Digital Egypt Pioneers

Ministry of Communications and Information Technology

Exploratory Data Analysis (EDA) Report for House Price Prediction Project

1. Introduction

The dataset used in this analysis, cleaned_house_data.csv, contains various features that can influence house prices. These features include both categorical and numerical data. The main target variable is Property_Sale_Price, which represents the sale price of properties. The goal of this EDA is to explore the dataset, identify important patterns, and detect relationships between features and house prices.

2. Dataset Overview

The dataset contains several features with both numerical and categorical variables. Here is an overview:

- **Total columns:** It contains a variety of attributes (numerical and categorical) number of columns 81.
- **Total rows:** Based on the dataset of house_prices.csv number of rows are 1460

General Information:

- **Data types:**
    - **Numerical**: Features such as ['Id', 'Dwell_Type', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'Property_Sale_Price']
- **Categorical**: Features such as ['Zone_Class', 'Road_Type', 'Alley', 'Property_Shape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'Dwelling_Type', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond',

'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual', 'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature', 'SaleType', 'SaleCondition']

- **Null values:**
    - Significant missing values were detected for some columns. Example: MasVnrArea, Electrical, BsmtQual, GarageYrBlt (garage-related features), and others. These were highlighted and addressed.

4]:

	col	dtype	unique values	count unique	count null	percentage null
72	PoolQC	object	[nan, Ex, Fa, Gd]	3	1453	99.520548
74	MiscFeature	object	[nan, Shed, Gar2, Othr, TenC]	4	1406	96.301370
6	Alley	object	[nan, Grvl, Pave]	2	1369	93.767123
73	Fence	object	[nan, MnPrv, GdWo, GdPrv, MnWw]	4	1179	80.753425
25	MasVnrType	object	[BrkFace, nan, Stone, BrkCmn]	3	872	59.726027
57	FireplaceQu	object	[nan, TA, Gd, Fa, Ex, Po]	5	690	47.260274
3	LotFrontage	float64	[65.0, 80.0, 68.0, 60.0, 84.0, 85.0, 75.0, nan...	110	259	17.739726
58	GarageType	object	[Attchd, Detchd, BuiltIn, CarPort, nan, Basmen...	6	81	5.547945
59	GarageYrBlt	float64	[2003.0, 1976.0, 2001.0, 1998.0, 2000.0, 1993....	97	81	5.547945
60	GarageFinish	object	[RFn, Unf, Fin, nan]	3	81	5.547945
63	GarageQual	object	[TA, Fa, Gd, nan, Ex, Po]	5	81	5.547945
64	GarageCond	object	[TA, Fa, nan, Gd, Po, Ex]	5	81	5.547945
32	BsmtExposure	object	[No, Gd, Mn, Av, nan]	4	38	2.602740
35	BsmtFinType2	object	[Unf, BLQ, nan, ALQ, Rec, LwQ, GLQ]	6	38	2.602740
30	BsmtQual	object	[Gd, TA, Ex, nan, Fa]	4	37	2.534247
31	BsmtCond	object	[TA, Gd, nan, Fa, Po]	4	37	2.534247
33	BsmtFinType1	object	[GLQ, ALQ, Unf, Rec, BLQ, nan, LwQ]	6	37	2.534247
26	MasVnrArea	float64	[196.0, 0.0, 162.0, 350.0, 186.0, 240.0, 286.0...	327	8	0.547945
42	Electrical	object	[SBrkr, FuseF, FuseA, FuseP, Mix, nan]	5	1	0.068493
0	Id	int64	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	1460	0	0.000000

3. Descriptive Statistics

The summary statistics for numerical variables provide insights into the data's distribution. Some key highlights:

- **Mean and median (50th percentile)** help understand the central tendency.
- **Min and max values** show the range, and any extreme outliers can be observed.

- **Standard deviation** measures the spread of the data, with certain features exhibiting high variability.

  - **Descriptive analysis for numerical data:**

	count	mean	std	min	25%	50%	75%	max
Id	1460.0	730.500000	421.610009	1.0	365.75	730.5	1095.25	1460.0
Dwell_Type	1460.0	56.897260	42.300571	20.0	20.00	50.0	70.00	190.0
LotFrontage	1201.0	70.049958	24.284752	21.0	59.00	69.0	80.00	313.0
LotArea	1460.0	10516.828082	9981.264932	1300.0	7553.50	9478.5	11601.50	215245.0
OverallQual	1460.0	6.099315	1.382997	1.0	5.00	6.0	7.00	10.0
OverallCond	1460.0	5.575342	1.112799	1.0	5.00	5.0	6.00	9.0
YearBuilt	1460.0	1971.267808	30.202904	1872.0	1954.00	1973.0	2000.00	2010.0
YearRemodAdd	1460.0	1984.865753	20.645407	1950.0	1967.00	1994.0	2004.00	2010.0
MasVnrArea	1452.0	103.685262	181.066207	0.0	0.00	0.0	166.00	1600.0
BsmtFinSF1	1460.0	443.639726	456.098091	0.0	0.00	383.5	712.25	5644.0
BsmtFinSF2	1460.0	46.549315	161.319273	0.0	0.00	0.0	0.00	1474.0
BsmtUnfSF	1460.0	567.240411	441.866955	0.0	223.00	477.5	808.00	2336.0
TotalBsmtSF	1460.0	1057.429452	438.705324	0.0	795.75	991.5	1298.25	6110.0
1stFlrSF	1460.0	1162.626712	386.587738	334.0	882.00	1087.0	1391.25	4692.0
2ndFlrSF	1460.0	346.992466	436.528436	0.0	0.00	0.0	728.00	2065.0
LowQualFinSF	1460.0	5.844521	48.623081	0.0	0.00	0.0	0.00	572.0
GrLivArea	1460.0	1515.463699	525.480383	334.0	1129.50	1464.0	1776.75	5642.0
BsmtFullBath	1460.0	0.425342	0.518911	0.0	0.00	0.0	1.00	3.0
BsmtHalfBath	1460.0	0.057534	0.238753	0.0	0.00	0.0	0.00	2.0
FullBath	1460.0	1.565068	0.550916	0.0	1.00	2.0	2.00	3.0
HalfBath	1460.0	0.382877	0.502885	0.0	0.00	0.0	1.00	2.0
BedroomAbvGr	1460.0	2.866438	0.815778	0.0	2.00	3.0	3.00	8.0
KitchenAbvGr	1460.0	1.046575	0.220338	0.0	1.00	1.0	1.00	3.0
TotRmsAbvGrd	1460.0	6.517808	1.625393	2.0	5.00	6.0	7.00	14.0
Fireplaces	1460.0	0.613014	0.644666	0.0	0.00	1.0	1.00	3.0
GarageYrBlt	1379.0	1978.506164	24.689725	1900.0	1961.00	1980.0	2002.00	2010.0
GarageCars	1460.0	1.767123	0.747315	0.0	1.00	2.0	2.00	4.0
GarageArea	1460.0	472.980137	213.804841	0.0	334.50	480.0	576.00	1418.0
WoodDeckSF	1460.0	94.244521	125.338794	0.0	0.00	0.0	168.00	857.0
OpenPorchSF	1460.0	46.660274	66.256028	0.0	0.00	25.0	68.00	547.0
EnclosedPorch	1460.0	21.954110	61.119149	0.0	0.00	0.0	0.00	552.0
3SsnPorch	1460.0	3.409589	29.317331	0.0	0.00	0.0	0.00	508.0
ScreenPorch	1460.0	15.060959	55.757415	0.0	0.00	0.0	0.00	480.0
PoolArea	1460.0	2.758904	40.177307	0.0	0.00	0.0	0.00	738.0
MiscVal	1460.0	43.489041	496.123024	0.0	0.00	0.0	0.00	15500.0
MoSold	1460.0	6.321918	2.703626	1.0	5.00	6.0	8.00	12.0
YrSold	1460.0	2007.815753	1.328095	2006.0	2007.00	2008.0	2009.00	2010.0
Property_Sale_Price	1460.0	180921.195890	79442.502883	34900.0	129975.00	163000.0	214000.00	755000.0

- **Descriptive analysis for Categorical data**:

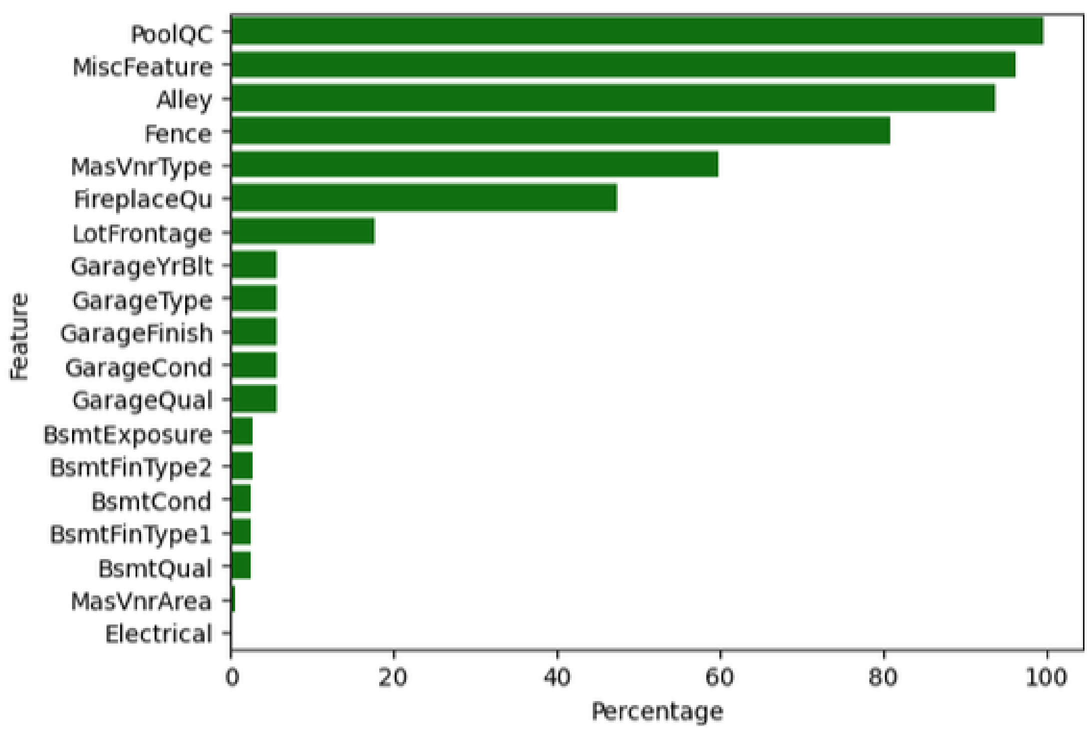
	count	unique	top	freq
Zone_Class	1460	5	RL	1151
Road_Type	1460	2	Pave	1454
Alley	91	2	Grvl	50
Property_Shape	1460	4	Reg	925
LandContour	1460	4	Lvl	1311
Utilities	1460	2	AllPub	1459
LotConfig	1460	5	Inside	1052
LandSlope	1460	3	Gtl	1382
Neighborhood	1460	25	NAmes	225
Condition1	1460	9	Norm	1260
Condition2	1460	8	Norm	1445
Dwelling_Type	1460	5	1Fam	1220
HouseStyle	1460	8	1Story	726
RoofStyle	1460	6	Gable	1141
RoofMatl	1460	8	CompShg	1434
Exterior1st	1460	15	VinylSd	515
Exterior2nd	1460	16	VinylSd	504
MasVnrType	588	3	BrkFace	445
ExterQual	1460	4	TA	906
ExterCond	1460	5	TA	1282
Foundation	1460	6	PConc	647
BsmtQual	1423	4	TA	649
BsmtCond	1423	4	TA	1311
BsmtExposure	1422	4	No	953
BsmtFinType1	1423	6	Unf	430
BsmtFinType2	1422	6	Unf	1256
Heating	1460	6	GasA	1428
HeatingQC	1460	5	Ex	741
CentralAir	1460	2	Y	1365
Electrical	1459	5	SBrkr	1334
KitchenQual	1460	4	TA	735
Functional	1460	7	Typ	1360
FireplaceQu	770	5	Gd	380
GarageType	1379	6	Attchd	870
GarageFinish	1379	3	Unf	605
GarageQual	1379	5	TA	1311
GarageCond	1379	5	TA	1326
PavedDrive	1460	3	Y	1340
PoolQC	7	3	Gd	3
Fence	281	4	MnPrv	157
MiscFeature	54	4	Shed	49
SaleType	1460	9	WD	1267
SaleCondition	1460	6	Normal	1198

4. Missing Data Analysis

Missing values were found in several features. Here's a breakdown:

- MasVnrArea and Electrical have missing values, affecting their usability.
- The basement-related features (BsmtCond, BsmtQual, BsmtFinType2, BsmtExposure) also show significant null percentages.
- Handling missing values for these critical features is important as they may impact model performance.

A bar plot was used to visualize the percentage of missing data for each feature, making it easier to identify columns with missing data.



5. Correlation Analysis (Numerical Variables)

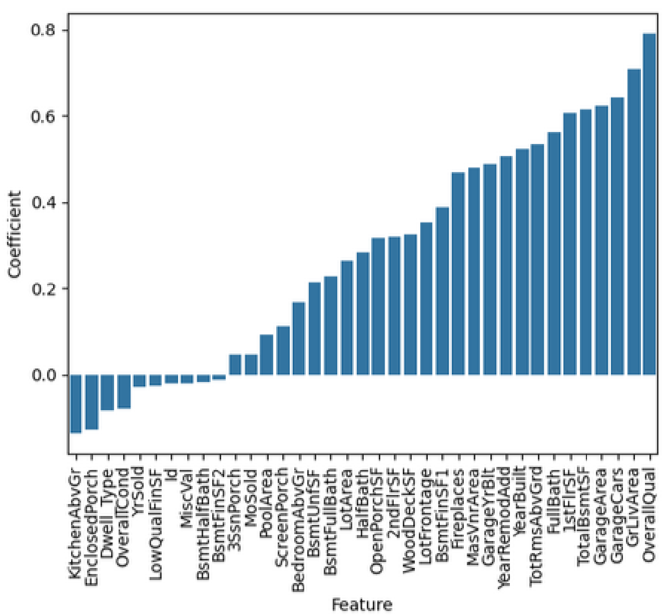
The correlation matrix was generated to analyze the relationships between numerical variables.

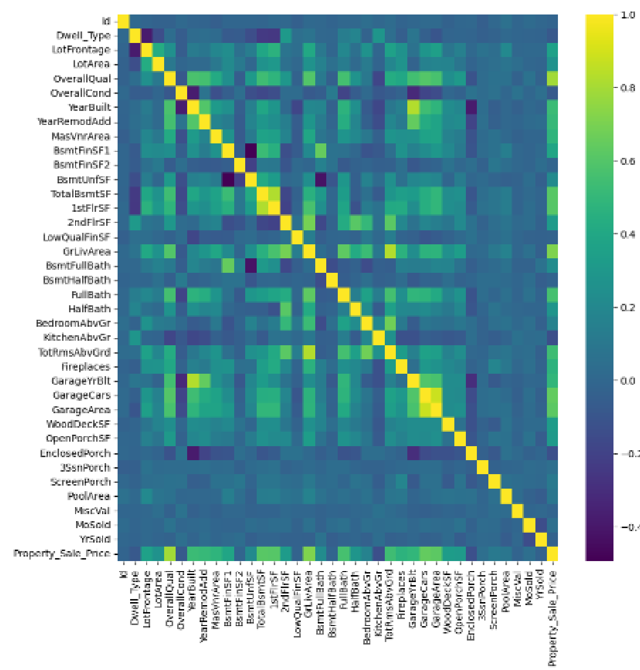
- **Highly correlated features with Property_Sale_Price:**

- Features like OverallQual, GrLivArea, and GarageCars show strong positive correlations with house prices.
- Conversely, features like EnclosedPorch and OverallCond have weaker correlations.

A **heatmap** was plotted to visualize the correlations between numerical variables. This revealed several strong relationships, guiding us toward feature importance for predictive modeling.

- **Top correlated features**: We used the absolute correlation matrix to filter the top 40 most correlated pairs, helping identify multicollinearity and important variables for the prediction model.



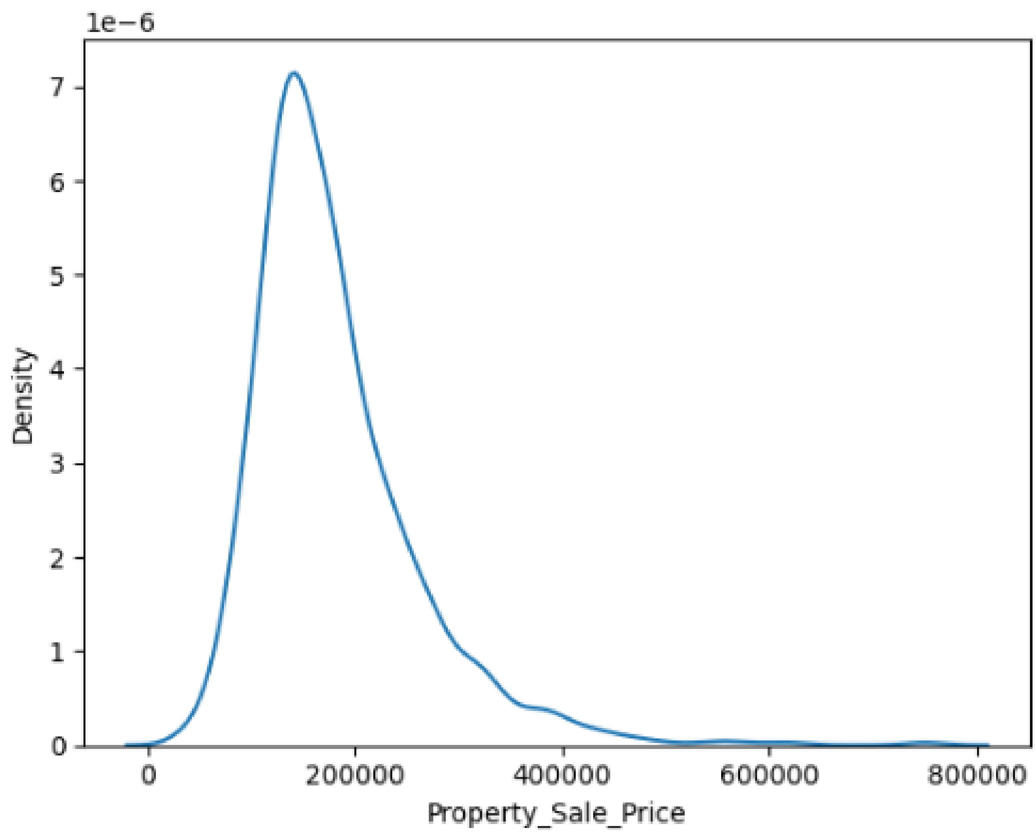



```
]: Property_Sale_Price    1.000000
   OverallQual            0.796088
   GrLivArea              0.693550
   GarageCars             0.648683
   GarageArea             0.631098
   TotalBsmtSF            0.609097
   1stFlrSF               0.599554
   FullBath               0.557775
   YearBuilt              0.534943
   TotRmsAbvGrd           0.533486
   YearRemodAdd           0.521141
   MasVnrArea             0.471845
   Fireplaces             0.463872
   BsmtFinSF1             0.373409
   LotFrontage            0.334225
   OpenPorchSF            0.324959
   WoodDeckSF             0.322209
   2ndFlrSF               0.296304
   HalfBath               0.282741
   GarageYrBlt            0.268311
   LotArea                0.265523
   BsmtFullBath           0.234306
   BsmtUnfSF              0.220669
   BedroomAbvGr           0.160533
   ScreenPorch            0.118327
   MoSold                 0.057045
   3SsnPorch              0.047416
   PoolArea               0.028630
```

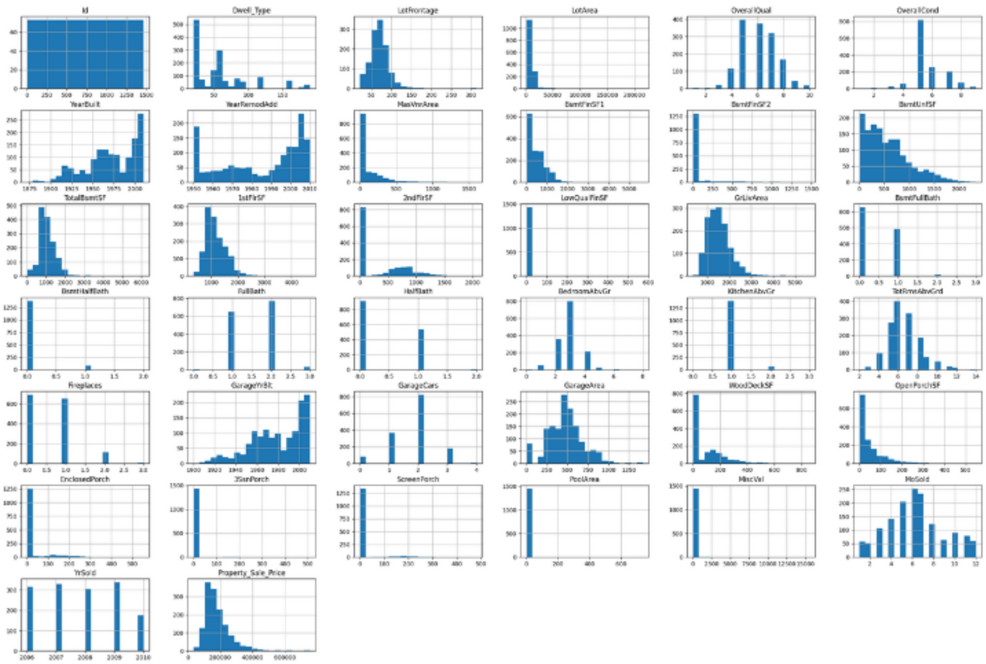
```
BsmtFinSF2          -0.008892
MiscVal             -0.021094
YrSold              -0.023725
LowQualFinSF        -0.025347
BsmtHalfBath        -0.036785
OverallCond         -0.080180
Dwell_Type          -0.088116
EnclosedPorch       -0.129757
KitchenAbvGr        -0.138839
Name: Property_Sale_Price, dtype: fl
```

6. Univariate Analysis

- **Numerical features:**
  - The distribution of the target variable (Property_Sale_Price) was visualized using a **histogram** and **kdeplot**. This revealed a positively skewed distribution, indicating that most houses are sold at lower prices, with fewer properties in higher price ranges.



- Other numerical variables such as LotFrontage, GarageYrBlt, and MasVnrArea were visualized using histograms to understand their distribution.



- **Categorical features:**
  - Each categorical feature was analyzed, showing the unique values and their counts.

```
Zone_Class
['RL' 'RM' 'C (all)' 'FV' 'RH']
Zone_Class
RL          1151
RM           218
FV            65
RH            16
C (all)       10
Name: count, dtype: int64
Road_Type
['Pave' 'Grvl']
Road_Type
Pave    1454
Grvl       6
Name: count, dtype: int64
Alley
[nan 'Grvl' 'Pave']
Alley
Grvl    50
Pave    41
Name: count, dtype: int64
Property_Shape
['Reg' 'IR1' 'IR2' 'IR3']
Property_Shape
Reg    925
IR1    484
IR2     41
IR3     10
Name: count, dtype: int64
```

```
LandContour
['Lvl' 'Bnk' 'Low' 'HLS']
LandContour
Lvl    1311
Bnk      63
HLS      50
Low      36
Name: count, dtype: int64
Utilities
['AllPub' 'NoSeWa']
Utilities
AllPub    1459
NoSeWa       1
Name: count, dtype: int64
LotConfig
['Inside' 'FR2' 'Corner' 'CulDSac' 'FR3']
LotConfig
Inside     1052
Corner      263
CulDSac      94
FR2          47
FR3           4
Name: count, dtype: int64
LandSlope
['Gtl' 'Mod' 'Sev']
LandSlope
Gtl    1382
Mod      65
Sev      13
Name: count, dtype: int64
```

```
Neighborhood
['CollgCr' 'Veenker' 'Crawfor' 'NoRidge' 'Mitchel' 'Somerst' 'NWAmes'
 'OldTown' 'BrkSide' 'Sawyer' 'NridgHt' 'NAmes' 'SawyerW' 'IDOTRR'
 'MeadowV' 'Edwards' 'Timber' 'Gilbert' 'StoneBr' 'ClearCr' 'NPkVill'
 'Blmngtn' 'BrDale' 'SWISU' 'Blueste']
Neighborhood
NAmes      225
CollgCr    150
OldTown    113
Edwards    100
Somerst     86
Gilbert     79
NridgHt     77
Sawyer      74
NWAmes      73
SawyerW     59
BrkSide     58
Crawfor     51
Mitchel     49
NoRidge     41
Timber      38
IDOTRR      37
ClearCr     28
SWISU       25
StoneBr     25
Blmngtn     17
MeadowV     17
BrDale      16
Veenker     11
NPkVill      9
Blueste      2
Name: count, dtype: int64
```

```
Condition1
['Norm' 'Feedr' 'PosN' 'Artery' 'RRAe' 'RRNn' 'RRAn' 'PosA' 'RRNe']
Condition1
Norm      1260
Feedr       81
Artery      48
RRAn        26
PosN        19
RRAe        11
PosA         8
RRNn         5
RRNe         2
Name: count, dtype: int64
Condition2
['Norm' 'Artery' 'RRNn' 'Feedr' 'PosN' 'PosA' 'RRAn' 'RRAe']
Condition2
Norm      1445
Feedr        6
Artery       2
RRNn         2
PosN         2
PosA         1
RRAn         1
RRAe         1
Name: count, dtype: int64
```

```
Dwelling_Type
['1Fam' '2fmCon' 'Duplex' 'TwnhsE' 'Twnhs']
Dwelling_Type
1Fam      1220
TwnhsE     114
Duplex      52
Twnhs       43
2fmCon      31
Name: count, dtype: int64
HouseStyle
['2Story' '1Story' '1.5Fin' '1.5Unf' 'SFoyer' 'SLvl' '2.5Unf' '2.5Fin']
HouseStyle
1Story    726
2Story    445
1.5Fin    154
SLvl       65
SFoyer     37
1.5Unf     14
2.5Unf     11
2.5Fin      8
Name: count, dtype: int64
RoofStyle
['Gable' 'Hip' 'Gambrel' 'Mansard' 'Flat' 'Shed']
RoofStyle
Gable      1141
Hip         286
Flat         13
Gambrel      11
Mansard       7
Shed          2
Name: count, dtype: int64
```

```
RoofMatl
['CompShg' 'WdShngl' 'Metal' 'WdShake' 'Membran' 'Tar&Grv' 'Roll'
 'ClyTile']
RoofMatl
CompShg    1434
Tar&Grv      11
WdShngl       6
WdShake       5
Metal         1
Membran       1
Roll          1
ClyTile       1
Name: count, dtype: int64
Exterior1st
['VinylSd' 'MetalSd' 'Wd Sdng' 'HdBoard' 'BrkFace' 'WdShing' 'CemntBd'
 'Plywood' 'AsbShng' 'Stucco' 'BrkComm' 'AsphShn' 'Stone' 'ImStucc'
 'CBlock']
Exterior1st
VinylSd    515
HdBoard    222
MetalSd    220
Wd Sdng    206
Plywood    108
CemntBd     61
BrkFace     50
WdShing     26
Stucco      25
AsbShng     20
BrkComm      2
Stone        2
AsphShn      1
ImStucc      1
CBlock       1
```

7. Bivariate Analysis

- **Sale Price vs Numerical Features:** A barplot showing the **correlation coefficients** between Property_Sale_Price and numerical features highlighted features like OverallQual, GrLivArea, and GarageCars as the most significant predictors for house prices.

- **Categorical Features vs Sale Price:** The Dash application allowed for interactive plotting of categorical features. A **count plot**, **scatter plot**, and **mean sale price plot** were implemented for deeper analysis:
  - Neighborhood, GarageType, and other categorical variables were analyzed for their impact on house prices.
  - Mean sale price per category helped understand which categories have higher average property prices.

8. Multivariate Analysis

By examining correlation between features

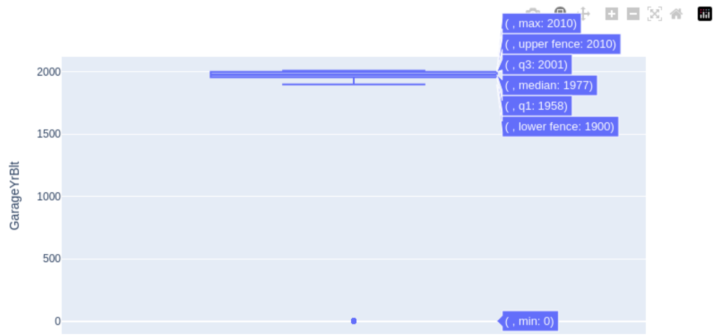
```
GarageArea          GarageCars           0.882475
GarageYrBlt         YearBuilt            0.825667
TotRmsAbvGrd        GrLivArea            0.825489
1stFlrSF            TotalBsmtSF          0.819530
OverallQual         Property_Sale_Price  0.790982
GrLivArea           Property_Sale_Price  0.708624
                    2ndFlrSF             0.687501
BedroomAbvGr        TotRmsAbvGrd         0.676620
BsmtFinSF1          BsmtFullBath         0.649212
GarageYrBlt         YearRemodAdd         0.642277
Property_Sale_Price GarageCars           0.640409
FullBath            GrLivArea            0.630012
Property_Sale_Price GarageArea           0.623431
TotRmsAbvGrd        2ndFlrSF             0.616423
TotalBsmtSF         Property_Sale_Price  0.613581
2ndFlrSF            HalfBath             0.609707
Property_Sale_Price 1stFlrSF             0.605852
OverallQual         GarageCars           0.600671
                    GrLivArea            0.593007
YearBuilt           YearRemodAdd         0.592855
dtype: float64
```

9. Outliers

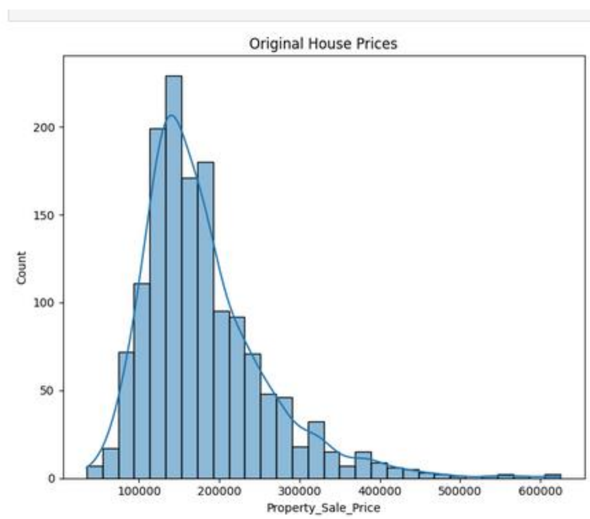
Outliers were identified using box plots, particularly in features like GrLivArea, OverallQual, and SaleCondition. These outliers could be influential in skewing the results of predictive models and may need to be addressed.

10. Feature Engineering Considerations

- Features such as GarageYrBlt features may need additional preprocessing due to their missing values and categorical nature.



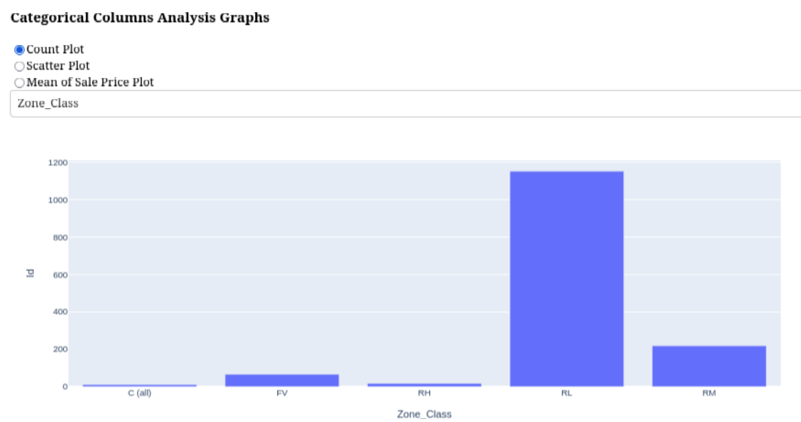
- Log transformation was performed on Property_Sale_Price to reduce skewness for model training.



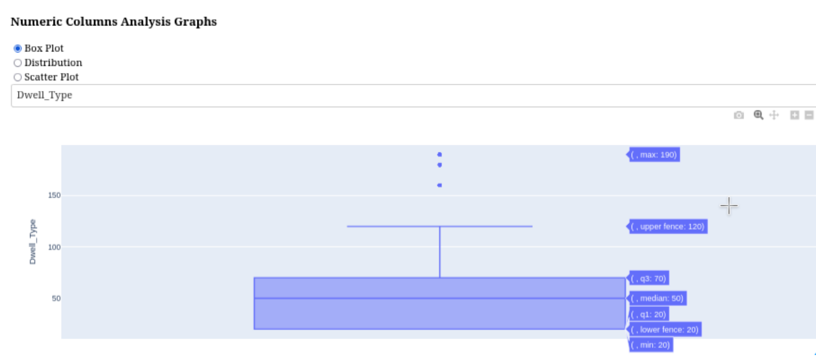
11. Visualizations Using Dash

To enable interactive analysis, a Dash application was built to visualize both categorical and numerical features:

- For categorical features, a count plot, scatter plot, and mean sale price plot were implemented to explore their distributions and relationships with sale prices.



- For numerical features, a box plot, distribution plot, and scatter plot were implemented to visualize the spread and relationships of various features with house prices.



12. Conclusions

The EDA reveals several key insights about house prices:

- **Strong predictive features**: OverallQual, GrLivArea, and GarageCars are strongly correlated with house prices and should be considered as primary features in any predictive model.
- **Missing values**: Critical features with missing data need to be handled properly before training the model.

Digital Egypt Pioneers

Ministry of Communications and Information Technology

Cleaning Data Report for House Price Prediction Project

Handling Outliers in the Dataset

Objective

In the context of house price prediction, outliers in key numerical features such as *OverallQual* and *GrLivArea* can skew the model's predictions and affect its performance. These outliers need to be treated effectively to ensure the dataset is well-prepared for model training. We developed a custom function to handle outliers flexibly and efficiently.

Methodology

To address outliers in the dataset, we utilized the **Interquartile Range (IQR)** method, which is effective for detecting values that fall significantly outside the typical range of a variable.

1. **IQR Calculation**:
   The IQR is the range between the 25th percentile (Q1) and the 75th percentile (Q3). Values that lie below Q1 by more than 1.5 times the IQR or above Q3 by more than 1.5 times the IQR are considered outliers. These are defined as:
   - **Lower Bound**: Q1−1.5×IQRQ1 - 1.5 \times IQRQ1−1.5×IQR
   - **Upper Bound**: Q3+1.5×IQRQ3 + 1.5 \times IQRQ3+1.5×IQR

Code Implementation

We developed a Python function, `handle_outlier`, to handle outliers using the IQR method. This function offers flexibility by allowing the user to either cap outliers or remove them based on the dataset's needs.

```
def handle_outlier(col,df):
  q1=df[col].quantile(0.25)
  q3=df[col].quantile(0.75)
  IQR=q3-q1
  lower_b=q1-1.5*IQR
  upper_b=q3+1.5*IQR
  for i in range(len(df)):
    if df.loc[i,col]>upper_b :  df.loc[i,col]=upper_b
    elif df.loc[i,col]<lower_b:  df.loc[i,col]=lower_b
handle_outlier('OverallQual',df)
handle_outlier('GrLivArea',df)
```

Conclusion

By using this outlier-handling approach, we effectively reduced the potential impact of extreme values in the dataset. This ensures the model can focus on learning meaningful patterns without being influenced by anomalies, leading to better prediction accuracy.

Handling Missing Values for Basement Features

Objective
To handle missing values in the basement-related categorical columns, we assigned a meaningful category to represent the absence of a basement in the dataset.

Methodology
The columns related to basement characteristics, including *BsmtQual*, *BsmtCond*, *BsmtExposure*, *BsmtFinType1*, and *BsmtFinType2*, had missing values. These missing values likely indicate the absence of a basement, so we replaced them with the category **'None'** to reflect this condition.

```
bsmt_str_cols = ['BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',
'BsmtFinType2']
df[bsmt_str_cols] = df[bsmt_str_cols].fillna('None')
```

Conclusion
By filling missing values with 'None', we ensured that the data accurately reflects the absence of basement features without introducing incorrect assumptions.

Handling Missing Values for Garage Features

Objective
To address missing values in garage-related categorical columns, we replaced the missing entries with a category that reflects the absence of a garage.

Methodology
The columns related to garage characteristics, including *GarageType*, *GarageFinish*, *GarageQual*, and *GarageCond*, had missing values. These missing values likely indicate that there is no garage for certain houses. Therefore, we replaced these missing values with the category **'None'** to signify the absence of a garage.

```
gar_str_cols = ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']
df[gar_str_cols] = df[gar_str_cols].fillna('None')
```

Conclusion
By imputing the missing values with 'None', we ensured the dataset accurately represents properties without garages, preventing any misinterpretation of missing data.

Handling Missing Values for Masonry Veneer Type

Objective
To handle missing values in the *MasVnrType* column, which indicates the type of masonry veneer used in a house, we assigned a value representing the absence of this feature.

Methodology
The *MasVnrType* column had missing values, likely indicating that no masonry veneer was used. To reflect this, the missing values were replaced with the category **'None'**.

```
df["MasVnrType"] = df["MasVnrType"].fillna("None")
```

Conclusion
By filling missing values with 'None', the data now correctly represents houses without masonry veneer, ensuring consistency in the dataset.

Handling Missing Values for Fireplace Quality

Objective
To address missing values in the *FireplaceQu* column, which represents the quality of a fireplace, we assigned a value that indicates the absence of a fireplace.

Methodology
The *FireplaceQu* column had missing values, likely indicating that a house does not have a fireplace. To capture this, the missing values were replaced with the category **'None'**.

```
df["FireplaceQu"] = df["FireplaceQu"].fillna("None")
```

Conclusion
By imputing the missing values with 'None', the dataset now properly reflects properties without fireplaces, maintaining consistency in the data.

Handling Missing Values for Electrical System

Objective
To ensure data completeness in the *Electrical* feature, which indicates the type of electrical system in a house, we removed rows with missing values for this feature.

Methodology
The *Electrical* column contained a small number of missing values. Since this feature is essential and categorical imputation might introduce incorrect assumptions, we opted to remove rows with missing values. This ensures that only rows with valid entries for the electrical system remain.

```
df = df.dropna(axis=0, subset=['Electrical'])
```

Conclusion
By dropping rows with missing *Electrical* values, we maintained the integrity of the dataset without introducing assumptions about unknown electrical systems.

Handling Missing Values for Garage Year Built

Objective
To handle missing values in the *GarageYrBlt* column, which represents the year a garage was built, we assigned a value that indicates the absence of a garage.

Methodology
The *GarageYrBlt* column had missing values, likely corresponding to houses without garages. Instead of imputing with a year, we replaced the missing values with **0** to clearly indicate that no garage exists for those properties.

```
df['GarageYrBlt'] = df['GarageYrBlt'].fillna(0)
```

Conclusion
By filling missing values with 0, we accurately represent properties without garages while ensuring the column remains numeric for further analysis.

Handling Missing Values for Lot Frontage

Objective
To address missing values in the *LotFrontage* column, which represents the linear feet of street-connected property, we imputed the missing values using the average *LotFrontage* for each neighborhood.

Methodology
The *LotFrontage* column contained missing values. Since lot sizes can vary by neighborhood, we used **group-based imputation** to fill missing values. Specifically, the mean *LotFrontage* for each neighborhood was calculated and used to impute the missing values, ensuring the imputation aligns with local property characteristics.

```
df['LotFrontage'] =
df.groupby('Neighborhood')['LotFrontage'].transform(lambda val:
val.fillna(val.mean()))
```

Conclusion
By using the neighborhood-specific mean to fill missing values, we ensured that the imputation reflects neighborhood characteristics, preserving the integrity of the *LotFrontage* feature for model training.

Handling Missing Values for Masonry Veneer Area

Objective
To handle missing values in the *MasVnrArea* column, which represents the area of masonry veneer in square feet, we assigned a value that indicates no masonry veneer.

Methodology
The *MasVnrArea* column had missing values, likely corresponding to houses without masonry veneer. To reflect this, the missing values were replaced with **0**, indicating that no masonry veneer is present.

```
df["MasVnrArea"] = df["MasVnrArea"].fillna(0)
```

Conclusion
By filling missing values with 0, we accurately captured properties without masonry veneer, ensuring consistency while keeping the column numeric for further analysis.

Handling Missing Values for Specific Entries in Masonry Veneer Area

Objective
To address missing values in the *MasVnrArea* column for specific entries, we filled them using the mean *MasVnrArea* value based on the corresponding *MasVnrType*.

Methodology
For the records with indices 688 and 1241, the *MasVnrArea* values were missing. Instead of filling these with a general value, we imputed the missing values with the mean *MasVnrArea* based on the respective *MasVnrType* (i.e., the type of masonry veneer). This ensures that the imputation aligns with the typical veneer area for each type.

```
df.loc[688, 'MasVnrArea'] = df_mean[df.loc[688, 'MasVnrType']]

df.loc[1241, 'MasVnrArea'] = df_mean[df.loc[1241, 'MasVnrType']]
```

Conclusion
By using the type-specific mean *MasVnrArea*, we provided a more accurate and contextually appropriate imputation for these specific missing values, preserving the integrity of the dataset.

Dropping Irrelevant or Sparse Columns

Objective
To streamline the dataset and remove columns with sparse or irrelevant information, we dropped several features that were not essential for predicting house prices.

Methodology
The following columns were dropped:

- **Id**: A unique identifier for each property, which does not contribute to the predictive power of the model.
- **PoolQC**, **MiscFeature**, **Alley**, **Fence**: These columns had a high percentage of missing values. Retaining these sparse features could introduce noise into the model without adding significant value.

```
df = df.drop(['Id', 'PoolQC', 'MiscFeature', 'Alley', 'Fence'], axis=1)
```

Conclusion
By dropping these columns, we reduced the complexity of the dataset and ensured that only relevant features are used for model training, enhancing model performance and interpretability.

Log Transformation of Property Sale Price

Objective
To normalize the distribution of the *Property_Sale_Price* column and reduce the effect of extreme values, we applied a natural logarithm transformation.

Methodology
The distribution of house prices (*Property_Sale_Price*) is typically right-skewed, with a few very high values. To address this skewness and make the data more suitable for modeling, we created a new column, **Property_Sale_Price_natural_log**, by applying the natural logarithm function. This transformation helps stabilize variance and makes the data more normally distributed, which can improve model performance.

```
df['Property_Sale_Price_natural_log'] = np.log(df['Property_Sale_Price'])
```

Conclusion
By applying a log transformation to the *Property_Sale_Price* column, we improved the data distribution, making it more suitable for regression-based predictive modeling.

Calculating Garage Age

Objective
To create a more meaningful feature for the model, we calculated the age of the garage by deriving it from the year it was built.

Methodology
Using the *GarageYrBlt* column (the year the garage was built), we computed a new feature called **GarageAge**. This feature represents the age of the garage in 2024 by subtracting the garage's construction year from the current year.

```
df['GarageAge'] = 2024 - df['GarageYrBlt']
```

Conclusion
By creating the *GarageAge* feature, we provided a more interpretable and relevant variable for modeling the influence of garage age on house prices, enhancing the dataset's predictive capabilities.

Dropping the Garage Year Built Column

Objective
After calculating the garage age, we removed the redundant *GarageYrBlt* column to simplify the dataset.

Methodology
The *GarageYrBlt* column was used to calculate the new feature *GarageAge*, which provides a more relevant and interpretable variable for modeling. Since *GarageYrBlt* is now redundant, it was dropped from the dataset to reduce complexity and avoid duplication.

```
df = df.drop(['GarageYrBlt'], axis=1)
```

Conclusion
By dropping *GarageYrBlt*, we streamlined the dataset and retained only the more meaningful *GarageAge* feature, which better contributes to the model.

One-Hot Encoding of Categorical Variables

Objective
To convert categorical features into a format suitable for machine learning algorithms, we applied one-hot encoding to the dataset.

Methodology
Categorical variables cannot be directly used in most machine learning models. To address

this, we applied **one-hot encoding**, which converts each category into a separate binary feature (0 or 1) for each unique category in the original column. This was achieved using the `pd.get_dummies()` function, ensuring that all categorical variables were transformed into numerical representations.

```
df = pd.get_dummies(df, dtype='int')
```

Conclusion
By applying one-hot encoding, we transformed all categorical features into numerical format, making the dataset ready for model training without losing any valuable categorical information.

Exporting the Cleaned Dataset

Objective
To preserve the cleaned and preprocessed dataset for future use, we exported it to a CSV file.

Methodology
After completing the data cleaning and preprocessing steps, the final dataset was saved to a CSV file named **cleaned_house_data.csv** using the `to_csv()` function. This ensures the processed data can be easily reloaded for analysis or model training.

```
df.to_csv('cleaned_house_data.csv')
```

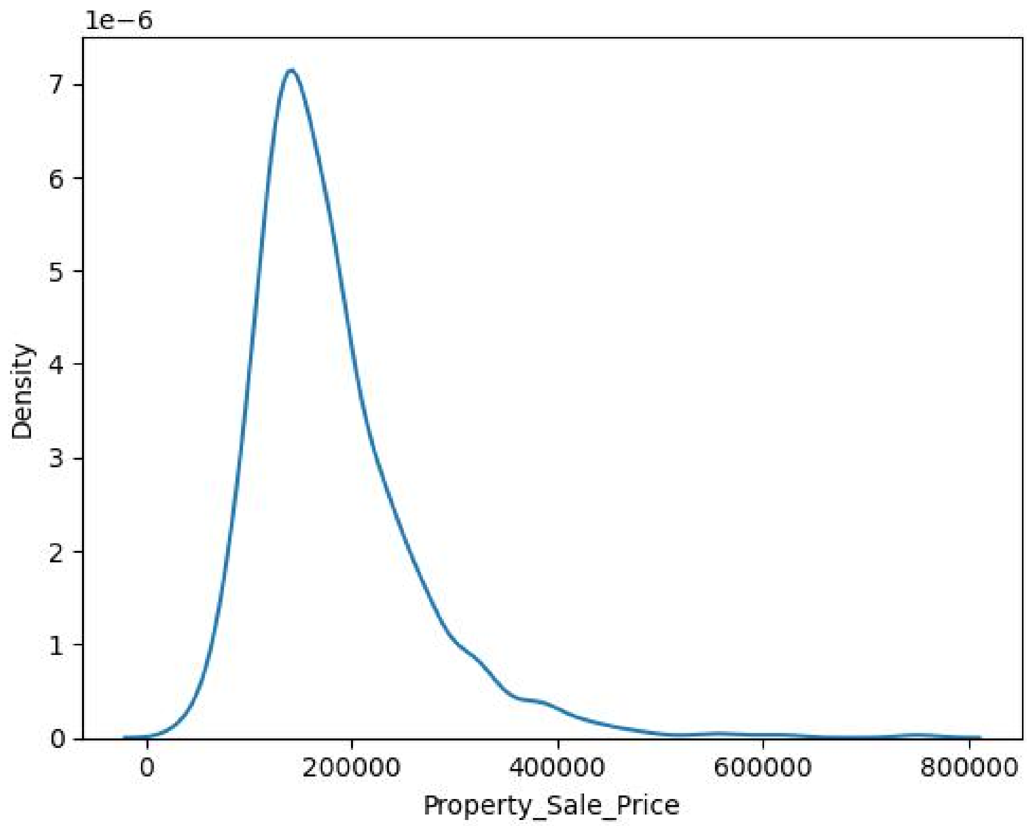
Conclusion
By exporting the dataset, we ensured that the cleaned data is stored in a reusable format, ready for further analysis and modeling tasks.

Modeling Report for House Price Prediction Project

The target variable "Property_Sale_Price":

```
sns.kdeplot(df['Property_Sale_Price'])
```

```
<Axes: xlabel='Property_Sale_Price', ylabel='Density'>
```



The target variable, "Property_Sale_Price," appears to be right-skewed, meaning that most of the property prices are concentrated towards the lower end, with a few higher-priced properties extending the distribution's tail. This skewness can affect the performance of regression models, particularly those that assume normally distributed residuals, like linear regression.

To handle this skewness, you might consider applying a transformation to the target variable, such as a log transformation (`log(Property_Sale_Price)`), to make the distribution more normal. This can help improve model performance and make the predictions more accurate for a wider range of property prices.

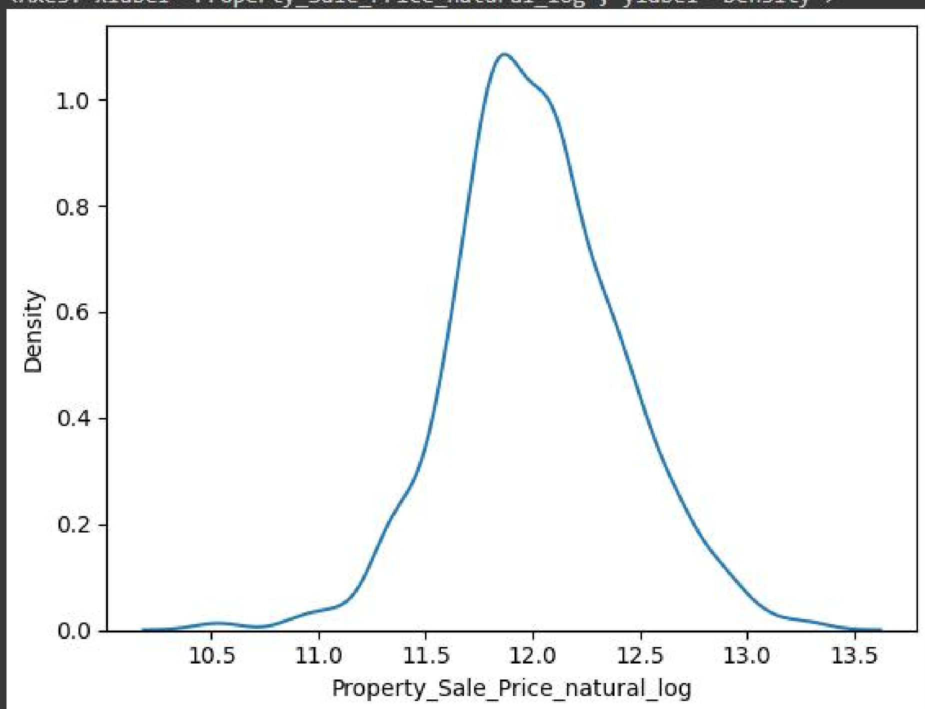
Trandsform Data

Use log Transform on Property_Sale_Price for better distribution

```
df['Property_Sale_Price_natural_log'] = np.log(df['Property_Sale_Price'])
```

```
[26] sns.kdeplot(df['Property_Sale_Price_natural_log'])
```

```
<Axes: xlabel='Property_Sale_Price_natural_log', ylabel='Density'>
```



We use **'Property_Sale_Price_natural_log':**

The log transformation is often used to reduce skewness of a measurement variable.

1- **AdaBoost Regressor:**

n_estimators: Controls the number of weak learners (decision trees by default). A higher number may improve performance but could also increase training time.(Impact on performance and trade-offs)

```
model = AdaBoostRegressor()
```

```
param_grid = {"n_estimators": [1, 40]}
```

```
grid = GridSearchCV(model,param_grid=param_grid, cv=5,
                    scoring='neg_mean_squared_error', n_jobs=-1)
```

Evaluation using original house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.82	19.54	13.682

Evaluation using Log Transform of house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.836	1.416	1.077

2- Decision Tree Regressor:

- `max_depth`: The maximum depth of the tree. Higher values increase model complexity.
- `min_samples_split`: The minimum number of samples required to split an internal node.
- `min_samples_leaf`: The minimum number of samples that must be present in a leaf node

```
model = DecisionTreeRegressor()
```

```
param_grid = {
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Evaluation using original house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.77	21.9	13.449

Evaluation using Log Transform of house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.836	1.415	1.04

3- **Gradient Boosting Regressor:**

- `n_estimators`: Number of boosting stages.
- `learning_rate`: Shrinks the contribution of each tree. Smaller values make the model more robust to overfitting but require more iterations.
- `max_depth`: Limits the depth of the trees to control overfitting.
- `min_samples_split`: Controls minimum samples required to split.
- `loss`: Loss function to be minimized.

```
]: model = GradientBoostingRegressor()
```

```
]: param_grid = {
       'n_estimators': [100, 200, 300],
       'learning_rate': [0.01, 0.1, 0.2],
       'max_depth': [3, 5],
       "min_samples_split": [5],
        "loss": ["squared_error"]
   }
```

Evaluation using original house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.899	14.67	9.29

Evaluation using Log Transform of house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.93	0.93	0.7

4- KNeighborsRegressor :

- `n_neighbors`: Number of neighbors to use for predictions. Smaller numbers can increase model variance, larger numbers reduce it.

```
model = KNeighborsRegressor()
```

```
param_grid = {'n_neighbors': range(1, 21)}
```

Evaluation using original house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.74	23.67	14.76

Evaluation using Log Transform of house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.75	1.75	1.23

5- **ElasticNet :**

- `alpha`: Regularization strength.
- `l1_ratio`: Mix between L1 (Lasso) and L2 (Ridge) regularization. 1 means Lasso, 0 means Ridge.

```
model = ElasticNet()
```

```
param_grid = {'alpha':[0.1,1,5,10,50,100],
              'l1_ratio':[.1, .5, .7, .9, .95, .99, 1]}
```

Evaluation using original house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.89	15.37	9.7

Evaluation using Log Transform of house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.92	0.96	0.7

6- RandomForestRegressor :

- `n_estimators`: Number of trees in the forest.
- `max_depth`: Limits the depth of the trees.
- `min_samples_split`: Minimum samples to split a node.

```
model = RandomForestRegressor()
```

```
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10]
}
```

Evaluation using original house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.88	15.82	9.82

Evaluation using Log Transform of house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.9	1.07	0.75

7- **Support Vector Regressor (SVR) :**

- `C`: Regularization parameter. Larger values give higher accuracy but risk overfitting.
- `kernel`: Kernel type (e.g., 'linear', 'rbf').
- `gamma`: Kernel coefficient for non-linear models.
- `degree`: Degree of the polynomial kernel function (used for 'poly' kernel).
- `epsilon`: Epsilon in the epsilon-SVR model.

```
svr = SVR()
```

```
param_grid = {'C':[100,200,300],
             'kernel':['linear','rbf'],
              'gamma':['scale','auto'],
              'degree':[1,2],
              'epsilon':[0.1,1,2,3]}
```

Evaluation using original house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.89	15.06	9.097

Evaluation using Log Transform of house price

r2-score	root_mean_squared_error % from mean	mean_absolute_error % from mean
0.93	0.93	0.63

Conclusion:

- **Gradient Boosting** and **Support Vector Regression** stand out as the best-performing models, showing lower error percentages and higher R^2 scores, which indicate better predictive accuracy.
- **Random Forest** and **Linear Regression** are also strong performers, offering a balance between simplicity and effectiveness.
- **Adaboost**, **Decision Tree**, and **K-Nearest Neighbors (KNN)** perform worse, with higher errors and lower R^2 scores, suggesting they may need further tuning or could be excluded from further analysis.

algorithm	house price	Log house price
Gradient Boosting	0.899	0.93
Support Vector Regression	0.89	0.93
Random Forest	0.88	0.9
Linear Regression	0.89	0.92
Adaboost	0.82	0.836
Decision Tree	0.77	0.836
KNN	0.74	0.75