



PROJECT REPORT

HOUSING PRICE PREDICTION

Prepared For
Dr. Amir Rahnama

Prepared By
Mahmood Hossain. ID : c0896079
Chanpreet kaur. ID : c0907021
Nilesh Khurana. ID : c0894394
Rajia Bano. ID : c0907016

ABSTRACT

This project explores the application of machine learning and 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 machine learning and 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.



OBJECTIVE

- Data Preprocessing: Clean and preprocess the dataset to handle missing values, encode categorical variables, and scale numerical features.
- Feature Engineering: Explore and engineer features to enhance the model's predictive power.
- Model Development: Design and implement various deep learning architectures to find the most effective model for price prediction.
- Model Evaluation: Assess the performance of the models using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared.
- Feature Analysis: Investigate the impact of different features on the prediction outcomes to gain insights into the key drivers of property prices.
- Optimization and Tuning: Optimize the model through hyperparameter tuning to achieve the best possible performance.
- Deployment: Create a deployment strategy for the model to be used in real-world applications, providing valuable predictions for stakeholders.



DATASET OVERVIEW

The dataset has been collected from an ongoing Kaggle competition called “House Prices - Advanced Regression Techniques”. This dataset consists the house property attributes of IOWA Ames, USA. Total records of the dataset are **2919** and it has **81** columns. This dataset is a mix of numerical and categorical features with inconsistencies which we had to deal with. Following are the features of the dataset:

- SalePrice - the property's sale price in dollars. This is the target variable that you're trying to predict.
- MSSubClass: The building class
- MSZoning: The general zoning classification
- LotFrontage: Linear feet of street connected to property
- LotArea: Lot size in square feet
- Street: Type of road access
- Alley: Type of alley access
- LotShape: General shape of property
- LandContour: Flatness of the property
- Utilities: Type of utilities available
- LotConfig: Lot configuration
- LandSlope: Slope of property
- Neighborhood: Physical locations within Ames city limits
- Condition1: Proximity to main road or railroad
- Condition2: Proximity to main road or railroad (if a second is present)
- BldgType: Type of dwelling
- HouseStyle: Style of dwelling
- OverallQual: Overall material and finish quality
- OverallCond: Overall condition rating
- YearBuilt: Original construction date
- YearRemodAdd: Remodel date
- RoofStyle: Type of roof
- RoofMatl: Roof material
- Exterior1st: Exterior covering on house
- Exterior2nd: Exterior covering on house (if more than one material)
- MasVnrType: Masonry veneer type
- MasVnrArea: Masonry veneer area in square feet
- ExterQual: Exterior material quality
- ExterCond: Present condition of the material on the exterior
- Foundation: Type of foundation
- BsmtQual: Height of the basement
- BsmtCond: General condition of the basement
- BsmtExposure: Walkout or garden level basement walls
- BsmtFinType1: Quality of basement finished area
- BsmtFinSF1: Type 1 finished square feet
- BsmtFinType2: Quality of second finished area (if present)
- BsmtFinSF2: Type 2 finished square feet
- BsmtUnfSF: Unfinished square feet of basement area
- TotalBsmtSF: Total square feet of basement area
- Heating: Type of heating
- HeatingQC: Heating quality and condition

DATASET OVERVIEW

- CentralAir: Central air conditioning
- Electrical: Electrical system
- 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: Number of bedrooms above basement level
- Kitchen: Number of kitchens
- KitchenQual: Kitchen quality
- TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)
- Functional: Home functionality rating
- Fireplaces: Number of fireplaces
- FireplaceQu: Fireplace quality
- GarageType: Garage location
- GarageYrBlt: Year garage was built
- GarageFinish: Interior finish of the garage
- GarageCars: Size of garage in car capacity
- GarageArea: Size of garage in square feet
- GarageQual: Garage quality
- GarageCond: Garage condition
- PavedDrive: Paved driveway
- 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
- Fence: Fence quality
- MiscFeature: Miscellaneous feature not covered in other categories
- MiscVal: \$Value of miscellaneous feature
- MoSold: Month Sold
- YrSold: Year Sold
- SaleType: Type of sale
- SaleCondition: Condition of sale

MISSING VALUE PERCENTAGE

There are some values missing in some features

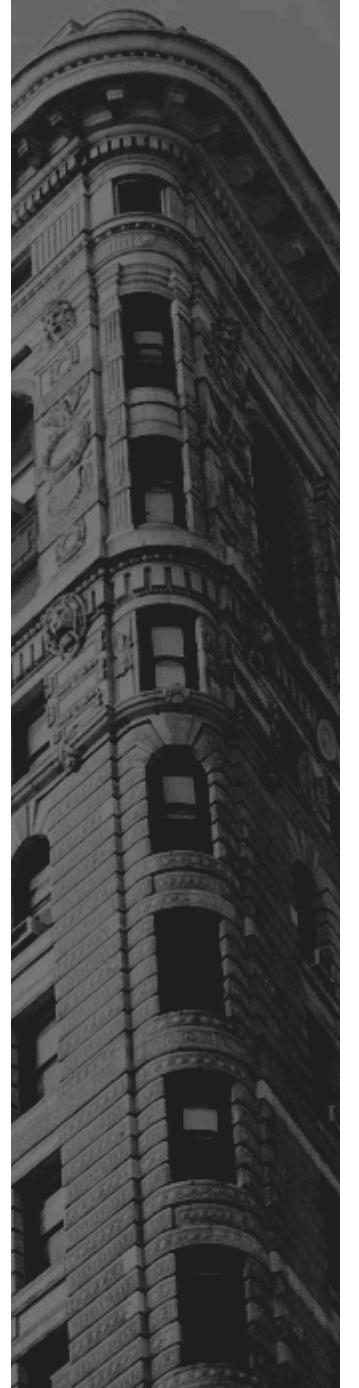
Id	0.000000	BsmtUnfSF	0.034258
MSSubClass	0.000000	TotalBsmtSF	0.034258
MSZoning	0.137033	Heating	0.000000
LotFrontage	16.649538	HeatingQC	0.000000
LotArea	0.000000	CentralAir	0.000000
Street	0.000000	Electrical	0.034258
Alley	93.216855	1stFlrSF	0.000000
LotShape	0.000000	2ndFlrSF	0.000000
LandContour	0.000000	LowQualFinSF	0.000000
Utilities	0.068517	GrLivArea	0.000000
LotConfig	0.000000	BsmtFullBath	0.068517
LandSlope	0.000000	BsmtHalfBath	0.068517
Neighborhood	0.000000	FullBath	0.000000
Condition1	0.000000	HalfBath	0.000000
Condition2	0.000000	BedroomAbvGr	0.000000
BldgType	0.000000	KitchenAbvGr	0.000000
HouseStyle	0.000000	KitchenQual	0.034258
OverallQual	0.000000	TotRmsAbvGrd	0.000000
OverallCond	0.000000	Functional	0.068517
YearBuilt	0.000000	Fireplaces	0.000000
YearRemodAdd	0.000000	FireplaceQu	48.646797
RoofStyle	0.000000	GarageType	5.378554
RoofMatl	0.000000	GarageYrBlt	5.447071
Exterior1st	0.034258	GarageFinish	5.447071
Exterior2nd	0.034258	GarageCars	0.034258
MasVnrType	60.500171	GarageArea	0.034258
MasVnrArea	0.787941	GarageQual	5.447071
ExterQual	0.000000	GarageCond	5.447071
ExterCond	0.000000	PavedDrive	0.000000
Foundation	0.000000	WoodDecksSF	0.000000
BsmtQual	2.774923	OpenPorchSF	0.000000
BsmtCond	2.809181	EnclosedPorch	0.000000
BsmtExposure	2.809181	3SsnPorch	0.000000
BsmtFinType1	2.706406	ScreenPorch	0.000000
BsmtFinSF1	0.034258	PoolArea	0.000000
BsmtFinType2	2.740665	PoolQC	99.657417
BsmtFinSF2	0.034258	Fence	80.438506
		MiscFeature	96.402878
		MiscVal	0.000000
		MoSold	0.000000
		YrSold	0.000000
		SaleType	0.034258
		SaleCondition	0.000000
		SalePrice	49.982871

DATA CLEANING

There are variety of features in the dataset. We will take a look at the features but before that we need to deal 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.

```
# remove features with a high number of missing values  
  
rem_cols = ['Alley', 'PoolQC', 'Fence', 'MiscFeature']  
df.drop(columns=rem_cols, inplace=True)
```

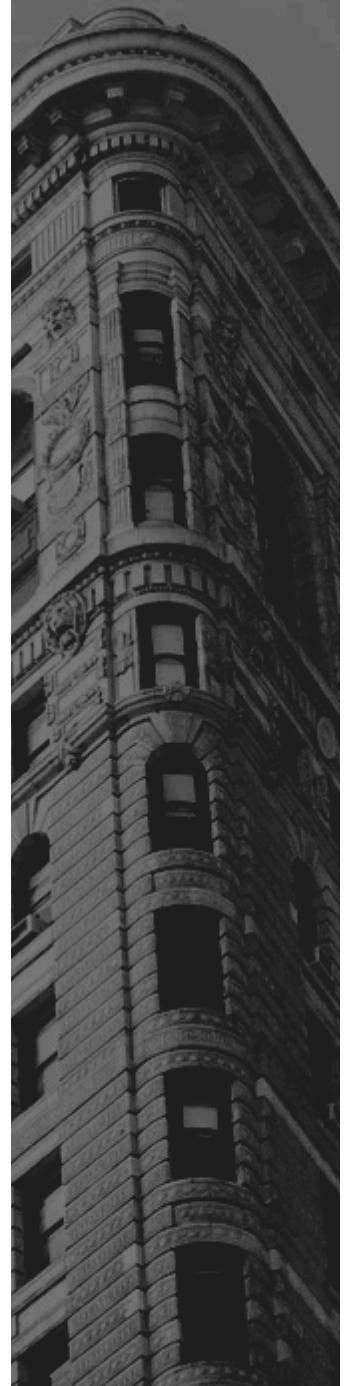
FireplaceQu and MasVnrType columns have 40 - 60% data missing. We need to deal with these columns but before that lets take a look at what they mean. MasVnrType means Masonry veneer type and FireplaceQu means Fireplace Quality. For MasVnrType, we will use 'mode' and for FireplaceQu we will use 'mean'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



DATA CLEANING

We checked the numeric and non numeric features so that we can encode them.

Id	int64	MSZoning	object
MSSubClass	int64	Street	object
LotFrontage	float64	LotShape	object
LotArea	int64	LandContour	object
OverallQual	int64	Utilities	object
OverallCond	int64	LotConfig	object
YearBuilt	int64	LandSlope	object
YearRemodAdd	int64	Neighborhood	object
MasVnrArea	float64	Condition1	object
BsmtFinSF1	float64	Condition2	object
BsmtFinSF2	float64	BldgType	object
BsmtUnfSF	float64	HouseStyle	object
TotalBsmtSF	float64	RoofStyle	object
1stFlrSF	int64	RoofMatl	object
2ndFlrSF	int64	Exterior1st	object
LowQualFinSF	int64	Exterior2nd	object
GrLivArea	int64	MasVnrType	object
BsmtFullBath	float64	ExterQual	object
BsmtHalfBath	float64	ExterCond	object
FullBath	int64	Foundation	object
HalfBath	int64	BsmtQual	object
BedroomAbvGr	int64	BsmtCond	object
KitchenAbvGr	int64	BsmtExposure	object
TotRmsAbvGrd	int64	BsmtFinType1	object
Fireplaces	int64	BsmtFinType2	object
GarageYrBlt	float64	Heating	object
GarageCars	float64	HeatingQC	object
GarageArea	float64	CentralAir	object
WoodDeckSF	int64	Electrical	object
OpenPorchSF	int64	KitchenQual	object
EnclosedPorch	int64	Functional	object
3SsnPorch	int64	FireplaceQu	object
ScreenPorch	int64	GarageType	object
PoolArea	int64	GarageFinish	object
MiscVal	int64	GarageQual	object
MoSold	int64	GarageCond	object
YrSold	int64	PavedDrive	object
SalePrice	float64	SaleType	object
		SaleCondition	object



DATA CLEANING

Data Encoding for object type columns

```
# 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
```

Now lets first deal with the large null columns mentioned earlier

```
# 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)
```

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

```
# 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.

THE GEOPY LIBRARY WAS SUCCESSFULL FILLING IN THE COORDINATES OF THE LOCATIONS. HOWEVER, IT COULD NOT FIND SOME OF THEM. WE WILL BE DOING SOME RESEARCH ON OUR OWN ON GOOGLE MAP TO FIND OUT THOSE LOCATIONS AND MANUALLY FILL IN THE COORDINATES. FINALLY WE WERE ABLE TO FILL IN ALL THE NULL VALUES IN COORDINATES

```
# 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)
```

```
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' : '42.036681, -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
```

```
# 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
```

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

```
# 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=443): Error
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
"""

# Print the error messages
print(error_messages)
```

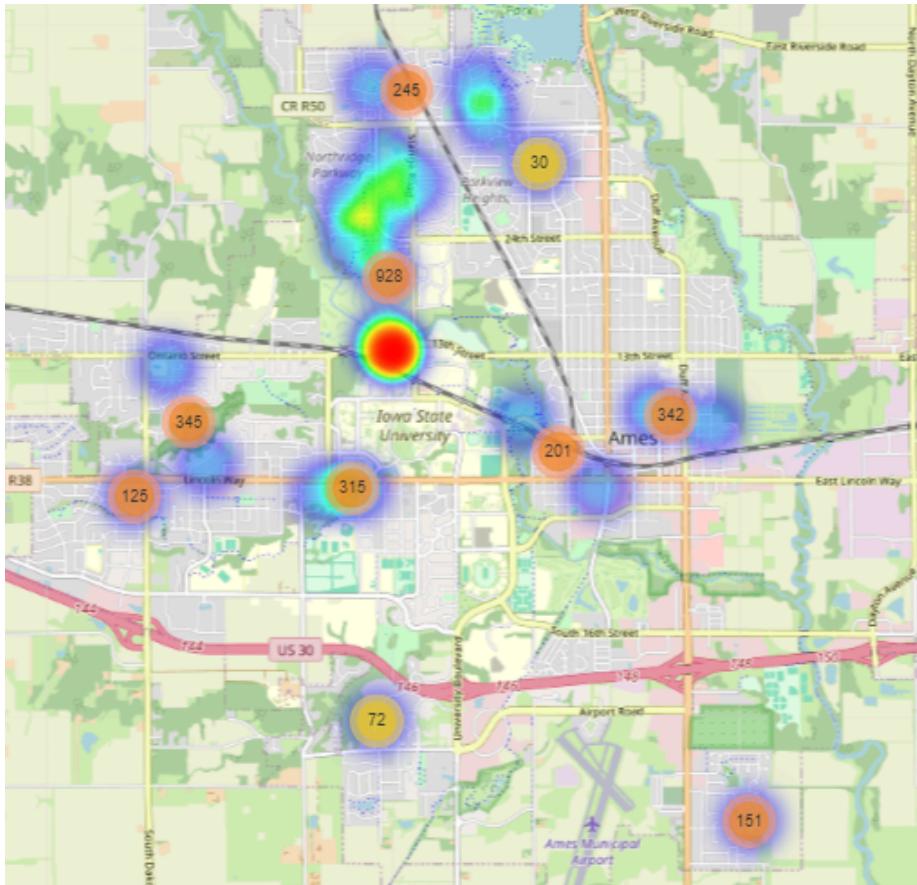
```
# Check for NaN values in Latitude and Longitude
nan_coords = df[df['Latitude'].isna() | df['Longitude'].isna()]

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

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

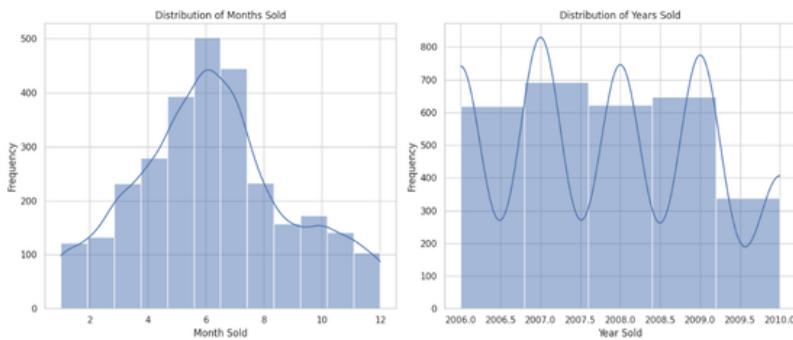
# Print the rows with NaN values in Latitude and Longitude along with Neighborhood_Full
unique_nan_neighborhoods = nan_coords['Neighborhood_Full'].unique()
print(unique_nan_neighborhoods)
```

PROPERTY LOCATIONS



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.

DISTRIBUTION OF MONTHLY AND YEARLY SALES AND SALE PRICE ANALYSIS

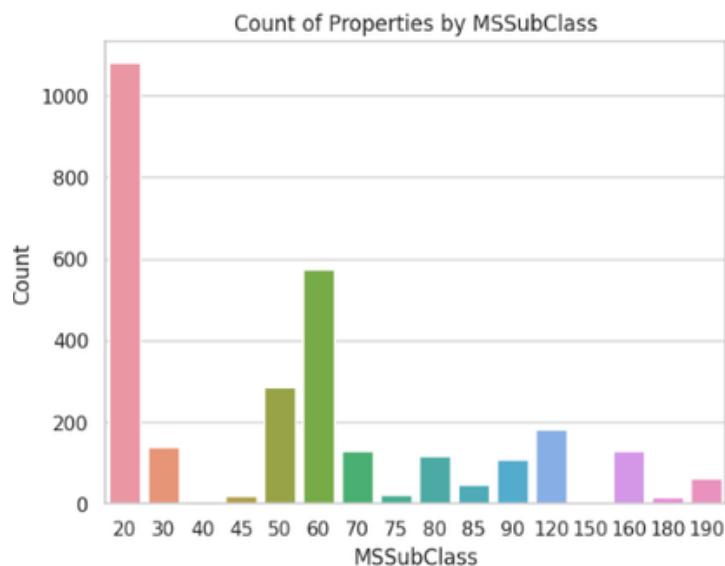


The monthly sell represents a normal distribution which indicates that the highest number of sells happen in the middle of the year. The most number of houses were sold in the years 2007 and 2009.



- 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 beyond 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



MOST TYPES OF HOUSES IN THE DATASET ARE :

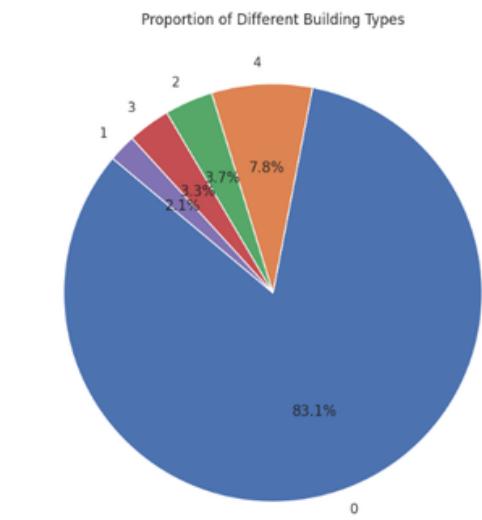
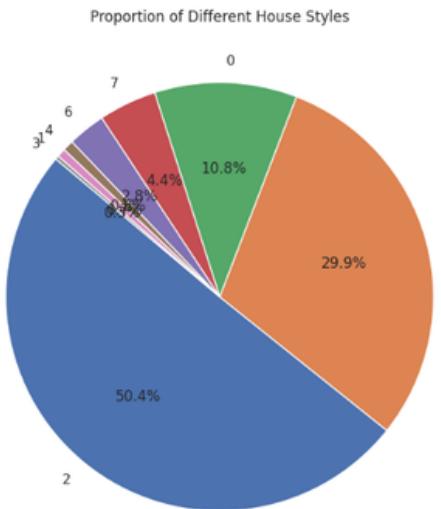
1-STORY 1946 & NEWER ALL STYLES
2-STORY 1946 & NEWER
1-1/2 STORY FINISHED ALL AGES
1-STORY PUD (PLANNED UNIT DEVELOPMENT) - 1946 & NEWER

LEAST TYPES OF HOUSES IN THE DATASET ARE :

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



PROPORTION OF DIFFERENT TYPES IN THE DATASET



CLASS 2 REPRESENTS "1STORY" HOUSES WHICH ACCOUNT FOR MORE THAN 50% IN THE DATASET. FOLLOWED BY CLASS 5 WHICH REPRESENTS '2STORY' HOUSES.

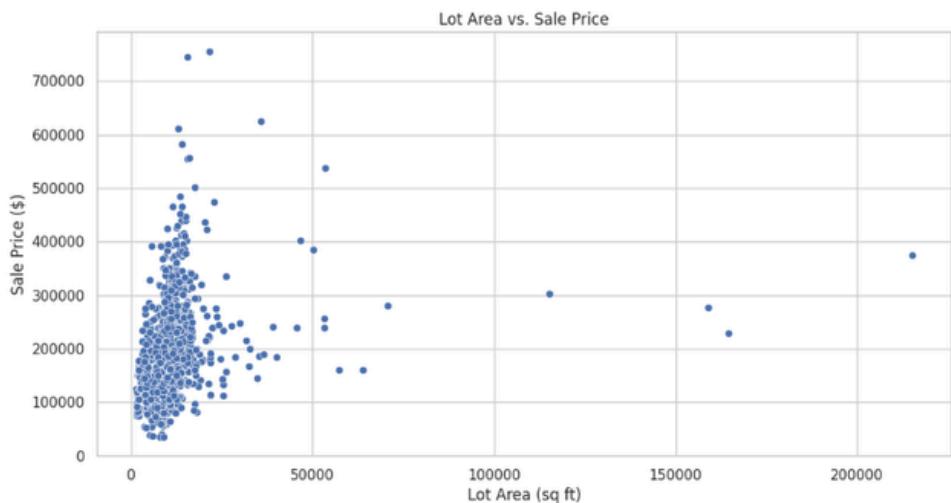
THE ABOVE VISUALIZATION SHOWS THAT CLASS 0 THAT MEANS SINGLE-FAMILY DETACHED BUILDINGS ARE MOSTLY IN THE DATASET WHICH IS EXTREMELY HIGH IN NUMBER FOLLOWED BY CLASS 4 - TOWNHOUSE INSIDE UNIT TYPES OF BUILDINGS.

OVERALL QUALITY VS SALES PRICE



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

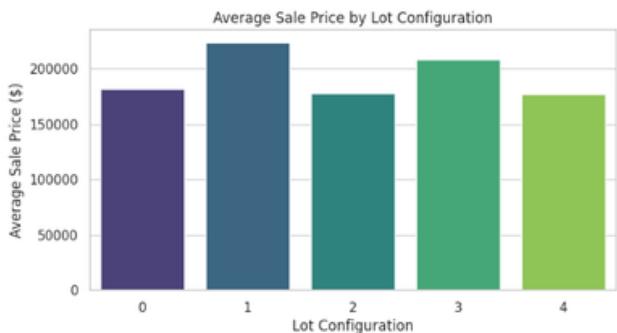


1. The plot represents the relationship between the size of a property's lot (measured in square feet) and its sale price (in dollars).
2. Most data points cluster at the lower end, indicating smaller lots with lower prices.
3. Larger lots show a slight trend toward higher prices

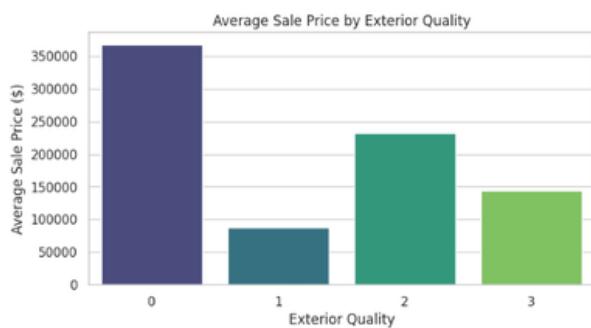
COMPARISON OF THE AVERAGE SALEPRICE ACROSS DIFFERENT LOTSHAPE, LOTCONFIG CATEGORIES AND EXTERIOR QUALITIES



- REGULAR SHAPES REPRESENTING 0 IS COMPARATIVELY CHEAPER
- SLIGHTLY IRREGULAR AND MODERATELY IRREGULAR SHAPES ARE COMPARATIVELY HIGHER IN PRICE

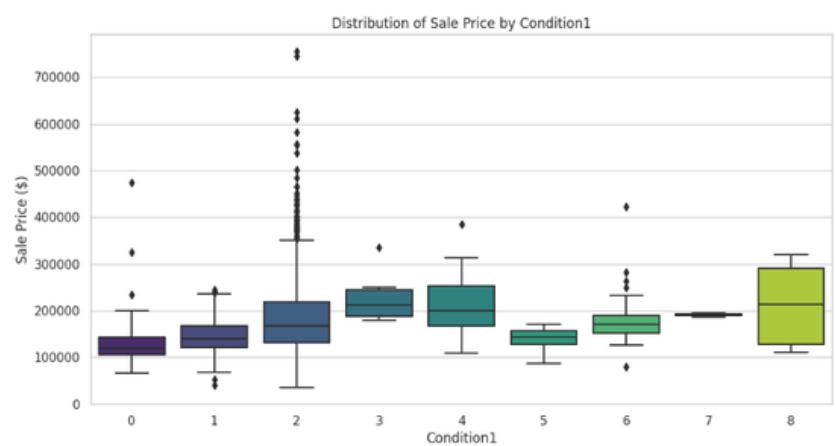
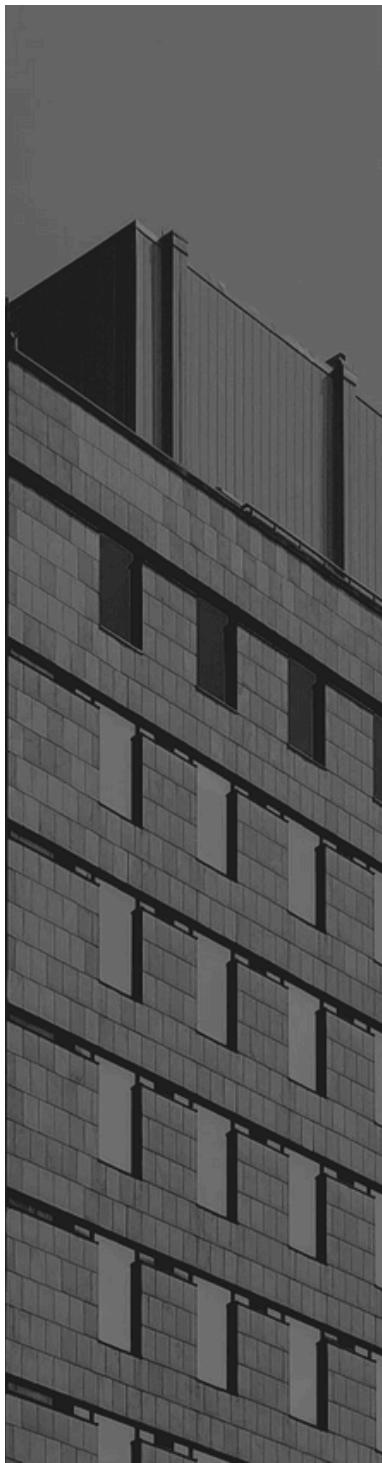


- CUL-DE-SAC HOUSE SITS ON A DEAD-END ROUNDED STREET, FACING OTHER HOUSES AND CREATING A CLOSE-KNIT FEELING BETWEEN NEIGHBORS. THAT REPRESENTS CLASS 1 WHICH ARE HIGHER IN PRICE.
- FRONTEAGE ON 3 SIDES OF PROPERTY TYPES HAVE SECOND HIGHEST AVERAGE PRICE.
- OTHER CONFIGURATIONS ARE "FRONTEAGE ON 2 SIDES OF PROPERTY", "INSIDE LOT" AND "CORNER LOT" WHICH HAVE A SIMILAR AVERAGE PRICE.



- 0 REPRESENTS THE 'EXCELLEN'T 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.

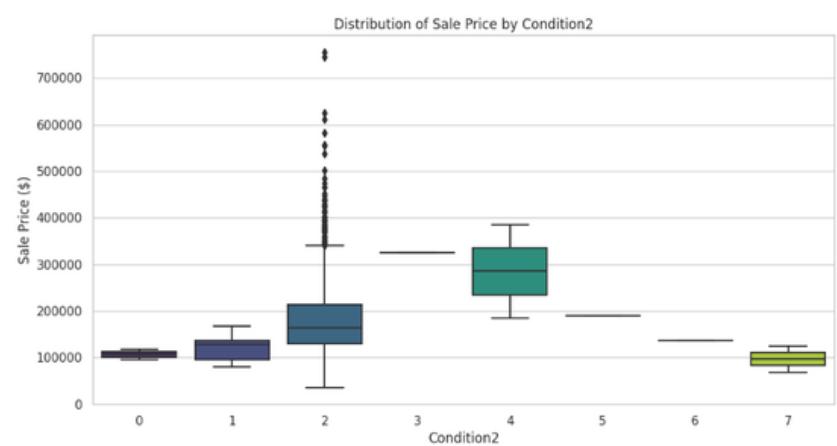
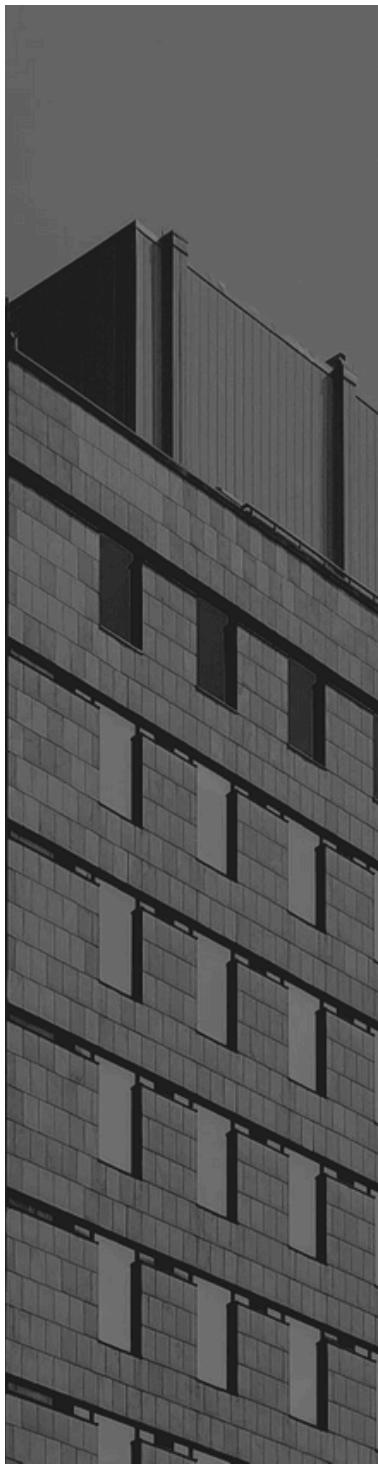
DISTRIBUTION OF SALEPRICE BY CONDITION1 AND CONDITION2



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.

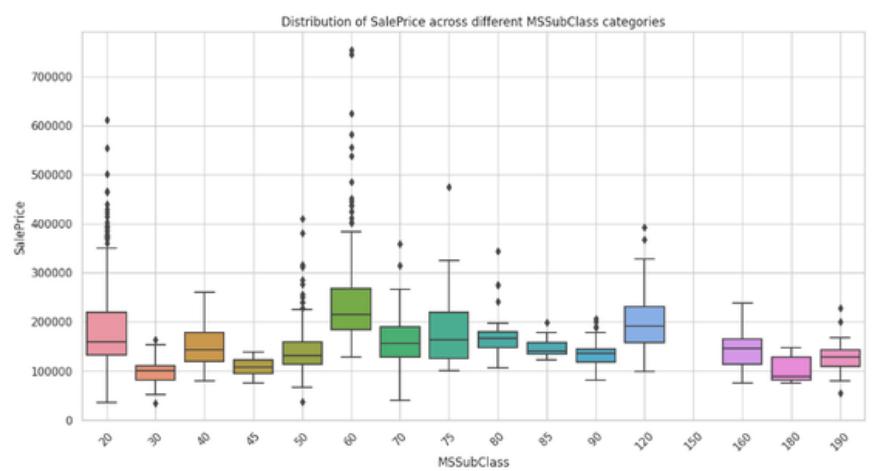
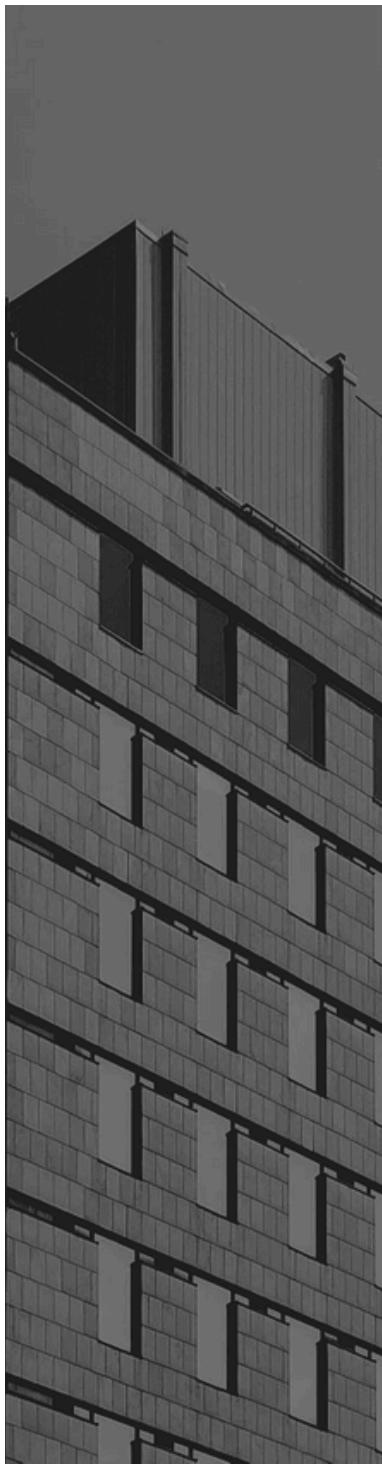
DISTRIBUTION OF SALEPRICE BY CONDITION1 AND CONDITION2



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.

COMPARE SALEPRICE ACROSS DIFFERENT MSSUBCLASS CATEGORIES



- 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

STATISTICAL ANALYSIS

Average, Standard Deviation and Percentiles of the Features

	count	mean	std	min	25%	50%	75%	max
Id	2919.000000	1460.000000	842.787043	1.000000	730.500000	1460.000000	2189.500000	2919.000000
MSSubClass	2919.000000	57.137718	42.517628	20.000000	20.000000	50.000000	70.000000	190.000000
MSZoning	2919.000000	3.030490	0.662386	0.000000	3.000000	3.000000	3.000000	5.000000
LotFrontage	2919.000000	69.305795	21.312345	21.000000	60.000000	69.305795	78.000000	313.000000
LotArea	2919.000000	10168.114080	7886.996359	1300.000000	7478.000000	9453.000000	11570.000000	215245.000000
Street	2919.000000	0.995889	0.063996	0.000000	1.000000	1.000000	1.000000	1.000000
LotShape	2919.000000	1.947585	1.409721	0.000000	0.000000	3.000000	3.000000	3.000000
LandContour	2919.000000	2.776978	0.704391	0.000000	3.000000	3.000000	3.000000	3.000000
Utilities	2919.000000	0.001713	0.055510	0.000000	0.000000	0.000000	0.000000	2.000000
LotConfig	2919.000000	3.055841	1.604472	0.000000	2.000000	4.000000	4.000000	4.000000
LandSlope	2919.000000	0.053786	0.248750	0.000000	0.000000	0.000000	0.000000	2.000000
Neighborhood	2919.000000	12.437136	5.957992	0.000000	7.000000	12.000000	17.000000	24.000000
Condition1	2919.000000	2.040425	0.874047	0.000000	2.000000	2.000000	2.000000	8.000000
Condition2	2919.000000	2.002055	0.209431	0.000000	2.000000	2.000000	2.000000	7.000000
BldgType	2919.000000	0.505653	1.206513	0.000000	0.000000	0.000000	0.000000	4.000000
HouseStyle	2919.000000	3.026721	1.912937	0.000000	2.000000	2.000000	5.000000	7.000000
OverallQual	2919.000000	6.089072	1.409947	1.000000	5.000000	6.000000	7.000000	10.000000
OverallCond	2919.000000	5.564577	1.113131	1.000000	5.000000	5.000000	6.000000	9.000000
YearBuilt	2919.000000	1971.312778	30.291442	1872.000000	1953.500000	1973.000000	2001.000000	2010.000000
YearRemodAdd	2919.000000	1984.264474	20.894344	1950.000000	1965.000000	1993.000000	2004.000000	2010.000000
RoofStyle	2919.000000	1.396369	0.820906	0.000000	1.000000	1.000000	1.000000	5.000000
RoofMatl	2919.000000	1.063035	0.539210	0.000000	1.000000	1.000000	1.000000	7.000000
Exterior1st	2919.000000	9.625214	3.200303	0.000000	8.000000	12.000000	12.000000	15.000000
Exterior2nd	2919.000000	10.337102	3.552133	0.000000	8.000000	13.000000	13.000000	16.000000
MasVnrType	2919.000000	2.286742	0.926533	0.000000	1.000000	3.000000	3.000000	3.000000
MasVnrArea	2919.000000	102.201312	178.626089	0.000000	0.000000	0.000000	163.500000	1600.000000
ExterQual	2919.000000	2.530661	0.702245	0.000000	2.000000	3.000000	3.000000	3.000000
ExterCond	2919.000000	3.708804	0.773641	0.000000	4.000000	4.000000	4.000000	4.000000
Foundation	2919.000000	1.393285	0.727061	0.000000	1.000000	1.000000	2.000000	5.000000
BsmtQual	2919.000000	2.288112	0.922771	0.000000	2.000000	2.000000	3.000000	4.000000
BsmtCond	2919.000000	2.835903	0.700631	0.000000	3.000000	3.000000	3.000000	4.000000
BsmtExposure	2919.000000	2.327509	1.151168	0.000000	2.000000	3.000000	3.000000	4.000000
BsmtFinType1	2919.000000	2.846865	1.862342	0.000000	2.000000	2.000000	5.000000	6.000000
BsmtFinSF1	2919.000000	441.423235	455.532750	0.000000	0.000000	369.000000	733.000000	5644.000000
BsmtFinType2	2919.000000	4.714628	1.012141	0.000000	5.000000	5.000000	5.000000	6.000000
BsmtFinSF2	2919.000000	49.582248	169.176615	0.000000	0.000000	0.000000	0.000000	1526.000000
BsmtUnfSF	2919.000000	560.772104	439.468337	0.000000	220.000000	467.000000	805.000000	2336.000000
TotalBsmtSF	2919.000000	1051.777587	440.690726	0.000000	793.000000	990.000000	1302.000000	6110.000000
Heating	2919.000000	1.025351	0.245678	0.000000	1.000000	1.000000	1.000000	5.000000
HeatingQC	2919.000000	1.533744	1.742548	0.000000	0.000000	0.000000	4.000000	4.000000
CentralAir	2919.000000	0.932854	0.250318	0.000000	1.000000	1.000000	1.000000	1.000000
Electrical	2919.000000	3.685509	1.047746	0.000000	4.000000	4.000000	4.000000	5.000000
1stFlrSF	2919.000000	1159.581706	392.362079	334.000000	876.000000	1082.000000	1387.500000	5095.000000
2ndFlrSF	2919.000000	336.483727	428.701456	0.000000	0.000000	0.000000	704.000000	2065.000000
LowQualFinSF	2919.000000	4.694416	46.396825	0.000000	0.000000	0.000000	0.000000	1064.000000
GrlivArea	2919.000000	1500.759849	506.051045	334.000000	1126.000000	1444.000000	1743.500000	5642.000000
BsmtFullBath	2919.000000	0.429894	0.524556	0.000000	0.000000	0.000000	1.000000	3.000000

STATISTICAL ANALYSIS

Average, Standard Deviation and Percentiles of the Features

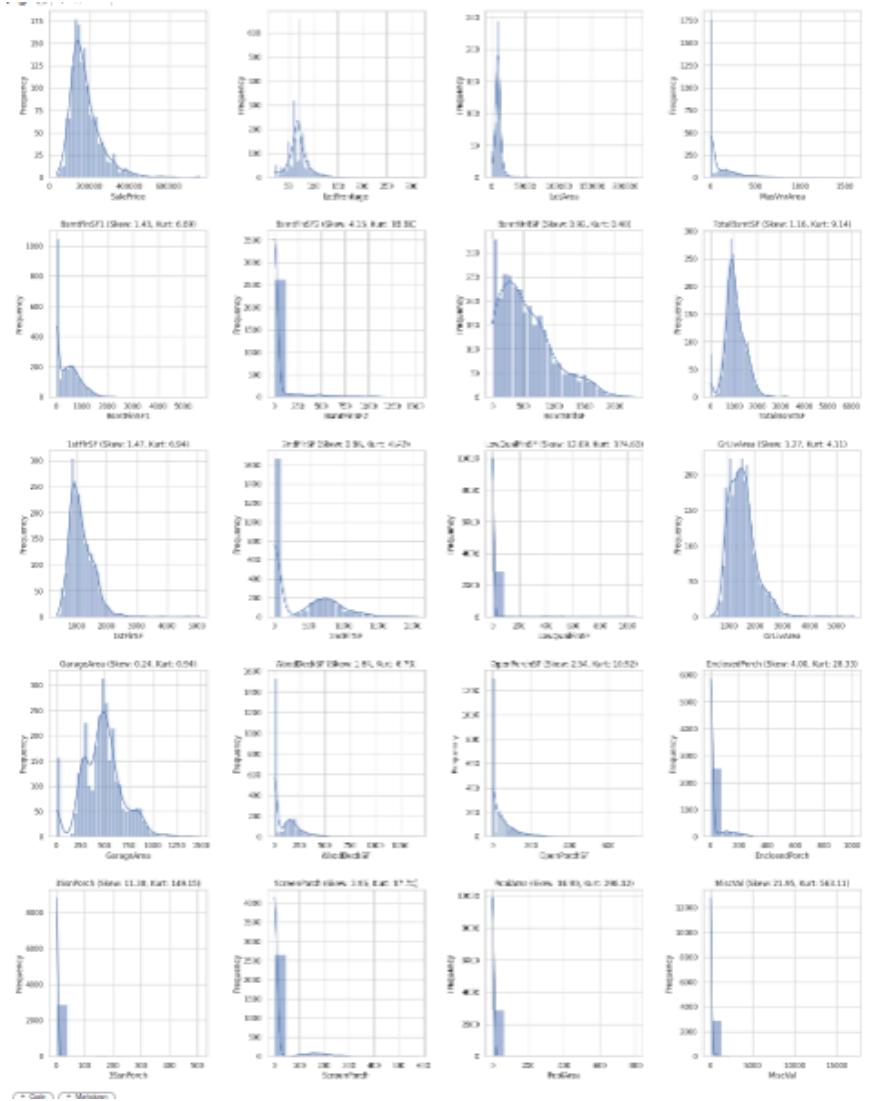
BsmtHalfBath	2919.000000	0.061364	0.245603	0.000000	0.000000	0.000000	0.000000	2.000000
FullBath	2919.000000	1.568003	0.552969	0.000000	1.000000	2.000000	2.000000	4.000000
HalfBath	2919.000000	0.380267	0.502872	0.000000	0.000000	0.000000	1.000000	2.000000
BedroomAbvGr	2919.000000	2.860226	0.822693	0.000000	2.000000	3.000000	3.000000	8.000000
KitchenAbvGr	2919.000000	1.044536	0.214462	0.000000	1.000000	1.000000	1.000000	3.000000
KitchenQual	2919.000000	2.347379	0.834847	0.000000	2.000000	3.000000	3.000000	4.000000
TotRmsAbvGrd	2919.000000	6.451524	1.569379	2.000000	5.000000	6.000000	7.000000	15.000000
Functional	2919.000000	5.760534	0.935847	0.000000	6.000000	6.000000	6.000000	7.000000
Fireplaces	2919.000000	0.597122	0.646129	0.000000	0.000000	1.000000	1.000000	4.000000
FireplaceQu	2919.000000	3.825968	1.398569	0.000000	2.000000	4.000000	5.000000	5.000000
GarageType	2919.000000	2.483727	1.932814	0.000000	1.000000	1.000000	5.000000	6.000000
GarageYrBlt	2919.000000	1978.113406	24.867762	1895.000000	1961.500000	1978.113406	2001.000000	2207.000000
GarageFinish	2919.000000	1.284001	0.897327	0.000000	1.000000	1.000000	2.000000	3.000000
GarageCars	2919.000000	1.766621	0.761494	0.000000	1.000000	2.000000	2.000000	5.000000
GarageArea	2919.000000	472.874572	215.357904	0.000000	320.000000	480.000000	576.000000	1488.000000
GarageQual	2919.000000	3.904762	0.692049	0.000000	4.000000	4.000000	4.000000	5.000000
GarageCond	2919.000000	3.959233	0.568221	0.000000	4.000000	4.000000	4.000000	5.000000
PavedDrive	2919.000000	1.830764	0.537299	0.000000	2.000000	2.000000	2.000000	2.000000
WoodDeckSF	2919.000000	93.709832	126.526589	0.000000	0.000000	0.000000	168.000000	1424.000000
OpenPorchSF	2919.000000	47.486811	67.575493	0.000000	0.000000	26.000000	70.000000	742.000000
EnclosedPorch	2919.000000	23.098321	64.244246	0.000000	0.000000	0.000000	0.000000	1012.000000
3SsnPorch	2919.000000	2.602261	25.188169	0.000000	0.000000	0.000000	0.000000	508.000000
ScreenPorch	2919.000000	16.062350	56.184365	0.000000	0.000000	0.000000	0.000000	576.000000
PoolArea	2919.000000	2.251799	35.663946	0.000000	0.000000	0.000000	0.000000	800.000000
MiscVal	2919.000000	50.825968	567.402211	0.000000	0.000000	0.000000	0.000000	17000.000000
MoSold	2919.000000	6.213087	2.714762	1.000000	4.000000	6.000000	8.000000	12.000000
YrSold	2919.000000	2007.792737	1.314964	2006.000000	2007.000000	2008.000000	2009.000000	2010.000000
SaleType	2919.000000	7.491607	1.593719	0.000000	8.000000	8.000000	8.000000	9.000000
SaleCondition	2919.000000	3.779034	1.078241	0.000000	4.000000	4.000000	4.000000	5.000000
SalePrice	1460.000000	180921.195890	79442.502883	34900.000000	129975.000000	163000.000000	214000.000000	755000.000000
Latitude	2919.000000	42.036187	0.023631	41.990092	42.022005	42.033483	42.049205	42.107339
Longitude	2919.000000	-93.644576	0.020378	-93.680265	-93.652025	-93.648429	-93.635387	-93.601829

- 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.

SKEWNESS ANALYSIS

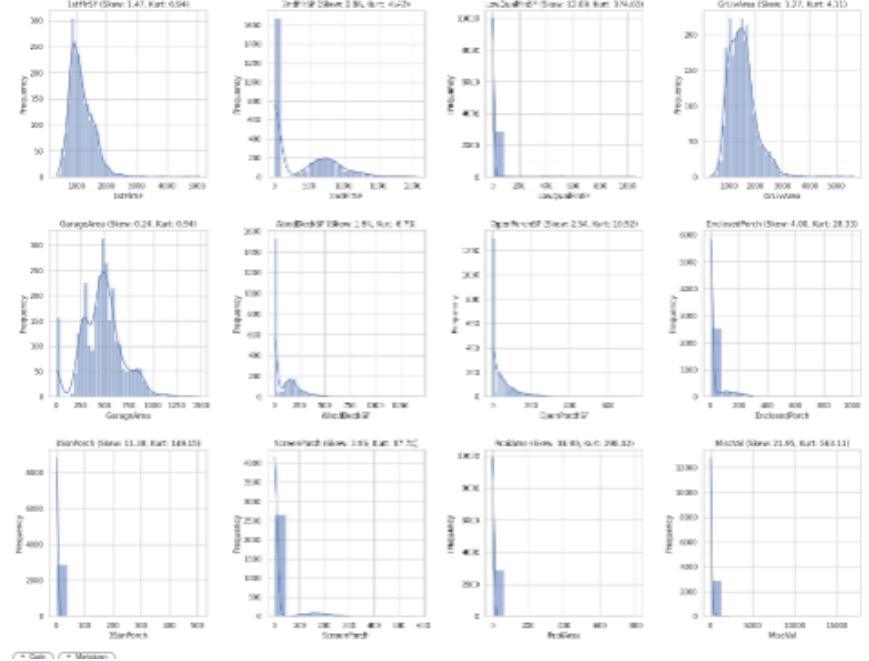
Skewness:

- 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.

```
# List of columns to exclude from outlier handling
excluded_columns = ['Id', 'Neighborhood', 'MoSold', 'YrSold', 'Neighborhood_Decoded', 'Neighborhood_Full', 'Latitude', 'Longitude']

# 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)
    interquartile_range = quartile3 - quartile1
    up_limit = quartile3 + 1.5 * interquartile_range
    low_limit = quartile1 - 1.5 * interquartile_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_limit)][col_name]
    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.")
```

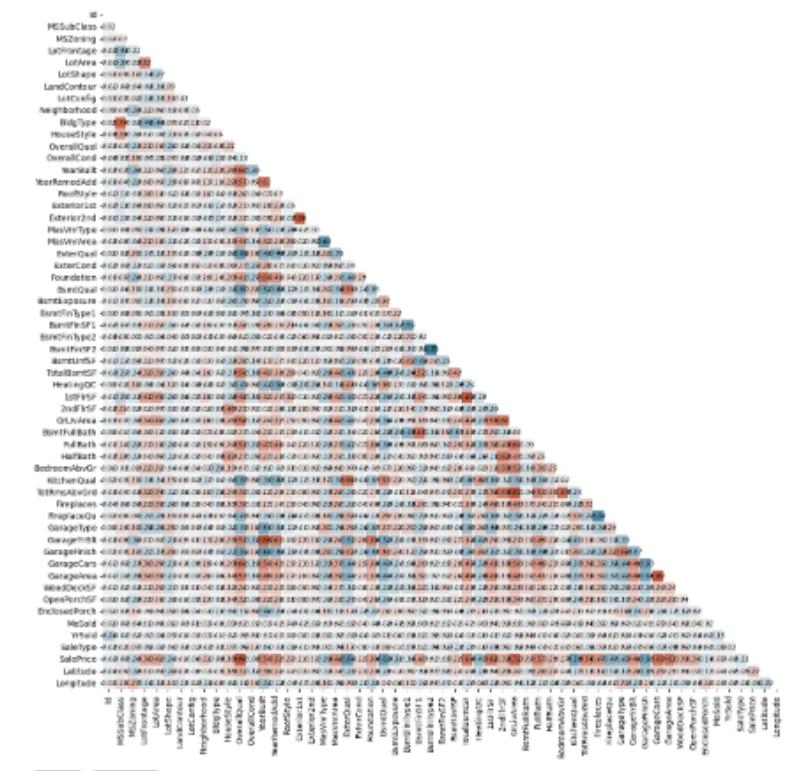
CORRELATION MATRIX

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.

```
unique_value_counts = df.nunique()
variables_with_one_unique_value = unique_value_counts[unique_value_counts == 1].index.tolist()
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', 'BsmtHlfBath', 'KitchenAbvGr', 'Functional', 'GarageQual', 'GarageCond', 'PavedDrive', 'SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'SaleCondition']

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



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 have created few features based on the context

```
# 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' else 0)  
  
# 14. HouseStyleQuality  
df['HouseStyleQuality'] = df['HouseStyle'].apply(lambda x: 1 if x in ['2Story', '1.5Fin'] else 0)  
  
# 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 OVERTFITTING

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.

MOVING HIGH CORRELATED FEATURE AND OTHER CATEGORICAL FEATURES

```
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.index if abs(upper.loc[row, column]) > threshold]

    return high_corr_features
```

```
high_corr_features = find_high_correlation_features(df, threshold=0.95)
print(high_corr_features)

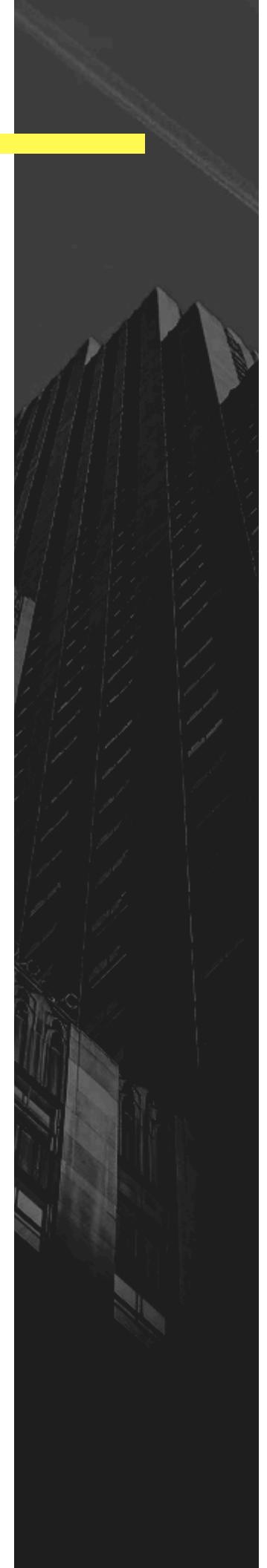
[('HouseArea', 'YearBuilt'), ('YearsSinceRemodel', 'YearRemodAdd')]
```

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

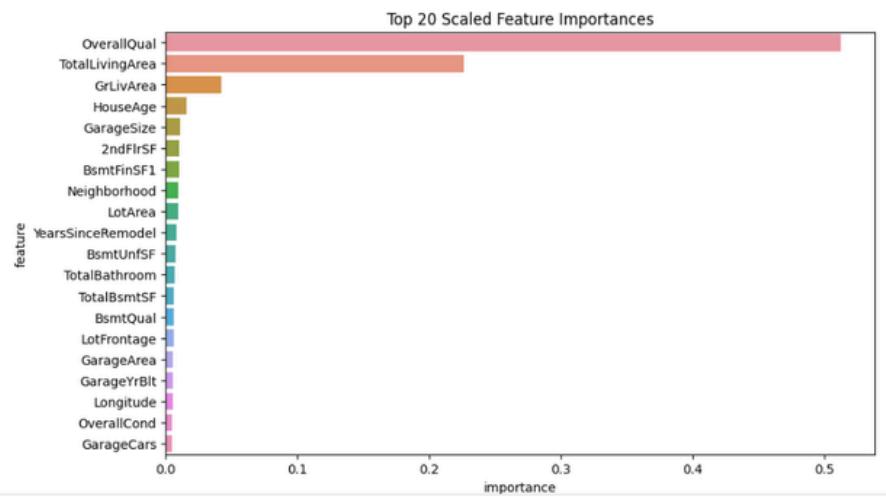
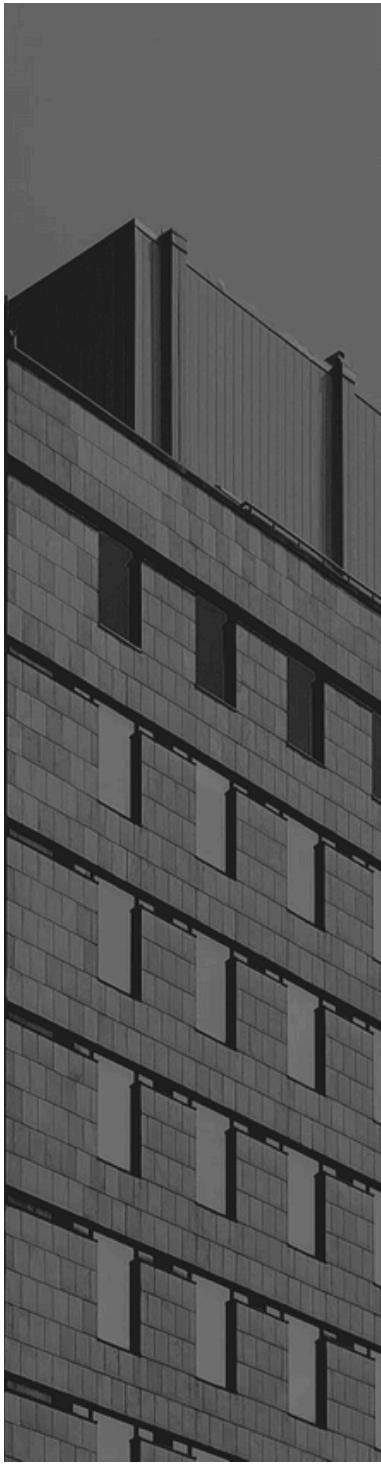
```
# removing
df = df.drop(['YearBuilt', 'YearRemodAdd', 'Id', 'Neighborhood_Decoded', 'Neighborhood_Full'], axis=1)
```

FEATURE IMPORTANCE AND FEATURE SELECTION

*We tried different
techniques for extracting
important features.*



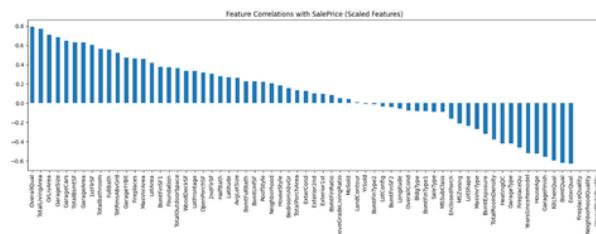
FEATURE IMPORTANCE FROM RANDOM FOREST



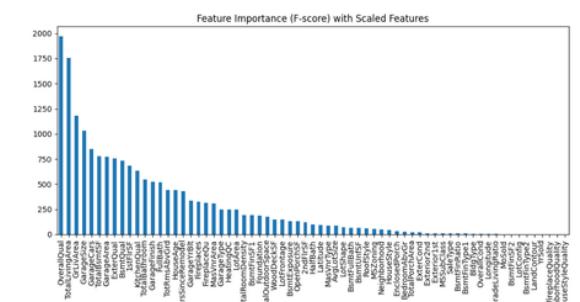
We wanted to try other feature importance techniques and finally take the decision after comparing one to another.

OTHER TECHNIQUES TO GET FEATURE IMPORTANCE

USING CORRELATION FOR FEATURE IMPORTANCE



F-SCORE FOR FEATURE IMPORTANCE



WE TRIED TO FIND OUT THE COMMON FEATURES IN THESE TWO TECHNIQUES

```

common_features = set(top_correlated.index).intersection(set(f_scores.index))

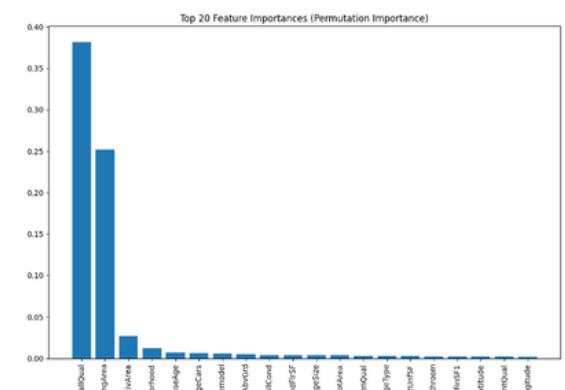
# Convert the result to a list if needed
common_features_list = list(common_features)

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

Common Features: ['foundation', 'LotShape', 'BedsBreakfast', 'TotalBsmtSF', 'HasHSArea', 'GarageType', 'HouseAge', 'TotalBldgDensity', 'ExterQual', 'StreetQual', 'AvgGrsqft', 'ExtWalls', 'TotalPorchArea', 'TotalPorchSF', 'LotArea', 'Bathfullsf', 'KitchenID', 'GarageCars', 'TotalLivingArea', 'GarageYrBlt', 'TotalPorchSF', 'EnclosedPorch', '2ndFlrSF', 'BsmtUnfSF', 'GarageArea', 'LotFrontage', 'Fireplaces', 'FireplaceQu', 'Latitude', 'YearSinceModel', 'WoodDeckSF', 'BsmtFinSF', 'ExterCond', 'BsmtQual', 'OverallQual', 'GarageFinish', 'GrLvlAcs', 'HouseStyle', 'RoofStyle', 'TotalDwSpace']

```

WE ALSO TRIED A TECHNIQUE CALLED PERMUTATION FEATURE IMPORTANCE



FEATURE IMPORTANCE DECISION

we had extracted features using different kinds methods. Seeing the results, using top correlated features made more sense to us.

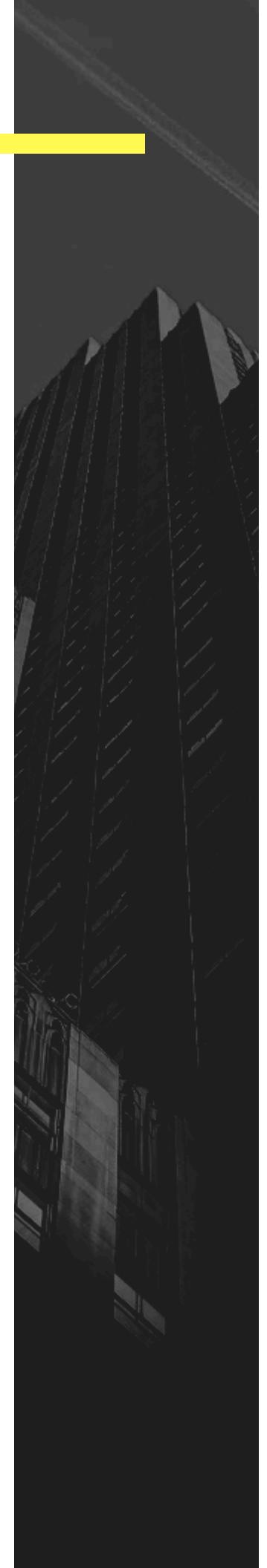
Using top correlated features gave us the opportunity to use all high correlated features which were recommended by other techniques too.



MODELLING

For modelling, we tried different methodologies

- **Baseline 1 :** At this stage, we used all the features in three models : Linear Regression, Decision Tree and Neural Network
- **Baseline 2 :** Here we have used selected features which we got from our correlation features. Then we have applied **log transformation** and **scaling**. Then we applied the models below :
 - Linear Regression
 - Ridge Regression
 - Lasso Regression
 - ElasticNet,
 - Decision Tree
 - Random Forest
 - Gradient Boosting
 - SVR
 - KNN
 - XGBoost
 - LightGBM
 - Neural Network
- **Final Model :** At this stage, we applied hyperparameter tuning to the best two machine learning models that we got in baseline 2 - **Gradient Boosting and Random Forest**. We evaluated the results of these two models along with the **Neural Network** model. Finally the result of the Gradient Boosting model was the best.



HYPERPARAMETER TUNING

APPLIED HYPERPARAMETER TUNING AFTER BASELINE 2 USING GRIDSEARCHCV

Tuning Gradient Boosting

```
# 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_jobs=-1)
    grid_search.fit(X_train, y_train)
    return grid_search.best_params_
```

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

Tuning Random Forest

```
# 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_jobs=-1)
    grid_search.fit(X_train, y_train)

    return grid_search.best_params_
```

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

MODEL EVALUATION

Baseline 1

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

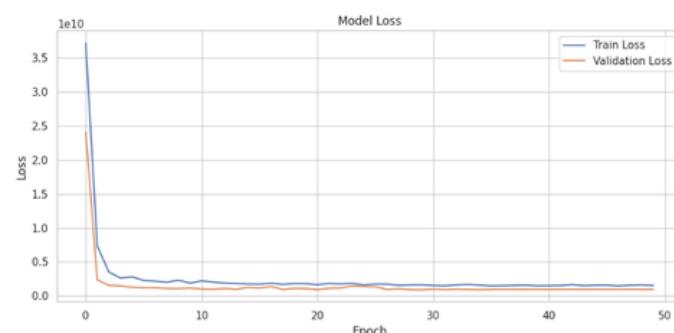
EVALUATING DECISION TREE:
CROSS-VALIDATION RMSE: 37708.3540 (+/- 7471.5025)
CROSS-VALIDATION R2 SCORE: 0.7556 (+/- 0.0922)
VALIDATION RMSE: 37209.4743
VALIDATION R2 SCORE: 0.7936

EVALUATING NEURAL NETWORK
TEST LOSS: 943303104.0000
TEST MAE: 21773.9258

Baseline 2

Model	Train RMSE	Train R2	Validation RMSE	Validation R2	\
Gradient Boosting	16802.865097	0.950413	25725.422160	0.901334	
Random Forest	10826.503980	0.979414	26046.141330	0.898859	
LightGBM	12208.428742	0.973823	27086.585549	0.890617	
XGBoost	1462.162601	0.999625	27739.142647	0.885283	
Linear Regression	31209.603918	0.828928	31251.098562	0.854396	
Lasso Regression	31209.611482	0.828928	31251.423325	0.854393	
Ridge Regression	31211.332498	0.828909	31253.878490	0.854370	
ElasticNet	32651.597633	0.812755	32930.519558	0.838326	
KNN	27640.195598	0.865821	32989.866727	0.837743	
Decision Tree	171.728222	0.999995	34021.001562	0.827441	
SVR	77101.362465	-0.044061	82823.164384	-0.022695	

```
{'Model': 'Deep Learning (TensorFlow)', 'Train RMSE': 30390.47702245533, 'Train R2': 0.8377902525300349,  
'Validation RMSE': 32910.75214965297, 'Validation R2': 0.8385202175947031,
```



METRICS AFTER TUNING

Gradient Boosting

- 'MODEL': 'GRADIENT BOOSTING (BEST PARAMETERS)',
- 'TRAIN RMSE': 9635.141982088855,
- 'TRAIN R2': 0.9836950995468902,
- 'VALIDATION RMSE': 25395.863406245982,
- 'VALIDATION R2': 0.903845745552171

Random Forest

- 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

PACKAGING THE BEST MODEL WITH PICKLE

Using Pickle, we saved the model, selected features, skewed features and scaler to use for inference.

```
# 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)
```

DEPLOYMENT USING FLASK AND STREAMLIT

At this stage we had the best model saved and wanted to deploy our model using flask and streamlit. Streamlit creates an UI and pushes the input to the Flask server and returns the output. The codes for flask and streamlit are below.

```
# 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])

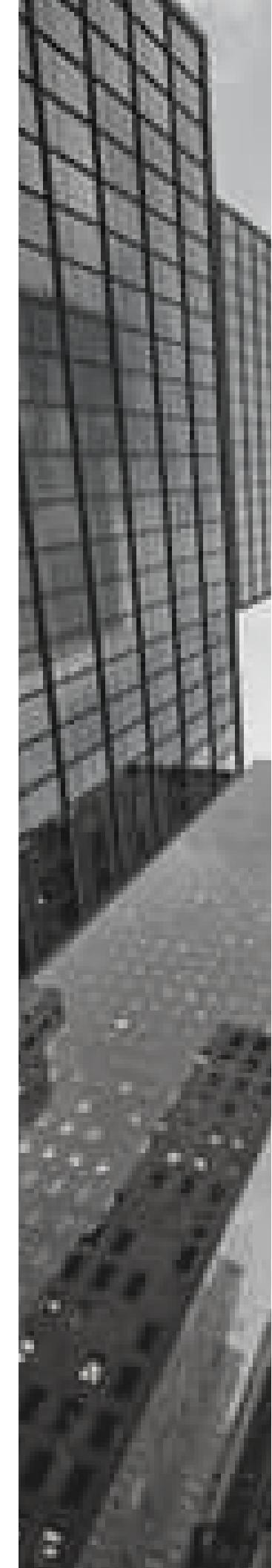
    # Handle infinite values
    input_selected = input_selected.replace([np.inf, -np.inf], np.nan)
    input_selected = input_selected.fillna(input_selected.mean())

    # 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)
```



DEPLOYMENT USING FLASK AND STREAMLIT

THE STREAMLIT APP DECODES THE CATEGORICAL FEATURE INPUTS AS PER THE ENCODING BY LABEL ENCODER AND THEN IT PUSHES THE INPUT VARIABLES TO THE FLASK SERVER.

```
# 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, 'Detchd': 5},
    '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')

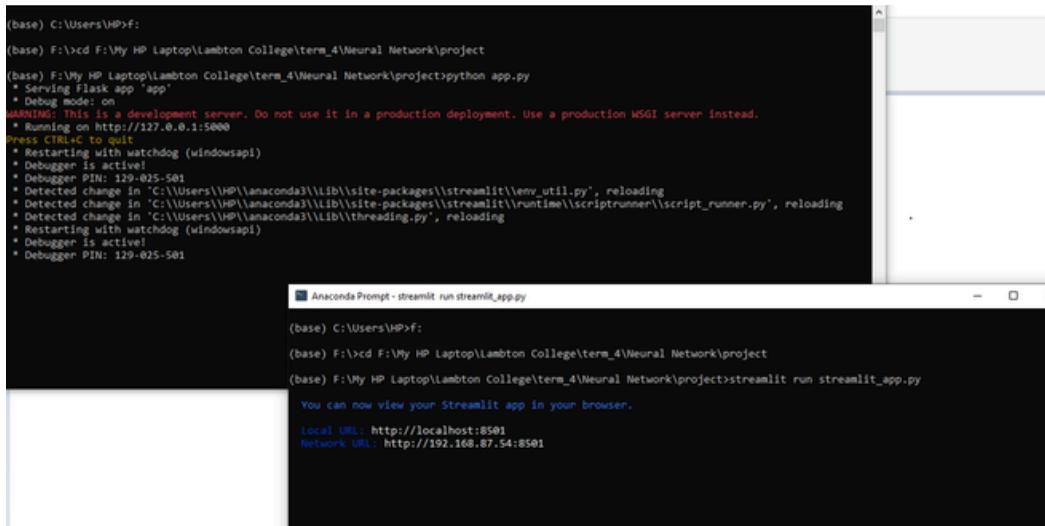
# Create input fields for each feature
overall_qual = st.slider('Overall Quality', 1, 10, 5)
total_living_area = st.number_input('Total Living Area (sq ft)', min_value=0)
gr_liv_area = st.number_input('Above Ground Living Area (sq ft)', min_value=0)
garage_size = st.number_input('Garage Size (cars)', 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_fir_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,
        'FirstFlrSF': first_fir_sf,
        'TotalBathrooms': 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.')
```

USER INTERFACE OF THE APPLICATION



The image shows two terminal windows side-by-side. The left terminal window is titled '(base) C:\Users\HP>f:' and displays the command 'python app.py' being run. It outputs the following text:

```
(base) C:\Users\HP>f:  
(base) F:\>cd F:\My HP Laptop\Lambton College\term_4\Neural Network\project  
(base) F:\My HP Laptop\Lambton College\term_4\Neural Network\project>python app.py  
* Serving Flask app 'app'  
* Debug mode: on  
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
* Running on http://127.0.0.1:5000  
Press CTRL+C to quit  
* Restarting with watchdog (windowsapi)  
Debugger is active!  
Debugger PIN: 129-025-501  
* Reloading changes in 'C:\Users\HP\anaconda3\lib\site-packages\streamlit\env_util.py', reloading  
* Detected change in 'C:\Users\HP\anaconda3\lib\site-packages\streamlit\runtime\script_runner.py', reloading  
* Detected change in 'C:\Users\HP\anaconda3\lib\threading.py', reloading  
* Restarting with watchdog (windowsapi)  
Debugger is active!  
Debugger PIN: 129-025-501
```

The right terminal window is titled 'Anconda Prompt - streamlit run streamlit_app.py' and displays the command 'streamlit run streamlit_app.py' being run. It outputs the following text:

```
Anconda Prompt - streamlit run streamlit_app.py  
(base) C:\Users\HP>f:  
(base) F:\>cd F:\My HP Laptop\Lambton College\term_4\Neural Network\project  
(base) F:\My HP Laptop\Lambton College\term_4\Neural Network\project>streamlit run streamlit_app.py  
You can now view your Streamlit app in your browser.  
Local URL: http://localhost:8501  
Network URL: http://192.168.87.54:8501
```

First we run our Flask app in a terminal. Then we run the Streamlit app in another terminal



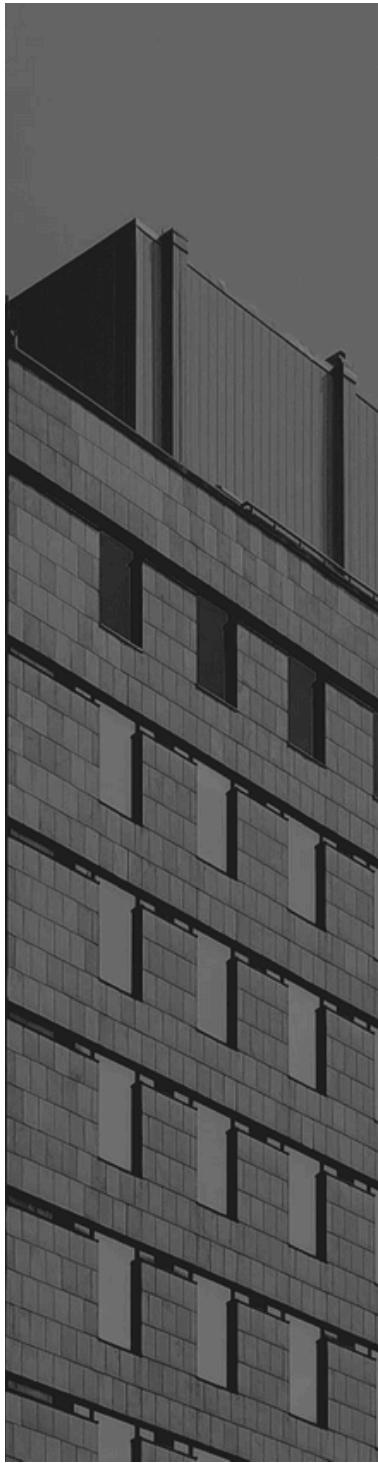
USER INTERFACE OF THE APPLICATION

House Price Predictor in Iowa

Overall Quality	<input type="range" value="6"/>
Total Living Area (sq ft)	<input type="text" value="0"/> - +
Above Ground Living Area (sq ft)	<input type="text" value="0"/> - +
Garage Size (cars)	<input type="text" value="0"/> - +
Garage Cars	<input type="text" value="0"/> - +
Total Basement Area (sq ft)	<input type="text" value="0"/> - +
Garage Area (sq ft)	<input type="text" value="0"/> - +
First Floor Area (sq ft)	<input type="text" value="0"/> - +
Total Bathrooms	<input type="text" value="0"/> - +
Garage Type	<input type="text" value="2Types"/> - +
Garage Finish	<input type="text" value="Fin"/> - +
Kitchen Quality	<input type="text" value="Gd"/> - +
Basement Quality	<input type="text" value="Fa"/> - +
Exterior Quality	<input type="text" value="Gd"/> - +
<input type="button" value="Predict Price"/>	
Predicted House Price: \$86,976.60	

After running both the apps, we will be able to see the application in our browser which will be able to take inputs provide the predicted price based on our Gradient Boosting model.

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 engineering, feature selection, hyperparameter tuning, modelling and deployment. Each in step we have learnt new things and we have tried to push ourselves to reach out to better results. We hope to continue to learn more and make better AI products in the future.

APPENDIX

Github Repository

https://www.github.com/farsim-hossain/house_price_prediction_ames_iowa.git

Dataset

<https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data>

