International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

# Exploring the Paradigm Shift: Harnessing Data Analytics for Real - World Applications

**D. Diana Julie M. E[1], Abinesh Mathivanan[2]**

[1, 2]KIT – Kalaignar Karunanidhi Institute of Technology, Department of Artificial Intelligence and Data Science, Coimbatore, India

Email: dianajuliekit*[at]gmail.com*

Email: *abineshmathivanan31[at]gmail.com*

**Abstract:** ***Data handling and extraction are essential steps in any data science project. In Python, there are a variety of libraries that can be used to perform these tasks. This paper reviews the most popular libraries for handling and extracting datasets, including Pandas, NumPy, SciPy, Matplotlib, and scikit - learn. Pandas is a powerful library for data analysis and manipulation. It provides high - level data structures and tools for working with structured and time series data. NumPy is a library for scientific computing. It provides high - performance multidimensional arrays and a wide range of mathematical functions. SciPy is a collection of scientific computing tools that build on top of NumPy. It provides functions for linear algebra, numerical integration, and signal processing. Matplotlib is a library for creating visualizations. It provides a wide range of plotting functions for 2D and 3D data. The paper discusses the implementation of these libraries into a real world application and provides examples of how they can be used to handle and extract datasets. The paper also discusses how these libraries can be used to run simulation models of the data using ML Models. The paper concludes by discussing the future of data handling and extraction in Python, and how the development of new libraries is likely to continue to improve the tools available to data scientists.***

**Keywords**: Python, data handling, data extraction, Pandas, NumPy, SciPy, Matplotlib, scikit - learn, machine learning, simulation models

## 1. Introduction

In today's data - driven world, the ability to handle and process data effectively has become crucial for making informed decisions and driving meaningful outcomes. This paper aims to provide a comprehensive yet accessible overview of methods for handling and processing data sets using Python libraries, focusing on their practical implementation in real - life scenarios.

The presented techniques are designed to enable researchers, practitioners, and data enthusiasts to leverage the power of Python libraries in their data analysis endeavors. By exploring different methods of data handling and processing, we aim to bridge the gap between raw data and actionable insights, making the implementation of data sets in real - life applications more attainable.

Throughout the paper, we will delve into a real world application model which includes data cleaning, prepro - cessing, feature extraction, and transformation. We will showcase how Python libraries, such as scikit - learn, provide a wealth of tools and functions that simplify these processes. By utilizing these libraries, users can streamline the handling and processing of data sets, making them suitable for implementation in real - life scenarios.

The focus of this paper is to present the methods and techniques in a clear and accessible manner, ensuring that readers with varying levels of expertise can easily grasp and implement them. We will provide practical examples and step - by - step guidelines to facilitate understanding and promote practical application.

Overall, by providing an accessible overview of data handling and processing methods, we aim to empower readers to unlock the full potential of their data and drive meaningful outcomes in their respective fields.

## 2. Libraries for Data Handling and Manipulation

Python has several widely used libraries for data handling and manipulation. Among the most popular ones are:

**1) Numpy:** Numpy is a Python library commonly used for numerical and scientific computing. It provides a powerful and efficient way to manipulate large multidimensional arrays and matrices. A key feature of Numpy is the N - d array (N - dimensional array) object that allows efficient storage and manipulation of homogeneous data. Numpy's array manipulation routines, such as reshaping, slicing, and indexing, enable efficient data manipulation and extraction. These operations provide a convenient way to extract subsets of data, reorder elements, and perform transformations on arrays. Numpy arrays can also be easily combined, split, and stacked, allowing for flexible data manipulation and aggregation.

Moreover, Numpy integrates seamlessly with other Python libraries and tools, such as Matplotlib for data visualization and Pandas for data analysis. Numpy arrays can be directly used as input for plotting functions and can be easily converted to Pandas Data Frames for further data exploration and analysis.

Numpy's advanced features, such as linear algebra operations, Fourier transforms, and random number generation, make it a versatile tool for scientific computing, data analysis, and numerical simulations. These features enable users to perform complex computations and analyses with ease.

**2) Scipy:** Scipy offers a range of numerical algorithms for solving mathematical problems, including inte - gration techniques and optimization routines. Its integration modules provide efficient implementations of numerical integration and quadrature methods, enabling accurate computation of definite and indefinite in - tegrals. Scipy's optimization routines are useful in various applications, such as parameter estimation and model fitting.

It also includes modules for interpolation, enabling the generation of smooth curves and surfaces from sparse or irregularly spaced data points. Its interpolation methods include linear, polynomial, spline, and radial basis function interpolation, providing a variety of options for one - dimensional and multi - dimensional data.

Linear algebra operations are another strength of Scipy, with functions for solving systems of linear equa - tions, computing matrix decompositions, and performing matrix operations like inversion, determinant com - putation, and eigenvalue calculations. These capabilities make Scipy well - suited for tasks involving linear algebra, such as regression analysis and principal component analysis.

Furthermore, Scipy includes a comprehensive statistics module that provides numerous statistical functions and probability distributions. It allows for statistical analysis, hypothesis testing, descriptive statistics, and other statistical computations. The integration with Numpy arrays and other Scipy modules facilitates seam - less data handling and analysis within a consistent scientific computing environment.

**3) Matplotlib:** Matplotlib is a popular Python library often used for visualization and graph creation. It pro - vides a flexible and comprehensive set of functions for generating a variety of static, animated and interactive plots, diagrams and figures.

It allows you to create line charts, scatter plots, bar charts, histograms, pie charts, heat maps and many other types of visualizations. This library provides fine - grained control over many aspects of plotting, including axes, labels, titles, colors, line styles, markers, annotations, and more. This allows for precise tuning and design of visualizations to your specific requirements. It supports a variety of output formats, allowing users to save plots as image files (PNG, JPEG, SVG, etc.) or directly embed them in various file formats (PDF, LaTeX, etc.). This makes it ideal for producing publication - quality figures in research papers, reports, presentations, websites, etc.

In addition to its core functionality, Matplotlib seamlessly integrates with other libraries such as NumPy and Pandas to enable efficient data processing and manipulation. It can directly represent data from NumPy arrays, pandas dataframes, and other data structures, simplifying the process of visualizing data stored in these formats.

Matplotlib is well documented and offers an extensive collection of examples, tutorials and documentation. This allows users to easily get started, learn the capabilities of the library, and find solutions to specific plotting challenges.

**4) Scikit - learn:** Scikit - learn is a widely adopted and highly regarded Python library that provides a rich set of functionalities for machine learning tasks, data analysis, and predictive modeling. With its intuitive and consistent API, extensive algorithm collection, and efficient implementation, scikit - learn has become a go - to choice for researchers and practitioners in the field of machine learning.

One of the key strengths of scikit - learn lies in its comprehensive support for various stages of the machine learning pipeline. It offers a wide range of tools for data preprocessing, feature extraction, feature selection, and feature scaling, allowing users to effectively handle and manipulate datasets of different sizes and com - plexities. Additionally, scikit - learn provides robust implementations of diverse machine learning algorithms, including both supervised and unsupervised approaches, making it suitable for a broad range of applications. It seamlessly integrates with other popular Python libraries such as NumPy, SciPy, and Matplotlib, allowing for seamless data exchange, visualization, and advanced scientific computing capabilities.

The versatility of scikit - learn extends beyond traditional machine learning tasks. It also supports more ad - vanced techniques such as model evaluation, parameter tuning, and ensemble learning, facilitating the devel - opment of robust and accurate predictive models. Additionally, scikit - learn offers utilities for working with multi - class classification, text data, and imbalanced datasets, addressing various challenges encountered in real - world scenarios.

**5) Seaborn:** Seaborn is a popular Python library that enhances the capabilities of data visualization, build - ing upon the functionality provided by Matplotlib. With its intuitive and user - friendly interface, Seaborn simplifies the process of creating visually appealing and informative statistical graphics. It offers a diverse range of built - in themes, color palettes, and specialized plot types, allowing researchers and data scientists to effectively communicate complex information through visual representations.

One of the key advantages of Seaborn is its seamless integration with Pandas, a powerful data manipulation library. This integration enables users to effortlessly handle and analyze datasets, making data exploration and analysis more efficient. Moreover, Seaborn provides extensive support for multi - dimensional datasets, allowing researchers to uncover intricate relationships and patterns within their data.

The rich set of visualization options offered by Seaborn empowers researchers to generate compelling visualizations that effectively convey insights and facilitate data - driven decision - making. Whether it's exploring data distributions, examining relationships between variables, or visualizing statistical models, Seaborn pro - vides a versatile toolkit that enhances the visual exploration and understanding of data.

## 3. Real world Implementation with ML and Data Analysis

The below code is focused on performing Exploratory analysis on House Prices: Advanced Regression Techniques You could find the dataset for the Housing Prices in this link https: //www.kaggle. com/competitions/ house - prices - advanced - regression - techniques/data

We could start the analysis by importing the necessary libraries for Data Analysis and manipulation of the housing price:

```
i m p o r t numpy as np i m p o r t pandas as pd
import m a t p l o t l i b. p y p l o t as p l t i m p o r t s e a b o r n as s n s
from s c i p y i m p o r t s t a t s
from s c i p y. s t a t s i m p o r t norm, skew
from s k l e a r n. p r e p r o c e s s i n g i m p o r t S t a n d a r d S c a l e r i m p o r t os
i m p o r t w a r n i n g s
w a r n i n g s. f i l t e r w a r n i n g s (' i g n o r e ')

s n s. s e t s t y l e (" d a r k g r i d ")
p l t. s t y l e. use (" f i v e t h i r t y e i g h t ")
pd. pandas. s e t o p t i o n (' d i s p l a y. max columns ', None)
%m a t p l o t l i b i n l i n e
```

The above snippet sets the visual styles for plots using the "darkgrid" style for Seaborn and the "fivethirtyeight" style for Matplotlib. It also configures Pandas to display all columns in DataFrames and enables inline plotting in Jupyter Notebook.

```
t r a i n = pd. r e a d c s v (" t r a i n. csv ") t e s t = pd. r e a d c s v (" t e s t. csv ")
t r a i n. shape
t r a i n. columns
```

The above snippet reads, "train. csv" and "test. csv", and stores the data in separate Data Frames "train" and "test". The **train. shape** line returns the dimensions of the "train" DataFrame, while **train. columns** returns the column names of the "train" DataFrame, **train. head ()** displays a preview of the first few rows of the "train" DataFrame, providing a quick glimpse into the data's structure and values.

**Output:**

(1460, 81)

Index (['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street', 'Alley', 'LotShape', 'Land - Contour', 'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle', 'RoofMatl', 'Exte - rior1st', 'Exterior2nd', 'MasVnrType', 'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual', 'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC', 'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType', 'SaleCondition', 'SalePrice'], dtype='object')

Out of the 81 features, some may not contribute significantly to the prediction. After careful evaluation, it is observed that one additional feature **TotRmsAbvGrd**, could also play a crucial role in predicting the SalePrice. However, it is noted that the **Neighborhood** feature does not provide substantial information in this context. The key features to focus on are **OverallQual, YearBuilt, TotalBsmtSF, GrLivArea,** and **TotRmsAbvGrd.**

```
t r a i n [" S a l e P r i c e "]. d e s c r i b e ()
```

**Output:**

count 1460.000000
mean 180921.195890
std 79442.502883
min 34900.000000
25% 129975.000000
50% 163000.000000
75% 214000.000000
max 755000.000000
Name: SalePrice, dtype: float64

Based on the statistical summary of the feature, we can observe a noticeable skewness without the need for visualizing the histogram or KDE plot. The discrepancy between the mean and median values (25%) indicates the presence of skewness. Additionally, while 75% of the price values fall within the range of 35000 to 215000, the maximum value significantly deviates from this range. To validate our observation, we will plot the data and confirm our conclusion,

```
= s n s. d i s t p l o t (t r a i n. S a l e P r i c e)
```

**Output:**

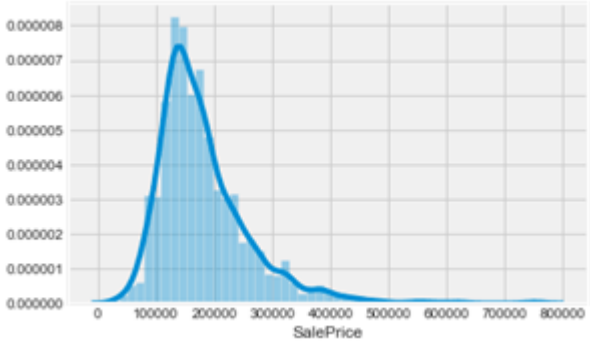


**Figure 1:** Simulation of Housing Sale Prices using Matplotlib

Our initial observation was accurate as the data exhibits a right - skewed distribution, with the tail extending towards the right - hand side. To quantify the extent of skewness, we

will further investigate the data,

```
print(f" Skewness: {train['SalePrice'].skew()}")
print(f" Kurtosis: {train['SalePrice'].kurt()}")
```

**Output:**

Skewness: 1.8828757597682129

Kurtosis: 6.536281860064529

***Relationship between Selected Features***

Relation with Numerical Variables

The below code snippet shows the scatter plot on the linear relationship between **SalePrice** and **GrLivArea** with a slight exponential relationship between **ToralBsmtSF**

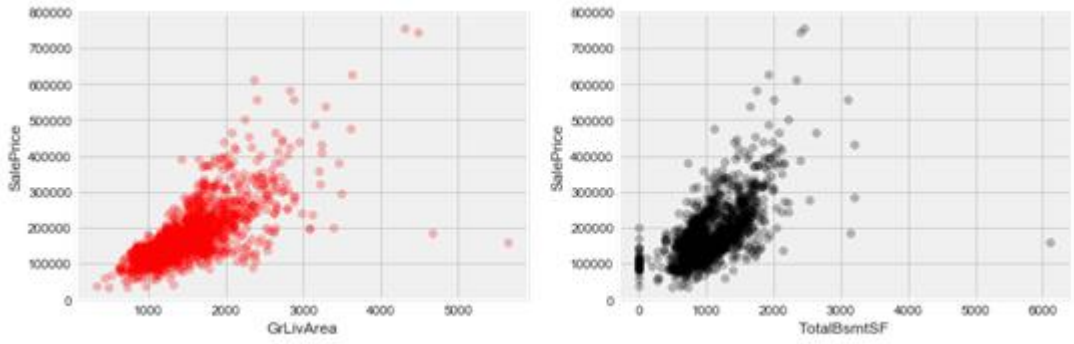


**Figure 2:** Scatter plot comparison between **SalePrice** and **GrLivArea**

**Relation with Categorial Variables**

The below snippet generates a boxplot relationship between the 'OverallQual' feature and the corresponding 'SalePrice' in the 'train' dataset. The boxplot is created by specifying 'OverallQual' as the x - axis variable and 'SalePrice' as the y - axis variable. The plot is then displayed with a restricted y - axis range of 0 to 800, 000 and it confirms that higher the quality, higher the price.

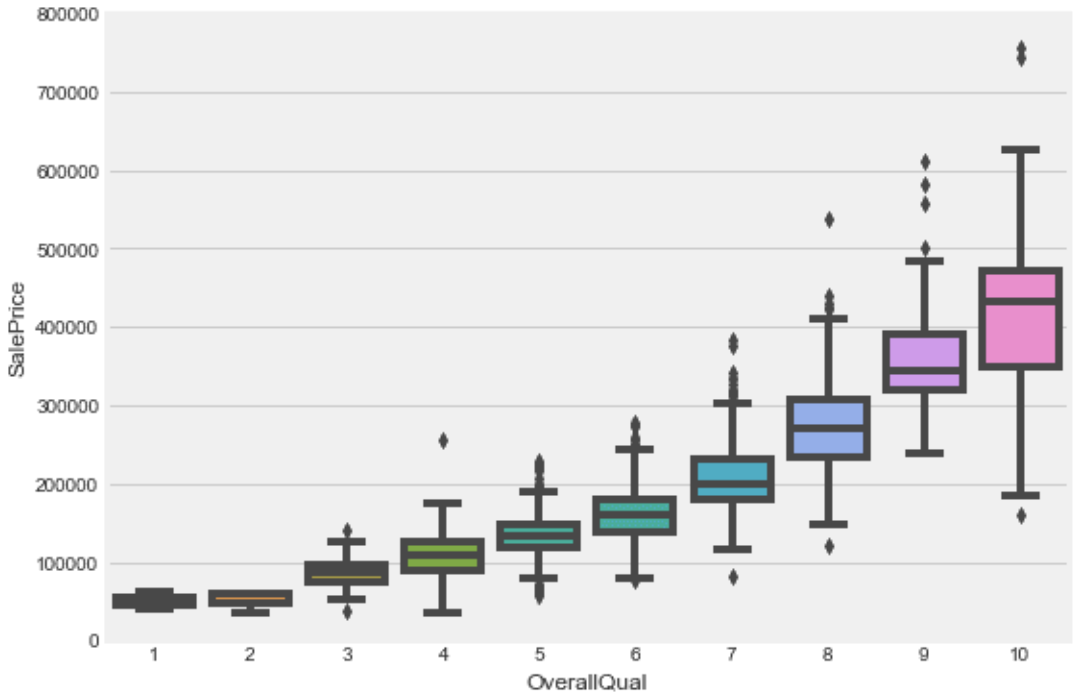


**Figure 3:** Boxplot comparison between **OverallQual** and **SalePrice**.

```
plt.figure(figsize = (20, 8))
var = 'YearBuilt'
data = pd.concat([train['SalePrice'], train[var]], axis = 1)
fig = sns.boxplot(x= var, y=" SalePrice", data = data)
fig.axis(ymin = 0, ymax = 800000)
= plt.xticks(rotation = 90)
```

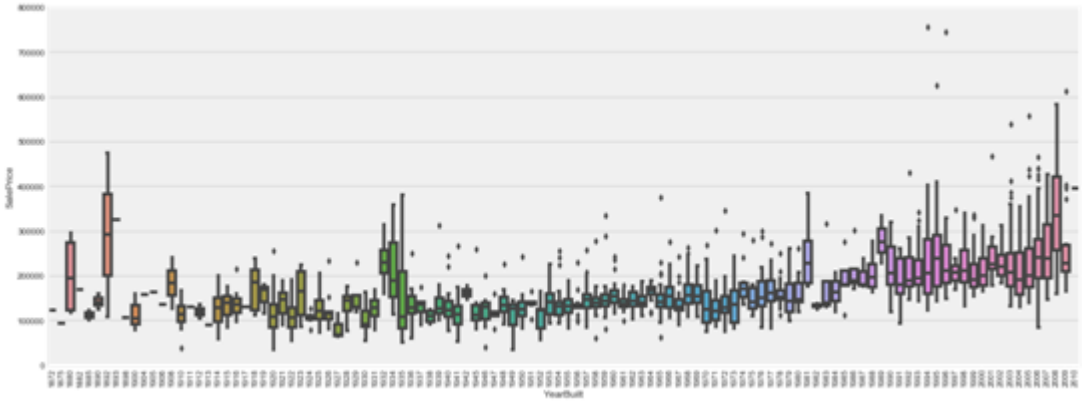

**Figure 4:** Boxplot comparison between **YearBuilt** and **SalePrice**. p l t. f i g u r e (f i g s i z e = (20, 6))

v a r = ' TotRmsAbvGrd ' p l t. s u b p l o t (1, 2, 1)
d a t a = pd. c o n c a t ([t r a i n [' S a l e P r i c e '], t r a i n [v a r]], a x i s = 1)
= s n s. b o x p l o t (x= var, y=" S a l e P r i c e ", d a t a = d a t a) p l t. a x i s (ymin = 0, ymax = 800000)
p l t. show ()

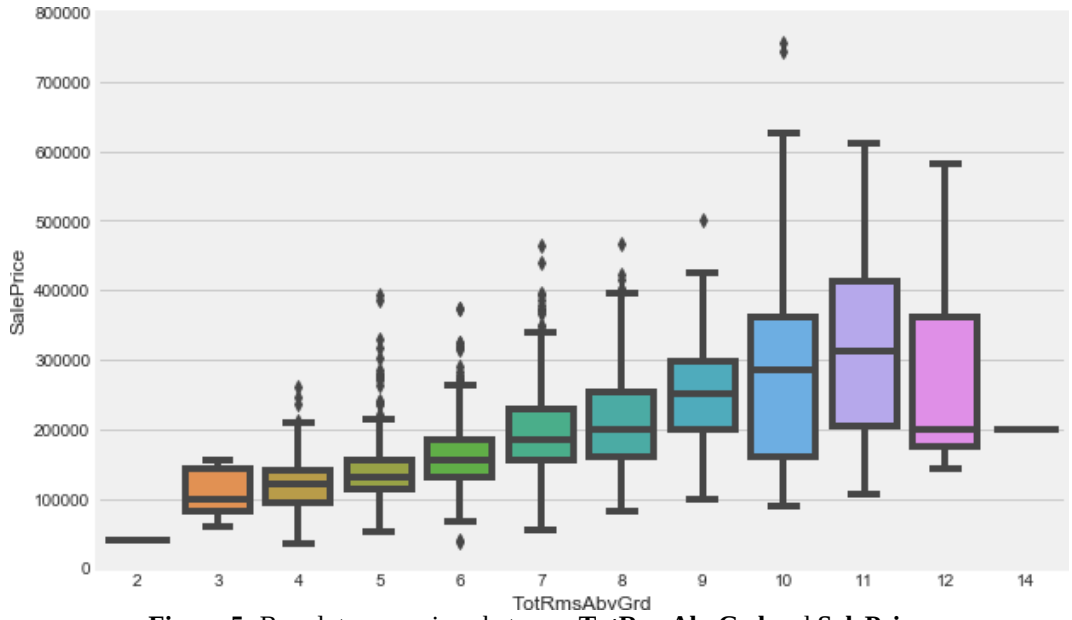


**Figure 5:** Boxplot comparison between **TotRmsAbvGrd** and **SalePrice**.

The variables **'GrLivArea'** and **'TotalBsmtSF'** show a positive linear relationship with **'SalePrice'**. Similarly, **'OverallQual'**, **'TotRmsAbvGrd'**, and **'YearBuilt'** also exhibit some correlation with **'SalePrice'**. Among these, the relationship is particularly strong for **'OverallQual'** as indicated by the box plot, showing an increase in sales prices with higher overall quality. The relationship is less pronounced for **'TotRmsAbvGrd'** compared to **'OverallQual'**.

***Analysis on Other Variables***

Let's have a look at all the available data subjectively,
p l t. f i g u r e (f i g s i z e = (18, 1 0))
c o r r m a t = t r a i n. drop (' Id ', 1). c o r r ()
= s n s. heatmap (corrmat, vmax = 1.0, s q u a r e = True, fmt = '.2 f ',
cmap= ' coolwarm ', annot kws ={ ' s i z e ': 8 })
p l t. show ()

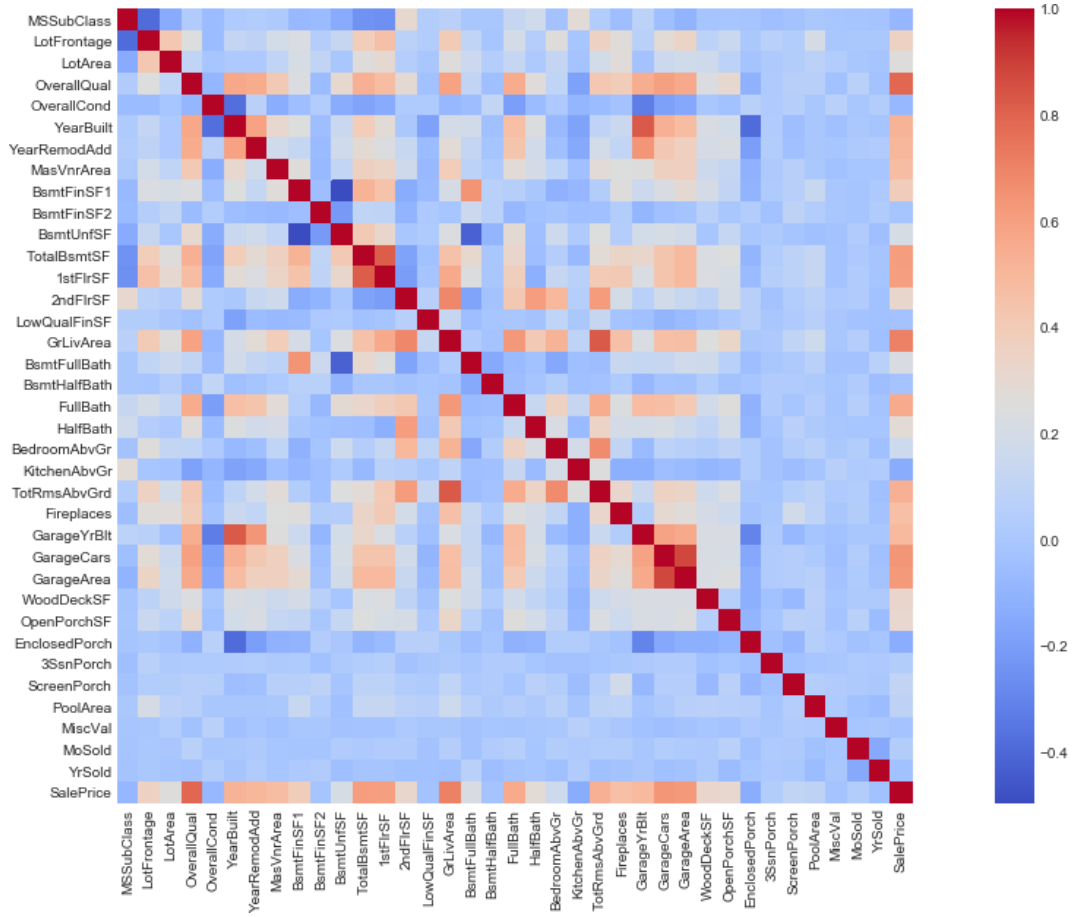


**Figure 6:** Heatmap visualization of the correlation matrix

As we can see, the above snippet generates a heatmap visualization of the correlation matrix of the dataset. The correlation matrix is calculated by excluding the 'Id' column from the 'train' dataset using the drop () function.

The heatmap reveals two notable correlations. The first is between **'TotalBsmtSF'** and **'1stFlrSF'**, indicating a strong relationship where both variables provide similar information, potentially leading to multicollinearity. The second correlation is observed among the **'GarageX'** variables. Additionally, we can observe significant correlations with **'SalePrice'** for **'GrLivArea'**, **'TotalBsmtSF'**, and **'OverallQual'**,

```
p l t. f i g u r e (f i g s i z e = (6, 6))

k = 10 # number of v a r i a b l e s f o r heatmap
c o l s = c o r r m a t. n l a r g e s t (k, ' S a l e P r i c e ') [' S a l e P r i c e ']. i n d e x cm = np. c o r r c o e f (t r a i n [c o l s]. v a l u e s. T)
s n s. s e t (f o n t s c a l e = 1.2 5)
hm = s n s. heatmap (cm, c b a r = True, a n n o t = True, s q u a r e = True, fmt = '.2 f ', cmap= ' coolwarm ', annot kws = ' s i z e ': 10, y t i c k l a b e l s = c o l s. v a l u e s, x t i c k l a b e l s = c o l s. v a l u e s) p l t. show ()
```

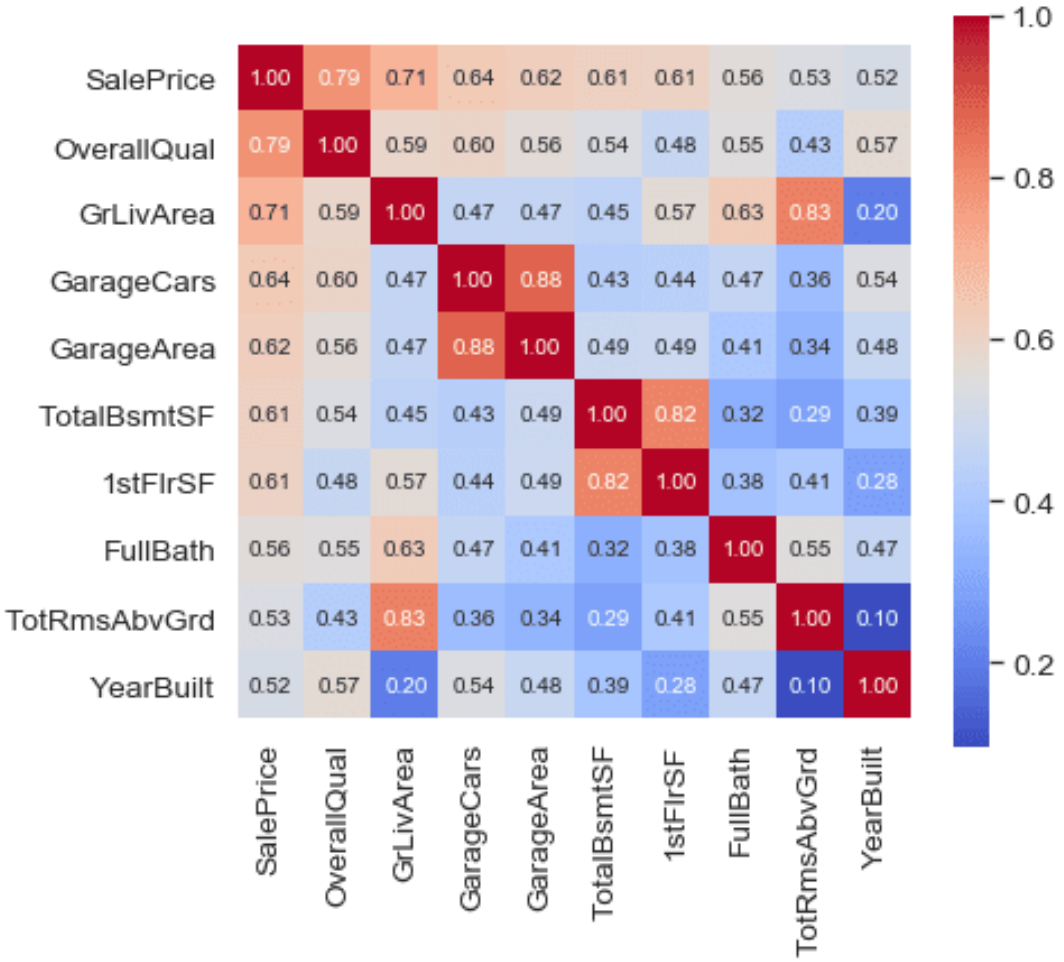


**Figure 7:** Heatmap of the correlation coefficients between the top 10 variables

The heatmap serves as a valuable tool for identifying such correlations and assisting in feature selection processes. It also highlights the relevance of other variables that should be considered in our analysis.

```
p l t. f i g u r e (f i g s i z e = (10, 8)
) s n s. s e t ()
c o l s = [' S a l e P r i c e ', ' O v e r a l l Q u a l ', ' Gr Liv Area ',
' Garage Cars ', ' Total Bsmt SF ', ' F u l l B a t h ', ' Y e a r B u i l t ']
= s n s. p a i r p l o t (t r a i n [c o l s], s i z e = 2.5, d i a g k i n d = ' kde ') p l t. show ()
```

The above snippet produces a pair plot that includes variables such as **'SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath', and 'YearBuilt'** and shows the relationship between them.

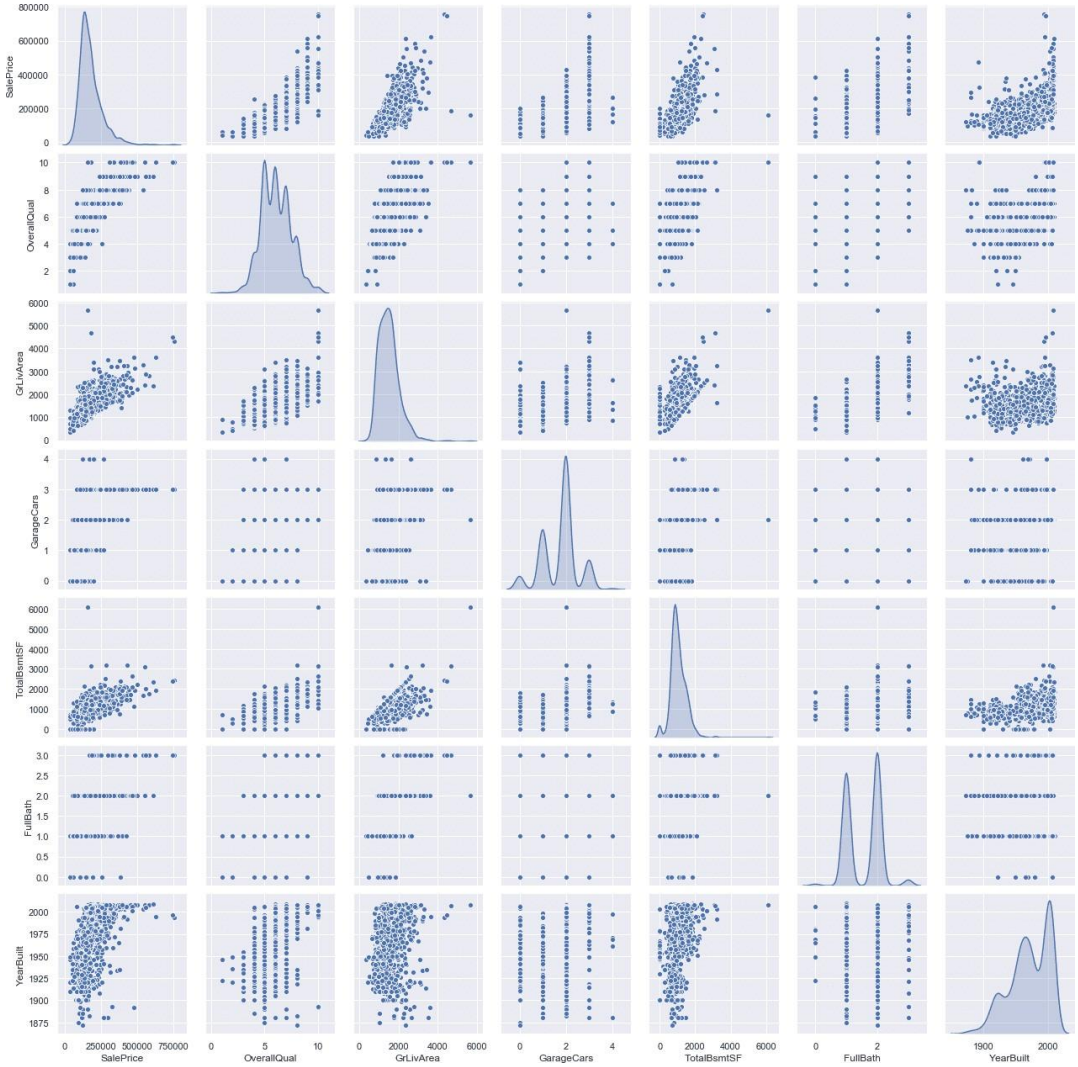


**Figure 8:** Pair plot comparing the necessary variables

*Missing Variables*

```
features with na = [features for features in train.columns if train[features].isnull().sum() >= 1]

a = pd.DataFrame(
'features': features with na,
'Total': [train[i].isnull().sum() for i in features with na], 'Missing PCT': [np.round(train[i].isnull().sum() /
train.shape[0], 4) for i in features with na]
).sort values(by = 'Missing PCT', ascending = False).reset index(drop = True
) a.style.background gradient(cmap='Reds')
```

We've now created a DataFrame 'a' that summarizes features in the 'train' dataset with missing values which calculates the total count and percentage of missing values for each feature and sorts them in descending order. The resulting DataFrame is styled with a red color gradient, indicating the extent of missing values in each feature.

| | features | Total | Missing_PCT |
|---|---|---|---|
| 0 | PoolQC | 1453 | 0.995200 |
| 1 | MiscFeature | 1406 | 0.963000 |
| 2 | Alley | 1369 | 0.937700 |
| 3 | Fence | 1179 | 0.807500 |
| 4 | FireplaceQu | 690 | 0.472600 |
| 5 | LotFrontage | 259 | 0.177400 |
| 6 | GarageType | 81 | 0.055500 |
| 7 | GarageYrBlt | 81 | 0.055500 |
| 8 | GarageFinish | 81 | 0.055500 |
| 9 | GarageQual | 81 | 0.055500 |
| 10 | GarageCond | 81 | 0.055500 |
| 11 | BsmtExposure | 38 | 0.026000 |
| 12 | BsmtFinType2 | 38 | 0.026000 |
| 13 | BsmtFinType1 | 37 | 0.025300 |
| 14 | BsmtCond | 37 | 0.025300 |
| 15 | BsmtQual | 37 | 0.025300 |
| 16 | MasVnrArea | 8 | 0.005500 |
| 17 | MasVnrType | 8 | 0.005500 |
| 18 | Electrical | 1 | 0.000700 |

**Figure 9:** Table showing the percentage of missing values

We have several options for handling features with missing values. Features with more than 50% missing values can be dropped since they may not provide significant information. For the **'GarageX'** and **'BSMTX'** variables, since they are

related to garages and basements respectively, and we already have important information captured by other variables, we can consider dropping or filling them with appropriate values. Similarly, **'MasVnrArea'** and **'MasVnrType'** can be considered non - essential due to their correlation with **'YearBuilt'** and **'OverallQual'**. Lastly, for the **'Electrical'** feature with just one missing value, we can either remove the corresponding row or fill it with the mode value.

In this dataset, we're replacing the missing values of the above variables with, "None", "Not available" and "No garage".

A grid of bar plots are generated where each bar plot corresponds to a feature that contains missing values in the dataset. The plots illustrate the median sale price for observations where the feature information is present versus those where it is missing. The presence or absence of a specific feature value can be visually compared to assess its impact on the sale price. The bar plots are accompanied by informative titles and properly labeled axes, facilitating clear interpretation of the results. The resulting visualizations serve as a valuable tool for understanding the relationship between missing data in certain features and their influence on the sale price.

```
p l t. f i g u r e (f i g s i z e = (20, 2 0))
f o r i, f e a t u r e i n enumerate (f e a t u r e s w i t h n a, 1):
p l t. s u b p l o t (5, 5, i)
d a t a = t r a i n. copy ()
d a t a [f e a t u r e] = np. where (d a t a [f e a t u r e]. i s n u l l (), 1, 0) temp = d a t a. groupby (f e a t u r e) [' S a l e P r i c e ']. median ()
= s n s. b a r p l o t (x= temp. index, y= temp. v a l u e s) p l t. t i t l e (f e a t u r e)
p l t. x l a b e l (" ")
p l t. x t i c k s ([0, 1], [" P r e s e n t ", " Missing "]) p l t. y l a b e l (" S a l e s P r i c e ", r o t a t i o n = 90)
p l t. t i g h t l a y o u t (h pad = 2, w pad = 2) p l t. show ()
```

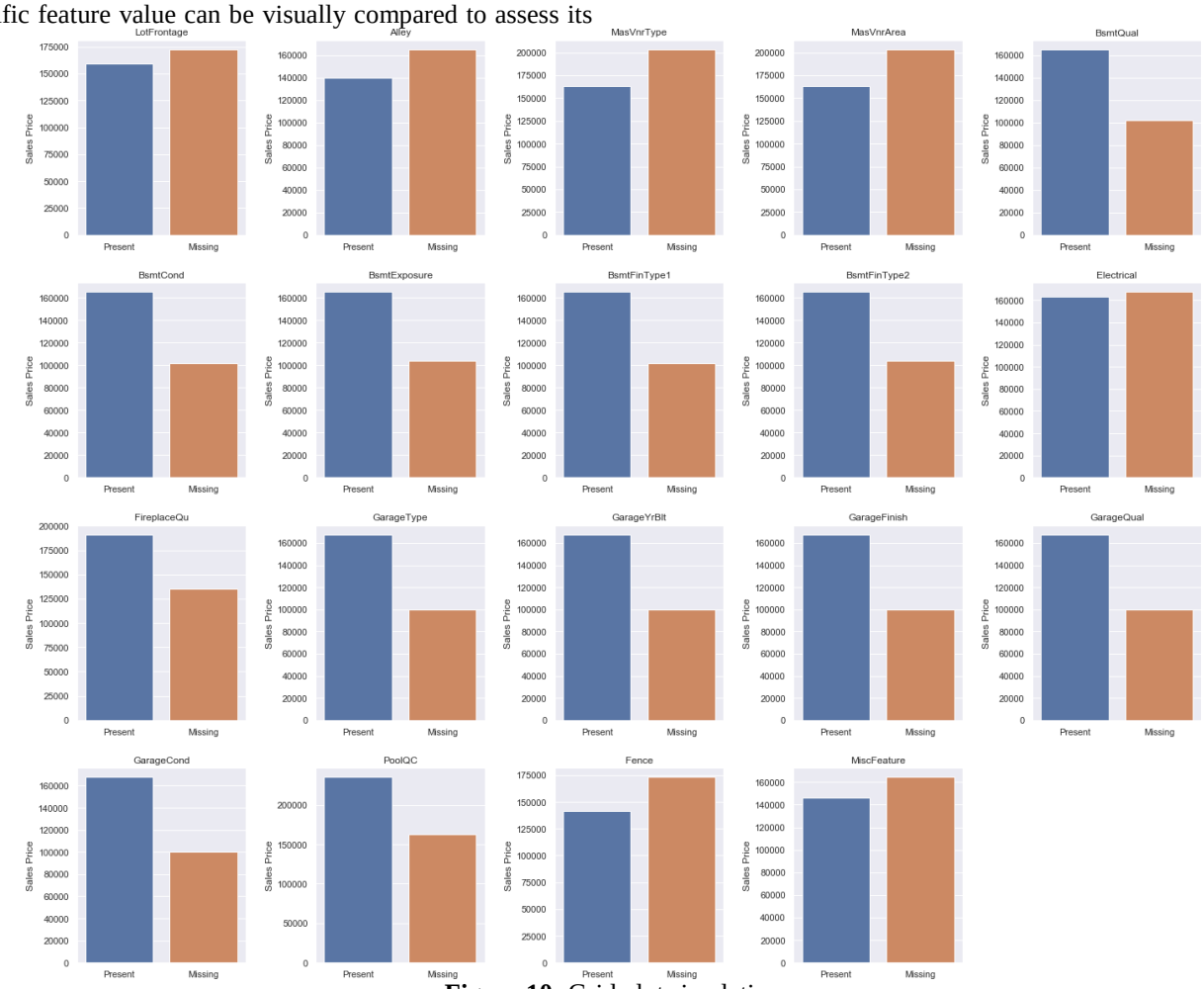


**Figure 10:** Grid plot simulation

***Numerical Variables***

Discrete Numerical Values:

Discrete numerical values are specific numeric data points that are separate and distinct, typically represented by whole numbers or integers. They are countable and finite, lacking intermediate values or decimals.

```
d i s c r e t e f e a t u r e s
```

**Output:**

['MSSubClass', 'OverallQual', 'OverallCond', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'Bed - roomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'PoolArea', 'MoSold']

Now, we've written a code that creates a grid of subplots to visualize the relationship between discrete features and the sale price. It iterates through each feature in the list of

discrete features and plots a bar chart showing the median sale price for each category of the feature. The data used for plotting is a copy of the training dataset. The resulting plot allows for a comparison of how different discrete features are associated with the sale price, providing insights into their impact on the target variable.


**Figure 11:** Relationship between Discrete Values and Sales Price The discrete numerical features exhibit varying effects on the **SalePrice**:

- Some features show a positive correlation, where an increase in feature value corresponds to higher SalePrice.
- Certain features have specific values that result in significantly higher or lower SalePrice.
- Several features have minimal variation in SalePrice across different feature values.
- Overall, the impact of most features on SalePrice is not immediately apparent, highlighting the need for a feature selection process to identify the most influential factors.

**Continuous Numerical Values**

Continuous numerical values in a dataset represent measurements or variables that can take on any value within a specific range. Unlike discrete numerical values, which have distinct categories or intervals, continuous numerical values can vary infinitely and exhibit a smooth progression.

p r i n t (f ” Number of C o n t i n u o u s n u m e r i c a l f e a t u r e: { l e n (c o n t i n u o u s n u m f e a t u r e s) } ”) c o n t i n u o u s n u m f e a t u r e s

**Output:**

Number of Continuous numerical feature: 19

[’LotFrontage’, ’LotArea’, ’MasVnrArea’, ’BsmtFinSF1’, ’BsmtFinSF2’, ’BsmtUnfSF’, ’TotalBsmtSF’, ’1stFlrSF’, ’2ndFlrSF’, ’LowQualFinSF’, ’GrLivArea’, ’GarageArea’, ’WoodDeckSF’, ’OpenPorchSF’, ’En - closedPorch’, ’3SsnPorch’, ’ScreenPorch’, ’MiscVal’, ’SalePrice’]

To visualize the distributions of continuous numerical features in the dataset: p l t. f i g u r e (f i g s i z e = (20, 2 0))

f o r i, f e a t u r e i n enumerate (c o n t i n u o u s n u m f e a t u r e s, 1): p l t. s u b p l o t (5, 4, i)

= s n s. d i s t p l o t (t r a i n [f e a t u r e], kde= F a l s e, rug = True) p l t. t i g h t l a y o u t (h pad = 2, w pad = 2)

p l t. show ()

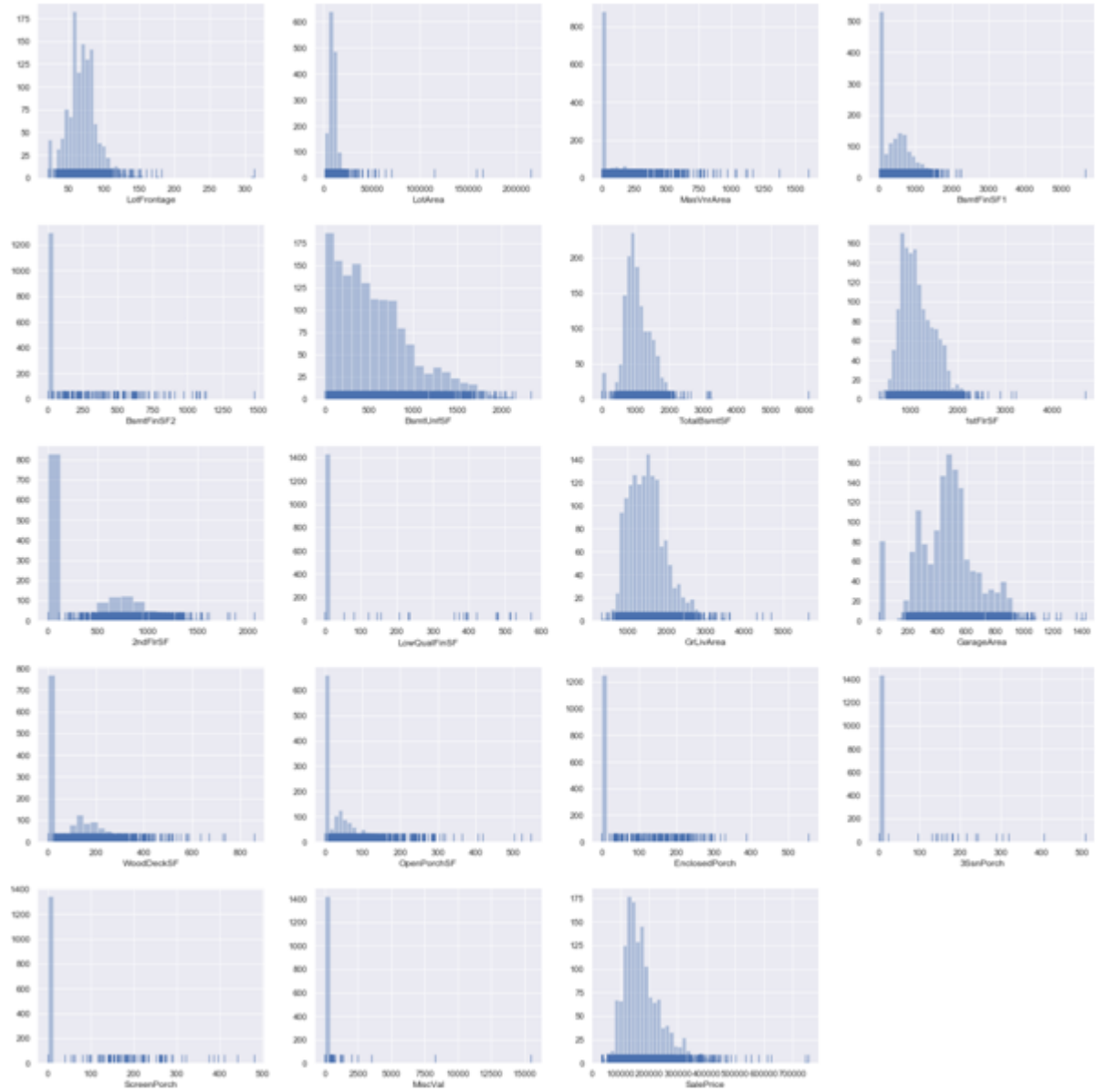


**Figure 12:** Simulation of Continuous Numerical Variables

The generated subplots reveal that the majority of continuous features do not exhibit a normal distribution.

Additionally, several features display significant skewness, indicating the need for data transformation to address this issue.

**Logarithmic Transformation**

Now we've to apply a log transformation to the continuous features to plot their correlation with the logarithm of **SalePrice**, and to visualize their distribution after transformation. This analysis helps to evaluate the relationship between the transformed features and the target variable, as well as the effectiveness of the log transformation in achieving a more normal distribution.

```
d a t a = t r a i n. copy ()

s a l e p r i c e = np. l o g (t r a i n [' S a l e P r i c e '])

f o r i, f e a t u r e i n enumerate (c o n t i n u o u s n u m f e a t u r e s [: − 1], 1)
: d a t a = t r a i n [f e a t u r e]. copy ()

i f 0 i n d a t a. u n i q u e (): # as l o g 0 i s u n d e f i n e d z c o n t i n u e
e l s e:
d a t a = np. l o g (d a t a
) d a t a. name = f e a t u r e
= p l t. f i g u r e (f i g s i z e = (15, 8)) p l t. s u b p l o t (1, 2, 1)
s n s. r e g p l o t (data, s a l e p r i c e , f i t r e g = True, s c a t t e r k w s = " c o l o r ": " b l a c k ",
l i n e k w s = " c o l o r ": " r e d "). s e t t i t l e (f " C o r r e l a t i o n: d a t a. c o r r (s a l e p r i c e) ")
p l t. s u b p l o t (1, 2, 2)
s n s. d i s t p l o t (d a t a). s e t t i t l e (f " Log t r a n s f o r m a t i o n of: f e a t u r e ") p l t. show ()
```

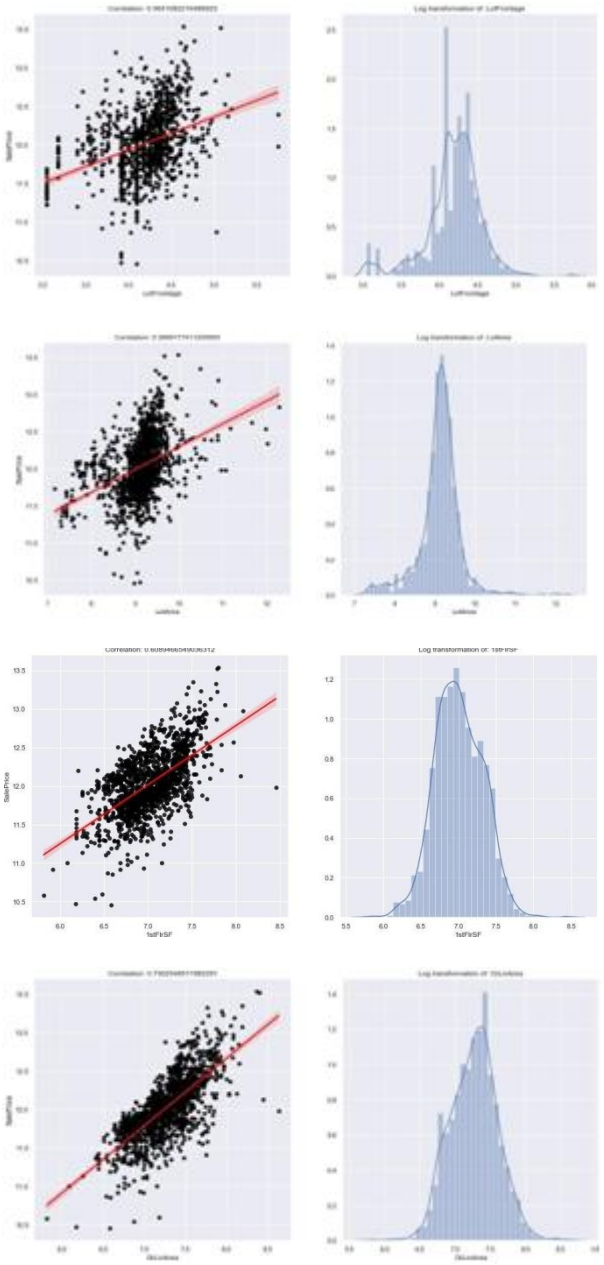

**Figure 13:** Logarithmic comparison between continuous values and **SalePrice**

The log transformation revealed interesting insights. While **LotArea** showed limited correlation with **SalePrice, GrLivArea and 1stFlrSF** exhibited strong correlations. However, the presence of zero values prevented log transformation for some features. Further analysis and preprocessing are necessary to address these irregularities and enhance the correlation between the features and the target variable

**Categorial Variables**

Categorical variables provide insights into the relationship between factors and SalePrice. Analyzing their distribution, impact, and handling of missing values helps identify influential categories and guides feature selection and modeling decisions for accurate predictions.

```
c a t e g o r i c a l f e a t u r e s = [f e a t u r e f o r f e a t u r e
i n t r a i n. columns i f t r a i n [f e a t u r e]. d t y p e s == 'O']
p r i n t (f " Number of c a t e g o r i c a l f e a t u r e: { l e n
(c a t e g o r i c a l f e a t u r e s) }")
```

**Output:**

Number of categorical feature: 43

['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Neigh - borhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtEx - posure', 'BsmtFinType1', 'BsmtFinType2', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual', 'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature', 'SaleType', 'SaleCondition']

To visualize the relationship between Categorial Variables and **SalePrice**

```
p l t. f i g u r e (f i g s i z e = (40, 3 0))
f o r i, f e a t u r e i n enumerate (c a t e g o r i c a l f e a t u r
e s, 1): d a t a = t r a i n. copy ()
temp = d a t a. groupby (f e a t u r e) [' S a l e P r i c e ']. median () p l t. s u b p l o t (9, 5, i)
s n s. b a r p l o t (temp. index, temp. v a l u e s) p l t. x t i c k s (r o t a t i o n = 45)
p l t. t i g h t l a y o u t (h pad = 1.2) p l t. show ()
```

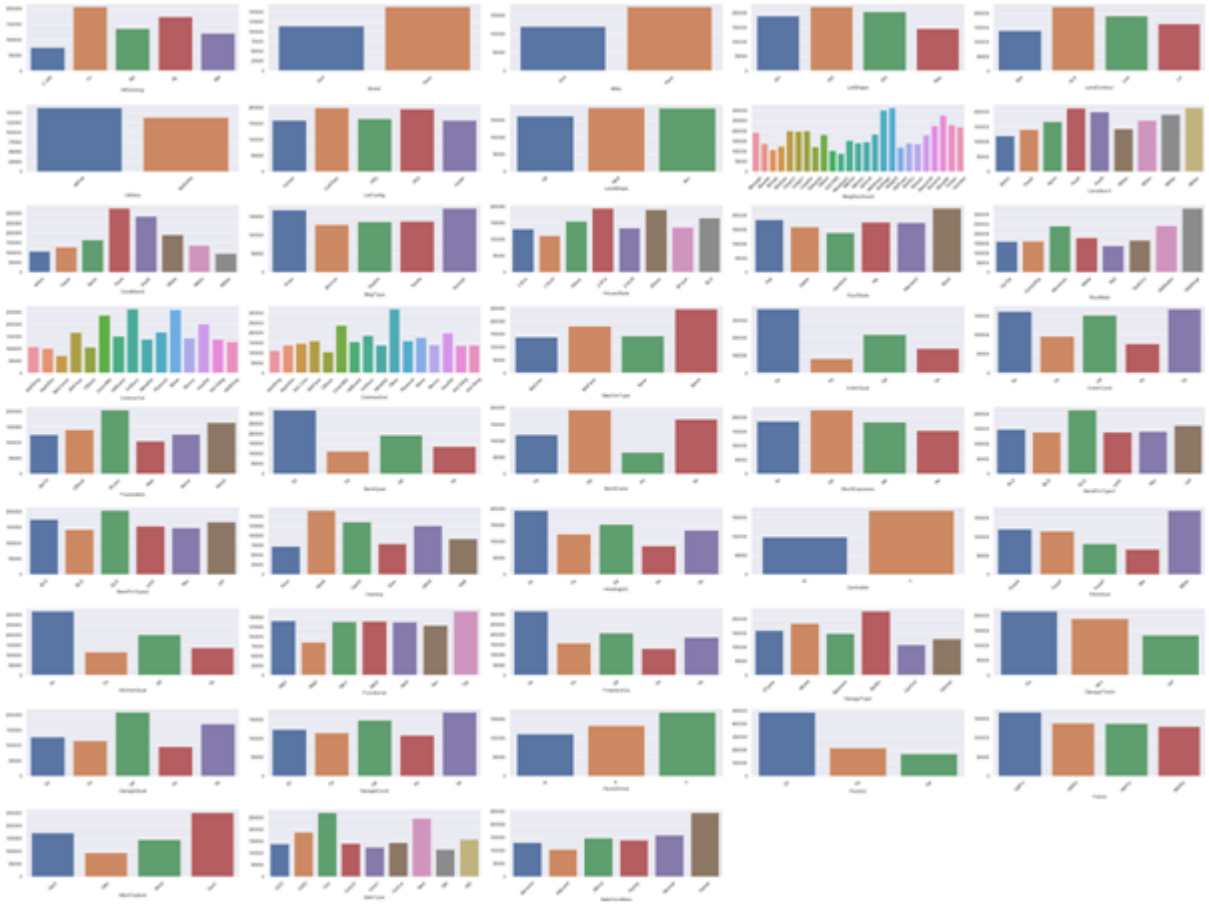

**Figure 14:** Comparison between Categorial Variables and **SalePrice**

- Some categorical variables have stronger impacts on the sale price, such as the neighborhood, exterior quality, kitchen quality, and the presence of a fireplace or central air conditioning.
- Other variables may not have significant effects, such as the type of electrical system or certain miscella - neous features.
- For some groups of related variables, selecting a representative feature with high correlation to sale price can be sufficient, such as GarageCars representing garage - related features.
- Overall, careful feature selection and analysis will help determine the most influential categorical variables for accurate modeling.

## 4. Conclusion

In conclusion, this study delved into the realm of data analysis and feature selection for house price prediction. Through extensive exploration and analysis of the dataset, we uncovered valuable insights regarding the variables that strongly influence the sale price of houses. Key factors such as overall quality, living area, garage capacity, and basement size emerged as significant contributors to the price. By addressing data quality issues like missing values and outliers, and applying feature engineering and transformation techniques, we were able to enhance the interpretability and predictive power of our models.

Looking towards the future, the field of data analysis holds great promise. Advancements in machine learning, artificial intelligence, and big data processing will pave the way for more sophisticated and accurate predictive models. Integration of diverse data sources, including geospatial and social media data, will further enrich our understanding of the factors shaping house prices. Automated feature selection and model interpretability methods will streamline analysis processes and enable better decision - making.

As we navigate this evolving landscape, it is important to strike a balance between comprehensive analysis and practical implementation. The findings from this study provide a solid foundation for developing robust house price prediction models. By leveraging data - driven insights and advanced analytical techniques, we can make more informed decisions in the real estate industry, empowering stakeholders with accurate pricing information and valuable insights. Ultimately, the potential for transformative advancements in predictive analytics within the real estate domain is substantial.

## References

[1] https: //www.kaggle. com/pmarcelino/comprehensive - data - exploration - with - python
[2] https: //www.kaggle. com/code/juliencs/a - study - on - regression - applied - to - the - ames - dataset/notebook
[3] https: //www.kaggle. com/code/siddheshpujari/eda - and - prediction - of - house - price/notebook
[4] Singh, Archana Sharma, Apoorva Dubey, Gaurav. (2020). Big data analytics predicting real estate prices. International Journal of System Assurance Engineering

and Management.11. 10.1007/s13198 - 020 - 00946 - 3.
[5] Wu, Chao ye, Xinyue Ren, Fu Wan, You Ning, Pengfei Du, Qingyun. (2016). Spatial and Social Media Data Analytics of Housing Prices in Shenzhen, China. PLOS ONE.11.10.1371/journal. pone.0164553.
[6] https: //www.kaggle. com/code/apapiu/regularized - linear - models/notebook
[7] Li, Mingzhao Bao, Zhifeng Sellis, Timos Yan, Shi Zhang, Rui. (2018). HomeSeeker: A Visual Analytics System of Real Estate Data. Journal of Visual Languages Computing.45.10.1016/j. jvlc.2018.02.001.