21      2
22      2
23      4
24      4
25      2
26      5
27      2
28      2
29      5
30      5
31      5
32      5
33      2
34      2
35      2
36      5
37      4
38      5
39      5
40      4
41      2
42      5
```

43	5
44	5
45	2
46	0
47	5
48	5
49	5
50	5
51	2
52	5
53	2
54	4
55	2
56	5
57	5
58	2
59	5
60	5
61	5
62	2
63	5
64	5
65	2
66	2
67	5
68	5
69	4
70	2
71	5
72	4
73	5
74	5
75	5
76	5
77	5
78	5
79	5
80	2
81	5
82	2
83	5
84	4
85	4
86	2
87	5
88	5
89	5
90	5
91	5
92	5
93	2
94	5
95	4
96	5
97	5
98	5
99	5
100	4
101	4
102	5
103	5
104	4
105	2
106	5
107	5
108	5

109	4
110	5
111	4
112	2
113	2
114	4
115	1
116	3
117	5
118	4
119	2
120	4
121	5
122	5
123	5
124	4
125	5
126	4
127	4
128	1
129	5
130	2
131	4
132	5
133	5
134	4
135	2
136	1
137	5
138	4
139	5
140	3
141	5
142	5
143	5
144	5
145	5
146	5
147	2
148	5
149	5
150	5
151	2
152	2
153	2
154	5
155	5
156	5
157	2
158	2
159	2
160	5
161	2
162	2
163	5
164	5
165	5
166	2
167	2
168	2
169	4
170	5
171	2
172	4
173	4
174	4

175	2
176	4
177	2
178	2
179	5
180	4
181	2
182	2
183	5
184	5
185	4
186	5
187	5
188	4
189	2
190	4
191	5
192	5
193	5
194	5
195	4
196	2
197	0
198	5
199	2
200	5
201	1
202	5
203	2
204	5
205	5
206	4
207	3
208	2
209	5
210	5
211	5
212	4
213	5
214	5
215	1
216	5
217	5
218	4
219	5
220	5
221	4
222	4
223	5
224	0
225	5
226	4
227	5
228	1
229	4
230	5
231	4
232	3
233	5
234	4
235	5
236	5
237	5
238	5
239	2
240	5

241	5
242	5
243	4
244	1
245	4
246	5
247	4
248	5
249	4
250	5
251	2
252	5
253	5
254	5
255	4
256	5
257	2
258	4
259	5
260	4
261	2
262	4
263	5
264	5
265	4
266	4
267	2
268	2
269	1
270	2
271	4
272	2
273	2
274	5
275	5
276	5
277	5
278	0
279	4
280	4
281	5
282	2
283	2
284	5
285	5
286	2
287	5
288	5
289	5
290	2
291	5
292	2
293	4
294	2
295	5
296	5
297	4
298	2
299	2
300	2
301	4
302	4
303	5
304	0
305	5
306	4

307	5
308	5
309	0
310	4
311	2
312	2
313	2
314	2
315	2
316	4
317	2
318	4
319	4
320	5
321	2
322	4
323	5
324	2
325	5
326	2
327	5
328	5
329	5
330	5
331	5
332	2
333	2
334	4
335	2
336	2
337	5
338	5
339	5
340	5
341	5
342	5
343	0
344	5
345	2
346	5
347	2
348	5
349	0
350	2
351	4
352	5
353	5
354	2
355	5
356	5
357	3
358	5
359	4
360	4
361	5
362	0
363	5
364	4
365	5
366	2
367	2
368	2
369	2
370	4
371	2
372	5

373	5
374	2
375	5
376	5
377	2
378	0
379	4
380	2
381	2
382	5
383	5
384	1
385	4
386	5
387	1
388	5
389	0
390	5
391	4
392	5
393	4
394	5
395	5
396	5
397	4
398	5
399	5
400	2
401	2
402	5
403	4
404	4
405	4
406	5
407	5
408	2
409	2
410	5
411	5
412	2
413	2
414	4
415	5
416	4
417	2
418	5
419	1
420	5
421	4
422	5
423	4
424	2
425	2
426	4
427	5
428	5
429	4
430	5
431	5
432	5
433	4
434	5
435	4
436	5
437	5
438	2

439	5
440	2
441	5
442	2
443	2
444	4
445	2
446	2
447	4
448	2
449	5
450	5
451	4
452	5
453	5
454	5
455	4
456	5
457	2
458	2
459	4
460	5
461	5
462	3
463	2
464	5
465	4
466	3
467	2
468	2
469	5
470	5
471	2
472	5
473	0
474	5
475	5
476	4
477	2
478	2
479	5
480	2
481	2
482	2
483	5
484	5
485	2
486	5
487	4
488	2
489	5
490	2
491	4
492	5
493	1
494	5
495	5
496	2
497	5
498	5
499	5
500	5
501	5
502	5
503	4
504	1

505	5
506	4
507	5
508	2
509	5
510	4
511	2
512	5
513	5
514	5
515	2
516	4
517	4
518	5
519	2
520	5
521	2
522	2
523	2
524	4
525	4
526	5
527	2
528	5
529	4
530	1
531	5
532	3
533	5
534	2
535	5
536	5
537	5
538	1
539	4
540	2
541	4
542	4
543	5
544	2
545	5
546	2
547	5
548	5
549	2
550	5
551	5
552	2
553	5
554	2
555	2
556	4
557	5
558	4
559	4
560	2
561	4
562	2
563	2
564	4
565	5
566	2
567	5
568	2
569	4
570	5

571	5
572	5
573	4
574	5
575	5
576	2
577	1
578	5
579	5
580	2
581	2
582	5
583	2
584	5
585	2
586	5
587	5
588	2
589	5
590	5
591	2
592	5
593	5
594	5
595	4
596	5
597	2
598	4
599	4
600	2
601	2
602	4
603	5
604	4
605	1
606	5
607	5
608	2
609	5
610	0
611	4
612	4
613	5
614	5
615	5
616	2
617	5
618	2
619	2
620	5
621	4
622	5
623	4
624	4
625	5
626	4
627	2
628	4
629	5
630	5
631	2
632	4
633	5
634	5
635	5
636	3

637	5
638	5
639	2
640	2
641	4
642	4
643	5
644	2
645	5
646	5
647	2
648	4
649	5
650	5
651	2
652	4
653	5
654	4
655	5
656	5
657	2
658	2
659	5
660	4
661	4
662	4
663	5
664	0
665	4
666	3
667	1
668	4
669	2
670	5
671	5
672	4
673	4
674	2
675	4
676	5
677	5
678	2
679	5
680	4
681	5
682	2
683	4
684	5
685	4
686	5
687	5
688	2
689	2
690	4
691	0
692	2
693	5
694	5
695	4
696	5
697	5
698	2
699	5
700	2
701	5
702	2

703	3
704	5
705	5
706	4
707	2
708	2
709	5
710	5
711	5
712	2
713	5
714	5
715	4
716	5
717	1
718	4
719	3
720	4
721	5
722	5
723	5
724	2
725	5
726	2
727	5
728	5
729	5
730	4
731	2
732	4
733	4
734	5
735	4
736	5
737	2
738	5
739	5
740	5
741	5
742	5
743	4
744	4
745	4
746	4
747	2
748	4
749	5
750	5
751	5
752	5
753	2
754	5
755	5
756	5
757	4
758	5
759	4
760	5
761	5
762	5
763	2
764	2
765	2
766	4
767	5
768	2

769	2
770	5
771	5
772	4
773	5
774	2
775	5
776	5
777	1
778	4
779	5
780	4
781	5
782	5
783	4
784	2
785	2
786	5
787	5
788	5
789	5
790	2
791	4
792	4
793	5
794	4
795	4
796	4
797	5
798	2
799	4
800	5
801	5
802	2
803	2
804	5
805	5
806	5
807	4
808	2
809	5
810	1
811	2
812	5
813	5
814	5
815	5
816	2
817	2
818	5
819	2
820	5
821	5
822	2
823	2
824	2
825	2
826	5
827	4
828	5
829	5
830	2
831	5
832	4
833	5
834	5

835	5
836	5
837	5
838	5
839	5
840	5
841	3
842	5
843	5
844	2
845	4
846	4
847	1
848	2
849	4
850	5
851	4
852	2
853	1
854	2
855	5
856	5
857	4
858	4
859	2
860	2
861	5
862	5
863	5
864	5
865	5
866	2
867	5
868	4
869	4
870	5
871	5
872	5
873	2
874	5
875	2
876	5
877	2
878	5
879	5
880	5
881	4
882	4
883	5
884	5
885	4
886	5
887	5
888	4
889	4
890	3
891	4
892	5
893	2
894	5
895	4
896	5
897	5
898	2
899	4
900	5

901	5
902	2
903	2
904	5
905	5
906	2
907	2
908	5
909	2
910	5
911	5
912	5
913	5
914	5
915	5
916	5
917	5
918	4
919	4
920	5
921	5
922	2
923	1
924	4
925	5
926	2
927	2
928	4
929	4
930	5
931	5
932	2
933	5
934	2
935	5
936	5
937	2
938	5
939	4
940	5
941	4
942	5
943	5
944	2
945	5
946	4
947	2
948	4
949	4
950	5
951	5
952	5
953	3
954	5
955	5
956	4
957	5
958	5
959	5
960	5
961	4
962	4
963	5
964	2
965	2
966	4

967	5
968	5
969	5
970	5
971	5
972	4
973	5
974	5
975	5
976	5
977	5
978	5
979	5
980	5
981	4
982	2
983	2
984	5
985	5
986	5
987	2
988	4
989	5
990	4
991	2
992	1
993	5
994	2
995	5
996	5
997	4
998	2
999	5
1000	5
1001	5
1002	2
1003	5
1004	2
1005	5
1006	4
1007	5
1008	2
1009	5
1010	2
1011	5
1012	4
1013	5
1014	2
1015	5
1016	4
1017	0
1018	4
1019	2
1020	5
1021	5
1022	5
1023	2
1024	2
1025	5
1026	2
1027	2
1028	2
1029	5
1030	5
1031	4
1032	4

1033	5
1034	2
1035	5
1036	0
1037	4
1038	4
1039	5
1040	4
1041	5
1042	2
1043	0
1044	2
1045	2
1046	2
1047	5
1048	4
1049	5
1050	5
1051	2
1052	4
1053	2
1054	2
1055	4
1056	2
1057	2
1058	2
1059	2
1060	5
1061	5
1062	5
1063	2
1064	3
1065	5
1066	4
1067	5
1068	2
1069	5
1070	5
1071	5
1072	4
1073	5
1074	5
1075	2
1076	4
1077	5
1078	2
1079	5
1080	5
1081	5
1082	2
1083	2
1084	4
1085	5
1086	5
1087	2
1088	5
1089	2
1090	5
1091	5
1092	2
1093	5
1094	5
1095	2
1096	5
1097	5
1098	5

1099	4
1100	5
1101	5
1102	5
1103	1
1104	5
1105	4
1106	4
1107	2
1108	4
1109	2
1110	4
1111	2
1112	5
1113	5
1114	5
1115	2
1116	4
1117	5
1118	5
1119	5
1120	2
1121	5
1122	5
1123	5
1124	4
1125	4
1126	4
1127	2
1128	2
1129	5
1130	4
1131	5
1132	5
1133	4
1134	4
1135	2
1136	5
1137	5
1138	4
1139	2
1140	5
1141	4
1142	4
1143	5
1144	5
1145	2
1146	4
1147	2
1148	2
1149	5
1150	2
1151	2
1152	2
1153	5
1154	2
1155	1
1156	2
1157	2
1158	5
1159	4
1160	4
1161	2
1162	3
1163	5
1164	4

1165	5
1166	5
1167	4
1168	4
1169	4
1170	3
1171	4
1172	5
1173	2
1174	2
1175	4
1176	5
1177	5
1178	2
1179	2
1180	5
1181	2
1182	4
1183	2
1184	4
1185	5
1186	5
1187	1
1188	5
1189	4
1190	4
1191	5
1192	5
1193	5
1194	5
1195	5
1196	2
1197	2
1198	5
1199	3
1200	5
1201	5
1202	1
1203	4
1204	5
1205	4
1206	5
1207	5
1208	5
1209	2
1210	4
1211	5
1212	5
1213	5
1214	5
1215	5
1216	5
1217	5
1218	5
1219	5
1220	5
1221	4
1222	2
1223	5
1224	2
1225	5
1226	2
1227	5
1228	2
1229	5
1230	4

1231	5
1232	5
1233	5
1234	5
1235	5
1236	5
1237	2
1238	5
1239	2
1240	5
1241	5
1242	4
1243	2
1244	2
1245	4
1246	5
1247	5
1248	2
1249	5
1250	4
1251	2
1252	5
1253	2
1254	2
1255	2
1256	4
1257	5
1258	5
1259	5
1260	4
1261	5
1262	4
1263	2
1264	5
1265	5
1266	2
1267	2
1268	2
1269	5
1270	4
1271	2
1272	5
1273	2
1274	2
1275	5
1276	4
1277	2
1278	2
1279	5
1280	5
1281	4
1282	5
1283	5
1284	2
1285	2
1286	2
1287	5
1288	0
1289	2
1290	2
1291	5
1292	5
1293	4
1294	5
1295	5
1296	5

1297	5
1298	2
1299	5
1300	4
1301	2
1302	4
1303	5
1304	5
1305	0
1306	2
1307	5
1308	5
1309	5
1310	4
1311	5
1312	0
1313	4
1314	4
1315	2
1316	2
1317	2
1318	4
1319	5
1320	2
1321	5
1322	4
1323	5
1324	2
1325	5
1326	5
1327	0
1328	4
1329	4
1330	2
1331	5
1332	2
1333	5
1334	5
1335	4
1336	5
1337	5
1338	5
1339	5
1340	5
1341	5
1342	2
1343	4
1344	4
1345	5
1346	2
1347	2
1348	1
1349	5
1350	5
1351	2
1352	5
1353	0
1354	4
1355	4
1356	5
1357	1
1358	5
1359	2
1360	5
1361	2
1362	2

1363	2
1364	5
1365	5
1366	5
1367	4
1368	5
1369	4
1370	5
1371	4
1372	4
1373	2
1374	2
1375	2
1376	5
1377	5
1378	5
1379	5
1380	5
1381	2
1382	5
1383	5
1384	5
1385	5
1386	4
1387	2
1388	2
1389	2
1390	5
1391	5
1392	3
1393	4
1394	2
1395	2
1396	4
1397	5
1398	5
1399	2
1400	2
1401	4
1402	2
1403	5
1404	5
1405	2
1406	5
1407	5
1408	5
1409	4
1410	5
1411	5
1412	5
1413	2
1414	2
1415	4
1416	5
1417	4
1418	5
1419	4
1420	4
1421	1
1422	5
1423	2
1424	4
1425	5
1426	4
1427	4
1428	2

1429	4
1430	2
1431	5
1432	5
1433	4
1434	2
1435	2
1436	5
1437	2
1438	5
1439	4
1440	2
1441	4
1442	0
1443	2
1444	5
1445	5
1446	5
1447	4
1448	5
1449	5
1450	5
1451	2
1452	5
1453	5
1454	5
1455	4
1456	4
1457	2
1458	5
1459	5
1460	5
1461	5
1462	4
1463	2
1464	5
1465	4
1466	5
1467	2
1468	3
1469	5
1470	1
1471	5
1472	5
1473	4
1474	5
1475	2
1476	5
1477	2
1478	2
1479	2
1480	0
1481	2
1482	5
1483	5
1484	2
1485	2
1486	4
1487	2
1488	5
1489	2
1490	5
1491	5
1492	1
1493	4
1494	4

1495	2
1496	5
1497	5
1498	5
1499	5
1500	5
1501	5
1502	2
1503	4
1504	2
1505	5
1506	4
1507	4
1508	5
1509	5
1510	5
1511	5
1512	5
1513	5
1514	2
1515	2
1516	2
1517	5
1518	5
1519	5
1520	5
1521	2
1522	5
1523	5
1524	5
1525	5
1526	5
1527	5
1528	5
1529	4
1530	2
1531	2
1532	5
1533	5
1534	2
1535	5
1536	2
1537	5
1538	5
1539	5
1540	5
1541	5
1542	2
1543	5
1544	5
1545	2
1546	1
1547	5
1548	5
1549	2
1550	2
1551	4
1552	5
1553	5
1554	2
1555	5
1556	5
1557	5
1558	5
1559	2
1560	5

1561	5
1562	5
1563	4
1564	4
1565	4
1566	5
1567	4
1568	2
1569	4
1570	4
1571	5
1572	4
1573	5
1574	4
1575	2
1576	5
1577	5
1578	5
1579	4
1580	5
1581	5
1582	2
1583	4
1584	5
1585	5
1586	5
1587	5
1588	5
1589	1
1590	5
1591	2
1592	2
1593	5
1594	2
1595	4
1596	5
1597	5
1598	4
1599	4
1600	5
1601	5
1602	5
1603	2
1604	4
1605	1
1606	5
1607	4
1608	5
1609	5
1610	4
1611	5
1612	2
1613	3
1614	5
1615	5
1616	5
1617	5
1618	5
1619	5
1620	5
1621	5
1622	4
1623	1
1624	5
1625	2
1626	4

1627	2
1628	4
1629	2
1630	5
1631	5
1632	4
1633	4
1634	4
1635	4
1636	4
1637	4
1638	4
1639	4
1640	4
1641	0
1642	4
1643	2
1644	1
1645	4
1646	4
1647	5
1648	5
1649	5
1650	5
1651	5
1652	5
1653	4
1654	1
1655	4
1656	4
1657	4
1658	5
1659	4
1660	2
1661	2
1662	2
1663	2
1664	2
1665	2
1666	2
1667	2
1668	2
1669	2
1670	2
1671	2
1672	2
1673	2
1674	2
1675	2
1676	2
1677	0
1678	2
1679	2
1680	2
1681	2
1682	4
1683	2
1684	2
1685	2
1686	2
1687	2
1688	2
1689	2
1690	4
1691	2
1692	5

1693	4
1694	4
1695	4
1696	4
1697	4
1698	2
1699	4
1700	4
1701	5
1702	2
1703	2
1704	2
1705	2
1706	5
1707	5
1708	2
1709	2
1710	2
1711	2
1712	2
1713	5
1714	5
1715	5
1716	5
1717	5
1718	5
1719	5
1720	5
1721	5
1722	5
1723	5
1724	4
1725	5
1726	5
1727	5
1728	5
1729	4
1730	5
1731	5
1732	5
1733	5
1734	5
1735	5
1736	2
1737	4
1738	4
1739	2
1740	2
1741	5
1742	5
1743	2
1744	5
1745	2
1746	4
1747	4
1748	1
1749	5
1750	4
1751	5
1752	5
1753	4
1754	2
1755	3
1756	5
1757	2
1758	2

1759	4
1760	2
1761	4
1762	3
1763	5
1764	5
1765	4
1766	2
1767	5
1768	2
1769	5
1770	5
1771	5
1772	2
1773	5
1774	5
1775	5
1776	5
1777	5
1778	5
1779	2
1780	5
1781	5
1782	5
1783	5
1784	5
1785	4
1786	5
1787	5
1788	5
1789	5
1790	5
1791	5
1792	3
1793	5
1794	5
1795	2
1796	5
1797	5
1798	5
1799	5
1800	5
1801	5
1802	5
1803	5
1804	1
1805	5
1806	5
1807	5
1808	5
1809	2
1810	5
1811	5
1812	5
1813	5
1814	5
1815	5
1816	5
1817	5
1818	2
1819	2
1820	5
1821	5
1822	5
1823	2
1824	2

1825	5
1826	5
1827	2
1828	5
1829	5
1830	5
1831	5
1832	5
1833	5
1834	5
1835	5
1836	5
1837	5
1838	5
1839	5
1840	5
1841	5
1842	3
1843	1
1844	4
1845	4
1846	4
1847	5
1848	5
1849	5
1850	2
1851	2
1852	5
1853	5
1854	3
1855	4
1856	4
1857	5
1858	5
1859	4
1860	5
1861	4
1862	4
1863	4
1864	2
1865	2
1866	2
1867	2
1868	2
1869	5
1870	2
1871	2
1872	4
1873	5
1874	5
1875	4
1876	5
1877	4
1878	5
1879	5
1880	5
1881	2
1882	5
1883	5
1884	4
1885	2
1886	4
1887	2
1888	2
1889	5
1890	5

1891	5
1892	5
1893	5
1894	4
1895	2
1896	2
1897	2
1898	2
1899	2
1900	4
1901	5
1902	2
1903	2
1904	2
1905	4
1906	2
1907	4
1908	4
1909	4
1910	4
1911	2
1912	5
1913	5
1914	2
1915	5
1916	4
1917	4
1918	1
1919	5
1920	0
1921	5
1922	4
1923	1
1924	5
1925	2
1926	5
1927	3
1928	5
1929	5
1930	1
1931	5
1932	4
1933	4
1934	4
1935	5
1936	4
1937	5
1938	4
1939	5
1940	4
1941	5
1942	4
1943	2
1944	2
1945	2
1946	2
1947	4
1948	0
1949	5
1950	1
1951	4
1952	3
1953	4
1954	5
1955	2
1956	2

1957	2
1958	5
1959	5
1960	5
1961	5
1962	5
1963	5
1964	5
1965	5
1966	5
1967	0
1968	2
1969	2
1970	0
1971	2
1972	2
1973	2
1974	2
1975	4
1976	4
1977	2
1978	2
1979	2
1980	2
1981	2
1982	2
1983	5
1984	4
1985	4
1986	2
1987	2
1988	2
1989	2
1990	2
1991	4
1992	2
1993	4
1994	4
1995	4
1996	4
1997	4
1998	4
1999	4
2000	2
2001	2
2002	2
2003	2
2004	2
2005	5
2006	2
2007	5
2008	5
2009	5
2010	5
2011	5
2012	5
2013	5
2014	5
2015	5
2016	5
2017	4
2018	4
2019	5
2020	5
2021	4
2022	2

```
2023    2
2024    0
2025    5
2026    5
2027    5
2028    5
2029    4
2030    4
2031    4
2032    4
2033    5
2034    4
2035    5
2036    5
2037    4
2038    1
2039    2
2040    2
2041    4
2042    5
2043    4
2044    2
2045    5
2046    5
2047    5
2048    5
2049    5
2050    5
2051    5
2052    5
2053    5
2054    5
2055    2
2056    5
2057    2
2058    2
2059    5
2060    4
2061    5
2062    5
2063    5
2064    5
2065    2
2066    4
2067    5
2068    5
2069    5
2070    5
2071    2
2072    1
2073    2
2074    5
2075    4
2076    5
2077    5
2078    5
2079    5
2080    5
2081    5
2082    5
2083    5
2084    5
2085    5
2086    5
2087    5
2088    5
```

2089	5
2090	5
2091	5
2092	2
2093	4
2094	2
2095	5
2096	5
2097	3
2098	5
2099	5
2100	5
2101	5
2102	5
2103	5
2104	5
2105	5
2106	2
2107	5
2108	5
2109	3
2110	5
2111	2
2112	5
2113	5
2114	2
2115	1
2116	5
2117	2
2118	5
2119	2
2120	5
2121	5
2122	5
2123	2
2124	5
2125	5
2126	5
2127	5
2128	5
2129	5
2130	2
2131	5
2132	5
2133	2
2134	5
2135	5
2136	5
2137	5
2138	1
2139	5
2140	3
2141	5
2142	5
2143	5
2144	5
2145	4
2146	4
2147	5
2148	5
2149	2
2150	2
2151	2
2152	4
2153	5
2154	5

2155	4
2156	4
2157	4
2158	4
2159	2
2160	2
2161	2
2162	2
2163	1
2164	2
2165	5
2166	4
2167	5
2168	5
2169	5
2170	5
2171	1
2172	5
2173	4
2174	4
2175	2
2176	2
2177	4
2178	5
2179	5
2180	5
2181	5
2182	5
2183	5
2184	5
2185	3
2186	1
2187	4
2188	2
2189	5
2190	5
2191	5
2192	5
2193	5
2194	5
2195	5
2196	5
2197	2
2198	2
2199	2
2200	5
2201	2
2202	2
2203	2
2204	5
2205	4
2206	2
2207	4
2208	4
2209	4
2210	5
2211	5
2212	5
2213	3
2214	5
2215	5
2216	5
2217	5
2218	5
2219	5
2220	2

2221	2
2222	5
2223	4
2224	5
2225	4
2226	4
2227	5
2228	2
2229	5
2230	5
2231	4
2232	1
2233	4
2234	5
2235	2
2236	2
2237	5
2238	5
2239	5
2240	5
2241	5
2242	4
2243	5
2244	3
2245	5
2246	5
2247	5
2248	5
2249	5
2250	5
2251	2
2252	4
2253	4
2254	4
2255	5
2256	4
2257	5
2258	4
2259	4
2260	5
2261	4
2262	2
2263	2
2264	5
2265	2
2266	2
2267	2
2268	5
2269	4
2270	4
2271	4
2272	5
2273	4
2274	4
2275	1
2276	4
2277	5
2278	5
2279	5
2280	4
2281	4
2282	5
2283	5
2284	5
2285	5
2286	2

2287	2
2288	0
2289	2
2290	2
2291	2
2292	0
2293	0
2294	2
2295	2
2296	2
2297	2
2298	0
2299	2
2300	2
2301	5
2302	5
2303	5
2304	2
2305	2
2306	2
2307	2
2308	2
2309	2
2310	2
2311	5
2312	2
2313	5
2314	2
2315	2
2316	5
2317	5
2318	5
2319	5
2320	4
2321	2
2322	5
2323	2
2324	2
2325	4
2326	2
2327	2
2328	4
2329	4
2330	4
2331	4
2332	4
2333	4
2334	4
2335	4
2336	5
2337	2
2338	2
2339	2
2340	4
2341	5
2342	2
2343	5
2344	5
2345	5
2346	5
2347	5
2348	5
2349	2
2350	2
2351	2
2352	2

2353	5
2354	5
2355	5
2356	5
2357	5
2358	4
2359	5
2360	5
2361	4
2362	5
2363	4
2364	2
2365	5
2366	5
2367	2
2368	5
2369	4
2370	5
2371	5
2372	2
2373	4
2374	2
2375	4
2376	2
2377	5
2378	3
2379	5
2380	5
2381	4
2382	4
2383	2
2384	1
2385	5
2386	5
2387	5
2388	2
2389	4
2390	1
2391	5
2392	4
2393	5
2394	2
2395	4
2396	2
2397	5
2398	5
2399	5
2400	5
2401	5
2402	5
2403	4
2404	2
2405	5
2406	5
2407	5
2408	2
2409	4
2410	5
2411	4
2412	5
2413	5
2414	5
2415	5
2416	5
2417	5
2418	5

2419	4
2420	1
2421	5
2422	5
2423	2
2424	5
2425	5
2426	5
2427	4
2428	5
2429	5
2430	4
2431	5
2432	4
2433	5
2434	4
2435	5
2436	5
2437	3
2438	5
2439	5
2440	2
2441	2
2442	2
2443	2
2444	5
2445	5
2446	5
2447	2
2448	5
2449	5
2450	2
2451	2
2452	5
2453	4
2454	2
2455	5
2456	5
2457	4
2458	5
2459	2
2460	5
2461	2
2462	5
2463	5
2464	5
2465	5
2466	2
2467	5
2468	5
2469	2
2470	4
2471	2
2472	5
2473	5
2474	2
2475	5
2476	5
2477	4
2478	5
2479	3
2480	5
2481	5
2482	5
2483	5
2484	5

2485	2
2486	2
2487	2
2488	4
2489	2
2490	5
2491	4
2492	4
2493	4
2494	5
2495	2
2496	5
2497	5
2498	5
2499	5
2500	5
2501	2
2502	5
2503	3
2504	4
2505	2
2506	5
2507	2
2508	5
2509	2
2510	5
2511	5
2512	4
2513	1
2514	5
2515	5
2516	5
2517	5
2518	4
2519	1
2520	5
2521	4
2522	5
2523	1
2524	1
2525	5
2526	5
2527	5
2528	5
2529	5
2530	4
2531	2
2532	5
2533	4
2534	4
2535	4
2536	4
2537	5
2538	5
2539	2
2540	2
2541	5
2542	1
2543	5
2544	2
2545	5
2546	1
2547	5
2548	5
2549	2
2550	5

2551	5
2552	5
2553	5
2554	2
2555	1
2556	5
2557	5
2558	5
2559	2
2560	2
2561	2
2562	2
2563	4
2564	2
2565	4
2566	5
2567	4
2568	2
2569	4
2570	5
2571	5
2572	4
2573	2
2574	5
2575	5
2576	5
2577	5
2578	5
2579	5
2580	5
2581	5
2582	2
2583	3
2584	2
2585	2
2586	4
2587	5
2588	3
2589	4
2590	2
2591	5
2592	2
2593	5
2594	5
2595	0
2596	4
2597	2
2598	2
2599	5
2600	5
2601	5
2602	5
2603	5
2604	5
2605	1
2606	4
2607	4
2608	2
2609	5
2610	5
2611	5
2612	5
2613	5
2614	5
2615	4
2616	4

2617	2
2618	4
2619	4
2620	4
2621	4
2622	2
2623	2
2624	5
2625	4
2626	5
2627	2
2628	0
2629	2
2630	2
2631	2
2632	2
2633	2
2634	4
2635	4
2636	1
2637	0
2638	4
2639	5
2640	5
2641	5
2642	5
2643	1
2644	5
2645	5
2646	5
2647	3
2648	4
2649	5
2650	4
2651	2
2652	2
2653	5
2654	2
2655	2
2656	2
2657	2
2658	2
2659	2
2660	2
2661	2
2662	4
2663	2
2664	2
2665	2
2666	2
2667	5
2668	5
2669	2
2670	5
2671	2
2672	4
2673	4
2674	5
2675	2
2676	4
2677	5
2678	0
2679	4
2680	4
2681	0
2682	4

2683	4
2684	4
2685	2
2686	2
2687	5
2688	5
2689	2
2690	5
2691	5
2692	5
2693	5
2694	5
2695	5
2696	5
2697	5
2698	5
2699	5
2700	5
2701	5
2702	5
2703	4
2704	5
2705	5
2706	5
2707	5
2708	5
2709	3
2710	4
2711	4
2712	2
2713	5
2714	5
2715	5
2716	4
2717	4
2718	5
2719	5
2720	5
2721	5
2722	1
2723	5
2724	5
2725	5
2726	4
2727	2
2728	3
2729	5
2730	1
2731	2
2732	2
2733	2
2734	5
2735	2
2736	5
2737	5
2738	2
2739	5
2740	2
2741	4
2742	4
2743	4
2744	5
2745	5
2746	2
2747	5
2748	4

2749	5
2750	5
2751	2
2752	5
2753	2
2754	5
2755	5
2756	5
2757	5
2758	2
2759	4
2760	2
2761	5
2762	4
2763	2
2764	2
2765	5
2766	5
2767	5
2768	5
2769	5
2770	5
2771	5
2772	5
2773	5
2774	4
2775	5
2776	5
2777	3
2778	5
2779	5
2780	5
2781	5
2782	5
2783	5
2784	5
2785	5
2786	5
2787	5
2788	5
2789	5
2790	5
2791	5
2792	2
2793	5
2794	5
2795	5
2796	2
2797	5
2798	5
2799	5
2800	5
2801	4
2802	5
2803	2
2804	4
2805	5
2806	5
2807	4
2808	5
2809	5
2810	4
2811	1
2812	5
2813	5
2814	2

2815	2
2816	4
2817	5
2818	5
2819	2
2820	4
2821	2
2822	2
2823	1
2824	0
2825	5
2826	5
2827	4
2828	5
2829	5
2830	5
2831	2
2832	5
2833	5
2834	2
2835	5
2836	3
2837	5
2838	4
2839	5
2840	5
2841	4
2842	5
2843	5
2844	5
2845	5
2846	4
2847	4
2848	5
2849	4
2850	4
2851	4
2852	4
2853	5
2854	5
2855	5
2856	5
2857	5
2858	5
2859	5
2860	5
2861	2
2862	5
2863	0
2864	5
2865	5
2866	5
2867	5
2868	5
2869	2
2870	2
2871	5
2872	5
2873	5
2874	5
2875	2
2876	5
2877	2
2878	4
2879	5
2880	4

```
2881    2
2882    2
2883    2
2884    2
2885    2
2886    5
2887    5
2888    5
2889    5
2890    5
2891    5
2892    5
2893    5
2894    2
2895    2
2896    4
2897    5
2898    5
2899    5
2900    2
2901    5
2902    2
2903    2
2904    5
2905    5
2906    4
2907    5
2908    5
2909    5
2910    5
2911    4
2912    5
2913    5
2914    5
2915    5
2916    4
2917    5
2918    4
Name: FireplaceQu, dtype: int64
```

Rest of the null values in other columns are not significantly bigger. So we can use 'mean' to fill the rest.

In [ ]:
```python
# checking if there is any null values
df.isnull().sum()
```

Out[ ]:
```
Id                 0
MSSubClass         0
MSZoning           0
LotFrontage      486
LotArea            0
Street             0
LotShape           0
LandContour        0
Utilities          0
LotConfig          0
LandSlope          0
Neighborhood       0
Condition1         0
Condition2         0
BldgType           0
HouseStyle         0
OverallQual        0
OverallCond        0
YearBuilt          0
YearRemodAdd       0
```

```
RoofStyle           0
RoofMatl            0
Exterior1st         0
Exterior2nd         0
MasVnrType          0
MasVnrArea         23
ExterQual           0
ExterCond           0
Foundation          0
BsmtQual            0
BsmtCond            0
BsmtExposure        0
BsmtFinType1        0
BsmtFinSF1          1
BsmtFinType2        0
BsmtFinSF2          1
BsmtUnfSF           1
TotalBsmtSF         1
Heating             0
HeatingQC           0
CentralAir          0
Electrical          0
1stFlrSF            0
2ndFlrSF            0
LowQualFinSF        0
GrLivArea           0
BsmtFullBath        2
BsmtHalfBath        2
FullBath            0
HalfBath            0
BedroomAbvGr        0
KitchenAbvGr        0
KitchenQual         0
TotRmsAbvGrd        0
Functional          0
Fireplaces          0
FireplaceQu         0
GarageType          0
GarageYrBlt       159
GarageFinish        0
GarageCars          1
GarageArea          1
GarageQual          0
GarageCond          0
PavedDrive          0
WoodDeckSF          0
OpenPorchSF         0
EnclosedPorch       0
3SsnPorch           0
ScreenPorch         0
PoolArea            0
MiscVal             0
MoSold              0
YrSold              0
SaleType            0
SaleCondition       0
SalePrice        1459
dtype: int64
```

In [ ]:
```python
# fill null values with mean for all columns except 'SalePrice'
def fill_na_with_mean(df, col):
    df[col] = df[col].fillna(df[col].mean())
    return df

# Apply the function to each column except 'SalePrice'
```

```
for col in df.columns:
    if col != 'SalePrice':
        df = fill_na_with_mean(df, col)
```

Data Exploration

We are very interested to see the locations of the properties in a map. We have to decode the encoded values in the Neighborhood column. Then we will use folium and geopy libraries to get the coordinates of the locations and then we will see them on the map. We have commented out the geolocating codes because we had saved the encoded + geolocated files and read that file later. Since the geolocation process takes a long time, we decided to run the process once, save the file and read the file again.

In [ ]:
```
# # Define the decoding mapping we got from the decode output previously
# decode_map = {
#     0: 'Blmngtn', 1: 'Blueste', 2: 'BrDale', 3: 'BrkSide', 4: 'ClearCr', 5: 'CollgCr',
#     7: 'Edwards', 8: 'Gilbert', 9: 'IDOTRR', 10: 'MeadowV', 11: 'Mitchel', 12: 'NAmes'
#     14: 'NWAmes', 15: 'NoRidge', 16: 'NridgHt', 17: 'OldTown', 18: 'SWISU', 19: 'Sawye
#     21: 'Somerst', 22: 'StoneBr', 23: 'Timber', 24: 'Veenker'
# }

# # Decode the 'Neighborhood' column
# df['Neighborhood_Decoded'] = df['Neighborhood'].map(decode_map)
```

In [ ]:
```
# # We need the full forms of the location to give actual location to geopy
# full_form_map = {
#     'Blmngtn': 'Bloomington Heights', 'Blueste': 'Bluestem', 'BrDale': 'Briardale', 'B
#     'ClearCr': 'Clear Creek', 'CollgCr': 'College Creek', 'Crawfor': 'Crawford', 'Edwa
#     'Gilbert': 'Gilbert', 'IDOTRR': 'Iowa DOT and Rail Road', 'MeadowV': 'Meadow Villa
#     'NAmes': 'North Ames', 'NoRidge': 'Northridge', 'NPkVill': 'Northpark Villa', 'Nri
#     'NWAmes': 'Northwest Ames', 'OldTown': 'Old Town', 'SWISU': 'South & West of Iowa
#     'Sawyer': 'Sawyer', 'SawyerW': 'Sawyer West', 'Somerst': 'Somerset', 'StoneBr': 'S
#     'Timber': 'Timberland', 'Veenker': 'Veenker'
# }

# # Apply the mapping to create a new column with full forms
# df['Neighborhood_Full'] = df['Neighborhood_Decoded'].map(full_form_map)
```

In [ ]:
```
# df.head()
```

Now we will create a function to get actual co-ordinates of the locations

In [ ]:
```
# # Initialize Geocoder
# geolocator = Nominatim(user_agent="geoapiExercises")

# # Function to geocode neighborhood
# def geocode_neighborhood(neighborhood):
#     try:
#         location = geolocator.geocode(neighborhood + ', Ames, IA')
#         if location:
#             return location.latitude, location.longitude
#         else:
#             return None, None
#     except Exception as e:
#         print(f"Error geocoding {neighborhood}: {e}")
#         return None, None
```

In [ ]:
```
# # Apply geocoding to full form neighborhoods
# df[['Latitude', 'Longitude']] = df['Neighborhood_Full'].apply(lambda x: pd.Series(geoc
```

The geopy library was successfull filling in the co-ordinates of the locations and returned some errors. However, it could not find some of them. We will be doing some research on our own on google to find out those locations and manually fill in the co-ordinates.

In [ ]:
```
# # extracting the locations from error message



# # Sample error messages
# error_messages = """
# Error geocoding College Creek: Non-successful status code 403
# Error geocoding North Ames: Non-successful status code 403
# Error geocoding South & West of Iowa State University: Non-successful status code 403
# Error geocoding South & West of Iowa State University: Non-successful status code 403
# Error geocoding Northwest Ames: Non-successful status code 403
# Error geocoding Edwards: Non-successful status code 403
# Error geocoding Mitchell: HTTPSConnectionPool(host='nominatim.openstreetmap.org', port
# Error geocoding Old Town: Non-successful status code 403
# Error geocoding South & West of Iowa State University: Non-successful status code 403
# Error geocoding South & West of Iowa State University: Non-successful status code 403
# Error geocoding South & West of Iowa State University: Non-successful status code 403
# Error geocoding Clear Creek: Non-successful status code 403
# Error geocoding South & West of Iowa State University: Non-successful status code 403
# Error geocoding South & West of Iowa State University: Non-successful status code 403
# Error geocoding Somerset: Non-successful status code 403
# Error geocoding South & West of Iowa State University: Non-successful status code 403
# Error geocoding South & West of Iowa State University: Non-successful status code 403
# Error geocoding Old Town: Non-successful status code 403
# Error geocoding Iowa DOT and Rail Road: Non-successful status code 403
# Error geocoding Sawyer West: Non-successful status code 403
# Error geocoding Edwards: Non-successful status code 403
# Error geocoding Meadow Village: Non-successful status code 403
# Error geocoding Old Town: Non-successful status code 403
# Error geocoding Old Town: HTTPSConnectionPool(host='nominatim.openstreetmap.org', port
# Error geocoding South & West of Iowa State University: Non-successful status code 403
# Error geocoding Sawyer: HTTPSConnectionPool(host='nominatim.openstreetmap.org', port=4
# Error geocoding College Creek: Non-successful status code 403
# Error geocoding Edwards: Non-successful status code 403
# Error geocoding Edwards: HTTPSConnectionPool(host='nominatim.openstreetmap.org', port=
# Error geocoding Edwards: HTTPSConnectionPool(host='nominatim.openstreetmap.org', port=
# """

# # Use regex to extract the location names from the error messages
# pattern = r"Error geocoding ([\w\s&]+):"
# locations_not_found = re.findall(pattern, error_messages)

# # Remove duplicates
# locations_not_found = list(set(locations_not_found))

# # Display the list
# print(locations_not_found)
```

In [ ]:
```
# coordinates = {
#     'Gilbert' : '42.107339, -93.650046',
#     'Timberland' : '42.000054, -93.649546',
#     'Edwards' : '42.024238, -93.671078',
#     'South & West of Iowa State University' : '42.021641, -93.656344',
#     'Old Town' : '42.029275, -93.614412',
#     'North Ames': '42.034866, -93.647473',
#     'Clear Creek' : '42.036081, -93.648845',
#     'Brookside' : '42.028438, -93.631153',
#     'Somerset' : '42.050756, -93.644471'
# }
# # Split coordinates and assign to respective columns
# for neighborhood, coord in coordinates.items():
#     lat, lon = map(float, coord.split(', '))
#     df.loc[df['Neighborhood_Full'] == neighborhood, 'Latitude'] = lat
#     df.loc[df['Neighborhood_Full'] == neighborhood, 'Longitude'] = lon
```

In [ ]:
```
# # Check for NaN values in Latitude and Longitude
# nan_coords = df[df['Latitude'].isna() | df['Longitude'].isna()]

# # Print the rows with NaN values in Latitude and Longitude along with Neighborhood_Ful
# unique_nan_neighborhoods = nan_coords['Neighborhood_Full'].unique()
# print(unique_nan_neighborhoods)
```

There are still some locations without the coordinates. We will fill in with the same manual process.

In [ ]:
```
# coordinates = {
#     'Northwest Ames' : '42.049205, -93.652850',
#     'Sawyer West' : '42.021202, -93.680265',
#     'Iowa DOT and Rail Road' : '42.021948, -93.621307',
#     'Meadow Village' : '41.992291, -93.603508',
#     'Stone Brook' : '42.060080, -93.636868',
#     'Northpark Villa' : '42.053359, -93.648615',
#     'Northridge' : '42.048305, -93.648429',
#     'Northridge Heights' : '42.059853, -93.650201',
#     'Crawford' : '42.028077, -93.607049',
#     'Bloomington Heights' : '42.056526, -93.635387',
#     'Bluestem' : '42.045443, -93.652500',
#     'Briardale' : '42.052624, -93.628840',
#     'Veenker' : '42.042389, -93.648557',
#     'College Creek' : '42.022005, -93.652025',
#     'Mitchell' : '41.990092, -93.601829',
#     'Sawyer' : '42.033483, -93.676200'
# }

# # Split coordinates and assign to respective columns
# for neighborhood, coord in coordinates.items():
#     lat, lon = map(float, coord.split(', '))
#     df.loc[df['Neighborhood_Full'] == neighborhood, 'Latitude'] = lat
#     df.loc[df['Neighborhood_Full'] == neighborhood, 'Longitude'] = lon
```

In [ ]:
```
# # checking again for the null values in the coordinates
# nan_coords = df[df['Latitude'].isna() | df['Longitude'].isna()]

# # Print the rows with NaN values in Latitude and Longitude along with Neighborhood_Ful
# unique_nan_neighborhoods = nan_coords['Neighborhood_Full'].unique()
# print(unique_nan_neighborhoods)
```

In [ ]:
```
# saving the encoded file
# df.to_csv('data_encoded.csv', index = False)
```

PERFECT ! All the coordinates are filled. Now we will try to see the locations on map.

In [ ]:
```
# loading the encoded file
df = pd.read_csv('/kaggle/input/data-encoded/data_encoded.csv')
```

In [ ]:
```
df.head(10)
```

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	Lan
0	1	60	3	65.000000	8450	1	3	3	0	4	
1	2	20	3	80.000000	9600	1	3	3	0	2	

2	3	60	3	68.000000	11250	1	0	3	0	4
3	4	70	3	60.000000	9550	1	0	3	0	0
4	5	60	3	84.000000	14260	1	0	3	0	2
5	6	50	3	85.000000	14115	1	0	3	0	4
6	7	20	3	75.000000	10084	1	3	3	0	4
7	8	60	3	69.305795	10382	1	0	3	0	0
8	9	50	4	51.000000	6120	1	3	3	0	4
9	10	190	3	50.000000	7420	1	3	3	0	0

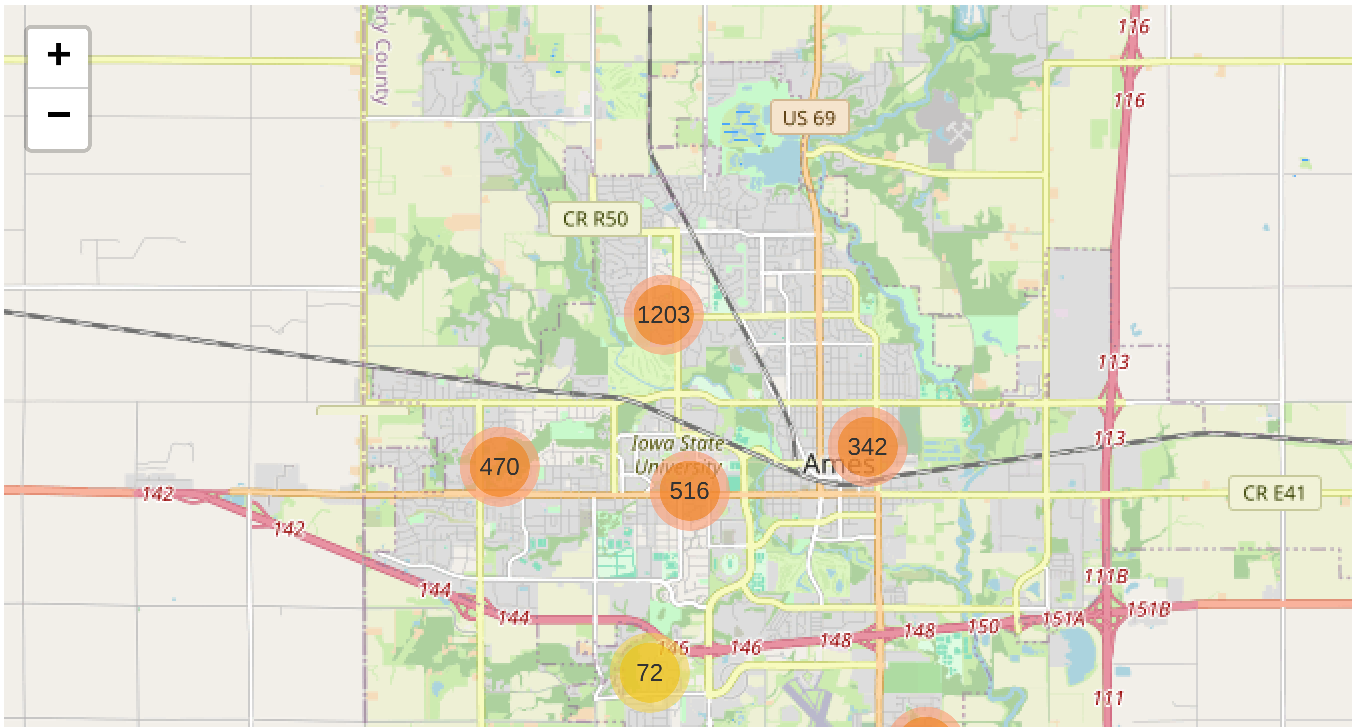
We will create two types of maps.

Marker Clusters:

- MarkerCluster: Creates a cluster of markers that group together when zoomed out. Add Markers: Adds each location as a marker to the cluster with the corresponding neighborhood name in the popup. Heatmaps:
- HeatMap: Visualizes the density of points using a color gradient.

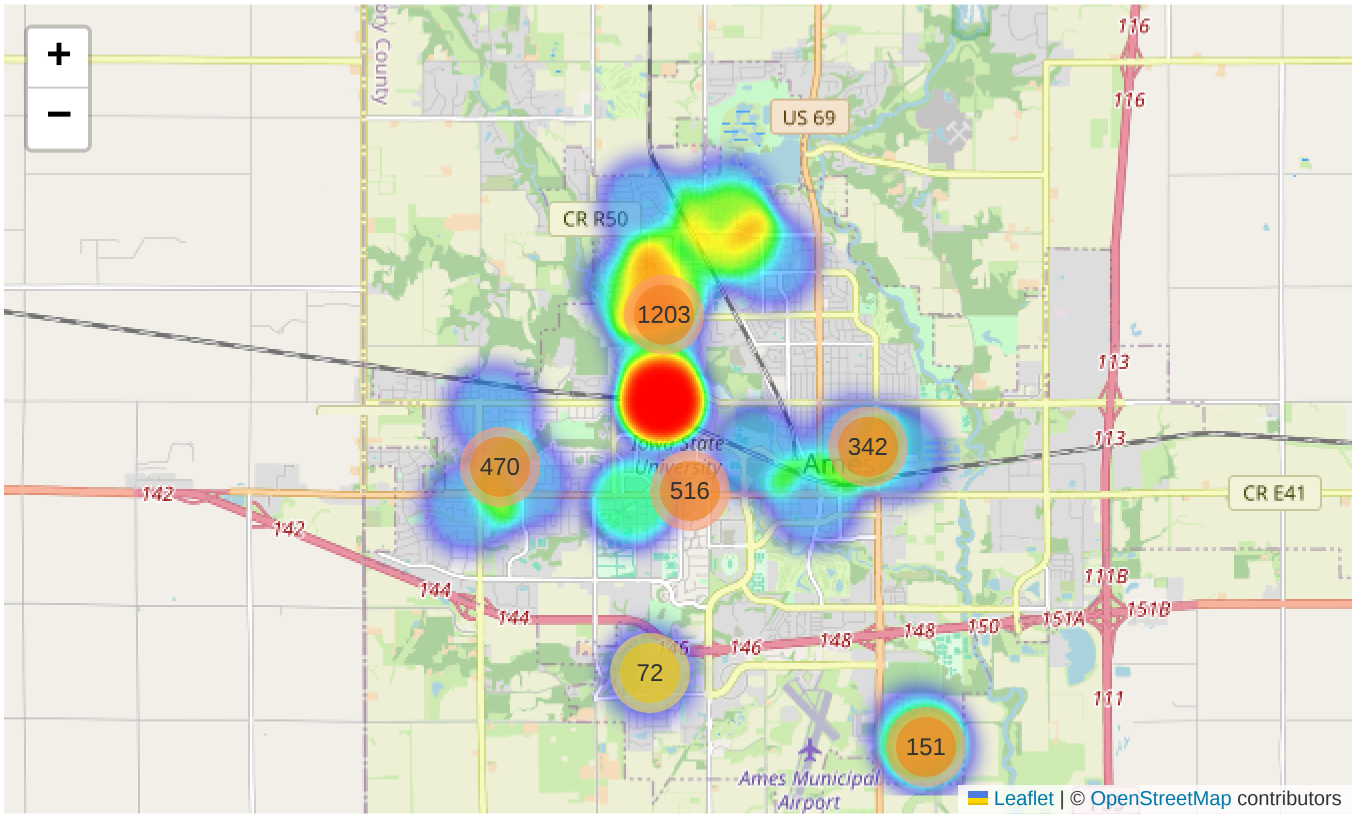
In [ ]:
```python
# Create a folium map centered around Ames, IA
map_ames = folium.Map(location=[42.034866, -93.647473], zoom_start=12)

# Create a marker cluster
marker_cluster = MarkerCluster().add_to(map_ames)

# Add markers to the cluster
for index, row in df.iterrows():
    folium.Marker(
        location=(row['Latitude'], row['Longitude']),
        popup=row['Neighborhood_Full']
    ).add_to(marker_cluster)

# Display the map
map_ames.save("ames_marker_cluster_map.html")  # Save to an HTML file
map_ames
```

Out[ ]:



In [ ]:
```python
# Prepare data for heatmap
heat_data = [[row['Latitude'], row['Longitude']] for index, row in df.iterrows()]

# Add heatmap to the map
HeatMap(heat_data).add_to(map_ames)

# Display the map
map_ames.save("ames_heatmap.html")  # Save to an HTML file
map_ames
```

Out[ ]:



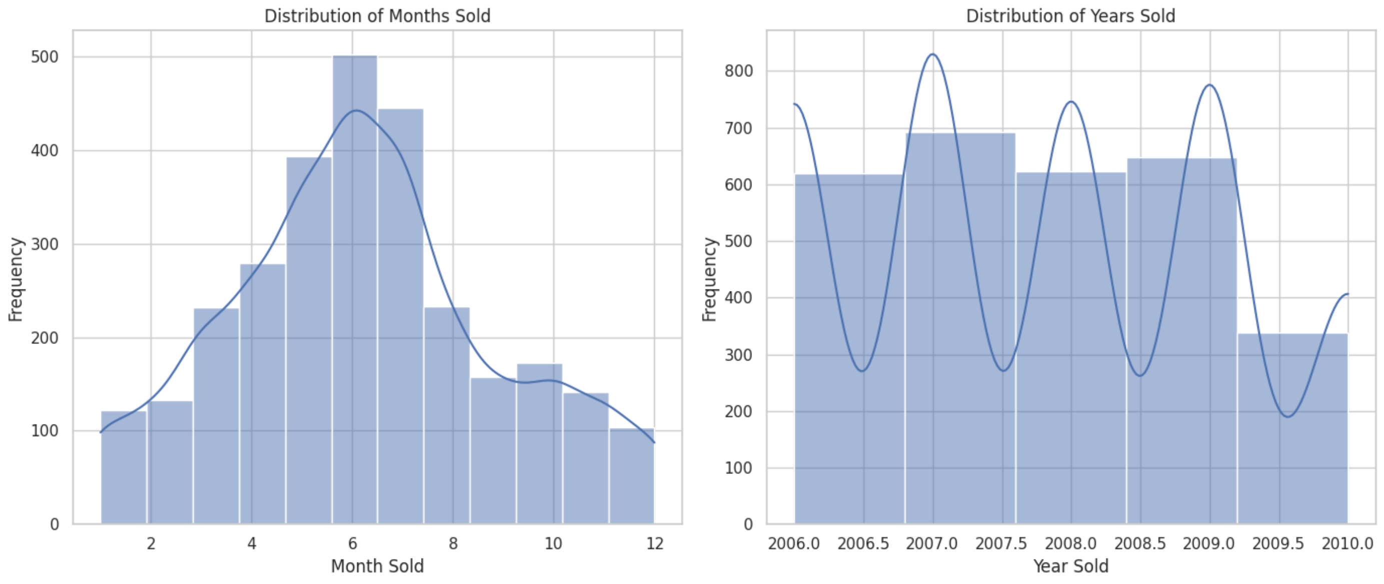
The most number of listed properties in the dataset belong to the central, northern side and south west side of the region. Next we will see the yearly and monthly sales of each property.

Yearly and Monthly Sales

In [ ]:
```python
# Set the style of the visualization
sns.set(style="whitegrid")

# Create a figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Plot the distribution of months sold
sns.histplot(df['MoSold'], bins=12, kde=True, ax=axes[0])
axes[0].set_title('Distribution of Months Sold')
axes[0].set_xlabel('Month Sold')
axes[0].set_ylabel('Frequency')

# Plot the distribution of years sold
sns.histplot(df['YrSold'], bins=len(df['YrSold'].unique()), kde=True, ax=axes[1])
axes[1].set_title('Distribution of Years Sold')
axes[1].set_xlabel('Year Sold')
axes[1].set_ylabel('Frequency')

# Display the plots
```

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

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

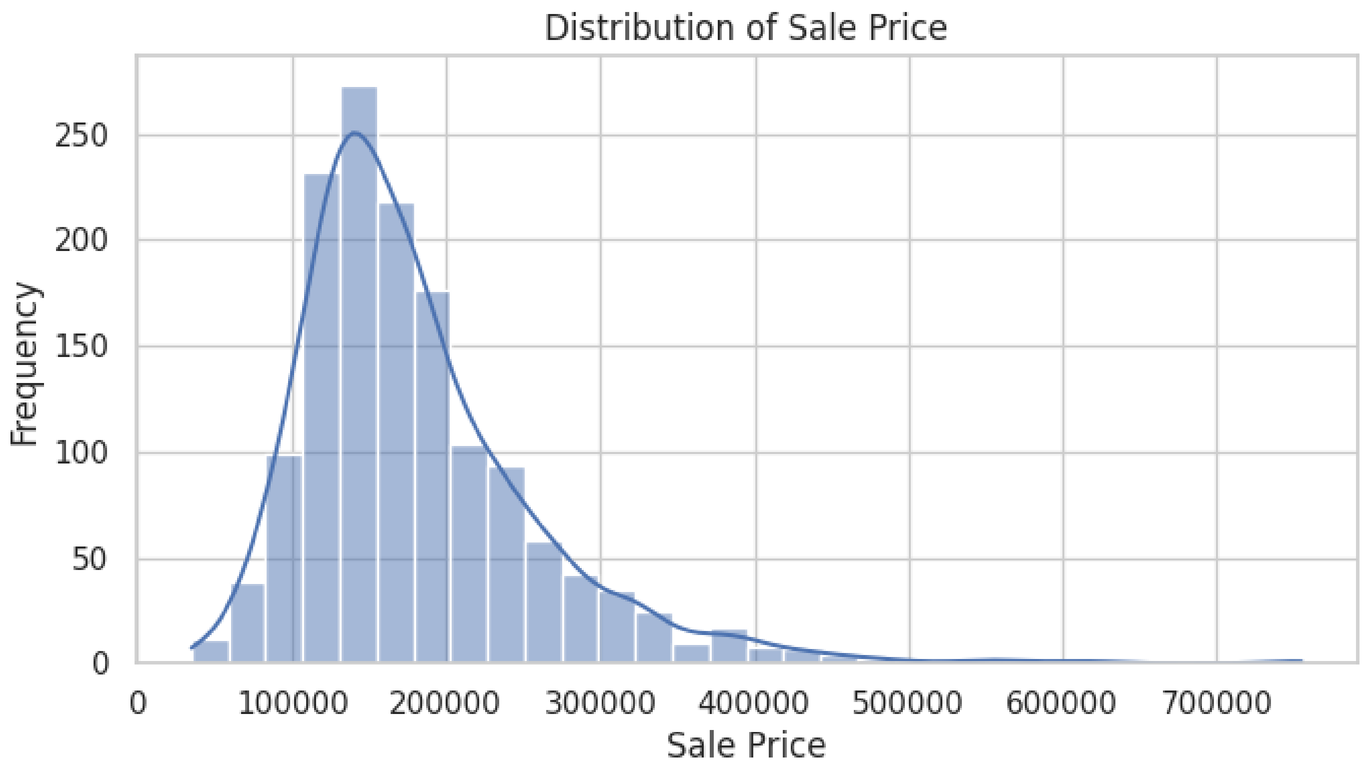


The monthly sell represents a normal distribution which indicates that the highest number of sells happen in the middle of the year. The most number of houses were sold in the years 2007 and 2009.

Sale Price Analysis

In [ ]:
```
# Set the size of the plot
plt.figure(figsize=(8, 4))

# Plotting the histogram and distribution plot for SalePrice
sns.histplot(df['SalePrice'], kde=True, bins=30)

# Adding titles and labels
plt.title('Distribution of Sale Price')
plt.xlabel('Sale Price')
plt.ylabel('Frequency')

# Display the plot
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

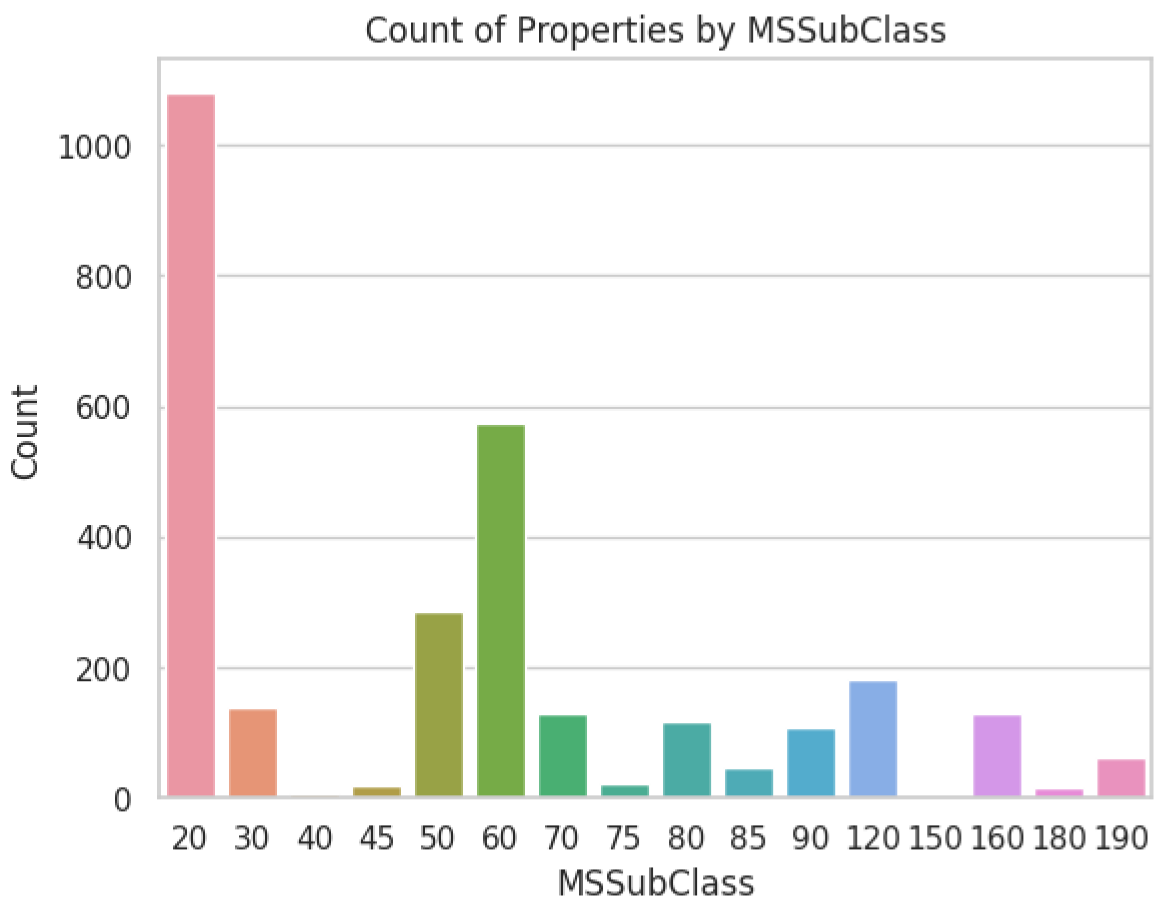


- Most houses are sold which have prices around 200,000 dollars.
- The count is extremely high comparing to other price ranges.
- Next highest count is for the price range between '100k- 200k'.
- More expensive houses beyoind 200k are least sold.
- It gives us an idea about the earning capacity of the population of Ames Iowa. Looking at the plot, this is our assumption that mostly the higher middle class people live in that area followed by the middle class, lower middle class and extremely rich people.

Building Characteristics

Distribution of building classes

```
In [ ]: # Plotting the bar plot for MSSubClass
        sns.countplot(data=df, x='MSSubClass')

        # Adding titles and labels
        plt.title('Count of Properties by MSSubClass')
        plt.xlabel('MSSubClass')
        plt.ylabel('Count')

        # Display the plot
        plt.show()
```



Most types of houses in the dataset are :

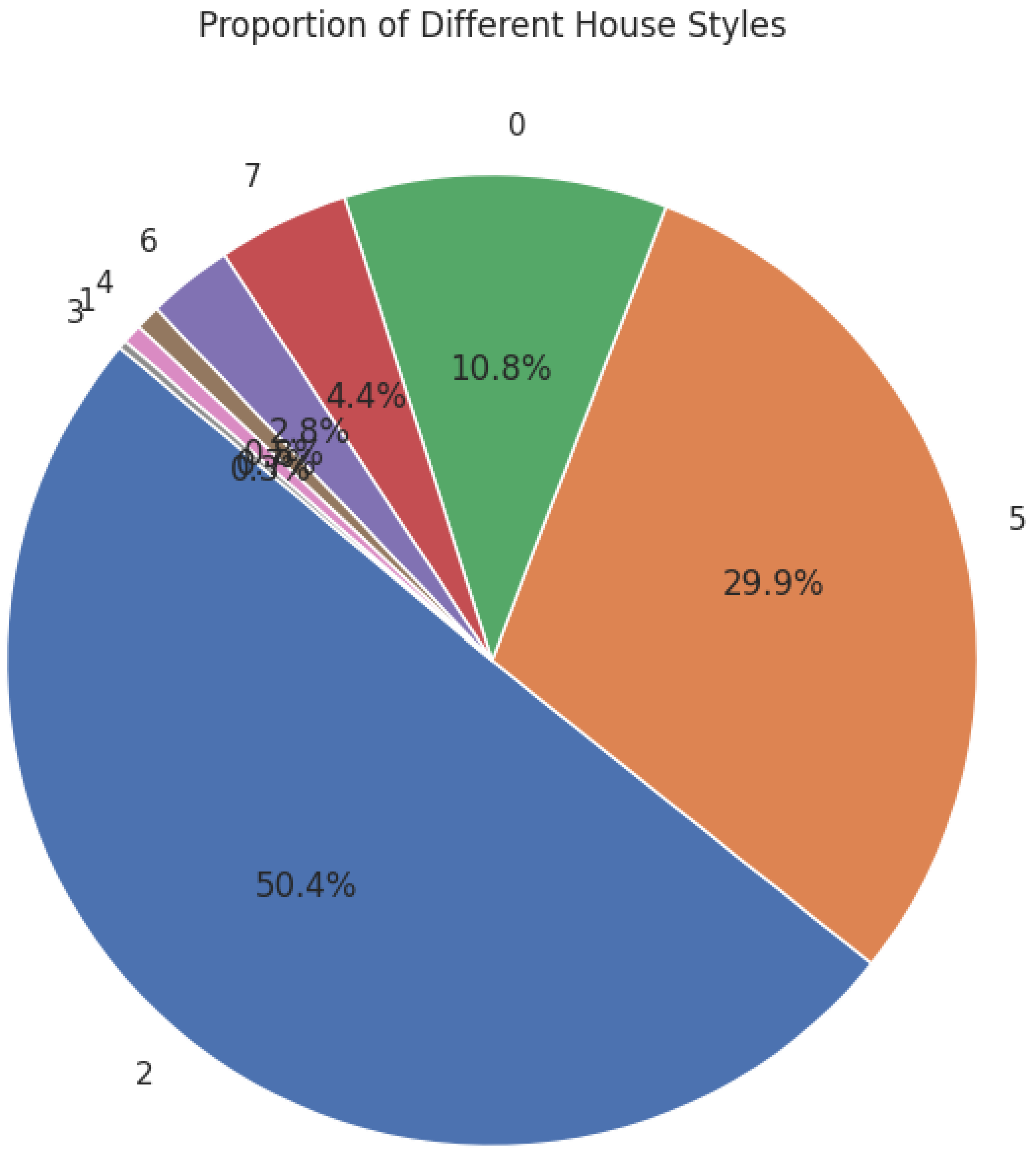
1. 1-STORY 1946 & NEWER ALL STYLES
2. 2-STORY 1946 & NEWER
3. 1-1/2 STORY FINISHED ALL AGES
4. 1-STORY PUD (Planned Unit Development) - 1946 & NEWER

Least types of houses in the dataset are :

1. 1-1/2 STORY - UNFINISHED ALL AGES
2. 2-1/2 STORY ALL AGES
3. PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
4. SPLIT FOYER

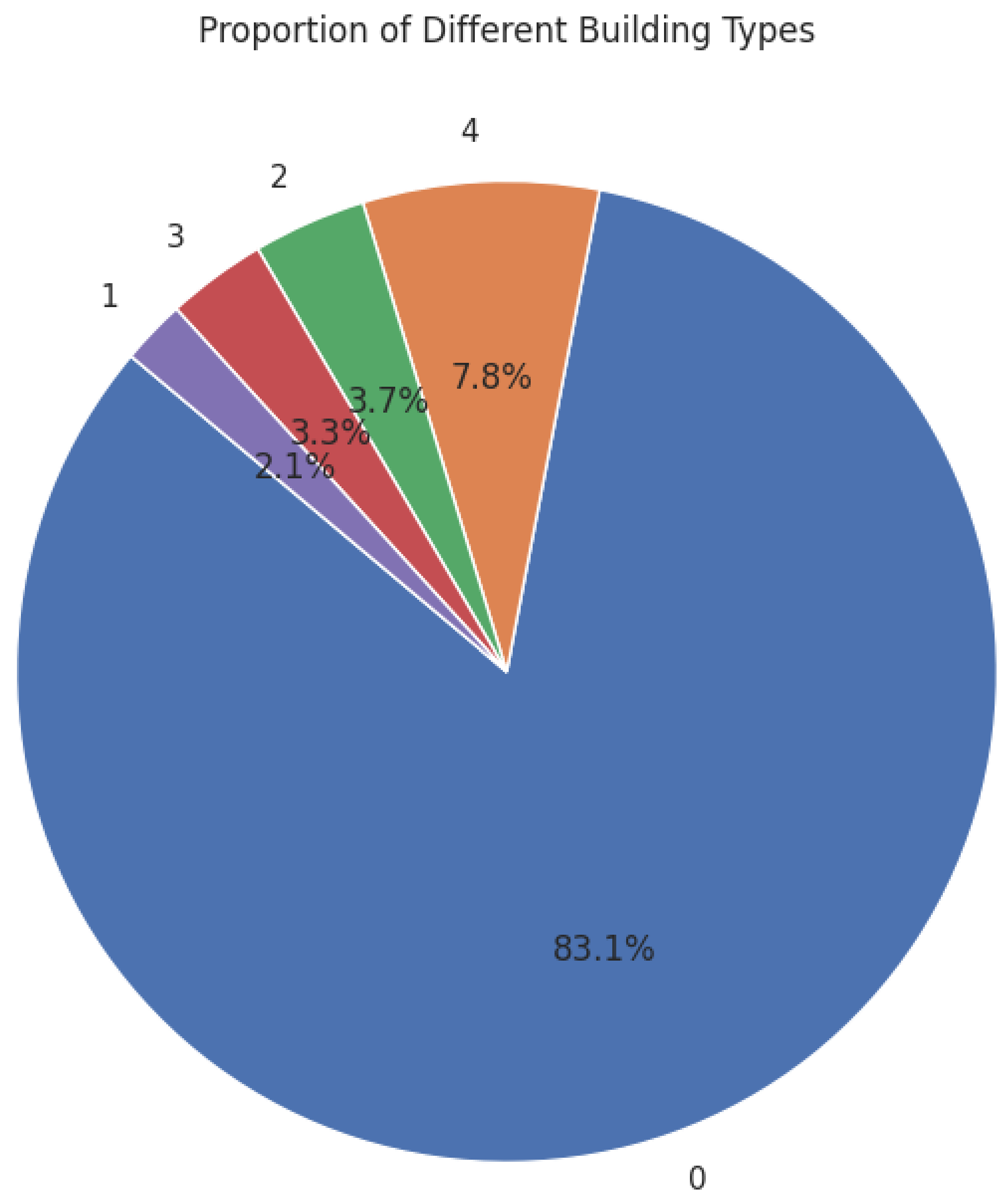
Proportion of Different Types in The Dataset

In [ ]:
```
# Calculate the count of each HouseStyle
house_style_counts = df['HouseStyle'].value_counts()

# Plotting the pie chart
plt.figure(figsize=(10, 8))
plt.pie(house_style_counts, labels=house_style_counts.index, autopct='%1.1f%%', startang

# Adding title
plt.title('Proportion of Different House Styles')

# Display the plot
plt.show()
```



Class 2 represents "1Story" houses which account for more than 50% in the dataset. Followed by class 5 which represents '2Story' houses.

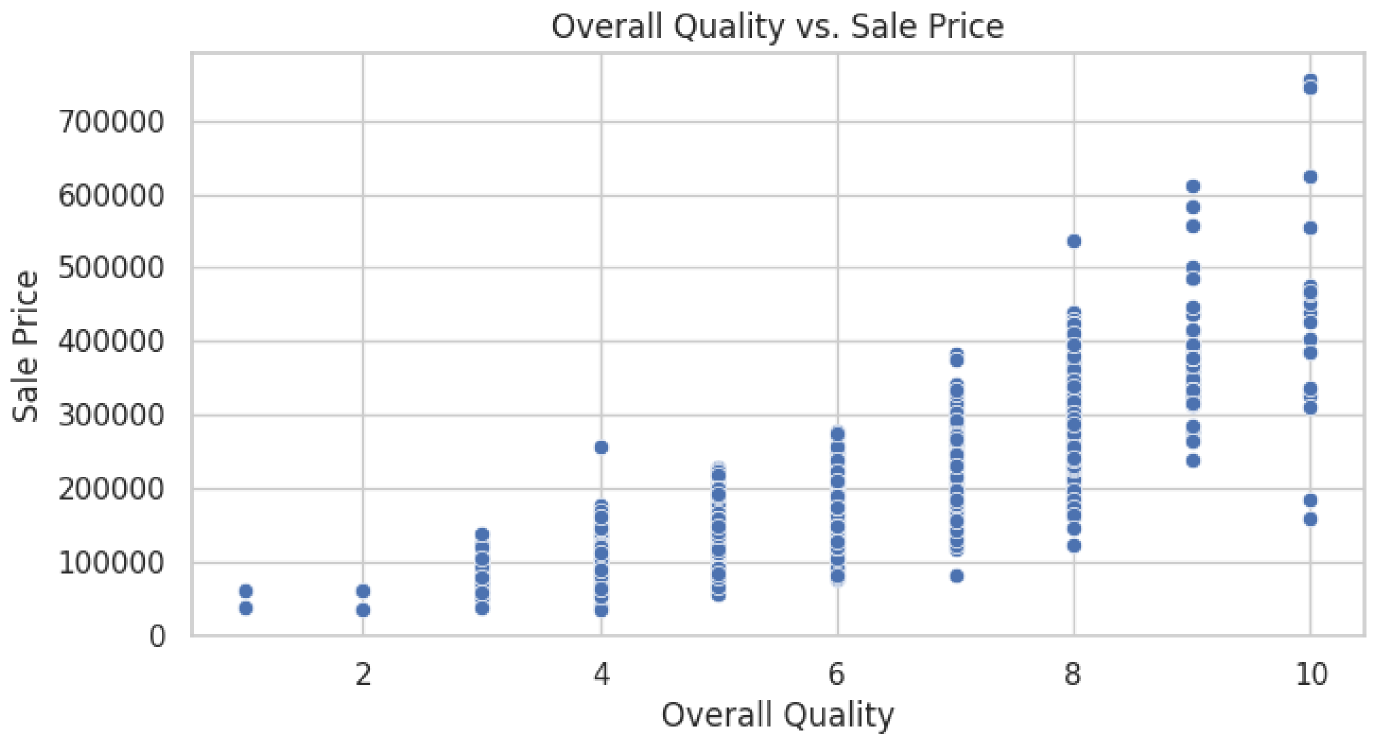
```
In [ ]: # Calculate the count of each BldgType
        bldg_type_counts = df['BldgType'].value_counts()

        # Plotting the pie chart
        plt.figure(figsize=(10, 8))
        plt.pie(bldg_type_counts, labels=bldg_type_counts.index, autopct='%1.1f%%', startangle=1

        # Adding title
        plt.title('Proportion of Different Building Types')

        # Display the plot
        plt.show()
```



The above visualization shows that class 0 that means Single-family Detached buildings are mostly in the dataset which is extremely high in number followed by class 4 - Townhouse Inside Unit types of buildings.

Overall Quality Vs Sales Price

In [ ]:
```
# Set the size of the plot
plt.figure(figsize=(8, 4))

# Plotting the scatter plot
sns.scatterplot(data=df, x='OverallQual', y='SalePrice')

# Adding titles and labels
plt.title('Overall Quality vs. Sale Price')
plt.xlabel('Overall Quality')
plt.ylabel('Sale Price')

# Display the plot
plt.show()
```

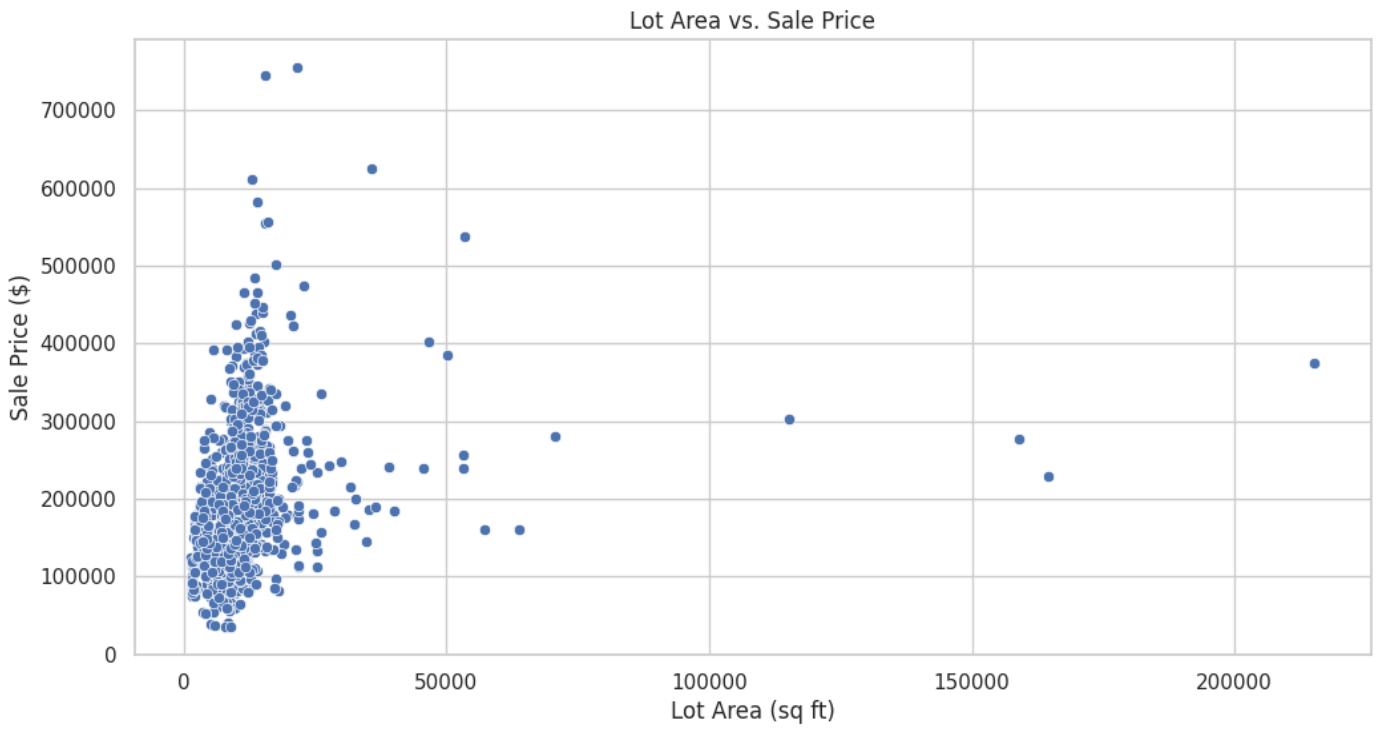


The above visualization shows that :

1. Properties ranging from 100K - 200K include most various quality. Extremely good quality houses can be found as well within that range. That makes sense why most properties cost that range.
2. The properties above 200k dollars are extremely good quality properties.

How lot size influences property prices

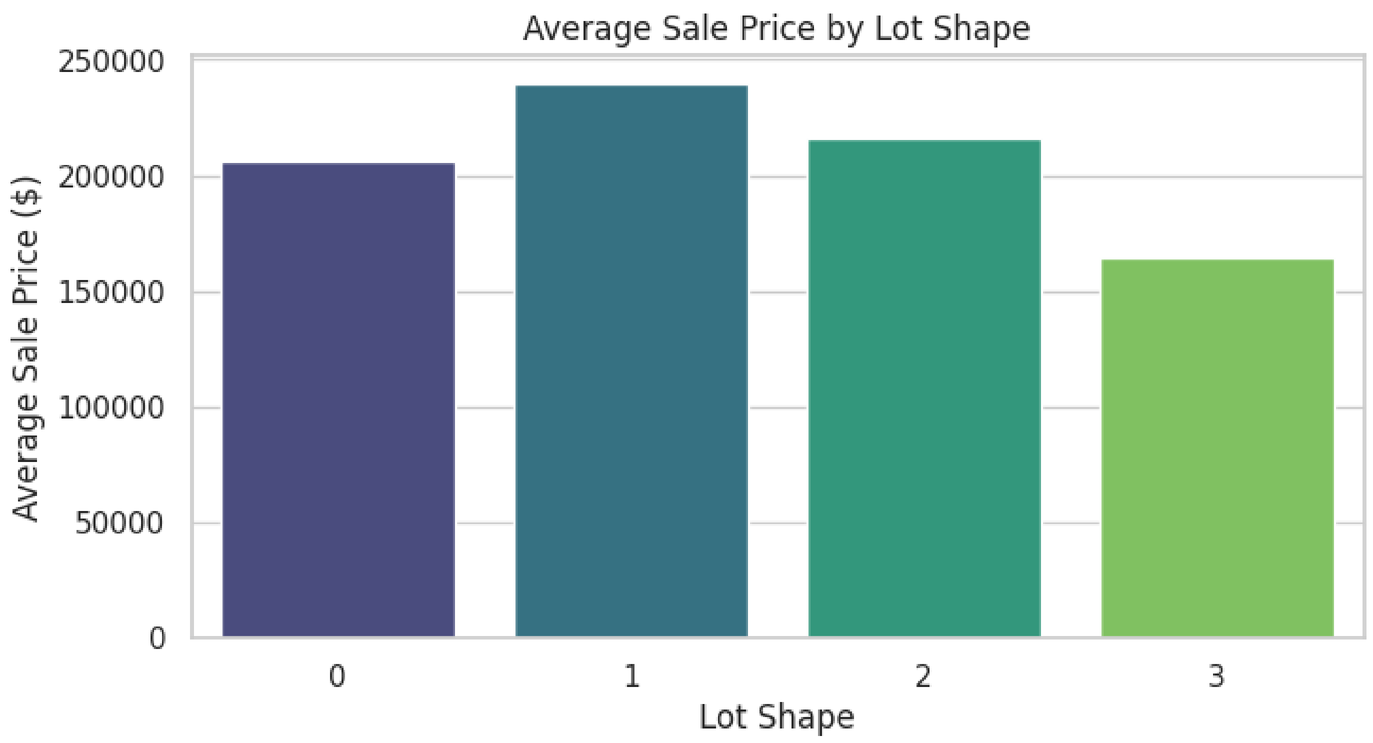
In [ ]:
```python
# Set the size of the plot
plt.figure(figsize=(12, 6))

# Plotting the scatter plot
sns.scatterplot(data=df, x='LotArea', y='SalePrice')

# Adding titles and labels
plt.title('Lot Area vs. Sale Price')
plt.xlabel('Lot Area (sq ft)')
plt.ylabel('Sale Price ($)')

# Display the plot
plt.show()
```



- The plot represents the relationship between the size of a property's lot (measured in square feet) and its sale price (in dollars).
- Most data points cluster at the lower end, indicating smaller lots with lower prices.
- Larger lots show a slight trend toward higher prices

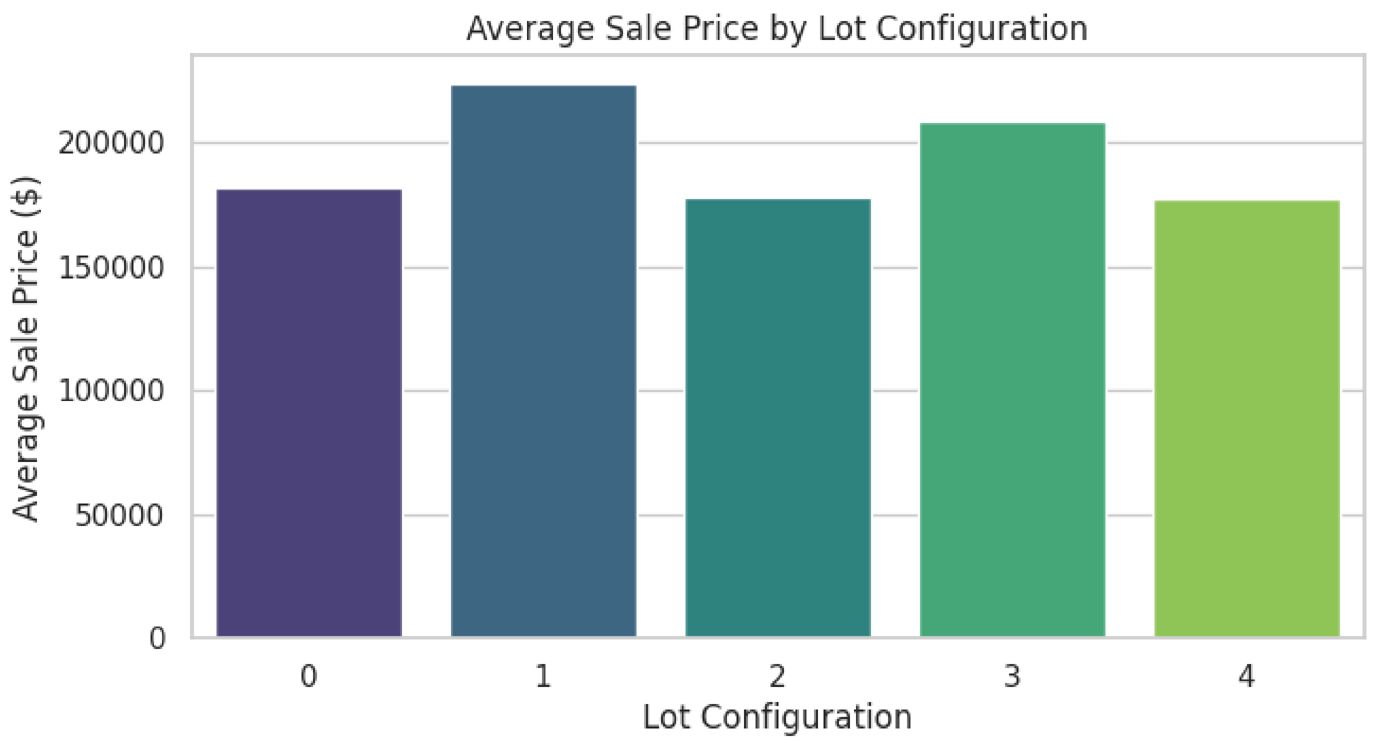
Compare the average SalePrice across different LotShape and LotConfig categories

In [ ]:
```
# Calculate the average SalePrice for each LotShape
avg_price_lotshape = df.groupby('LotShape')['SalePrice'].mean().reset_index()

# Set the size of the plot
plt.figure(figsize=(8, 4))

# Plotting the bar plot
sns.barplot(data=avg_price_lotshape, x='LotShape', y='SalePrice', palette='viridis')

# Adding titles and labels
plt.title('Average Sale Price by Lot Shape')
plt.xlabel('Lot Shape')
plt.ylabel('Average Sale Price ($)')

# Display the plot
plt.show()
```



- Regular shapes representing 0 is comparatively cheaper
- Slightly Irregular and Moderately Irregular shapes are comparatively higher in price

In [ ]:
```
# Calculate the average SalePrice for each LotConfig
avg_price_lotconfig = df.groupby('LotConfig')['SalePrice'].mean().reset_index()

# Set the size of the plot
plt.figure(figsize=(8, 4))

# Plotting the bar plot
sns.barplot(data=avg_price_lotconfig, x='LotConfig', y='SalePrice', palette='viridis')

# Adding titles and labels
plt.title('Average Sale Price by Lot Configuration')
plt.xlabel('Lot Configuration')
plt.ylabel('Average Sale Price ($)')

# Display the plot
plt.show()
```

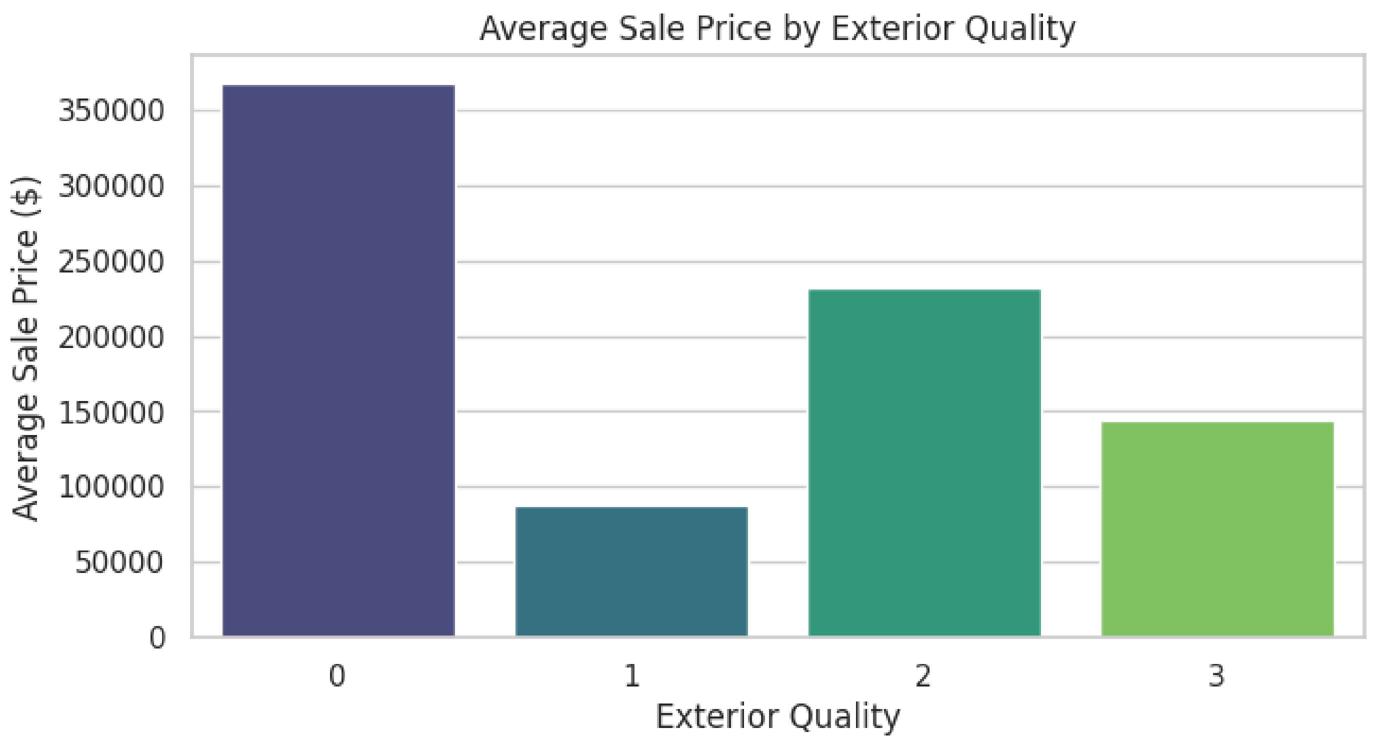


- Cul-de-sac house sits on a dead-end rounded street, facing other houses and creating a close-knit feeling between neighbors. That represents class 1 which are higher in price.
- Frontage on 3 sides of property types have second highest average price.
- Other configurations are "Frontage on 2 sides of property", "Inside lot" and "Corner lot" which have a similar average price.

Compare The Average SalePrice Across Different ExterQual, BsmtQual, and GarageQual Categories.

In [ ]:

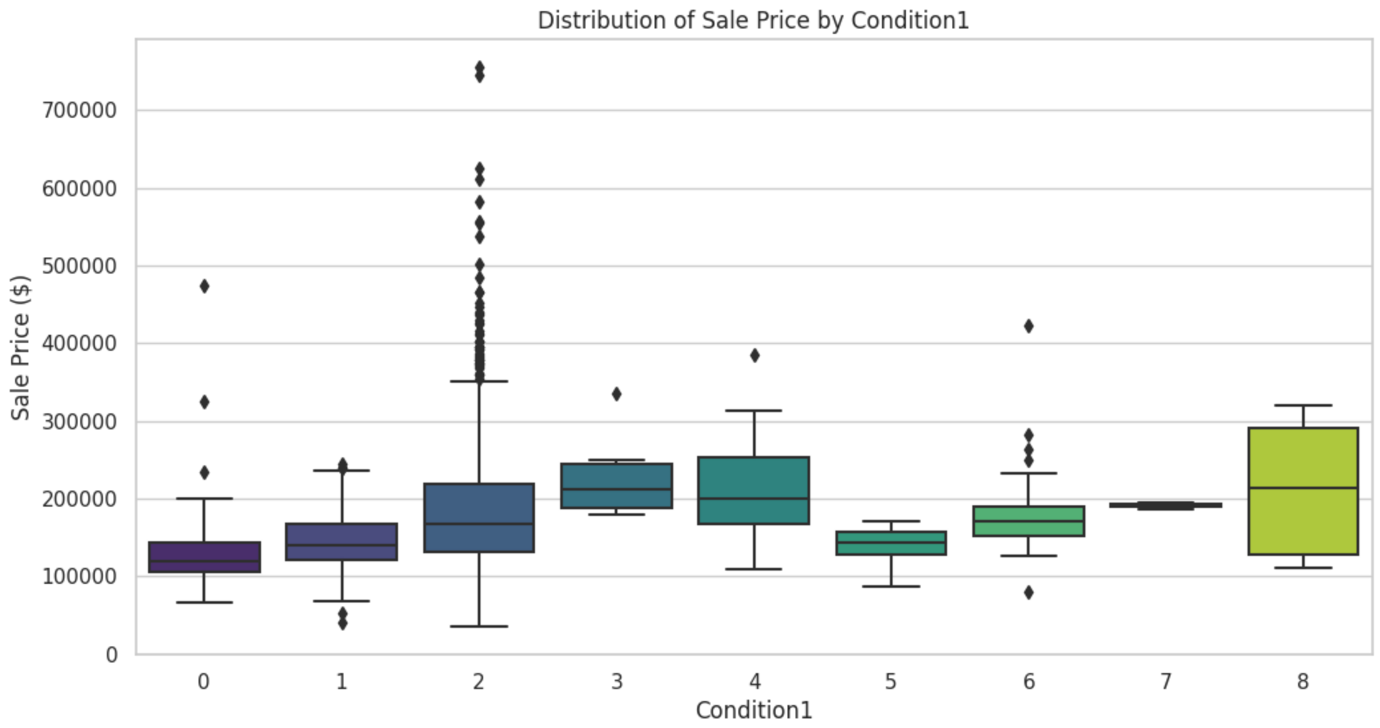
```
# Calculate the average SalePrice for each ExterQual
avg_price_exterqual = df.groupby('ExterQual')['SalePrice'].mean().reset_index()

# Set the size of the plot
plt.figure(figsize=(8, 4))

# Plotting the bar plot
sns.barplot(data=avg_price_exterqual, x='ExterQual', y='SalePrice', palette='viridis')

# Adding titles and labels
plt.title('Average Sale Price by Exterior Quality')
plt.xlabel('Exterior Quality')
plt.ylabel('Average Sale Price ($)')

# Display the plot
plt.show()
```



- 0 represents the 'Excellenet' quality properties which are obviously higher in price.
- 2 is the good quality properties which costs lower than the excellent ones.
- 3 is the typical or average types which comes next and the least expensive properties are 'fair' quality ones. There are no poor quality properties listed in the dataset.

Distribution of SalePrice by Condition1 and Condition2

In [ ]:
```
# Set the size of the plot
plt.figure(figsize=(12, 6))

# Plotting the box plot
sns.boxplot(data=df, x='Condition1', y='SalePrice', palette='viridis')

# Adding titles and labels
plt.title('Distribution of Sale Price by Condition1')
plt.xlabel('Condition1')
plt.ylabel('Sale Price ($)')

# Display the plot
plt.show()
```

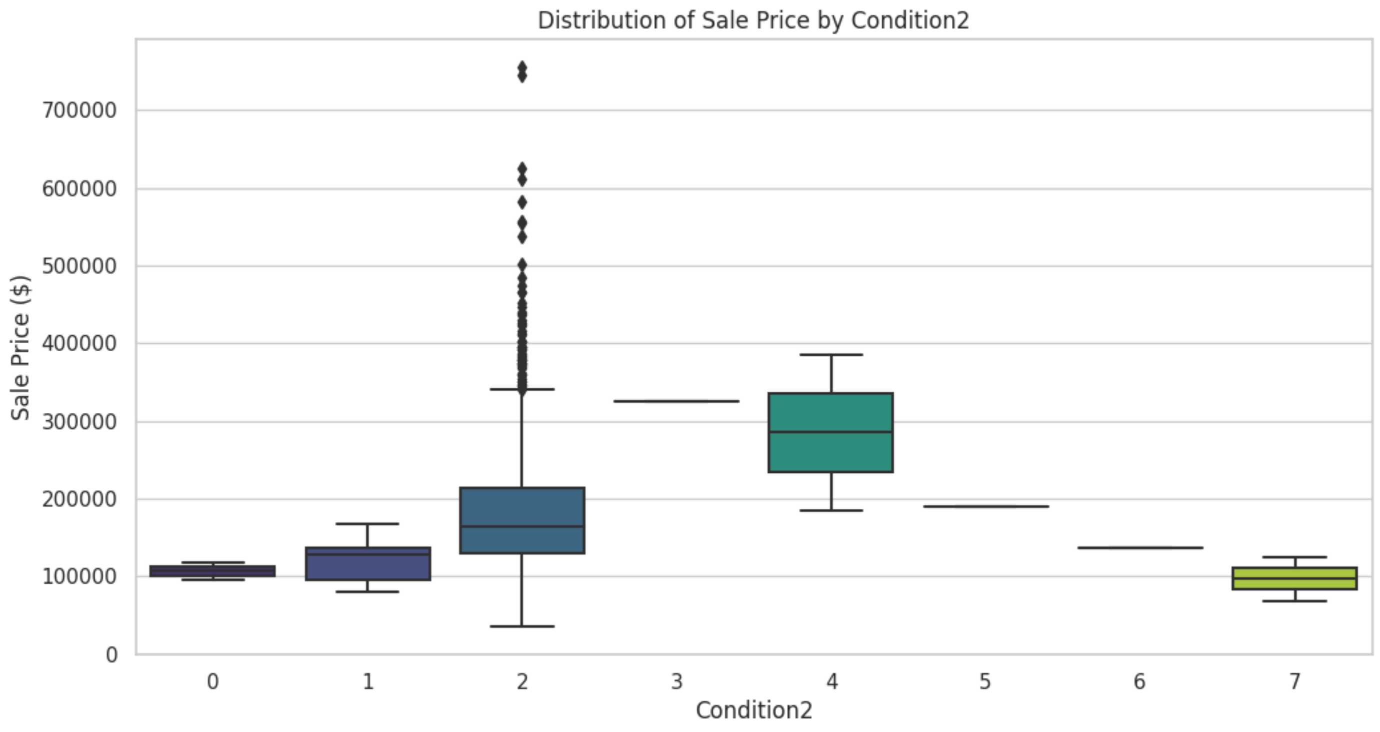


Condition 1 is the Proximity to various conditions as follows :

```
Artery   Adjacent to arterial street
Feedr    Adjacent to feeder street
Norm Normal
RRNn Within 200' of North-South Railroad
RRAn Adjacent to North-South Railroad
PosN Near positive off-site feature--park, greenbelt, etc.
PosA Adjacent to postive off-site feature
RRNe Within 200' of East-West Railroad
RRAe Adjacent to East-West Railroad
```

- 2 represents the Normal condition properties which have a lot of outliers : extremely minimum and maximum sales price. That means normal type of properties are mostly sold in various price ranges.
- 0 represents the properties which are Adjacent to arterial street. These properties are sold in various prices , sometimes in extremely high prices.

In [ ]:
```
# Set the size of the plot
plt.figure(figsize=(12, 6))

# Plotting the box plot
sns.boxplot(data=df, x='Condition2', y='SalePrice', palette='viridis')

# Adding titles and labels
plt.title('Distribution of Sale Price by Condition2')
plt.xlabel('Condition2')
plt.ylabel('Sale Price ($)')

# Display the plot
plt.show()
```



Condition 2 represents the Proximity to various conditions (if more than one is present). From the plot above, we can observe number of things ;

- Like condition 1, the normal properties are sold in extremely lower and higher prices than average
- For class 0 , the result is similar like condition 1 properties.
- For class 4 , properties near positive off-site feature--park, greenbelt, etc, these are sold in higher prices comparing to having a single condition.
- For properties within 200' of North-South Railroad, if they have multiple conditions present, they are sold in less prices comparing to having a single condition.

Statistical Analysis

In [ ]:
```
df.describe().T.style.background_gradient(axis=0, cmap='Reds')
```

Out[ ]:

	count	mean	std	min	25%	50%	
Id	2919.000000	1460.000000	842.787043	1.000000	730.500000	1460.000000	2189
MSSubClass	2919.000000	57.137718	42.517628	20.000000	20.000000	50.000000	70
MSZoning	2919.000000	3.030490	0.662386	0.000000	3.000000	3.000000	3
LotFrontage	2919.000000	69.305795	21.312345	21.000000	60.000000	69.305795	78
LotArea	2919.000000	10168.114080	7886.996359	1300.000000	7478.000000	9453.000000	11570
Street	2919.000000	0.995889	0.063996	0.000000	1.000000	1.000000	1
LotShape	2919.000000	1.947585	1.409721	0.000000	0.000000	3.000000	3
LandContour	2919.000000	2.776978	0.704391	0.000000	3.000000	3.000000	3
Utilities	2919.000000	0.001713	0.055510	0.000000	0.000000	0.000000	0
LotConfig	2919.000000	3.055841	1.604472	0.000000	2.000000	4.000000	4
LandSlope	2919.000000	0.053786	0.248750	0.000000	0.000000	0.000000	0
Neighborhood	2919.000000	12.437136	5.957992	0.000000	7.000000	12.000000	17
Condition1	2919.000000	2.040425	0.874047	0.000000	2.000000	2.000000	2
Condition2	2919.000000	2.002055	0.209431	0.000000	2.000000	2.000000	2

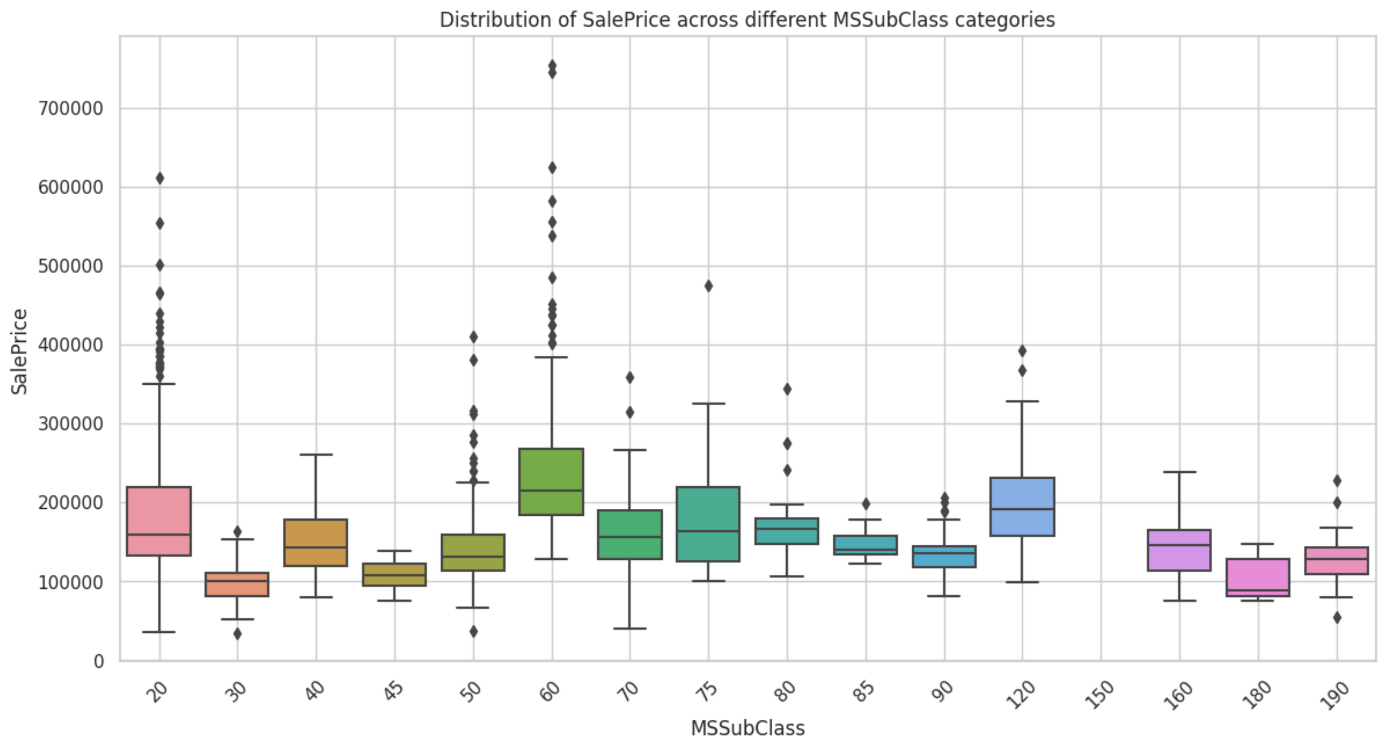
BldgType	2919.000000	0.505653	1.206513	0.000000	0.000000	0.000000	0
HouseStyle	2919.000000	3.026721	1.912937	0.000000	2.000000	2.000000	5
OverallQual	2919.000000	6.089072	1.409947	1.000000	5.000000	6.000000	7
OverallCond	2919.000000	5.564577	1.113131	1.000000	5.000000	5.000000	6
YearBuilt	2919.000000	1971.312778	30.291442	1872.000000	1953.500000	1973.000000	2001
YearRemodAdd	2919.000000	1984.264474	20.894344	1950.000000	1965.000000	1993.000000	2004
RoofStyle	2919.000000	1.396369	0.820906	0.000000	1.000000	1.000000	1
RoofMatl	2919.000000	1.063035	0.539210	0.000000	1.000000	1.000000	1
Exterior1st	2919.000000	9.625214	3.200303	0.000000	8.000000	12.000000	12
Exterior2nd	2919.000000	10.337102	3.552133	0.000000	8.000000	13.000000	13
MasVnrType	2919.000000	2.286742	0.926533	0.000000	1.000000	3.000000	3
MasVnrArea	2919.000000	102.201312	178.626089	0.000000	0.000000	0.000000	163
ExterQual	2919.000000	2.530661	0.702245	0.000000	2.000000	3.000000	3
ExterCond	2919.000000	3.708804	0.773641	0.000000	4.000000	4.000000	4
Foundation	2919.000000	1.393285	0.727061	0.000000	1.000000	1.000000	2
BsmtQual	2919.000000	2.288112	0.922771	0.000000	2.000000	2.000000	3
BsmtCond	2919.000000	2.835903	0.700631	0.000000	3.000000	3.000000	3
BsmtExposure	2919.000000	2.327509	1.151168	0.000000	2.000000	3.000000	3
BsmtFinType1	2919.000000	2.846865	1.862342	0.000000	2.000000	2.000000	5
BsmtFinSF1	2919.000000	441.423235	455.532750	0.000000	0.000000	369.000000	733
BsmtFinType2	2919.000000	4.714628	1.012141	0.000000	5.000000	5.000000	5
BsmtFinSF2	2919.000000	49.582248	169.176615	0.000000	0.000000	0.000000	0
BsmtUnfSF	2919.000000	560.772104	439.468337	0.000000	220.000000	467.000000	805
TotalBsmtSF	2919.000000	1051.777587	440.690726	0.000000	793.000000	990.000000	1302
Heating	2919.000000	1.025351	0.245678	0.000000	1.000000	1.000000	1
HeatingQC	2919.000000	1.533744	1.742548	0.000000	0.000000	0.000000	4
CentralAir	2919.000000	0.932854	0.250318	0.000000	1.000000	1.000000	1
Electrical	2919.000000	3.685509	1.047746	0.000000	4.000000	4.000000	4
1stFlrSF	2919.000000	1159.581706	392.362079	334.000000	876.000000	1082.000000	1387
2ndFlrSF	2919.000000	336.483727	428.701456	0.000000	0.000000	0.000000	704
LowQualFinSF	2919.000000	4.694416	46.396825	0.000000	0.000000	0.000000	0
GrLivArea	2919.000000	1500.759849	506.051045	334.000000	1126.000000	1444.000000	1743
BsmtFullBath	2919.000000	0.429894	0.524556	0.000000	0.000000	0.000000	1
BsmtHalfBath	2919.000000	0.061364	0.245603	0.000000	0.000000	0.000000	0
FullBath	2919.000000	1.568003	0.552969	0.000000	1.000000	2.000000	2
HalfBath	2919.000000	0.380267	0.502872	0.000000	0.000000	0.000000	1
BedroomAbvGr	2919.000000	2.860226	0.822693	0.000000	2.000000	3.000000	3
KitchenAbvGr	2919.000000	1.044536	0.214462	0.000000	1.000000	1.000000	1
KitchenQual	2919.000000	2.347379	0.834847	0.000000	2.000000	3.000000	3
TotRmsAbvGrd	2919.000000	6.451524	1.569379	2.000000	5.000000	6.000000	7

Functional	2919.000000	5.760534	0.935847	0.000000	6.000000	6.000000	6
Fireplaces	2919.000000	0.597122	0.646129	0.000000	0.000000	1.000000	1
FireplaceQu	2919.000000	3.825968	1.398569	0.000000	2.000000	4.000000	5
GarageType	2919.000000	2.483727	1.932814	0.000000	1.000000	1.000000	5
GarageYrBlt	2919.000000	1978.113406	24.867762	1895.000000	1961.500000	1978.113406	2001
GarageFinish	2919.000000	1.284001	0.897327	0.000000	1.000000	1.000000	2
GarageCars	2919.000000	1.766621	0.761494	0.000000	1.000000	2.000000	2
GarageArea	2919.000000	472.874572	215.357904	0.000000	320.000000	480.000000	576
GarageQual	2919.000000	3.904762	0.692049	0.000000	4.000000	4.000000	4
GarageCond	2919.000000	3.959233	0.568221	0.000000	4.000000	4.000000	4
PavedDrive	2919.000000	1.830764	0.537299	0.000000	2.000000	2.000000	2
WoodDeckSF	2919.000000	93.709832	126.526589	0.000000	0.000000	0.000000	168
OpenPorchSF	2919.000000	47.486811	67.575493	0.000000	0.000000	26.000000	70
EnclosedPorch	2919.000000	23.098321	64.244246	0.000000	0.000000	0.000000	0
3SsnPorch	2919.000000	2.602261	25.188169	0.000000	0.000000	0.000000	0
ScreenPorch	2919.000000	16.062350	56.184365	0.000000	0.000000	0.000000	0
PoolArea	2919.000000	2.251799	35.663946	0.000000	0.000000	0.000000	0
MiscVal	2919.000000	50.825968	567.402211	0.000000	0.000000	0.000000	0
MoSold	2919.000000	6.213087	2.714762	1.000000	4.000000	6.000000	8
YrSold	2919.000000	2007.792737	1.314964	2006.000000	2007.000000	2008.000000	2009
SaleType	2919.000000	7.491607	1.593719	0.000000	8.000000	8.000000	8
SaleCondition	2919.000000	3.779034	1.078241	0.000000	4.000000	4.000000	4
SalePrice	1460.000000	180921.195890	79442.502883	34900.000000	129975.000000	163000.000000	214000
Latitude	2919.000000	42.036187	0.023631	41.990092	42.022005	42.033483	42
Longitude	2919.000000	-93.644576	0.020378	-93.680265	-93.652025	-93.648429	-93

- High standard deviation for features like LotArea,MiscVal, WoodDeckSF, GrLivArea, 2ndFlrSF, 1stFlrSF indicate that they have a very wide range which means these property features have a great variety in the market.
- In features like LotArea, MasVnrArea, BsmtFinSF1 and TotalBsmtSF etc the IQR range is very high which indicates that there could be outliers.

Compare SalePrice Across Different MSSubClass categories

In [ ]:
```
# Box Plot to visualize the distribution of SalePrice across different MSSubClass catego
plt.figure(figsize=(14, 7))
sns.boxplot(x='MSSubClass', y='SalePrice', data=df)
plt.title('Distribution of SalePrice across different MSSubClass categories')
plt.xlabel('MSSubClass')
plt.ylabel('SalePrice')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



Practical Implication:

Different building classes (MSSubClass) have significantly different average SalePrices. This suggests that the type of building class is an important factor affecting property prices. Further investigation into which specific classes have higher or lower average SalePrices can provide insights for real estate pricing strategies.

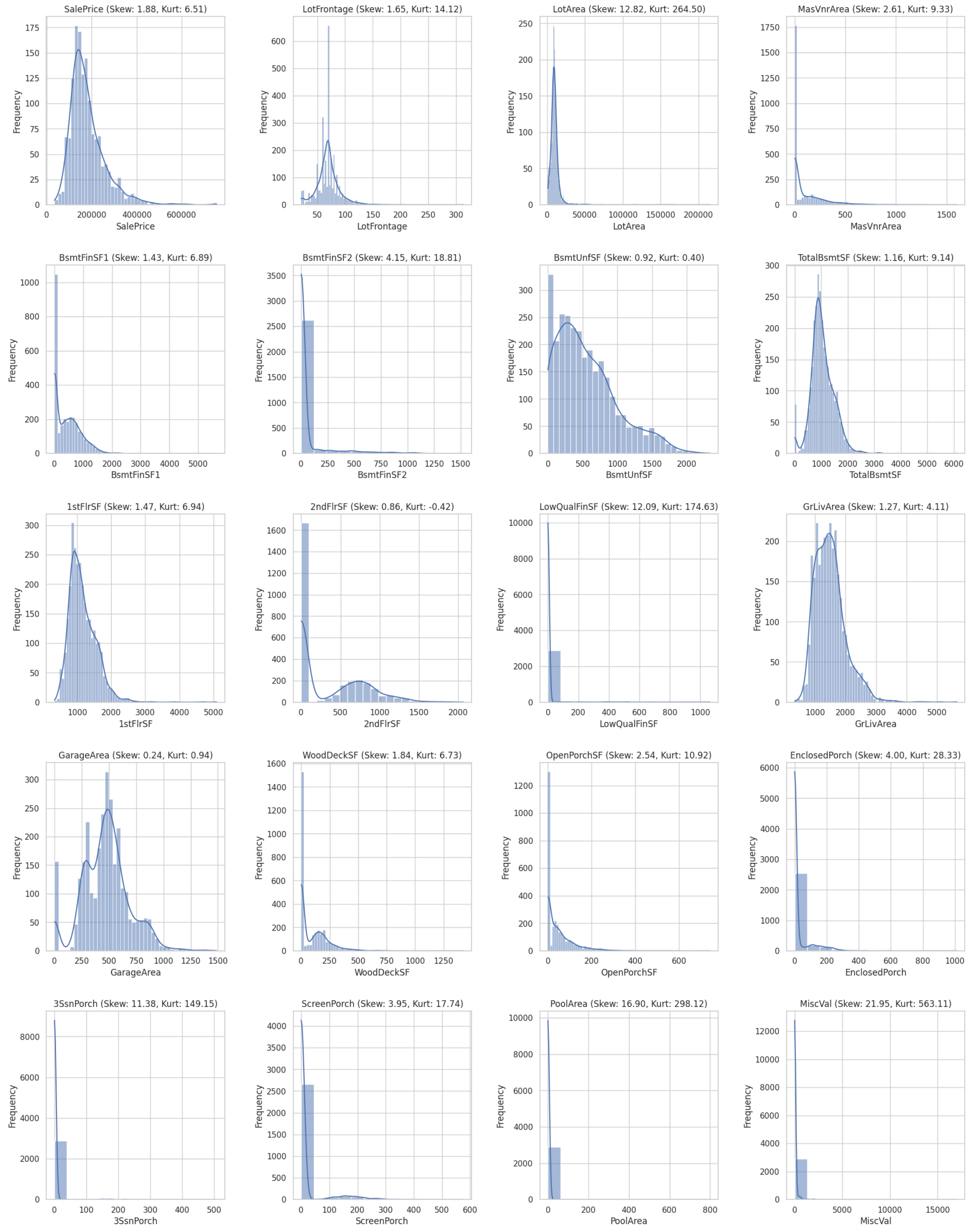
Skewness, Kurtosis and Normality Tests

We will assess the distributions and check for normality for few continuous features

```
In [ ]: # Features to assess
features = [
    'SalePrice', 'LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
    'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
    'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
    'ScreenPorch', 'PoolArea', 'MiscVal'
]

# Initialize dictionaries to store results
skewness_results = {}
kurtosis_results = {}
shapiro_results = {}

# Number of rows and columns for subplots
n_rows = 5
n_cols = 4

# Create a figure and a set of subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 25))
fig.tight_layout(pad=5.0)

# Flatten axes for easy iteration
axes = axes.flatten()

# Calculate skewness, kurtosis, and Shapiro-Wilk test, and plot distributions
for i, feature in enumerate(features):
    skewness_results[feature] = stats.skew(df[feature].dropna())
    kurtosis_results[feature] = stats.kurtosis(df[feature].dropna())
    shapiro_results[feature] = stats.shapiro(df[feature].dropna())

    sns.histplot(df[feature].dropna(), kde=True, ax=axes[i])
    axes[i].set_title(f'{feature} (Skew: {skewness_results[feature]:.2f}, Kurt: {kurtosi
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel('Frequency')

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
```

```
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf
_as_na option is deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [ ]:
```
# Display skewness and kurtosis
print("Skewness:")
print(skewness_results)

print("\nKurtosis:")
print(kurtosis_results)

# Display Shapiro-Wilk test results
```

```
print("\nShapiro-Wilk Test Results (W, p-value):")
print(shapiro_results)
```

```
Skewness:
{'SalePrice': 1.880940746034036, 'LotFrontage': 1.6455737855221888, 'LotArea': 12.822431
401556724, 'MasVnrArea': 2.6115493751087344, 'BsmtFinSF1': 1.4252334408817189, 'BsmtFinS
F2': 4.146033635959022, 'BsmtUnfSF': 0.9195083116601191, 'TotalBsmtSF': 1.16248374933319
72, '1stFlrSF': 1.4696044169256821, '2ndFlrSF': 0.8616747488436027, 'LowQualFinSF': 12.0
88761003370664, 'GrLivArea': 1.269357688230336, 'GarageArea': 0.241217781017102, 'WoodDe
ckSF': 1.8424328111184782, 'OpenPorchSF': 2.5351137294802557, 'EnclosedPorch': 4.0038912
20540856, '3SsnPorch': 11.376064682827481, 'ScreenPorch': 3.9466937029936977, 'PoolAre
a': 16.89832791614449, 'MiscVal': 21.9471948077491}

Kurtosis:
{'SalePrice': 6.509812011089439, 'LotFrontage': 14.120786759828889, 'LotArea': 264.49663
20739909, 'MasVnrArea': 9.33348272100738, 'BsmtFinSF1': 6.8943404611257595, 'BsmtFinSF
2': 18.809694890446103, 'BsmtUnfSF': 0.4020356208195319, 'TotalBsmtSF': 9.13752889580565
3, '1stFlrSF': 6.942514097204564, '2ndFlrSF': -0.4235925144377295, 'LowQualFinSF': 174.6
312561915874, 'GrLivArea': 4.112492367575526, 'GarageArea': 0.9374667685912672, 'WoodDec
kSF': 6.7279532273976965, 'OpenPorchSF': 10.916571954391017, 'EnclosedPorch': 28.3272684
78734087, '3SsnPorch': 149.1519757850386, 'ScreenPorch': 17.74421342304928, 'PoolArea':
298.11980075600536, 'MiscVal': 563.1067782423772}

Shapiro-Wilk Test Results (W, p-value):
{'SalePrice': ShapiroResult(statistic=0.869672954082489, pvalue=3.2072044604461286e-33),
'LotFrontage': ShapiroResult(statistic=0.8919751644134521, pvalue=3.2085530937645336e-4
1), 'LotArea': ShapiroResult(statistic=0.4345884323120117, pvalue=0.0), 'MasVnrArea': Sh
apiroResult(statistic=0.639785885810852, pvalue=0.0), 'BsmtFinSF1': ShapiroResult(statis
tic=0.8577058911323547, pvalue=1.401298464324817e-45), 'BsmtFinSF2': ShapiroResult(stati
stic=0.33327603340148926, pvalue=0.0), 'BsmtUnfSF': ShapiroResult(statistic=0.9284115433
692932, pvalue=2.938860428947383e-35), 'TotalBsmtSF': ShapiroResult(statistic=0.93872964
38217163, pvalue=3.9526188564231426e-33), '1stFlrSF': ShapiroResult(statistic=0.92306387
42446899, pvalue=2.8795466442184694e-36), '2ndFlrSF': ShapiroResult(statistic=0.76513379
8122406, pvalue=0.0), 'LowQualFinSF': ShapiroResult(statistic=0.07689714431762695, pvalu
e=0.0), 'GrLivArea': ShapiroResult(statistic=0.9338352680206299, pvalue=3.58456624872190
72e-34), 'GarageArea': ShapiroResult(statistic=0.9756833910942078, pvalue=6.415979670214
872e-22), 'WoodDeckSF': ShapiroResult(statistic=0.7558260560035706, pvalue=0.0), 'OpenPo
rchSF': ShapiroResult(statistic=0.7221224308013916, pvalue=0.0), 'EnclosedPorch': Shapir
oResult(statistic=0.41428905725479126, pvalue=0.0), '3SsnPorch': ShapiroResult(statistic
=0.07930713891983032, pvalue=0.0), 'ScreenPorch': ShapiroResult(statistic=0.318855702877
0447, pvalue=0.0), 'PoolArea': ShapiroResult(statistic=0.03639882802963257, pvalue=0.0),
'MiscVal': ShapiroResult(statistic=0.061026036739349365, pvalue=0.0)}
```

Interpretation of Statistical Tests

Skewness:

- A skewness value greater than 1 indicates a highly skewed distribution.
- Most variables have high skewness values, indicating that their distributions are not symmetrical.
- 'LotArea', 'LowQualFinSF', 'PoolArea', and 'MiscVal' have extremely high skewness values, indicating severe skewness.

Kurtosis:

- A kurtosis value greater than 3 indicates a leptokurtic distribution (heavy-tailed).
- Most variables have high kurtosis values, indicating that their distributions have heavy tails.
- 'LotArea', 'LowQualFinSF', 'PoolArea', and 'MiscVal' have extremely high kurtosis values, indicating extremely heavy-tailed distributions.

Shapiro-Wilk Test Results:

- The Shapiro-Wilk test is a normality test that checks if the data follows a normal distribution.

- The test produces a statistic (W) and a p-value.
- A p-value less than 0.05 indicates that the data does not follow a normal distribution.
- Most variables have p-values very close to 0, indicating that they do not follow a normal distribution.
- Only 'GarageArea' has a p-value greater than 0.05, indicating that it may follow a normal distribution.

Outlier Detection and Handling

We will create few functions to handle outliers. Instead of removing the datapoints completely, we will replace them with lower and higher threshold values. We will keep some features out from this process. For example, id, latitude, longitude etc.

In [ ]:
```
# List of columns to exclude from outlier handling
excluded_columns = ['Id', 'Neighborhood', 'MoSold', 'YrSold', 'Neighborhood_Decoded', 'N

# Selecting all numeric columns
numeric_vars = df.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Filtering out the excluded columns
numeric_vars = [col for col in numeric_vars if col not in excluded_columns]

# Function to calculate lower and upper thresholds
def outlier_thresholds(dataframe, col_name, q1=0.1, q3=0.9):
    quartile1 = dataframe[col_name].quantile(q1)
    quartile3 = dataframe[col_name].quantile(q3)
    interquantile_range = quartile3 - quartile1
    up_limit = quartile3 + 1.5 * interquantile_range
    low_limit = quartile1 - 1.5 * interquantile_range
    return low_limit, up_limit

# Function to check for outliers in a specific column
def check_outlier(dataframe, col_name):
    if pd.api.types.is_numeric_dtype(dataframe[col_name]):
        low_limit, up_limit = outlier_thresholds(dataframe, col_name)
        return dataframe[(dataframe[col_name] > up_limit) | (dataframe[col_name] < low_l
    else:
        return pd.Series([])

# Function to replace outliers with defined thresholds
def replace_with_thresholds(dataframe, variable, q1=0.1, q3=0.9):
    low_limit, up_limit = outlier_thresholds(dataframe, variable, q1, q3)

    # Get the current dtype of the column
    col_dtype = dataframe[variable].dtype

    # Cast thresholds to the appropriate dtype
    if pd.api.types.is_integer_dtype(col_dtype):
        low_limit = int(low_limit)
        up_limit = int(up_limit)

    dataframe.loc[(dataframe[variable] < low_limit), variable] = low_limit
    dataframe.loc[(dataframe[variable] > up_limit), variable] = up_limit

# Iterating through each numeric column to check and handle outliers
for col in numeric_vars:
    outliers = check_outlier(df, col)
    if not outliers.empty:
        print(f"Outliers found in {col}. Handling outliers...")
        replace_with_thresholds(df, col)

print("Outlier handling completed.")
```

```
Outliers found in MSZoning. Handling outliers...
Outliers found in LotFrontage. Handling outliers...
Outliers found in LotArea. Handling outliers...
Outliers found in Street. Handling outliers...
Outliers found in LandContour. Handling outliers...
Outliers found in Utilities. Handling outliers...
Outliers found in LandSlope. Handling outliers...
Outliers found in Condition1. Handling outliers...
Outliers found in Condition2. Handling outliers...
Outliers found in OverallCond. Handling outliers...
Outliers found in RoofMatl. Handling outliers...
Outliers found in MasVnrArea. Handling outliers...
Outliers found in ExterQual. Handling outliers...
Outliers found in BsmtCond. Handling outliers...
Outliers found in BsmtFinSF1. Handling outliers...
Outliers found in BsmtFinType2. Handling outliers...
Outliers found in BsmtFinSF2. Handling outliers...
Outliers found in TotalBsmtSF. Handling outliers...
Outliers found in Heating. Handling outliers...
Outliers found in CentralAir. Handling outliers...
Outliers found in Electrical. Handling outliers...
Outliers found in 1stFlrSF. Handling outliers...
Outliers found in LowQualFinSF. Handling outliers...
Outliers found in GrLivArea. Handling outliers...
Outliers found in BsmtFullBath. Handling outliers...
Outliers found in BsmtHalfBath. Handling outliers...
Outliers found in FullBath. Handling outliers...
Outliers found in BedroomAbvGr. Handling outliers...
Outliers found in KitchenAbvGr. Handling outliers...
Outliers found in KitchenQual. Handling outliers...
Outliers found in TotRmsAbvGrd. Handling outliers...
Outliers found in Functional. Handling outliers...
Outliers found in Fireplaces. Handling outliers...
Outliers found in GarageYrBlt. Handling outliers...
Outliers found in GarageQual. Handling outliers...
Outliers found in GarageCond. Handling outliers...
Outliers found in PavedDrive. Handling outliers...
Outliers found in WoodDeckSF. Handling outliers...
Outliers found in OpenPorchSF. Handling outliers...
Outliers found in EnclosedPorch. Handling outliers...
Outliers found in 3SsnPorch. Handling outliers...
Outliers found in ScreenPorch. Handling outliers...
Outliers found in PoolArea. Handling outliers...
Outliers found in MiscVal. Handling outliers...
Outliers found in SaleType. Handling outliers...
Outliers found in SaleCondition. Handling outliers...
Outliers found in SalePrice. Handling outliers...
Outlier handling completed.
```

Correlation Matrix

We will perform correlation matrix to identify the features which have high correlation with Salesprice but before that we need to remove those features which have only one unique value.

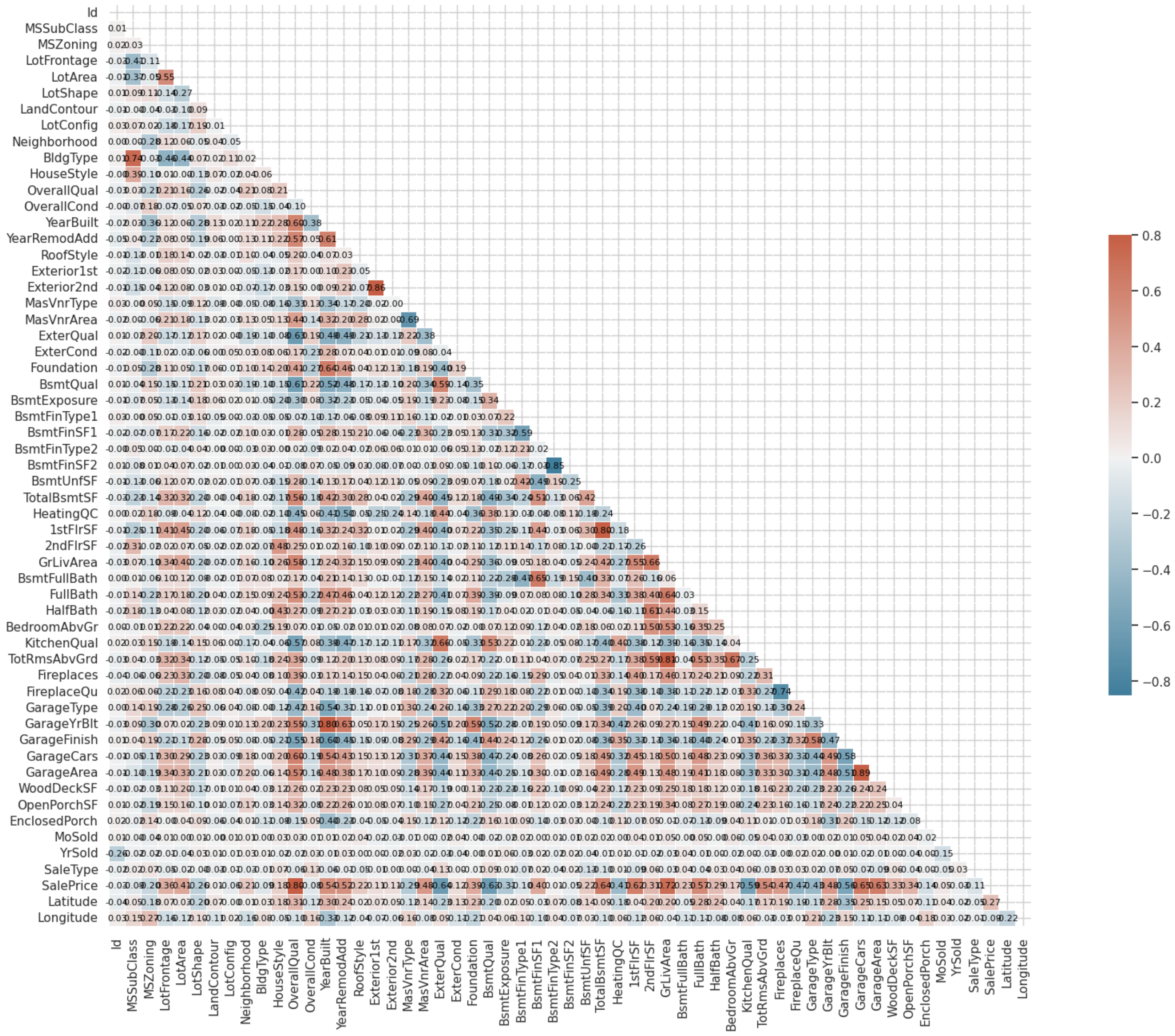
In [ ]:
```
unique_value_counts = df.nunique()
variables_with_one_unique_value = unique_value_counts[unique_value_counts == 1].index.to

print("Variables with unique value count of 1:")
print(variables_with_one_unique_value)
```

```
Variables with unique value count of 1:
['Street', 'Utilities', 'LandSlope', 'Condition1', 'Condition2', 'RoofMatl', 'BsmtCond',
'Heating', 'CentralAir', 'Electrical', 'LowQualFinSF', 'BsmtHalfBath', 'KitchenAbvGr',
'Functional', 'GarageQual', 'GarageCond', 'PavedDrive', '3SsnPorch', 'ScreenPorch', 'Poo
lArea', 'MiscVal', 'SaleCondition']
```

```
In [ ]: # dropping the features which have only one unique value
        df.drop(variables_with_one_unique_value, axis=1, inplace=True)
```

```
In [ ]: # correlation matrix
        fig, axis = plt.subplots(figsize=(25, 15))
        numeric_df = df.select_dtypes(include=[np.number])

        correlation = numeric_df.corr('pearson')
        mask = np.triu(np.ones_like(correlation, dtype=bool))
        cmap = sns.diverging_palette(230, 20, as_cmap=True)
        sns.heatmap(correlation, mask=mask, cmap=cmap, vmax=0.8, center=0, annot=True, fmt='.2f'
                    square=True, linewidths=.5, cbar_kws={"shrink": .5},
                    annot_kws={"size": 8, "color": 'black'})
```

```
Out[ ]: <Axes: >
```



Few features have high correlation pair. We will drop those features to avoid data leakage. We will do that after we create new features which will have high correlation.

Feature Engineering

We will be creating few features now

```
In [ ]: # 1. TotalLivingArea
        df['TotalLivingArea'] = df['GrLivArea'] + df['TotalBsmtSF'] + df['1stFlrSF']
```

```
# 2. TotalBathroom
df['TotalBathroom'] = df['FullBath'] + df['HalfBath']

# 3. TotalPorchArea
df['TotalPorchArea'] = df['OpenPorchSF'] + df['EnclosedPorch']

# 4. AvgLotSize
df['AvgLotSize'] = df['LotArea'] / df['LotFrontage']

# 5. HouseAge
df['HouseAge'] = df['YrSold'] - df['YearBuilt']

# 6. YearsSinceRemodel
df['YearsSinceRemodel'] = df['YrSold'] - df['YearRemodAdd']

# 7. TotalOutdoorSpace
df['TotalOutdoorSpace'] = df['WoodDeckSF'] + df['OpenPorchSF'] + df['EnclosedPorch']

# 8. BsmtFinRatio
df['BsmtFinRatio'] = (df['BsmtFinSF1'] + df['BsmtFinSF2']) / df['TotalBsmtSF']

# 9. AboveGradeLivingRatio
df['AboveGradeLivingRatio'] = df['GrLivArea'] / df['TotalLivingArea']

# 10. BathroomPerBedroom
df['BathroomPerBedroom'] = df['TotalBathroom'] / df['BedroomAbvGr']

# 11. FireplaceQuality
df['FireplaceQuality'] = df['FireplaceQu'].apply(lambda x: 1 if x == 'Ex' else 0)

# 12. GarageSize
df['GarageSize'] = df['GarageCars'] * df['GarageArea']

# 13. NeighborhoodQuality
df['NeighborhoodQuality'] = df['Neighborhood_Decoded'].apply(lambda x: 1 if x == 'High'

# 14. HouseStyleQuality
df['HouseStyleQuality'] = df['HouseStyle'].apply(lambda x: 1 if x in ['2Story', '1.5Fin'

# 15. TotalRoomDensity
df['TotalRoomDensity'] = df['TotRmsAbvGrd'] / df['TotalLivingArea']
```

We need to find high correlation feature pairs so that we can remove one from each pair so that our model can avoid overfitting

We will also remove the categorical features which were used for Data Analysis. For modelling we do not need them now. We will also remove the Id column.

In [ ]:
```
#### Checking for the high correlation pairs

def find_high_correlation_features(df, threshold=0.95):
    # Select only numerical columns
    numerical_df = df.select_dtypes(exclude=['object', 'datetime'])

    # Remove the Id column (assuming it's named 'Id')
    numerical_df = numerical_df.drop('Id', axis=1)

    # Calculate the correlation matrix
    corr_matrix = numerical_df.corr()

    # Get the upper triangle of the correlation matrix (excluding the diagonal)
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
    # Find the features with high correlation (above the threshold)
    high_corr_features = [(column, row) for column in upper.columns for row in upper.ind

    return high_corr_features
```

In [ ]:
```
high_corr_features = find_high_correlation_features(df, threshold=0.95)
print(high_corr_features)
```

```
[('HouseAge', 'YearBuilt'), ('YearsSinceRemodel', 'YearRemodAdd')]
```

Now that we have got the high correlation pairs, we will remove some of the features along with Id, and two object features that we have

In [ ]:
```
# removing
df = df.drop(['YearBuilt', 'YearRemodAdd' , 'Id', 'Neighborhood_Decoded', 'Neighborhood_
```

In [ ]:
```
# checking for null values one more time
df.isnull().sum()
```

Out[ ]:
```
MSSubClass              0
MSZoning                0
LotFrontage             0
LotArea                 0
LotShape                0
LandContour             0
LotConfig               0
Neighborhood            0
BldgType                0
HouseStyle              0
OverallQual             0
OverallCond             0
RoofStyle               0
Exterior1st             0
Exterior2nd             0
MasVnrType              0
MasVnrArea              0
ExterQual               0
ExterCond               0
Foundation              0
BsmtQual                0
BsmtExposure            0
BsmtFinType1            0
BsmtFinSF1              0
BsmtFinType2            0
BsmtFinSF2              0
BsmtUnfSF               0
TotalBsmtSF             0
HeatingQC               0
1stFlrSF                0
2ndFlrSF                0
GrLivArea               0
BsmtFullBath            0
FullBath                0
HalfBath                0
BedroomAbvGr            0
KitchenQual             0
TotRmsAbvGrd            0
Fireplaces              0
FireplaceQu             0
GarageType              0
GarageYrBlt             0
GarageFinish            0
GarageCars              0
```

```
GarageArea                0
WoodDeckSF                0
OpenPorchSF               0
EnclosedPorch             0
MoSold                    0
YrSold                    0
SaleType                  0
SalePrice              1459
Latitude                  0
Longitude                 0
TotalLivingArea           0
TotalBathroom             0
TotalPorchArea            0
AvgLotSize                0
HouseAge                  0
YearsSinceRemodel         0
TotalOutdoorSpace         0
BsmtFinRatio             78
AboveGradeLivingRatio     0
BathroomPerBedroom        2
FireplaceQuality          0
GarageSize                0
NeighborhoodQuality       0
HouseStyleQuality         0
TotalRoomDensity          0
dtype: int64
```

In [ ]:
```
# handling BsmtFinRatio null values with mean

df['BsmtFinRatio'] = df['BsmtFinRatio'].fillna(df['BsmtFinRatio'].mean())
```

In [ ]:
```
df['BathroomPerBedroom'] = df['BathroomPerBedroom'].fillna(df['BathroomPerBedroom'].mean
```

In [ ]:
```
# Separate rows with and without SalePrice values
X_test = df[df['SalePrice'].isnull()]  # rows without SalePrice values
X_train = df[df['SalePrice'].notnull()]  # rows with SalePrice values

y_train = X_train['SalePrice']
y_test = X_test['SalePrice']


X_train = X_train.drop('SalePrice', axis=1)
X_test = X_test.drop('SalePrice', axis=1)
```

In [ ]:
```
# Separate the actual test set (rows with null SalePrice)
X_test = df[df['SalePrice'].isnull()].drop('SalePrice', axis=1)

# Get the training data (rows with non-null SalePrice)
train_data = df[df['SalePrice'].notnull()]

# Separate features and target
X = train_data.drop('SalePrice', axis=1)
y = train_data['SalePrice']

# Split the data into train and validation sets (80% train, 20% validation)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

Checking the infinite features and dropping them

In [ ]:
```
inf_features = np.where(np.isinf(X_train).sum(axis=0) > 0)[0]
print(X_train.columns[inf_features])
```

```
Index(['BathroomPerBedroom'], dtype='object')
```

```
In [ ]: inf_features = np.where(np.isinf(X_val).sum(axis=0) > 0)[0]
        print(X_val.columns[inf_features])
```

```
Index(['BathroomPerBedroom'], dtype='object')
```

So we are dropping BathroomPerBedroom feature so that it does not create any problem in future.

```
In [ ]: X_train = X_train.drop('BathroomPerBedroom', axis=1)
        X_val = X_val.drop('BathroomPerBedroom', axis=1)
```

Feature Importance and Feature Selection

We will now determine which features are most important ones to predict SalePrice. We dont want a lot of features in our model.

We are using Random Forest Regresor for feature importance and selection The reason behind using this model are :

- Robustness to Overfitting:

Random Forests are an ensemble method, combining multiple decision trees. This makes them less prone to overfitting compared to single decision trees. The random sampling of both observations (bagging) and features at each split helps in creating a diverse set of trees, further reducing overfitting.

- Handles Non-linear Relationships:

Unlike linear methods (e.g., Lasso, Ridge regression), Random Forests can capture non-linear relationships between features and the target variable. This is particularly useful in real-world scenarios where relationships are often complex and non-linear.

- Implicit Feature Selection:

Random Forests perform feature selection inherently during the tree-building process. At each split, the algorithm chooses the best feature among a random subset, naturally prioritizing more important features.

- Measures Feature Interactions:

Random Forests can capture feature interactions, which simple linear models or correlation-based methods might miss. This is particularly useful in complex datasets where features might work together in non-obvious ways.

- Stability:

The feature importance scores from Random Forests are generally more stable compared to single decision trees. The aggregation of many trees helps smooth out the variability in importance scores.

- No Assumptions About Data Distribution:

Unlike parametric methods, Random Forests don't make assumptions about the underlying data distribution. This makes them versatile and applicable to a wide range of datasets. In our case, we have different kind of distributions of certain features which we have observed previously.

- Handles High-Dimensional Data:

Random Forests can effectively handle datasets with a large number of features relative to the number of observations. This is particularly useful in scenarios where you have many potential predictors. For our case, we have a lot of features already.

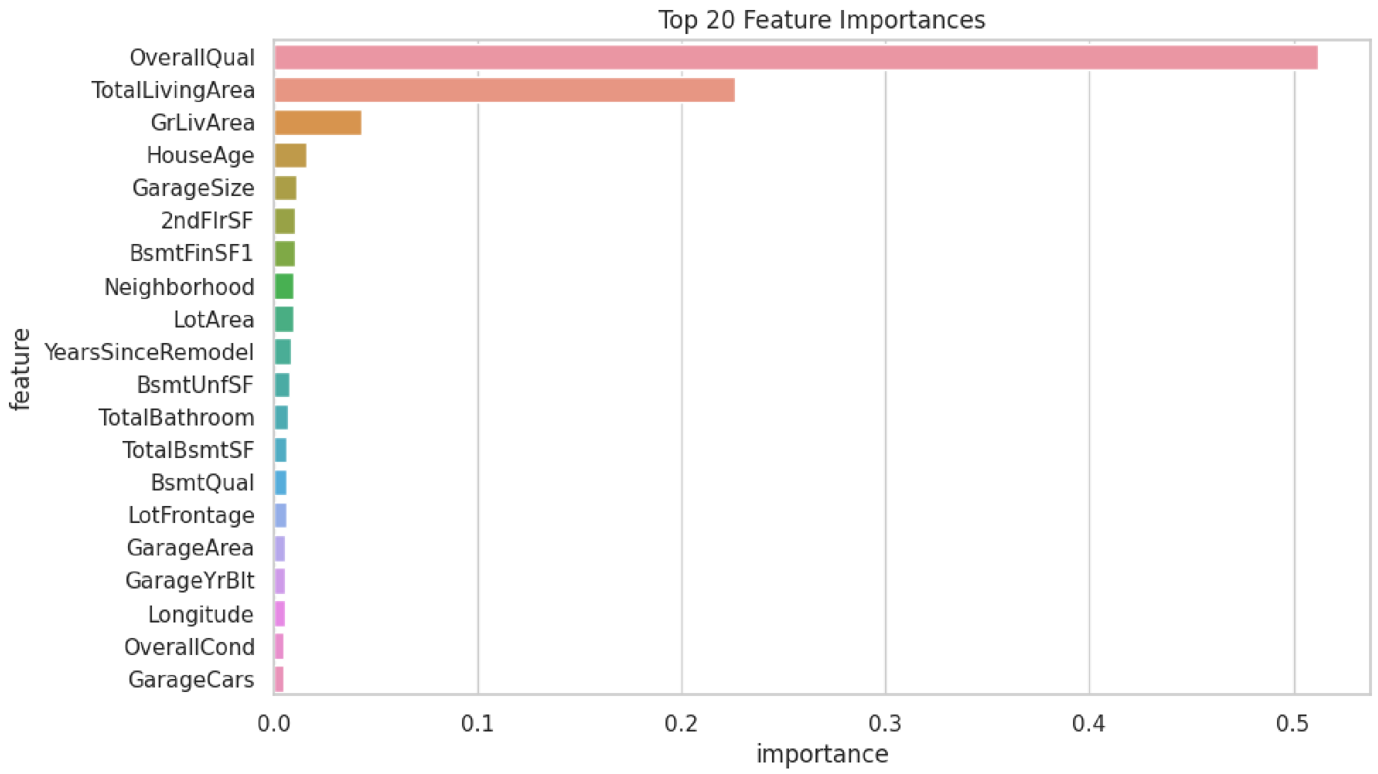
- Built-in Cross-Validation:

Random Forests use out-of-bag (OOB) samples for an internal cross-validation mechanism. This provides a reliable estimate of feature importance without needing a separate validation set.

- Resistance to Outliers:

We have seen that some of our features had outliers and we ahve handled them. However, we wanted to use Random forest for it's advanced algorithm. Random Forests are generally robust to outliers and noisy data, which can be beneficial when dealing with real-world datasets.

In [ ]:
```
rf = RandomForestRegressor(n_estimators=1000, random_state=42)
rf.fit(X_train, y_train)

# Get feature importances
importances = rf.feature_importances_
feature_importance = pd.DataFrame({'feature': X_train.columns, 'importance': importances
feature_importance = feature_importance.sort_values('importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importance.head(20))
plt.title('Top 20 Feature Importances')
plt.show()
```



In [ ]:
```
feature_importance = pd.DataFrame({'feature': X_train.columns, 'importance': rf.feature_
feature_importance = feature_importance.sort_values('importance', ascending=False)
feature_importance.head(20)
```

Out[ ]:

	feature	importance
10	OverallQual	0.512267
53	TotalLivingArea	0.225965
31	GrLivArea	0.042644

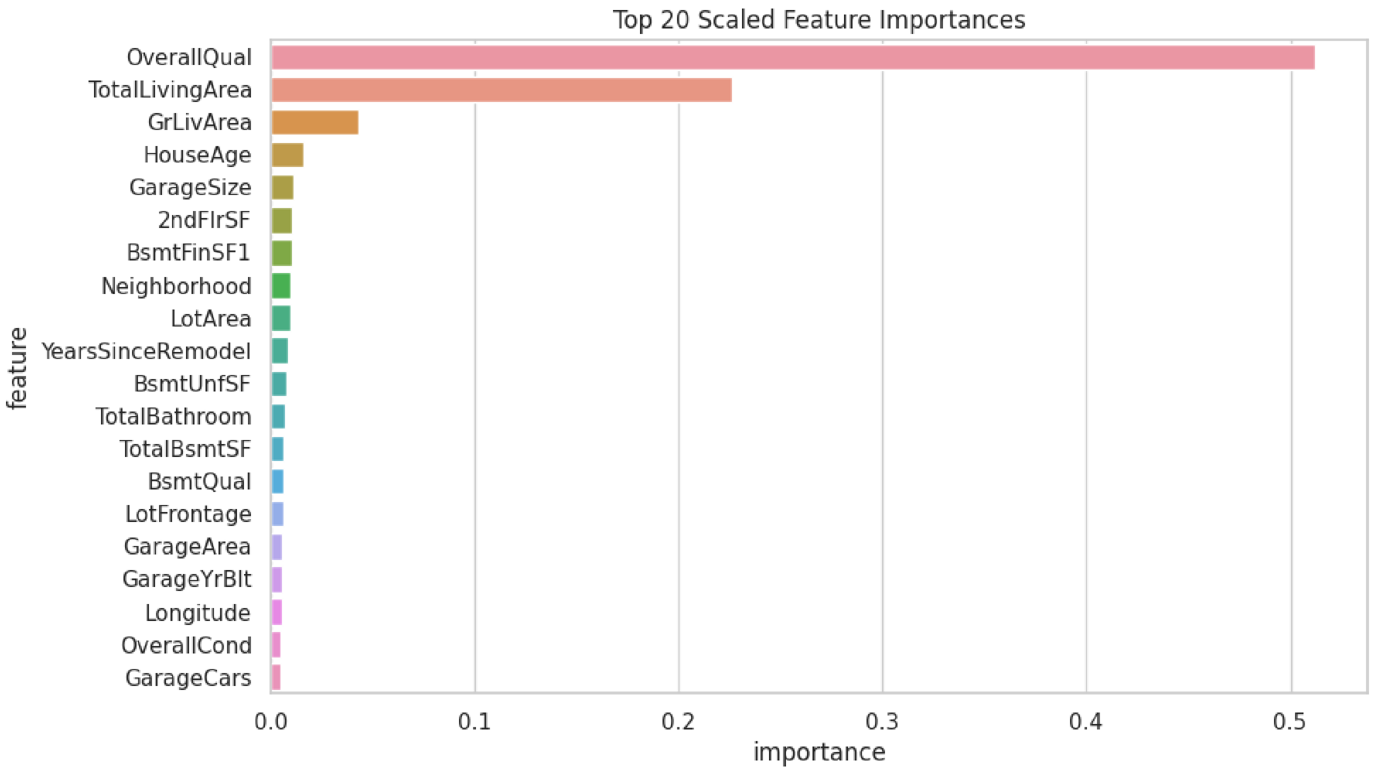
57	HouseAge	0.016196
63	GarageSize	0.011308
30	2ndFlrSF	0.010321
23	BsmtFinSF1	0.010297
7	Neighborhood	0.009895
3	LotArea	0.009539
58	YearsSinceRemodel	0.008367
26	BsmtUnfSF	0.007508
54	TotalBathroom	0.006762
27	TotalBsmtSF	0.006470
20	BsmtQual	0.006198
2	LotFrontage	0.006106
44	GarageArea	0.005274
41	GarageYrBlt	0.005267
52	Longitude	0.005191
11	OverallCond	0.005096
43	GarageCars	0.004966

We want to try some other methods for feature importance and selection. That is why we want to try scaling first.

In [ ]:
```
# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

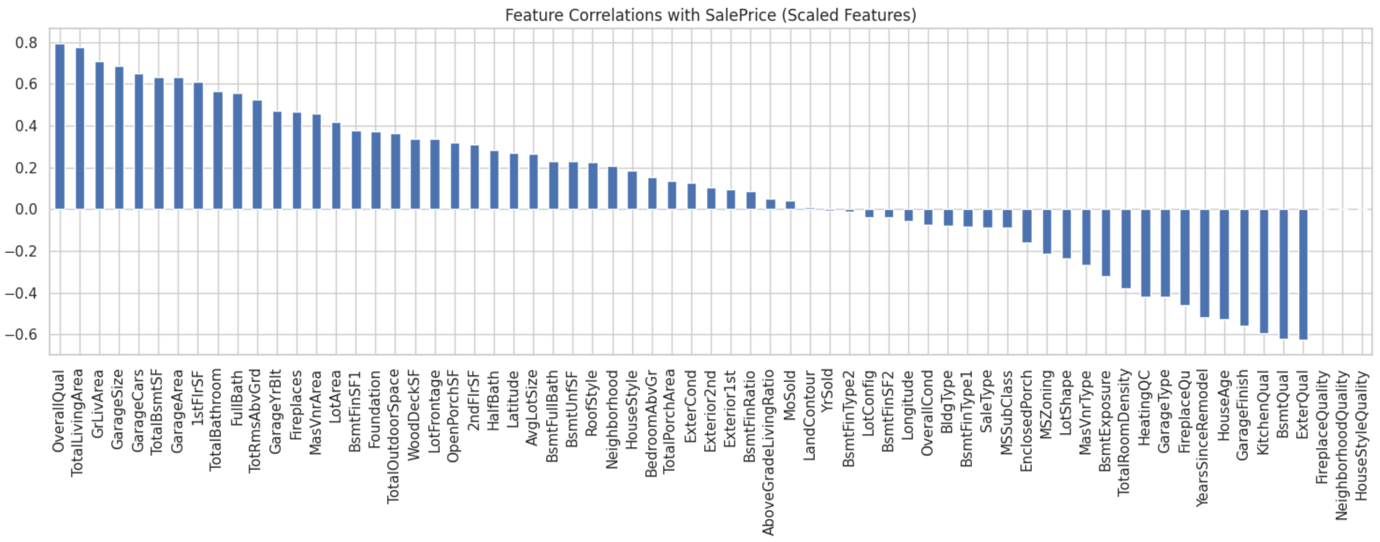
In [ ]:
```
# Fit Random Forest on scaled data
rf = RandomForestRegressor(n_estimators=1000, random_state=42)
rf.fit(X_train_scaled, y_train)

# Get feature importances
importances = rf.feature_importances_
feature_importance = pd.DataFrame({'feature': X_train.columns, 'importance': importances
feature_importance = feature_importance.sort_values('importance', ascending=False)
```

In [ ]:
```
# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importance.head(20))
plt.title('Top 20 Scaled Feature Importances')
plt.show()
```



Using Correlation For Feature Importance

```
In [ ]: X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns, in

        # Combine scaled features with target for correlation analysis
        data_scaled = pd.concat([X_train_scaled, y_train], axis=1)
        target_column = y_train.name

        # 1. Correlation with target variable
        correlations = data_scaled.corr()[target_column].sort_values(ascending=False)

        # Plot correlations with target
        plt.figure(figsize=(15, 6))
        correlations.drop(target_column).plot(kind='bar')
        plt.title(f'Feature Correlations with {target_column} (Scaled Features)')
        plt.tight_layout()
        plt.show()
```



From the visualization we can surely drop the last 3 features : FireplaceQuality, NeighborhoodQuality and HouseStyleQuality

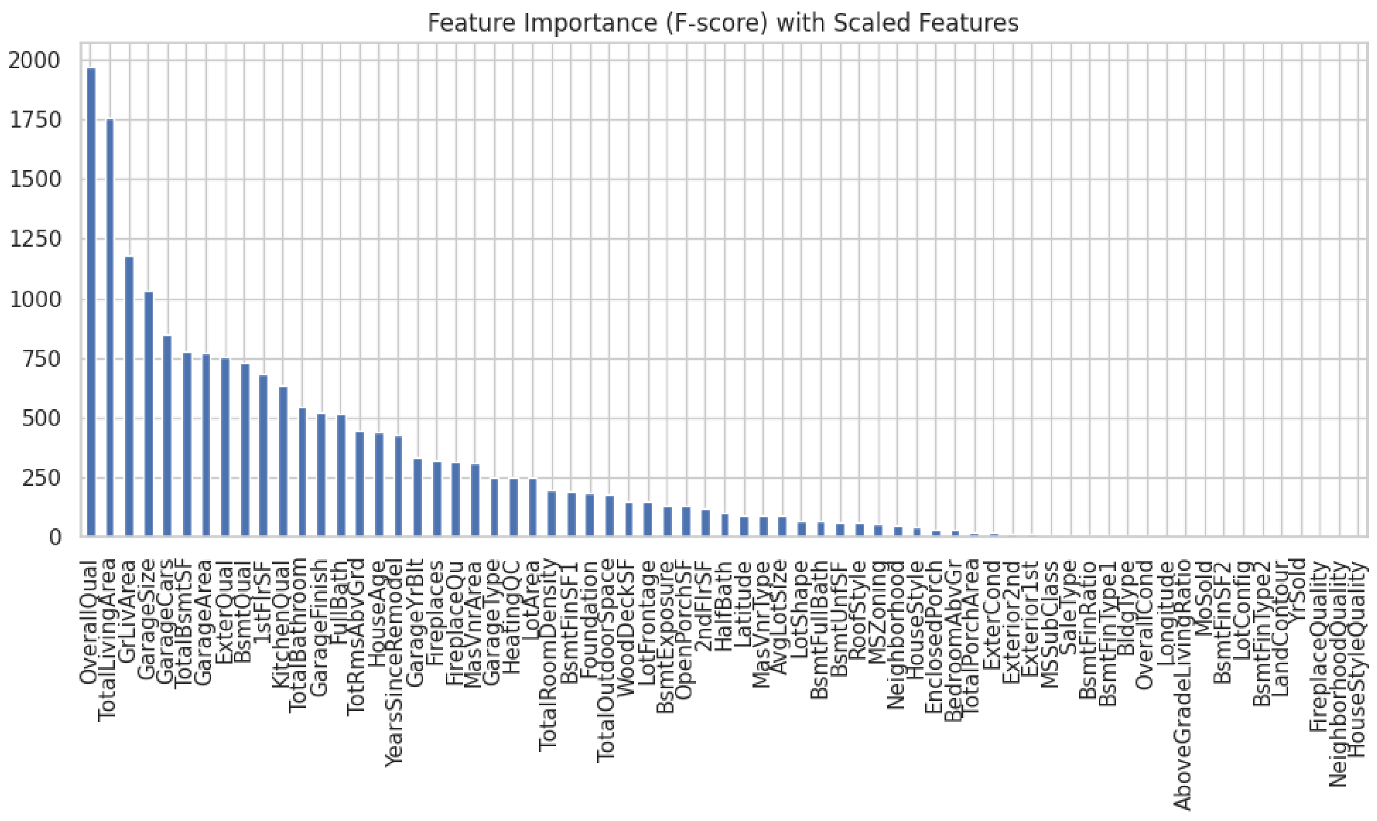
```
In [ ]: # 2. Select top correlated features
        correlation_threshold = 0.1  # Adjust this threshold as needed
        top_correlated = correlations[abs(correlations) > correlation_threshold].drop(target_col
        print("Top correlated features:")
        print(top_correlated)
```

```
Top correlated features:
OverallQual          0.792565
TotalLivingArea      0.775243
GrLivArea            0.709073
GarageSize           0.684887
GarageCars           0.649153
TotalBsmtSF          0.632634
GarageArea           0.631763
1stFlrSF             0.607727
TotalBathroom        0.564589
FullBath             0.555030
TotRmsAbvGrd         0.524959
GarageYrBlt          0.471778
Fireplaces           0.465982
MasVnrArea           0.458867
LotArea              0.418852
BsmtFinSF1           0.376029
Foundation           0.372983
TotalOutdoorSpace    0.362972
WoodDeckSF           0.335360
LotFrontage          0.335059
OpenPorchSF          0.318883
2ndFlrSF             0.307979
HalfBath             0.282335
Latitude             0.269035
AvgLotSize           0.265679
BsmtFullBath         0.228940
BsmtUnfSF            0.227727
RoofStyle            0.222893
Neighborhood         0.205323
HouseStyle           0.185174
BedroomAbvGr         0.155340
TotalPorchArea       0.135059
ExterCond            0.128628
Exterior2nd          0.102605
EnclosedPorch       -0.162007
MSZoning            -0.211929
LotShape            -0.237003
MasVnrType          -0.267733
BsmtExposure        -0.321298
TotalRoomDensity    -0.378298
HeatingQC           -0.419062
GarageType          -0.420556
FireplaceQu         -0.461916
YearsSinceRemodel   -0.518764
HouseAge            -0.524782
GarageFinish        -0.556890
KitchenQual         -0.593395
BsmtQual            -0.620791
ExterQual           -0.627290
Name: SalePrice, dtype: float64
```

F-score for feature importance

```
In [ ]: # 6. F-score for feature importance
        f_scores, _ = f_regression(X_train_scaled, y_train)
        f_scores = pd.Series(f_scores, index=X_train_scaled.columns)
        f_scores = f_scores.sort_values(ascending=False)
```

```
plt.figure(figsize=(10, 6))
f_scores.plot(kind='bar')
plt.title('Feature Importance (F-score) with Scaled Features')
plt.tight_layout()
plt.show()
```



We tried to find out the common features in these two techniques

In [ ]:
```
common_features = set(top_correlated.index).intersection(set(f_scores.index))

# Convert the result to a list if needed
common_features_list = list(common_features)

# Print the common features
print("Common features:", common_features_list)
```

```
Common features: ['BsmtQual', 'BsmtExposure', 'ExterQual', 'FullBath', 'WoodDeckSF', 'To
talBathroom', 'Neighborhood', 'Latitude', 'OverallQual', 'FireplaceQu', 'EnclosedPorch',
'BedroomAbvGr', 'TotalBsmtSF', 'MasVnrArea', 'HeatingQC', 'ExterCond', 'GarageArea', 'Gr
LivArea', '2ndFlrSF', 'GarageCars', 'LotShape', 'BsmtUnfSF', 'Exterior2nd', 'GarageFinis
h', 'RoofStyle', 'TotalPorchArea', 'TotRmsAbvGrd', 'TotalRoomDensity', 'AvgLotSize', 'To
talOutdoorSpace', 'KitchenQual', 'HouseStyle', 'LotArea', '1stFlrSF', 'MasVnrType', 'MSZ
oning', 'HouseAge', 'GarageYrBlt', 'Fireplaces', 'GarageType', 'BsmtFullBath', 'HalfBat
h', 'GarageSize', 'LotFrontage', 'BsmtFinSF1', 'YearsSinceRemodel', 'Foundation', 'Total
LivingArea', 'OpenPorchSF']
```

In [ ]:
```
len(common_features_list)
```

Out[ ]:
```
49
```

Permutation Feature Importance

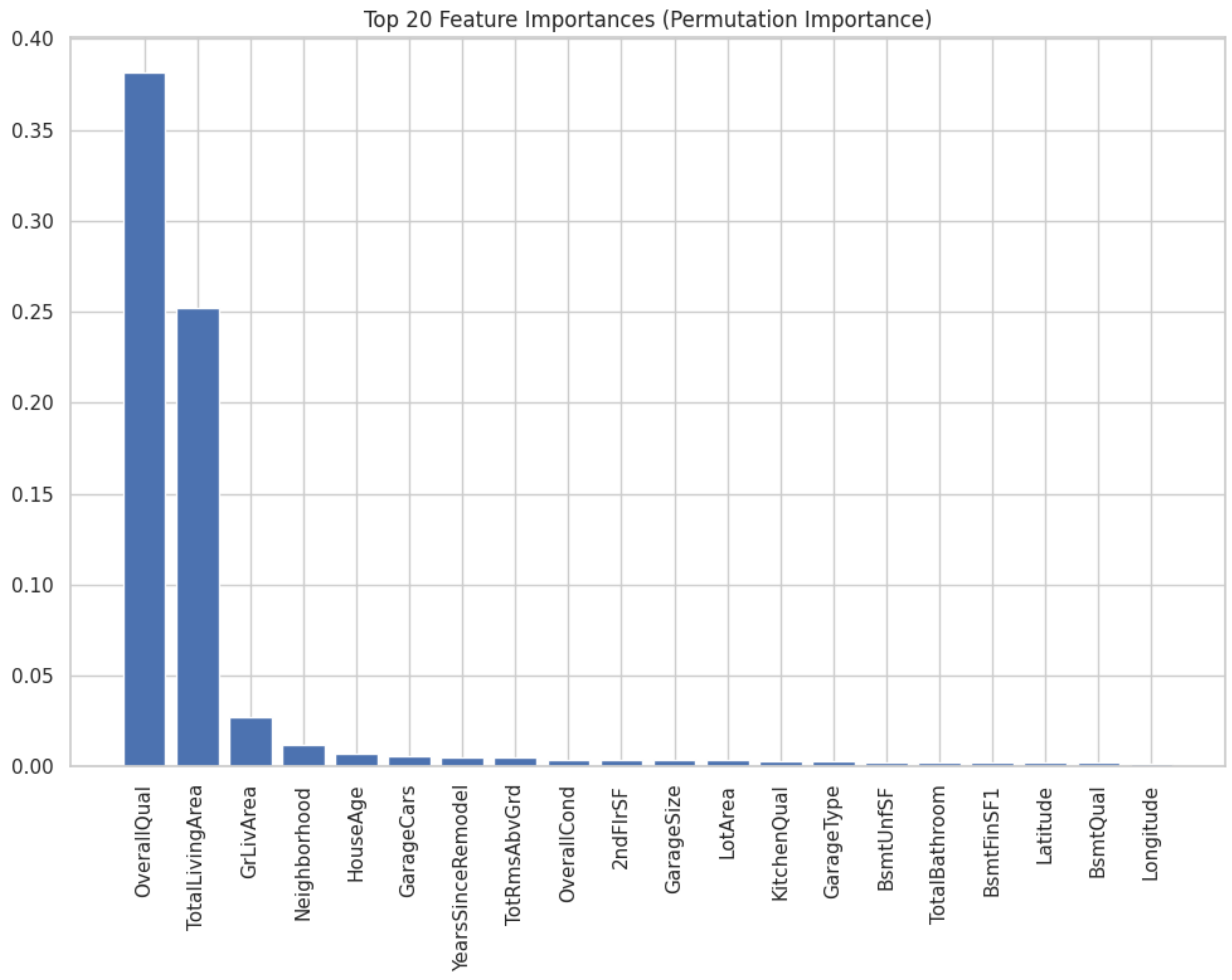
This is another technique that we wanted to try for getting feature importance

In [ ]:
```
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
# Calculate permutation importance
result = permutation_importance(
    model, X_val, y_val, n_repeats=60, random_state=42, n_jobs=-1
)

# Create a dataframe of feature importances
pi_feature_importance = pd.DataFrame({
    'feature': X_val.columns,
    'importance': result.importances_mean,
    'std': result.importances_std
})

# Sort features by importance
pi_feature_importance = pi_feature_importance.sort_values('importance', ascending=False)

# Print the top 20 most important features
print(pi_feature_importance.head(20))
```

```
              feature  importance       std
0         OverallQual    0.381588  0.032121
1     TotalLivingArea    0.252428  0.020324
2           GrLivArea    0.027131  0.003595
3        Neighborhood    0.012397  0.005694
4            HouseAge    0.007122  0.001592
5          GarageCars    0.006112  0.001922
6   YearsSinceRemodel    0.005358  0.001401
7        TotRmsAbvGrd    0.005170  0.001817
8         OverallCond    0.003913  0.000556
9            2ndFlrSF    0.003771  0.000948
10         GarageSize    0.003585  0.000700
11            LotArea    0.003504  0.001093
12        KitchenQual    0.002853  0.001069
13         GarageType    0.002765  0.000926
14          BsmtUnfSF    0.002702  0.000679
15      TotalBathroom    0.002645  0.000667
16         BsmtFinSF1    0.002636  0.000873
17           Latitude    0.002552  0.000888
18           BsmtQual    0.002142  0.000806
19          Longitude    0.001986  0.002927
```

In [ ]:
```
plt.figure(figsize=(10, 8))
plt.bar(pi_feature_importance['feature'][:20], pi_feature_importance['importance'][:20])
plt.xticks(rotation=90)
plt.title('Top 20 Feature Importances (Permutation Importance)')
plt.tight_layout()
plt.show()
```



Modelling

Now we will get into baseline modelling with all the features. For first baseline, we want to try with all the features.

```
In [ ]: # Prepare the data
        X_train_selected = X_train
        X_val_selected = X_val

        # Scale the features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train_selected)
        X_val_scaled = scaler.transform(X_val_selected)
```

Trying with Linear Regression and Decision Tree

```
In [ ]: # Combine train and validation sets for cross-validation
        X_combined = np.vstack((X_train_scaled, X_val_scaled))
        y_combined = np.concatenate((y_train, y_val))

        # Define models
        models = {
            'Linear Regression': LinearRegression(),
            'Decision Tree': DecisionTreeRegressor(random_state=42),
        }
```

```
# Perform cross-validation
n_splits = 5
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
```

In [ ]:
```
for name, model in models.items():
    print(f"\nEvaluating {name}:")

    try:
        rmse_scores = cross_val_score(model, X_combined, y_combined, cv=kf,
                                      scoring='neg_root_mean_squared_error', n_jobs=1)
        r2_scores = cross_val_score(model, X_combined, y_combined, cv=kf,
                                    scoring='r2', n_jobs=1)

        print(f"Cross-validation RMSE: {-rmse_scores.mean():.4f} (+/- {rmse_scores.std()
        print(f"Cross-validation R2 Score: {r2_scores.mean():.4f} (+/- {r2_scores.std()

        # Fit the model on the training data and evaluate on the validation data
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_val_scaled)

        mse = mean_squared_error(y_val, y_pred)
        rmse = np.sqrt(mse)
        r2 = r2_score(y_val, y_pred)

        print(f"Validation RMSE: {rmse:.4f}")
        print(f"Validation R2 Score: {r2:.4f}")
    except Exception as e:
        print(f"An error occurred: {str(e)}")
```

```
Evaluating Linear Regression:
Cross-validation RMSE: 29073.4837 (+/- 6090.6673)
Cross-validation R2 Score: 0.8558 (+/- 0.0456)
Validation RMSE: 27536.1105
Validation R2 Score: 0.8870

Evaluating Decision Tree:
Cross-validation RMSE: 37708.3540 (+/- 7471.5025)
Cross-validation R2 Score: 0.7556 (+/- 0.0922)
Validation RMSE: 37209.4743
Validation R2 Score: 0.7936
```

Trying with Neural Network

In [ ]:
```
# Define the model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(512, activation='relu'),
    Dropout(0.2),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(1)  # Output layer for regression
])

# Compile the model
learning_rate = 0.001
```

```python
optimizer = Adam(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mae'])

# Define callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=Tru
model_checkpoint = ModelCheckpoint('best_model.keras', save_best_only=True, monitor='val
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-6)

# Train the model
history = model.fit(
    X_train_scaled, y_train,
    validation_split=0.2,
    epochs=100,
    batch_size=64,
    callbacks=[early_stopping, model_checkpoint, reduce_lr],
    verbose=1
)
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/core/dense.py:87: UserWarning:
Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential mode
ls, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 33ms/step - loss: 37978447872.0000 - mae: 180904.3906 - val_l
oss: 37736488960.0000 - val_mae: 180851.4062 - learning_rate: 0.0010
Epoch 2/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 39739465728.0000 - mae: 182903.1094 - val_l
oss: 28170782720.0000 - val_mae: 155563.8594 - learning_rate: 0.0010
Epoch 3/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 18484809728.0000 - mae: 114644.6016 - val_l
oss: 2490172416.0000 - val_mae: 40965.0430 - learning_rate: 0.0010
Epoch 4/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 4519881216.0000 - mae: 54314.0938 - val_los
s: 1638055040.0000 - val_mae: 31189.0078 - learning_rate: 0.0010
Epoch 5/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 4092949248.0000 - mae: 43174.1562 - val_los
s: 1180075648.0000 - val_mae: 27339.9180 - learning_rate: 0.0010
Epoch 6/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 2528487424.0000 - mae: 37189.9883 - val_los
s: 1317465600.0000 - val_mae: 28474.7109 - learning_rate: 0.0010
Epoch 7/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 2414402048.0000 - mae: 36788.0742 - val_los
s: 1064576320.0000 - val_mae: 25743.6211 - learning_rate: 0.0010
Epoch 8/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 2360259072.0000 - mae: 35012.4531 - val_los
s: 982693056.0000 - val_mae: 24202.7207 - learning_rate: 0.0010
Epoch 9/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 2005766400.0000 - mae: 32260.5527 - val_los
s: 1318384256.0000 - val_mae: 27427.4785 - learning_rate: 0.0010
Epoch 10/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 2035124480.0000 - mae: 33636.3672 - val_los
s: 1781765376.0000 - val_mae: 32630.8418 - learning_rate: 0.0010
Epoch 11/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 2125516800.0000 - mae: 33380.7422 - val_los
s: 1379940864.0000 - val_mae: 27393.7324 - learning_rate: 0.0010
Epoch 12/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 2087159424.0000 - mae: 32963.0000 - val_los
s: 1256771328.0000 - val_mae: 26521.1992 - learning_rate: 0.0010
Epoch 13/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 1653435008.0000 - mae: 30645.6367 - val_los
s: 841795008.0000 - val_mae: 21937.4238 - learning_rate: 0.0010
Epoch 14/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1668977536.0000 - mae: 31012.4570 - val_los
s: 939712768.0000 - val_mae: 22717.3652 - learning_rate: 0.0010
Epoch 15/100
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1990177280.0000 - mae: 32988.5508 - val_los
s: 1394758528.0000 - val_mae: 28984.1973 - learning_rate: 0.0010
Epoch 16/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 1765204608.0000 - mae: 31785.4766 - val_los
s: 1623113600.0000 - val_mae: 31559.5078 - learning_rate: 0.0010
Epoch 17/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 2157406720.0000 - mae: 33973.2578 - val_los
s: 1228957696.0000 - val_mae: 26850.8574 - learning_rate: 0.0010
Epoch 18/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 1901825536.0000 - mae: 31610.3125 - val_los
s: 930592768.0000 - val_mae: 22863.6621 - learning_rate: 0.0010
Epoch 19/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 1585642496.0000 - mae: 29292.0039 - val_los
s: 749441280.0000 - val_mae: 20608.3828 - learning_rate: 2.0000e-04
Epoch 20/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1565241472.0000 - mae: 29322.1152 - val_los
s: 907262848.0000 - val_mae: 22593.0371 - learning_rate: 2.0000e-04
Epoch 21/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1917894272.0000 - mae: 31786.3438 - val_los
s: 1008547904.0000 - val_mae: 23844.3418 - learning_rate: 2.0000e-04
Epoch 22/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1802250752.0000 - mae: 31210.9336 - val_los
s: 839530432.0000 - val_mae: 21534.5078 - learning_rate: 2.0000e-04
Epoch 23/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1495472512.0000 - mae: 28265.4219 - val_los
s: 814797888.0000 - val_mae: 21228.9043 - learning_rate: 2.0000e-04
Epoch 24/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1600348288.0000 - mae: 29881.9551 - val_los
s: 903371136.0000 - val_mae: 22387.4043 - learning_rate: 2.0000e-04
Epoch 25/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1666441728.0000 - mae: 30478.1680 - val_los
s: 885007488.0000 - val_mae: 22155.9219 - learning_rate: 4.0000e-05
Epoch 26/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1770089984.0000 - mae: 30887.6777 - val_los
s: 861895296.0000 - val_mae: 21843.0410 - learning_rate: 4.0000e-05
Epoch 27/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 1661817344.0000 - mae: 30357.2520 - val_los
s: 860288576.0000 - val_mae: 21832.0547 - learning_rate: 4.0000e-05
Epoch 28/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1873338240.0000 - mae: 30870.2461 - val_los
s: 857580800.0000 - val_mae: 21793.0977 - learning_rate: 4.0000e-05
Epoch 29/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1564590976.0000 - mae: 29714.9414 - val_los
s: 845611648.0000 - val_mae: 21627.7637 - learning_rate: 4.0000e-05
Epoch 30/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1775487488.0000 - mae: 29858.5488 - val_los
s: 837113536.0000 - val_mae: 21524.1973 - learning_rate: 8.0000e-06
Epoch 31/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1563590144.0000 - mae: 29303.0039 - val_los
s: 835358976.0000 - val_mae: 21508.2461 - learning_rate: 8.0000e-06
Epoch 32/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1727647104.0000 - mae: 29391.4414 - val_los
s: 840858752.0000 - val_mae: 21573.2637 - learning_rate: 8.0000e-06
Epoch 33/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1593611648.0000 - mae: 29745.9492 - val_los
s: 847512256.0000 - val_mae: 21657.9570 - learning_rate: 8.0000e-06
Epoch 34/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1586875648.0000 - mae: 29773.2891 - val_los
s: 844943744.0000 - val_mae: 21621.9688 - learning_rate: 8.0000e-06
Epoch 35/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1425205888.0000 - mae: 27743.4883 - val_los
s: 845714048.0000 - val_mae: 21631.5762 - learning_rate: 1.6000e-06
Epoch 36/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1527588608.0000 - mae: 29028.4062 - val_los
s: 845921024.0000 - val_mae: 21633.8047 - learning_rate: 1.6000e-06
Epoch 37/100
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1966378368.0000 - mae: 29585.1699 - val_loss: 845904640.0000 - val_mae: 21632.7148 - learning_rate: 1.6000e-06
Epoch 38/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1643140480.0000 - mae: 30139.7734 - val_loss: 844918848.0000 - val_mae: 21619.8223 - learning_rate: 1.6000e-06
Epoch 39/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1582994560.0000 - mae: 29511.3145 - val_loss: 843883968.0000 - val_mae: 21607.1230 - learning_rate: 1.6000e-06
```

In [ ]:
```python
# Evaluate the model
test_loss, test_mae = model.evaluate(X_val_scaled, y_val, verbose=0)
print(f"Test Loss: {test_loss:.4f}")
print(f"Test MAE: {test_mae:.4f}")

# Make predictions
predictions = model.predict(X_val_scaled)
```

```
Test Loss: 993581504.0000
Test MAE: 22289.6504
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
```

The baseline models are giving poor results. We will now try with only those features which have high positive and negative correlations

Selected Features For Modelling

Highly correlated features only

In [ ]:
```python
top_correlated
```

Out[ ]:
```
OverallQual         0.792565
TotalLivingArea     0.775243
GrLivArea           0.709073
GarageSize          0.684887
GarageCars          0.649153
TotalBsmtSF         0.632634
GarageArea          0.631763
1stFlrSF            0.607727
TotalBathroom       0.564589
FullBath            0.555030
TotRmsAbvGrd        0.524959
GarageYrBlt         0.471778
Fireplaces          0.465982
MasVnrArea          0.458867
LotArea             0.418852
BsmtFinSF1          0.376029
Foundation          0.372983
TotalOutdoorSpace   0.362972
WoodDeckSF          0.335360
LotFrontage         0.335059
OpenPorchSF         0.318883
2ndFlrSF            0.307979
HalfBath            0.282335
Latitude            0.269035
AvgLotSize          0.265679
BsmtFullBath        0.228940
BsmtUnfSF           0.227727
RoofStyle           0.222893
Neighborhood        0.205323
HouseStyle          0.185174
BedroomAbvGr        0.155340
TotalPorchArea      0.135059
ExterCond           0.128628
```

```
Exterior2nd          0.102605
EnclosedPorch       -0.162007
MSZoning            -0.211929
LotShape            -0.237003
MasVnrType          -0.267733
BsmtExposure        -0.321298
TotalRoomDensity    -0.378298
HeatingQC           -0.419062
GarageType          -0.420556
FireplaceQu         -0.461916
YearsSinceRemodel   -0.518764
HouseAge            -0.524782
GarageFinish        -0.556890
KitchenQual         -0.593395
BsmtQual            -0.620791
ExterQual           -0.627290
Name: SalePrice, dtype: float64
```

In [ ]:
```python
top_cor_features = top_correlated[(top_correlated > 0.4) | (top_correlated < -0.4)]

# Convert the result to a list of feature names
top_cor_features_list = top_cor_features.index.tolist()
```

In [ ]:
```python
top_cor_features_list
```

Out[ ]:
```
['OverallQual',
 'TotalLivingArea',
 'GrLivArea',
 'GarageSize',
 'GarageCars',
 'TotalBsmtSF',
 'GarageArea',
 '1stFlrSF',
 'TotalBathroom',
 'FullBath',
 'TotRmsAbvGrd',
 'GarageYrBlt',
 'Fireplaces',
 'MasVnrArea',
 'LotArea',
 'HeatingQC',
 'GarageType',
 'FireplaceQu',
 'YearsSinceRemodel',
 'HouseAge',
 'GarageFinish',
 'KitchenQual',
 'BsmtQual',
 'ExterQual']
```

Transformation of selected features

Now we will try our baseline models again with these features. But before that we will handle any skewness and transform skewed features to log normal features

In [ ]:
```python
def select_features(X_train, X_val, selected_features_list):
    X_train_selected = X_train[selected_features_list]
    X_val_selected = X_val[selected_features_list]
    return X_train_selected, X_val_selected

def log_transform_skewed_features(X_train, X_val, threshold=0.5):
    numeric_feats = X_train.select_dtypes(include=['float64', 'int64']).columns
    skewed_feats = X_train[numeric_feats].apply(lambda x: skew(x.dropna()))
    skewed_feats = skewed_feats[abs(skewed_feats) > threshold]
    skewed_features = skewed_feats.index

    for feat in skewed_features:
        X_train[feat] = np.log1p(X_train[feat])
        X_val[feat] = np.log1p(X_val[feat])

    return X_train, X_val, list(skewed_features)

def handle_infinite_values(X_train, X_val):
    # Replace inf with NaN
    X_train = X_train.replace([np.inf, -np.inf], np.nan)
    X_val = X_val.replace([np.inf, -np.inf], np.nan)

    # Fill NaN with the mean of the column
    X_train = X_train.fillna(X_train.mean())
    X_val = X_val.fillna(X_train.mean())  # Use train mean for validation set

    return X_train, X_val

def scale_features(X_train, X_val):
    scaler = StandardScaler()
    X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns
    X_val_scaled = pd.DataFrame(scaler.transform(X_val), columns=X_val.columns, index=X_
    return X_train_scaled, X_val_scaled, scaler

def preprocess_data(X_train, X_val, selected_features_list, skew_threshold=0.5):
    # Select features
    X_train_selected, X_val_selected = select_features(X_train, X_val, selected_features

    # Log transform skewed features
    X_train_transformed, X_val_transformed, skewed_features = log_transform_skewed_featu

    # Handle infinite values
    X_train_cleaned, X_val_cleaned = handle_infinite_values(X_train_transformed, X_val_t

    # Scale features
    X_train_scaled, X_val_scaled, scaler = scale_features(X_train_cleaned, X_val_cleaned

    return X_train_scaled, X_val_scaled, skewed_features, scaler
```

In [ ]:
```
X_train_processed, X_val_processed, skewed_features, scaler = preprocess_data(X_train, X
```

```
/tmp/ipykernel_33/2513444776.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  X_train[feat] = np.log1p(X_train[feat])
/tmp/ipykernel_33/2513444776.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  X_val[feat] = np.log1p(X_val[feat])
/opt/conda/lib/python3.10/site-packages/pandas/core/arraylike.py:399: RuntimeWarning: di
vide by zero encountered in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Baseline Modelling (Second Try)

```
In [ ]: def try_different_models(X_train, X_val, y_train, y_val):
            models = {
                "Linear Regression": LinearRegression(),
                "Ridge Regression": Ridge(),
                "Lasso Regression": Lasso(),
                "ElasticNet": ElasticNet(),
                "Decision Tree": DecisionTreeRegressor(),
                "Random Forest": RandomForestRegressor(),
                "Gradient Boosting": GradientBoostingRegressor(),
                "SVR": SVR(),
                "KNN": KNeighborsRegressor(),
                "XGBoost": XGBRegressor(eval_metric='rmse'),
                "LightGBM": LGBMRegressor()
            }

            results = []
            trained_models = {}

            for name, model in models.items():
                start_time = time.time()

                # Train the model
                model.fit(X_train, y_train)

                # Store the trained model
                trained_models[name] = model

                # Make predictions
                train_predictions = model.predict(X_train)
                val_predictions = model.predict(X_val)

                # Calculate metrics
                train_mse = mean_squared_error(y_train, train_predictions)
                train_rmse = np.sqrt(train_mse)
                train_r2 = r2_score(y_train, train_predictions)

                val_mse = mean_squared_error(y_val, val_predictions)
                val_rmse = np.sqrt(val_mse)
                val_r2 = r2_score(y_val, val_predictions)

                end_time = time.time()
                training_time = end_time - start_time

                # Store results
                results.append({
                    "Model": name,
                    "Train RMSE": train_rmse,
                    "Train R2": train_r2,
                    "Validation RMSE": val_rmse,
                    "Validation R2": val_r2,
                    "Training Time": training_time
                })

            # Convert results to DataFrame
            results_df = pd.DataFrame(results)
            results_df = results_df.sort_values("Validation RMSE")

            return results_df, trained_models
```

```
In [ ]: # Usage
        results, trained_models = try_different_models(X_train_processed, X_val_processed, y_tra

        # Display results
        print(results)
```

```
# Access the best model (lowest Validation RMSE)
best_model_name = results.iloc[0]["Model"]
best_model = trained_models[best_model_name]
print(f"\nBest Model: {best_model_name}")
print(f"Best Model Validation RMSE: {results.iloc[0]['Validation RMSE']}")
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.
003516 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2280
[LightGBM] [Info] Number of data points in the train set: 1168, number of used features:
24
[LightGBM] [Info] Start training from score 181125.193065
                Model    Train RMSE  Train R2  Validation RMSE  Validation R2  \
6   Gradient Boosting  16802.865097  0.950413     25369.151208       0.904048
5       Random Forest  10571.858624  0.980371     25822.069038       0.900591
10           LightGBM  12208.428742  0.973823     27086.585549       0.890617
9             XGBoost   1462.162601  0.999625     27739.142647       0.885283
0   Linear Regression  31209.603918  0.828928     31251.098562       0.854396
2    Lasso Regression  31209.611482  0.828928     31251.423325       0.854393
1    Ridge Regression  31211.332498  0.828909     31253.878490       0.854370
3          ElasticNet  32651.597633  0.812755     32930.519558       0.838326
8                 KNN  27640.195598  0.865821     32989.866727       0.837743
4       Decision Tree    171.728222  0.999995     35789.649296       0.809033
7                 SVR  77101.362465 -0.044061     82823.164384      -0.022695

    Training Time
6        0.451351
5        1.206824
10       0.263505
9        0.538199
0        0.027055
2        0.079475
1        0.023777
3        0.026510
8        0.221559
4        0.066774
7        0.210081

Best Model: Gradient Boosting
Best Model Validation RMSE: 25369.15120774649
```

Hyperparameter Tuning

We will tune each model seperately to save compute resource. We are using GridSearch CV for tuning

Deep Learning Model

We are training our 8 layer deep learning model with selected features.

In [ ]:
```
def build_and_train_deep_model(X_train, X_val, y_train, y_val,
                               learning_rate=0.001,
                               batch_size=32,
                               epochs=100,
                               dropout_rate=0.2):
    # Define the model
    model = Sequential([
        Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
        Dropout(dropout_rate),
        Dense(128, activation='relu'),
        Dropout(dropout_rate),
        Dense(256, activation='relu'),
        Dropout(dropout_rate),
        Dense(512, activation='relu'),
        Dropout(dropout_rate),
        Dense(256, activation='relu'),
        Dropout(dropout_rate),
        Dense(128, activation='relu'),
        Dropout(dropout_rate),
        Dense(64, activation='relu'),
        Dropout(dropout_rate),
        Dense(32, activation='relu'),
        Dense(1)  # Output layer
    ])

    # Compile the model
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mean_squared_error')

    # Define callbacks
    early_stopping = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-

    # Train the model
    start_time = time.time()
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=epochs,
        batch_size=batch_size,
        callbacks=[early_stopping, reduce_lr],
        verbose=1
    )
    training_time = time.time() - start_time

    # Make predictions
    train_predictions = model.predict(X_train).flatten()
    val_predictions = model.predict(X_val).flatten()

    # Calculate metrics
    train_mse = mean_squared_error(y_train, train_predictions)
    train_rmse = np.sqrt(train_mse)
    train_r2 = r2_score(y_train, train_predictions)
    val_mse = mean_squared_error(y_val, val_predictions)
    val_rmse = np.sqrt(val_mse)
    val_r2 = r2_score(y_val, val_predictions)

    # Prepare results
    results = {
        "Model": "Deep Learning (TensorFlow)",
        "Train RMSE": train_rmse,
        "Train R2": train_r2,
        "Validation RMSE": val_rmse,
        "Validation R2": val_r2,
        "Training Time": training_time
    }

    return model, results, history
```

Trying with 500 epochs.

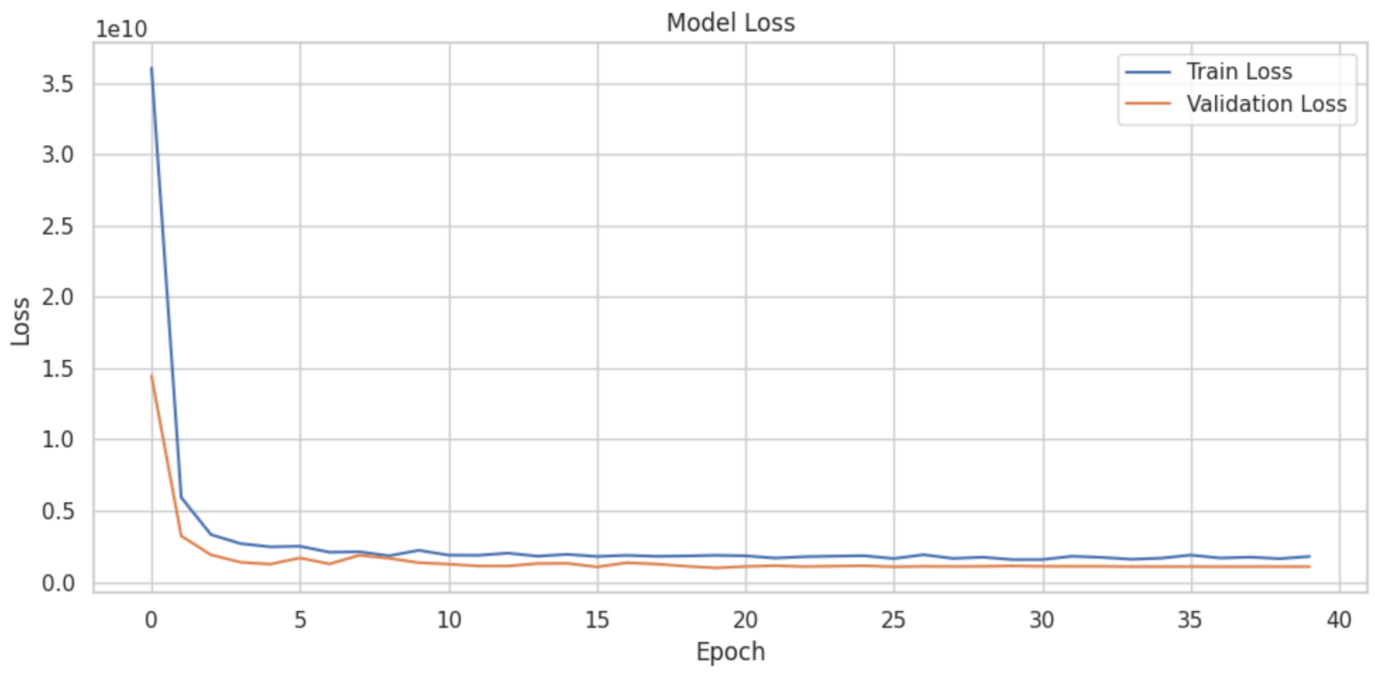
In [ ]:
```
deep_model, deep_results, history = build_and_train_deep_model(
    X_train_processed, X_val_processed, y_train, y_val,
    learning_rate=0.001,
    batch_size=32,
    epochs=500,
    dropout_rate=0.2
)

# Print results
print(deep_results)
```

```
Epoch 1/500
/opt/conda/lib/python3.10/site-packages/keras/src/layers/core/dense.py:87: UserWarning:
Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential mode
ls, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
37/37 □□□□□□□□□□□□□□□□□□□□ 4s 13ms/step - loss: 37493030912.0000 - val_loss: 14459078656.00
00 - learning_rate: 0.0010
Epoch 2/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 7ms/step - loss: 7597249536.0000 - val_loss: 3241426944.0000
 - learning_rate: 0.0010
Epoch 3/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 7ms/step - loss: 3430820608.0000 - val_loss: 1914221696.0000
 - learning_rate: 0.0010
Epoch 4/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 8ms/step - loss: 3170275328.0000 - val_loss: 1391904256.0000
 - learning_rate: 0.0010
Epoch 5/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 8ms/step - loss: 2492081920.0000 - val_loss: 1258509440.0000
 - learning_rate: 0.0010
Epoch 6/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 8ms/step - loss: 2458817536.0000 - val_loss: 1696397696.0000
 - learning_rate: 0.0010
Epoch 7/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 7ms/step - loss: 2214904832.0000 - val_loss: 1275124736.0000
 - learning_rate: 0.0010
Epoch 8/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 7ms/step - loss: 1868541056.0000 - val_loss: 1885150464.0000
 - learning_rate: 0.0010
Epoch 9/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 7ms/step - loss: 1790544384.0000 - val_loss: 1671299584.0000
 - learning_rate: 0.0010
Epoch 10/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 8ms/step - loss: 2559358464.0000 - val_loss: 1361264896.0000
 - learning_rate: 0.0010
Epoch 11/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 7ms/step - loss: 1715493504.0000 - val_loss: 1264197760.0000
 - learning_rate: 2.0000e-04
Epoch 12/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 8ms/step - loss: 1883373952.0000 - val_loss: 1129663872.0000
 - learning_rate: 2.0000e-04
Epoch 13/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 8ms/step - loss: 1828500736.0000 - val_loss: 1127907840.0000
 - learning_rate: 2.0000e-04
Epoch 14/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 7ms/step - loss: 2122530176.0000 - val_loss: 1303069696.0000
 - learning_rate: 2.0000e-04
Epoch 15/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 7ms/step - loss: 2056109312.0000 - val_loss: 1316326400.0000
 - learning_rate: 2.0000e-04
Epoch 16/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 7ms/step - loss: 1633779072.0000 - val_loss: 1063158976.0000
 - learning_rate: 2.0000e-04
Epoch 17/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 7ms/step - loss: 1761455616.0000 - val_loss: 1358696448.0000
 - learning_rate: 2.0000e-04
Epoch 18/500
37/37 □□□□□□□□□□□□□□□□□□□□ 0s 7ms/step - loss: 1891582592.0000 - val_loss: 1266745472.0000
 - learning_rate: 2.0000e-04
Epoch 19/500
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1830538496.0000 - val_loss: 1115181184.0000
 - learning_rate: 2.0000e-04
Epoch 20/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1825961856.0000 - val_loss: 994292544.0000 -
 learning_rate: 2.0000e-04
Epoch 21/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1774077056.0000 - val_loss: 1094765184.0000
 - learning_rate: 2.0000e-04
Epoch 22/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1643586176.0000 - val_loss: 1152294144.0000
 - learning_rate: 2.0000e-04
Epoch 23/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1810560512.0000 - val_loss: 1089627904.0000
 - learning_rate: 2.0000e-04
Epoch 24/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1806588160.0000 - val_loss: 1118782976.0000
 - learning_rate: 2.0000e-04
Epoch 25/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 2124066048.0000 - val_loss: 1145840128.0000
 - learning_rate: 2.0000e-04
Epoch 26/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1613044352.0000 - val_loss: 1075996032.0000
 - learning_rate: 4.0000e-05
Epoch 27/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1899206912.0000 - val_loss: 1104493952.0000
 - learning_rate: 4.0000e-05
Epoch 28/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1604207104.0000 - val_loss: 1096471424.0000
 - learning_rate: 4.0000e-05
Epoch 29/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1886380928.0000 - val_loss: 1105565824.0000
 - learning_rate: 4.0000e-05
Epoch 30/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1565191808.0000 - val_loss: 1134333824.0000
 - learning_rate: 4.0000e-05
Epoch 31/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1533719040.0000 - val_loss: 1110834944.0000
 - learning_rate: 8.0000e-06
Epoch 32/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1742881664.0000 - val_loss: 1104959360.0000
 - learning_rate: 8.0000e-06
Epoch 33/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 1618535680.0000 - val_loss: 1104407552.0000
 - learning_rate: 8.0000e-06
Epoch 34/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1476250880.0000 - val_loss: 1087660288.0000
 - learning_rate: 8.0000e-06
Epoch 35/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1713044352.0000 - val_loss: 1086939648.0000
 - learning_rate: 8.0000e-06
Epoch 36/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1825325312.0000 - val_loss: 1087803264.0000
 - learning_rate: 1.6000e-06
Epoch 37/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1536485248.0000 - val_loss: 1084937984.0000
 - learning_rate: 1.6000e-06
Epoch 38/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 1717892096.0000 - val_loss: 1086539392.0000
 - learning_rate: 1.6000e-06
Epoch 39/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1560178688.0000 - val_loss: 1087425024.0000
 - learning_rate: 1.6000e-06
Epoch 40/500
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1971701632.0000 - val_loss: 1087991936.0000
 - learning_rate: 1.6000e-06
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
{'Model': 'Deep Learning (TensorFlow)', 'Train RMSE': 30226.444687301828, 'Train R2': 0.
8395365782872805, 'Validation RMSE': 31532.403532728535, 'Validation R2': 0.851762974704
8806, 'Training Time': 15.300462007522583}
```

In [ ]:
```
# Plot training history
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()
```



We will take the best model that we got from Machine Learning models and The Deep Learning model for tuning

In [ ]:
```
# tuning gradient boosting

def tune_gradient_boosting(X_train, y_train):
    param_grid = {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 4, 5],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
    gb = GradientBoostingRegressor(random_state=42)
    grid_search = GridSearchCV(gb, param_grid, cv=5, scoring='neg_mean_squared_error', n
    grid_search.fit(X_train, y_train)
    return grid_search.best_params_
```

In [ ]:
```
best_gb_params = tune_gradient_boosting(X_train_processed, y_train)
print("Best Gradient Boosting parameters:", best_gb_params)
```

```
Best Gradient Boosting parameters: {'learning_rate': 0.1, 'max_depth': 4, 'min_samples_l
eaf': 1, 'min_samples_split': 10, 'n_estimators': 200}
```

Now we will do hyperparameter tuning on the second best model : Random Forest

In [ ]:
```
# tuning random forest

def tune_random_forest(X_train, y_train):
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'max_features': [1.0, 'sqrt', 'log2']  # Changed 'auto' to 1.0
    }

    rf = RandomForestRegressor(random_state=42)

    grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='neg_mean_squared_error', n
    grid_search.fit(X_train, y_train)

    return grid_search.best_params_
```

In [ ]:
```
# Use the function
best_rf_params = tune_random_forest(X_train_processed, y_train)

print("Best Random Forest parameters:", best_rf_params)
```

```
Best Random Forest parameters: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_
leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

Building Gradient Boosting Model based on the best hyperparameters

In [ ]:
```
def build_and_train_gradient_boosting(X_train, X_val, y_train, y_val):
    # Create the model with the best parameters
    best_params = {
        'learning_rate': 0.1,
        'max_depth': 4,
        'min_samples_leaf': 1,
        'min_samples_split': 10,
        'n_estimators': 200
    }

    gb_model = GradientBoostingRegressor(**best_params, random_state=42)

    # Train the model
    start_time = time.time()
    gb_model.fit(X_train, y_train)
    training_time = time.time() - start_time

    # Make predictions
    train_predictions = gb_model.predict(X_train)
    val_predictions = gb_model.predict(X_val)

    # Calculate metrics
    train_mse = mean_squared_error(y_train, train_predictions)
    train_rmse = np.sqrt(train_mse)
    train_r2 = r2_score(y_train, train_predictions)

    val_mse = mean_squared_error(y_val, val_predictions)
    val_rmse = np.sqrt(val_mse)
    val_r2 = r2_score(y_val, val_predictions)

    # Prepare results
    results = {
        "Model": "Gradient Boosting (Best Parameters)",
        "Train RMSE": train_rmse,
        "Train R2": train_r2,
        "Validation RMSE": val_rmse,
        "Validation R2": val_r2,
        "Training Time": training_time
    }

    return gb_model, results
```

In [ ]:
```
# Use the function
gb_model, gb_results = build_and_train_gradient_boosting(X_train_processed, X_val_proces

# Print results
print(gb_results)
```

```
{'Model': 'Gradient Boosting (Best Parameters)', 'Train RMSE': 9635.141982088855, 'Train
R2': 0.9836950995468902, 'Validation RMSE': 25395.863406245982, 'Validation R2': 0.90384
5745552171, 'Training Time': 1.111978530883789}
```

Building the Random Forest Model with Best parameters

We will build the Random Forest model with best parameters.

In [ ]:
```
best_rf_model = RandomForestRegressor(
    max_depth=None,
    max_features='sqrt',
    min_samples_leaf=1,
    min_samples_split=2,
    n_estimators=200,
    random_state=42
)

# Fit the model on the training data
best_rf_model.fit(X_train_processed, y_train)

# Make predictions on the training and validation sets
train_predictions = best_rf_model.predict(X_train_processed)
val_predictions = best_rf_model.predict(X_val_processed)

# Calculate metrics
train_mse = mean_squared_error(y_train, train_predictions)
train_rmse = np.sqrt(train_mse)
train_r2 = r2_score(y_train, train_predictions)

val_mse = mean_squared_error(y_val, val_predictions)
val_rmse = np.sqrt(val_mse)
val_r2 = r2_score(y_val, val_predictions)

# Print the results
print("Random Forest Model Performance:")
print(f"Training MSE: {train_mse:.4f}")
print(f"Training RMSE: {train_rmse:.4f}")
print(f"Training R-squared: {train_r2:.4f}")
print(f"\nValidation MSE: {val_mse:.4f}")
print(f"Validation RMSE: {val_rmse:.4f}")
print(f"Validation R-squared: {val_r2:.4f}")
```

```
Random Forest Model Performance:
Training MSE: 109816240.0885
Training RMSE: 10479.3244
Training R-squared: 0.9807

Validation MSE: 661492876.6941
Validation RMSE: 25719.5038
Validation R-squared: 0.9014
```

Since the validation R2 for Gradient Boosting model was higher than the Deep Learning model and Random Forest model, we wanted to use the best performing model. We have saved the model for using it. For the application, we will create seperate python files which will use the models that we have exported from this notebook.

Saving the Gradient Boosting Model with Pickle

As our best model is the Gradient Boosting model, we will use it. We will use Pickle to save the model.

In [ ]:
```
# Save the model
with open('gradient_boosting_model.pkl', 'wb') as f:
    pickle.dump(gb_model, f)

# Save the scaler
with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)

# Save the list of selected features
with open('selected_features.pkl', 'wb') as f:
    pickle.dump(top_cor_features_list, f)

# Save the list of skewed features
with open('skewed_features.pkl', 'wb') as f:
    pickle.dump(skewed_features, f)
```

Showing the model results after hyperparameter tuning

In [ ]:
```
def print_model_results_table(results_list):
    # Prepare the data for the table
    table_data = []
    headers = ["Model", "Train RMSE", "Train R2", "Validation RMSE", "Validation R2", "T

    for result in results_list:
        row = [
            result["Model"],
            f"{result['Train RMSE']:.4f}",
            f"{result['Train R2']:.4f}",
            f"{result['Validation RMSE']:.4f}",
            f"{result['Validation R2']:.4f}",
            f"{result['Training Time']:.2f}s"
        ]
        table_data.append(row)

    # Print the table
    print(tabulate(table_data, headers=headers, tablefmt="grid"))

# Collect all model results
all_results = [
    deep_results,
    gb_results,
    {
        "Model": "Random Forest",
        "Train RMSE": train_rmse,
        "Train R2": train_r2,
        "Validation RMSE": val_rmse,
        "Validation R2": val_r2,
        "Training Time": 0  # You didn't measure training time for RF, so we'll set it t
    }
]
```

```
# Print the table
print_model_results_table(all_results)
```

```
+--------------------------------------+--------------+------------+-------------------+-
----------------+-----------------+
| Model                                |   Train RMSE |   Train R2 |   Validation RMSE |
  Validation R2 | Training Time   |
+======================================+==============+============+===================+=
================+=================+
| Deep Learning (TensorFlow)           |     30226.4  |     0.8395 |           31532.4 |
         0.8518 | 15.30s          |
+--------------------------------------+--------------+------------+-------------------+-
----------------+-----------------+
| Gradient Boosting (Best Parameters) |      9635.14 |     0.9837 |           25395.9 |
         0.9038 | 1.11s           |
+--------------------------------------+--------------+------------+-------------------+-
----------------+-----------------+
| Random Forest                        |     10479.3  |     0.9807 |           25719.5 |
         0.9014 | 0.00s           |
+--------------------------------------+--------------+------------+-------------------+-
----------------+-----------------+
```

Flask Deployment With Streamlit

Now we will use Flask to deploy our model in a server. We will also create an user interface with Streamlit. The Streamlit app will hit the Flask server, the flask server will user the model and return the output to the Streamlit app which will show the results. We have run the codes in our local terminal. For reference, we are including our codes here.

In [ ]:
```
# flask app.py

from flask import Flask, request, jsonify
import pandas as pd
import pickle
import numpy as np

app = Flask(__name__)

# Load the model and other necessary components
with open('gradient_boosting_model.pkl', 'rb') as f:
    model = pickle.load(f)
with open('scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)
with open('selected_features.pkl', 'rb') as f:
    selected_features = pickle.load(f)
with open('skewed_features.pkl', 'rb') as f:
    skewed_features = pickle.load(f)

def preprocess_input(input_data):
    # Select features
    input_selected = input_data[selected_features]

    # Log transform skewed features
    for feat in skewed_features:
        input_selected[feat] = np.log1p(input_selected[feat])

    # Handle infinite values
    input_selected = input_selected.replace([np.inf, -np.inf], np.nan)
    input_selected = input_selected.fillna(input_selected.mean())

    # Scale features
    input_scaled = pd.DataFrame(scaler.transform(input_selected),
                                    columns=input_selected.columns,
                                    index=input_selected.index)

    return input_scaled

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json
    input_df = pd.DataFrame(data, index=[0])
    processed_input = preprocess_input(input_df)
    prediction = model.predict(processed_input)[0]
    return jsonify({'prediction': prediction})

if __name__ == '__main__':
    app.run(debug=True)
```

In [ ]:

```
# streamlit_app.py

import streamlit as st
import pandas as pd
import requests

# Define the encoding dictionaries
encodings = {
    'ExterQual': {'Ex': 0, 'Fa': 1, 'Gd': 2, 'TA': 3},
    'HeatingQC': {'Ex': 0, 'Fa': 1, 'Gd': 2, 'Po': 3, 'TA': 4},
    'GarageType': {'2Types': 0, 'Attchd': 1, 'Basment': 2, 'BuiltIn': 3, 'CarPort': 4, '
    'FireplaceQu': {'Ex': 0, 'Fa': 1, 'Gd': 2, 'Po': 3, 'TA': 4},
    'GarageFinish': {'Fin': 0, 'RFn': 1, 'Unf': 2},
    'KitchenQual': {'Ex': 0, 'Fa': 1, 'Gd': 2, 'TA': 3},
    'BsmtQual': {'Ex': 0, 'Fa': 1, 'Gd': 2, 'TA': 3}
}

st.title('House Price Predictor in Iowa')

# Create input fields for each feature
overall_qual = st.slider('Overall Quality', 1, 10, 5)
total_living_area = st.number_input('Total Living Area (sq ft)', min_value=0)
gr_liv_area = st.number_input('Above Ground Living Area (sq ft)', min_value=0)
garage_size = st.number_input('Garage Size (cars)', min_value=0)
garage_cars = st.number_input('Garage Cars', min_value=0)
total_bsmt_sf = st.number_input('Total Basement Area (sq ft)', min_value=0)
garage_area = st.number_input('Garage Area (sq ft)', min_value=0)
first_flr_sf = st.number_input('First Floor Area (sq ft)', min_value=0)
total_bathroom = st.number_input('Total Bathrooms', min_value=0)
full_bath = st.number_input('Full Bathrooms', min_value=0)
tot_rms_abv_grd = st.number_input('Total Rooms Above Ground', min_value=0)
garage_yr_blt = st.number_input('Garage Year Built', min_value=1900, max_value=2023)
fireplaces = st.number_input('Number of Fireplaces', min_value=0)
mas_vnr_area = st.number_input('Masonry Veneer Area (sq ft)', min_value=0)
lot_area = st.number_input('Lot Area (sq ft)', min_value=0)
heating_qc = st.selectbox('Heating Quality', list(encodings['HeatingQC'].keys()))
garage_type = st.selectbox('Garage Type', list(encodings['GarageType'].keys()))
fireplace_qu = st.selectbox('Fireplace Quality', list(encodings['FireplaceQu'].keys()))
years_since_remodel = st.number_input('Years Since Remodel', min_value=0)
house_age = st.number_input('House Age (years)', min_value=0)
garage_finish = st.selectbox('Garage Finish', list(encodings['GarageFinish'].keys()))
kitchen_qual = st.selectbox('Kitchen Quality', list(encodings['KitchenQual'].keys()))
bsmt_qual = st.selectbox('Basement Quality', list(encodings['BsmtQual'].keys()))
exter_qual = st.selectbox('Exterior Quality', list(encodings['ExterQual'].keys()))

if st.button('Predict Price'):
    # Prepare the input data
    input_data = {
        'OverallQual': overall_qual,
        'TotalLivingArea': total_living_area,
        'GrLivArea': gr_liv_area,
        'GarageSize': garage_size,
        'GarageCars': garage_cars,
        'TotalBsmtSF': total_bsmt_sf,
        'GarageArea': garage_area,
        '1stFlrSF': first_flr_sf,
        'TotalBathroom': total_bathroom,
        'FullBath': full_bath,
        'TotRmsAbvGrd': tot_rms_abv_grd,
        'GarageYrBlt': garage_yr_blt,
        'Fireplaces': fireplaces,
        'MasVnrArea': mas_vnr_area,
        'LotArea': lot_area,
        'HeatingQC': encodings['HeatingQC'][heating_qc],
        'GarageType': encodings['GarageType'][garage_type],
        'FireplaceQu': encodings['FireplaceQu'][fireplace_qu],
        'YearsSinceRemodel': years_since_remodel,
        'HouseAge': house_age,
        'GarageFinish': encodings['GarageFinish'][garage_finish],
        'KitchenQual': encodings['KitchenQual'][kitchen_qual],
        'BsmtQual': encodings['BsmtQual'][bsmt_qual],
        'ExterQual': encodings['ExterQual'][exter_qual]
    }

    # Send a POST request to the Flask API
    response = requests.post('http://localhost:5000/predict', json=input_data)

    if response.status_code == 200:
        prediction = response.json()['prediction']
        st.success(f'Predicted House Price: ${prediction:,.2f}')
    else:
        st.error('An error occurred while making the prediction.')
```

Conclusion

This project has enabled us to experiment and build a robust data product. We started from the scratch and went through the steps of data exploration, analysis, feature enginnering, feature selection, hyperparameter tuning, modelling and deployment. Each in step we have learnt new things and we have tried to push oursdelves to reach out to better results. We hope to continue to learn more and make better AI products in the future.