Capstone Project – 1

***Predicting Property Prices in a Specific Location Using Machine Learning***

**Submitted By**

**Ashish Pachauri**

**Batch**: - Data Science & AI (P-150924-E8-SS-30)

**Date of Submission:** - 04/04/2025

**Institution: -** Digicrome Academy

1. **INTRODUCTION TO THE PROJECT:-**

Our project aims to develop a machine learning model for accurately predicting property prices in a specific location. The accurate prediction of property prices is crucial for various stakeholders, including real estate investors, property developers, and homebuyers, as it facilitates informed decision-making in the real estate market.

1. **PROBLEM STATEMENT: -**

The real estate market is a complex and dynamic field influenced by numerous factors such as location, property size, neighbourhood features, and amenities. For buyers, sellers, and agents, accurately predicting property prices poses a significant challenge. Traditional methods may struggle to process large datasets and identify intricate patterns. This project aims to address this challenge by leveraging machine learning to predict property prices in a specific location with precision. The goal is to create a reliable, data-driven model that analyses key variables, handles missing data effectively, and produces accurate predictions to assist real estate stakeholders in making informed decisions.

1. **OBJECTIVE OF THE PROJECT**: -

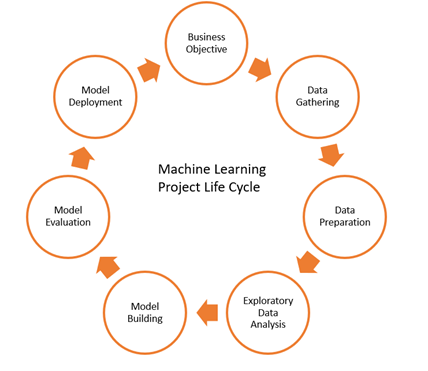
The primary objective of this project is to leverage machine learning techniques to accurately predict property prices in a specific location. This involves the following:

* Collecting and cleaning real estate data, handling missing data effectively.
* Explore and evaluate various regression algorithms, including Linear Regression, Decision Tree Regressor, Support Vector Regressor, and Random Forest Regressor, for their suitability in predicting property prices.
* Conducting exploratory data analysis (EDA) to identify key factors that influence property prices.
* Implementing techniques like scaling, PCA, and encoding to preprocess data effectively.
* Developing, training, and testing a machine learning model that uses selected variables for prediction.
* Evaluating and comparing model performance using metrics like R², MAE, and RMSE.
* Presenting insights into significant factors affecting property prices and delivering a comprehensive report with findings.

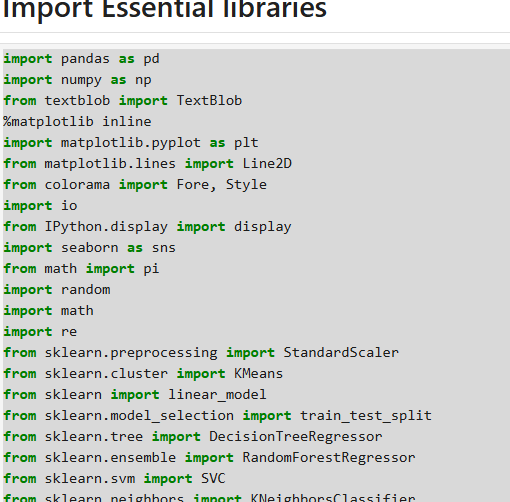
The overarching goal is to create a reliable machine learning model that supports real estate stakeholders by providing accurate property price predictions.

1. **FLOWCHART OF THE OPERATION:-**

* **Data Collection:** Gathering real estate data.
* **Data Cleaning:** Handling missing values and outliers.
* **Exploratory Data Analysis (EDA):** Identifying influential features.
* **Feature Engineering:** Creating new variables for improved model accuracy.
* **Model Selection:** Evaluating machine learning algorithms.
* **Model Training and Testing:** Training the model and assessing performance.
* **Model Evaluation:** Comparing metrics like MAE and RMSE.
* **Model Deployment:** Using the model for property price prediction.

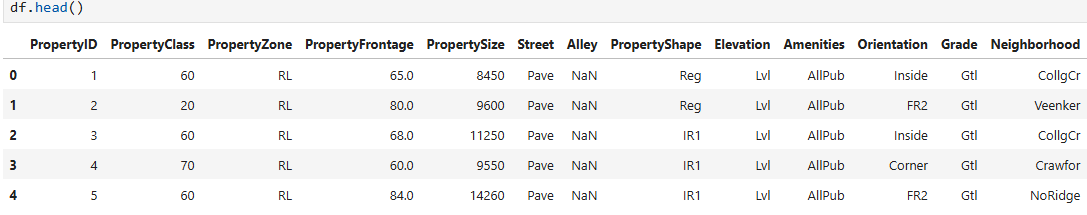


1. **Python Codes:-**



ALL THE CODE WHICH IS USING TO MAKE THIS PROJECT (MACHINE LEARNING: -

df = pd.read\_csv(r"C:\Users\Dell\Desktop\Property\_data.csv")



Print Shape of the Data

check all the columns in the data frame

df.columns

# Getting the information of the data

df.info()

finding the missing value in percentage of each column

total = df.isnull().sum().sort\_values(ascending=False)

percent = (df.isnull().sum() / df.shape[0] \* 100).round(2).sort\_values(ascending=False)

summary = pd.concat([total, percent], axis=1, keys=['Total null value', 'Percentage of null value'])

summary.style.background\_gradient(subset=['Percentage of null value'], cmap='coolwarm')

finding duplicate value of entire data

df.duplicated().sum()

Summary Statistics for numerical column

describe\_transposed = df2.describe().T

styled\_df2 = describe\_transposed.style.background\_gradient(cmap='hsv')

styled\_df2

Summary Statistics for categorical column

desc\_stats = df2.describe(include="object").T

def generate\_bright\_color():

return f"#{random.randint(100, 255):02x}{random.randint(100, 255):02x}{random.randint(100, 255):02x}"

row\_colors = [

{'selector': f'tr:nth-of-type({i+1})',

'props': [('background-color', generate\_bright\_color()), ('color', 'black')]}

for i in range(len(desc\_stats))

]

styled\_output = desc\_stats.style.set\_table\_styles(row\_colors)

display(styled\_output)

# Outliers Detection

numeric\_cols = df2.select\_dtypes(include=['number']).columns

plt.figure(figsize=(15, 70))

for i, col in enumerate(numeric\_cols):

ax = plt.subplot(len(numeric\_cols) // 2 + 1, 2, i + 1)

sns.distplot(df2[col],

color=sns.color\_palette("bright", len(numeric\_cols))[i],

kde=True,

ax=ax)

ax.set\_title(f'Distribution of {col}')

mean = df2[col].mean()

std\_dev = df2[col].std()

median = df2[col].median()

skewness = df2[col].skew()

ax.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')

ax.axvline(median, color='green', linestyle='-', label=f'Median: {median:.2f}')

ax.axvline(mean + std\_dev, color='blue', linestyle='-.', label=f'Mean+Std: {mean + std\_dev:.2f}')

ax.axvline(mean - std\_dev, color='blue', linestyle='-.', label=f'Mean-Std: {mean - std\_dev:.2f}')

skew\_line = Line2D([], [], color='purple', linestyle=':', label=f'Skew: {skewness:.2f}')

handles, labels = ax.get\_legend\_handles\_labels()

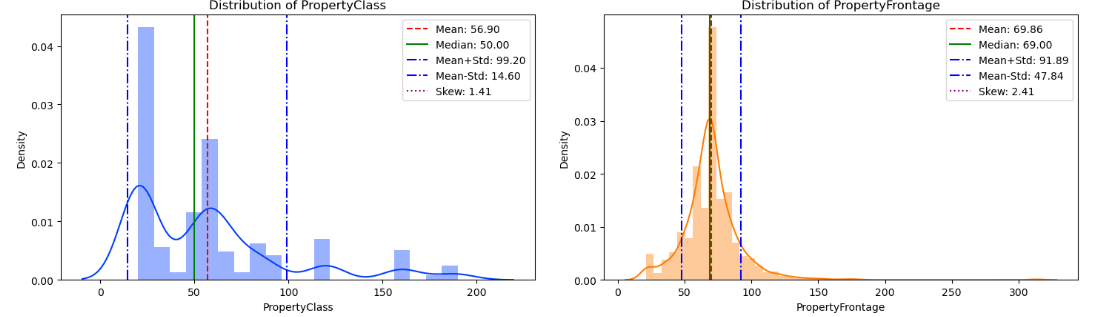
handles.append(skew\_line)

labels.append(f'Skew: {skewness:.2f}')

ax.legend(handles=handles, loc='upper right')

plt.tight\_layout()

plt.show()



**Boxplot**

#Here, we have checked for outliers using a boxplot

numeric\_cols = df2.select\_dtypes(include=['number']).columns

num\_rows = (len(numeric\_cols) + 1) // 2

fig, axes = plt.subplots(ncols=2, nrows=num\_rows, figsize=(20, num\_rows \* 5))

for i, col in enumerate(numeric\_cols):

sns.boxplot(data=df2[[col]], ax=axes[i//2][i%2], color=sns.color\_palette("bright", len(numeric\_cols))[i])

axes[i//2][i%2].set\_title(f'Boxplot of {col}')

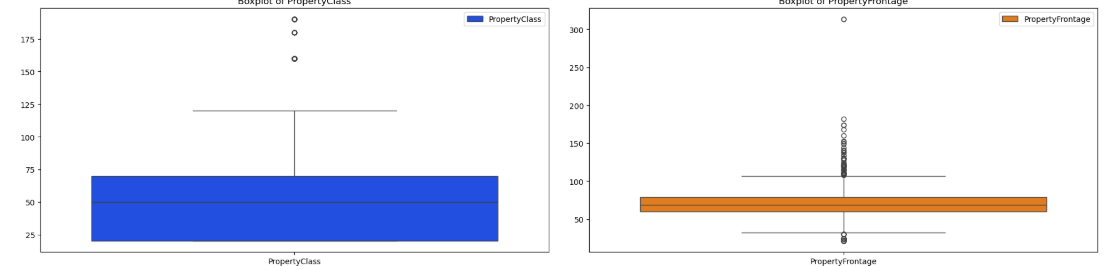
axes[i//2][i%2].legend([col], loc='upper right')

for j in range(len(numeric\_cols), num\_rows \* 2):

fig.delaxes(axes.flatten()[j])

plt.tight\_layout()

plt.show()



Here, we have calculated the number of outliers for each column

num\_cols = df2.select\_dtypes(include='number')

# Calculate Quartiles and IQR

Q1 = num\_cols.quantile(0.25)

Q3 = num\_cols.quantile(0.75)

IQR = Q3 - Q1

outliers = ((num\_cols < (Q1 - 1.5 \* IQR)) | (num\_cols > (Q3 + 1.5 \* IQR)))

num\_outliers = outliers.sum()

color\_mapping = {

"red": "\033[1;91m",

"yellow": "\033[1;93m",

"green": "\033[1;92m",

"cyan": "\033[1;96m",

"blue": "\033[1;94m",

"magenta": "\033[1;95m",

"white": "\033[1;97m"

}

reset\_color = "\033[0m"

color\_names = ["red", "yellow", "green", "cyan", "blue", "magenta", "white"]

print("Columns with potential outliers:")

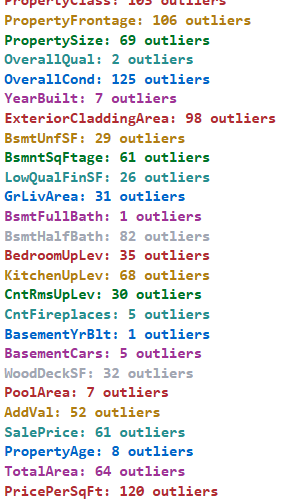
for i, (column, count) in enumerate(num\_outliers.items()):

if count > 0:

color\_name = color\_names[i % len(color\_names)]

row\_color = color\_mapping[color\_name]

print(f"{row\_color}{column}: {count} outliers{reset\_color}")



### Plot the scatter each Each numeric Features

numerical\_cols = df2.select\_dtypes(include=['number']).columns.drop('SalePrice').tolist()

colors = ['red', 'blue', 'green', 'orange', 'purple', 'pink', 'cyan', 'brown', 'yellow', 'lime']

plt.figure(figsize=(20, 70))

for i, col in enumerate(numerical\_cols, 1):

plt.subplot(len(numerical\_cols) // 3 + 1, 3, i)

sns.scatterplot(x=df2[col], y=df2['SalePrice'], color=colors[i % len(colors)])

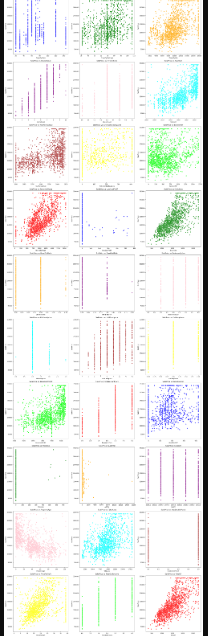
plt.title(f'SalePrice vs {col}')

plt.xlabel(col)

plt.ylabel('SalePrice')

plt.tight\_layout()

plt.show()



**Plot histplot**

key\_features = ['SalePrice', 'BedroomUpLev', 'TotalBathrooms', 'GrLivArea', 'TotalSF']

correlation\_matrix = df2[key\_features].corr()

plt.figure(figsize=(9, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='RdYlGn', fmt='.2f')

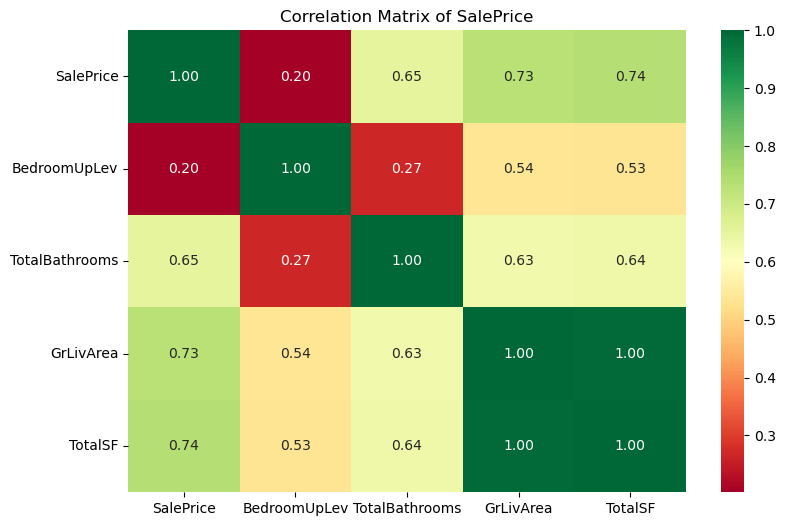
plt.title('Correlation Matrix of SalePrice')

plt.show()

sns.pairplot(df2[key\_features])

plt.title('Scatterplot Matrix of Key Features and SalePrice')

plt.show()

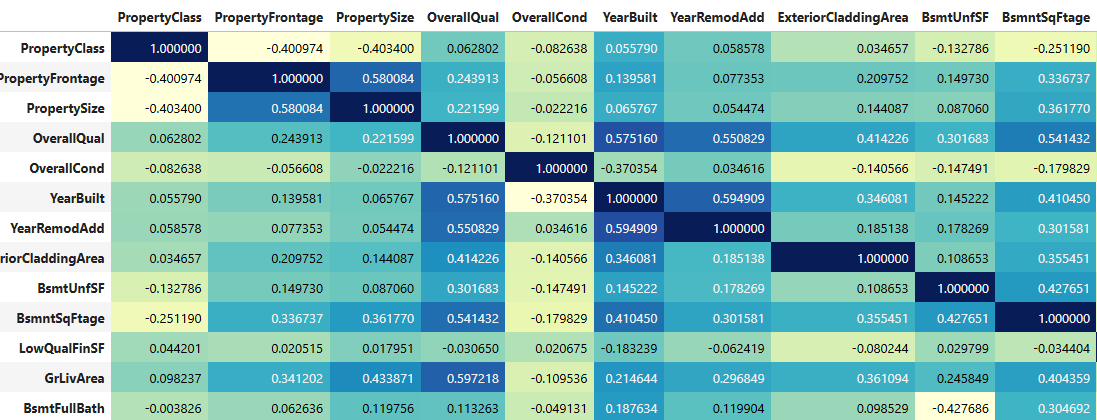


**Correlation: -**

corr = df2.select\_dtypes(include='number').corr()

corr = corr.style.background\_gradient(cmap='YlGnBu')

corr



**Model selection**

**here i am using Label Encoder for categorical column covert to number form**

categorical\_columns = df2.select\_dtypes(include=['object']).columns.tolist()

print("Categorical Columns:", categorical\_columns)

encoder = LabelEncoder()

for categorical\_col in categorical\_columns:

df2[categorical\_col] = encoder.fit\_transform(df2[categorical\_col])

#Separate features and target variable

features = df2.drop('SalePrice', axis=1)

target = df2["SalePrice"]

**#Scale the dataset means normalized the dataset**

Scaler = StandardScaler()

features = pd.DataFrame(Scaler.fit\_transform(features),columns=features.columns)

#Split the data into training and testing sets

X\_train,X\_test,y\_train,y\_test= train\_test\_split(features,target,test\_size=0.2,random\_state=42)

**# check data shape after train text split**

print("X\_train shape :", X\_train.shape)

print("y\_train shape :", y\_train.shape)

print("X\_test shape :", X\_test.shape)

print("y\_test shape :", y\_test.shape)

**# Feature extraction using PCA**

n\_components = 10

pca = PCA(n\_components=n\_components)

X\_pca\_train = pca.fit\_transform(X\_train)

X\_pca\_test = pca.transform(X\_test)

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

print("Explained variance ratio:", explained\_variance\_ratio)

# Combine PCA components with target variable for further analysis

pca\_train\_df = pd.DataFrame(X\_pca\_train, columns=[f"PC{i+1}" for i in range(n\_components)])

pca\_train\_df["SalePrice"] = y\_train.values

## **Creating a Function to Train Model using Different Regression Algorithms.**

model\_scores = []

def evaluate\_model\_with\_plot(model, model\_name, X\_train, y\_train, X\_test, y\_test):

model.fit(X\_train, y\_train)

y\_train\_pred = model.predict(X\_train)

y\_test\_pred = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_test\_pred)

mse = mean\_squared\_error(y\_test, y\_test\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_test\_pred)

adjusted\_r2 = 1 - ((1 - r2) \* (X\_test.shape[0] - 1) / (X\_test.shape[0] - X\_test.shape[1] - 1))

train\_score = model.score(X\_train, y\_train) \* 100

test\_score = model.score(X\_test, y\_test) \* 100

model\_scores.append({

"Model": model\_name,

"Training Score (%)": train\_score,

"Testing Score (%)": test\_score,

"MAE": mae,

"MSE": mse,

"RMSE": rmse,

"R2": r2,

"Adjusted R2": adjusted\_r2

})

print(f"Model: {model\_name}")

print(f"Training Score: {train\_score:.2f}%")

print(f"Testing Score: {test\_score:.2f}%")

print(f"R2: {r2:.2f}, Adjusted R2: {adjusted\_r2:.2f}")

print(f"MAE: {mae:.2f}, MSE: {mse:.2f}, RMSE: {rmse:.2f}")

print("\n")

plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)

plt.scatter(y\_train, y\_train\_pred, color='blue', label='Train')

plt.scatter(y\_test, y\_test\_pred, color='red', label='Test')

plt.xlabel('True values')

plt.ylabel('Predicted values')

plt.legend()

plt.title(f'{model\_name}: Scatter Plot', fontweight="bold", size=20, pad=10)

plt.subplot(1, 2, 2)

plt.scatter(y\_train\_pred, y\_train\_pred - y\_train, color='blue', label='Train')

plt.scatter(y\_test\_pred, y\_test\_pred - y\_test, color='red', label='Test')

plt.axhline(y=0, color='black', linestyle='--')

plt.xlabel('Predicted values')

plt.ylabel('Residuals')

plt.legend()

plt.title(f'{model\_name}: Residual Plot', fontweight="bold", size=20, pad=10)

plt.show()

def run\_all\_models\_with\_plot(models, X\_train, y\_train, X\_test, y\_test):

for model\_name, model in models.items():

evaluate\_model\_with\_plot(model, model\_name, X\_train, y\_train, X\_test, y\_test)

models = {

"Random Forest": RandomForestRegressor(n\_estimators=10),

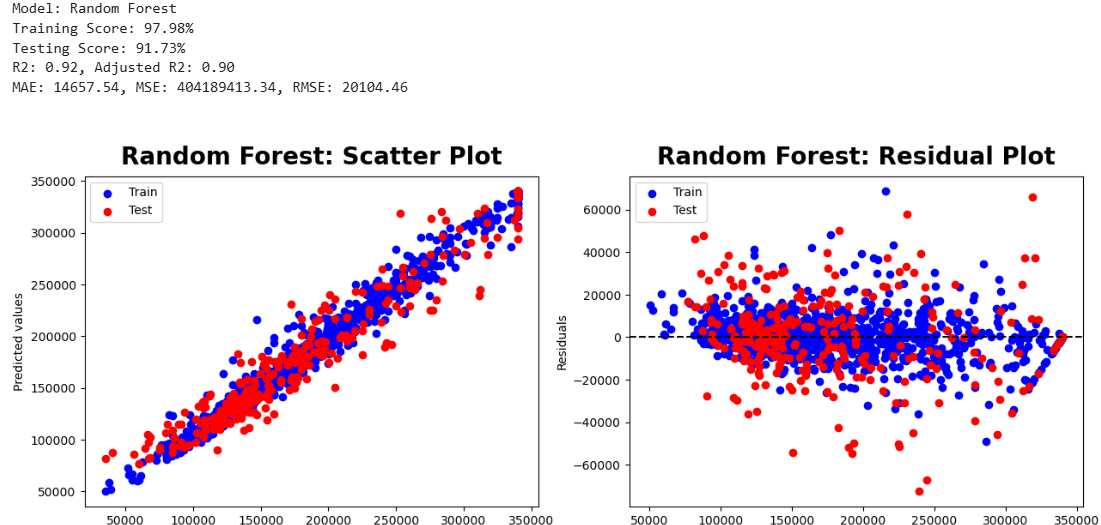
"Linear Regression": LinearRegression(),

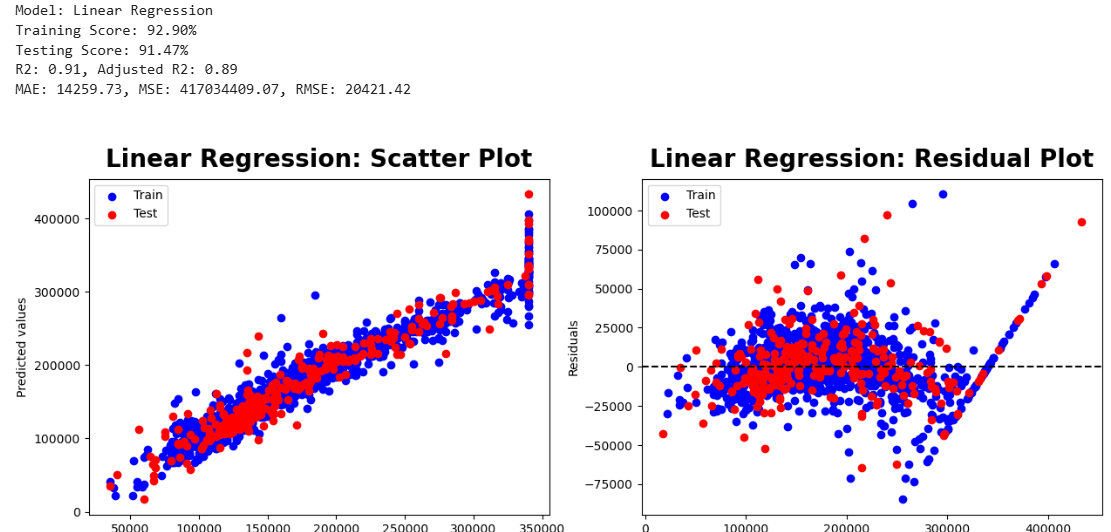
"Decision Tree": DecisionTreeRegressor()

}

run\_all\_models\_with\_plot(models, X\_train, y\_train, X\_test, y\_test)

pd.DataFrame(model\_scores)





# Train the model

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

plt.scatter(range(len(y\_pred)), y\_pred)

plt.scatter(range(len(y\_pred)),y\_test)

plt.show()

test\_predictions = RF.predict(X\_test)

MAE = mean\_absolute\_error(y\_test, test\_predictions).round(3)

MSE = mean\_squared\_error(y\_test, test\_predictions).round(3)

RMSE = np.sqrt(mean\_squared\_error(y\_test, test\_predictions)).round(3)

print(f"MAE: {MAE}, MSE: {MSE}, RMSE: {RMSE}")

kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

cv\_scores = cross\_val\_score(RF, X\_train, y\_train, cv=kf, scoring='neg\_mean\_squared\_error')

cv\_rmse\_scores = np.sqrt(-cv\_scores)

print("Cross-validation RMSE scores:", cv\_rmse\_scores)

print("Mean cross-validation RMSE:", cv\_rmse\_scores.mean())

model.fit(X\_train, y\_train)

y\_pred = RF.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Squared Error: {mse}')

print(f'R-squared: {r2}')

predictions\_df = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

print("\nPredictions:")

predictions\_df.to\_csv('predictions.csv', index=False)

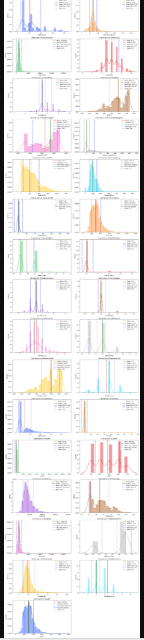
sns.kdeplot(data=test\_predictions, label="Price Predicted")

sns.kdeplot(data=y\_test, label="Actual Price")

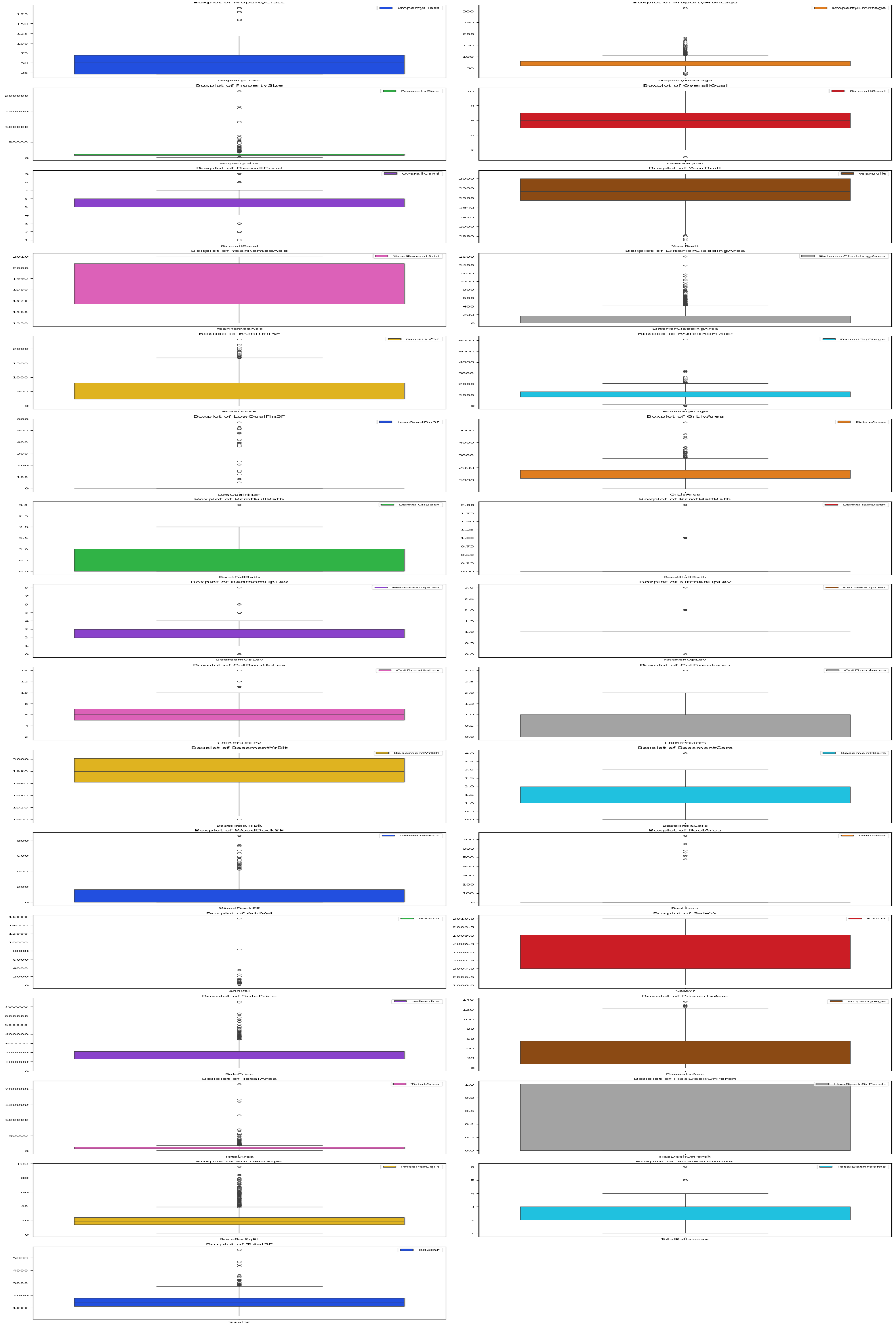
plt.legend()

plt.show()

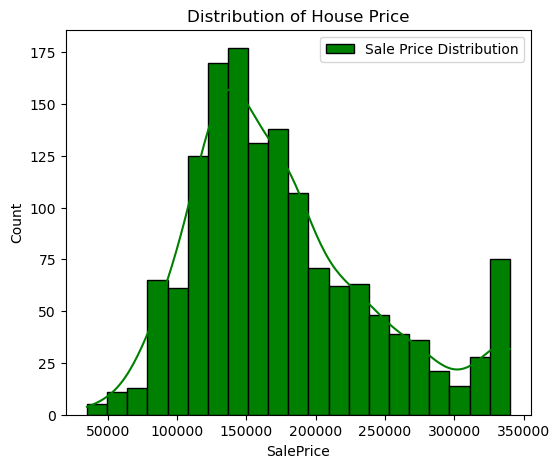
**Data Distribution**



**Data Visualization**

****

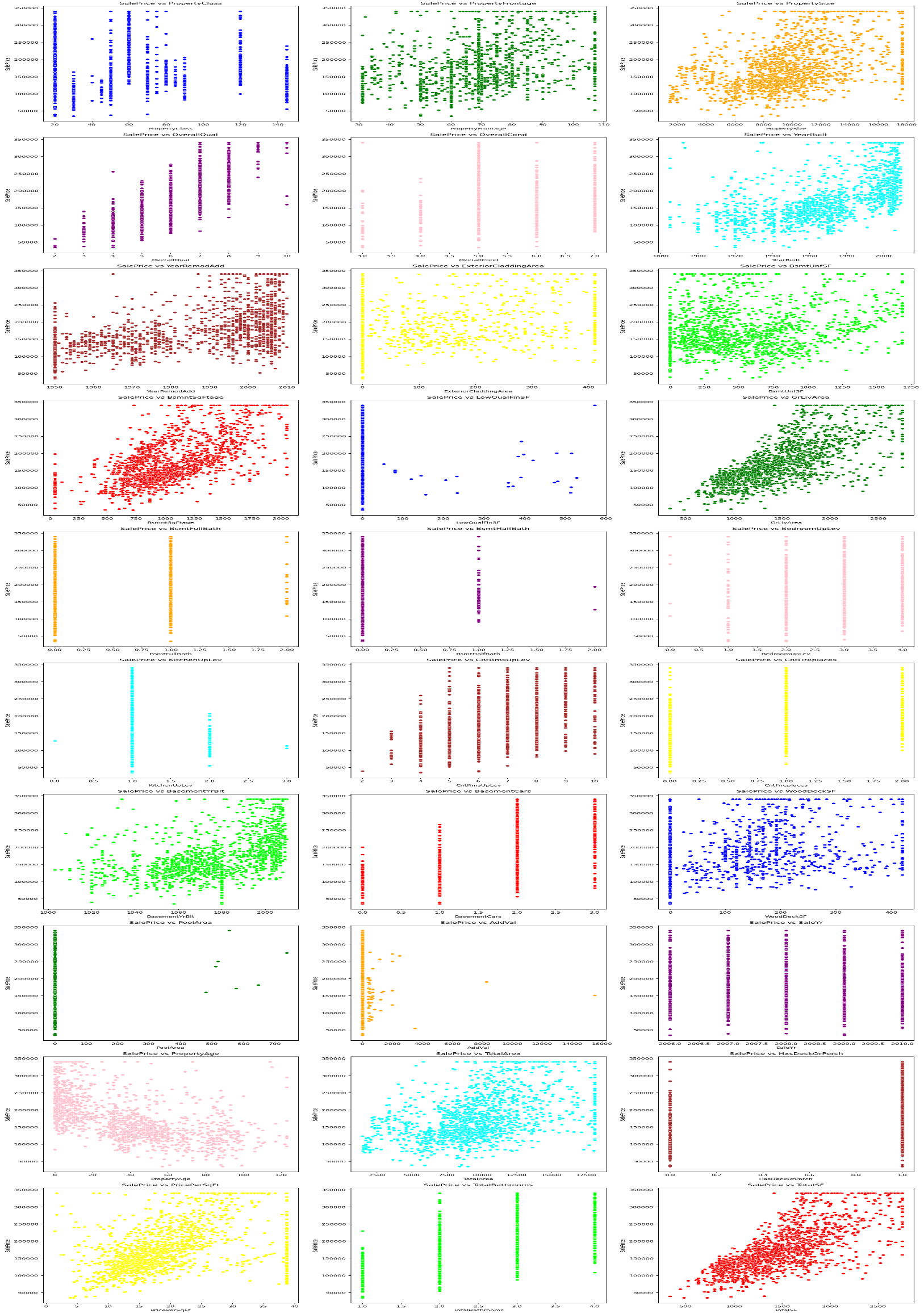
**Distribution of SalePrice**



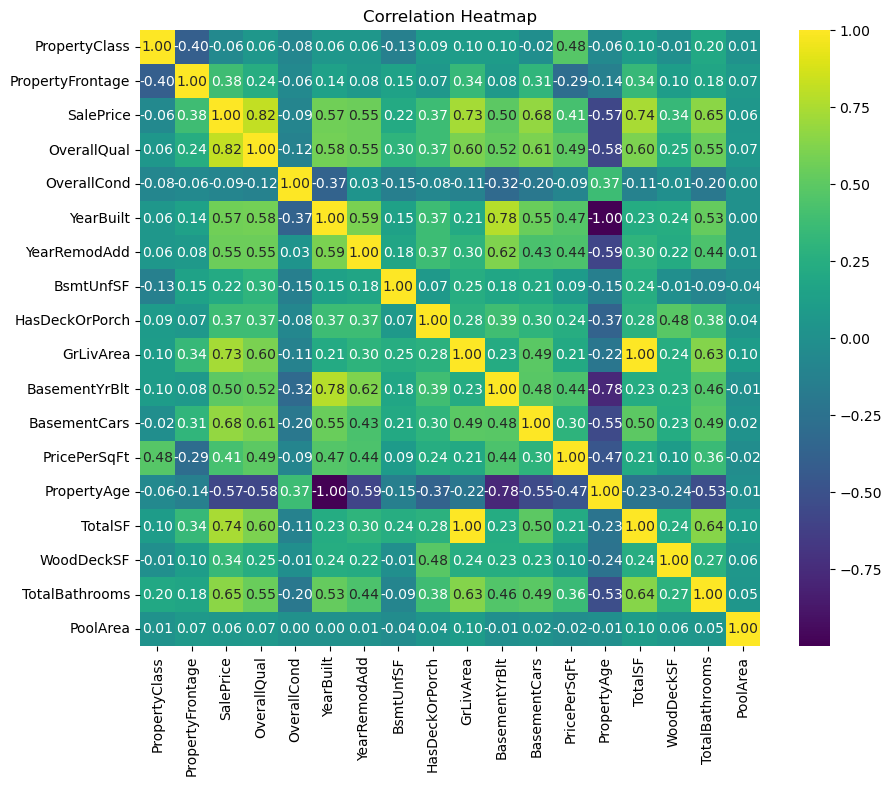
**Distribution Shape:**

The right-skewed distribution curve suggests that while most houses are sold at lower prices, a few high-priced properties pull the average price upward.

**Scatterplot for numerical colu**

****

**Heatmap**

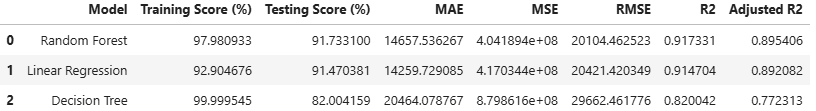
****

OverallQual correlates strongly with other high-value variables such as TotalSF and GrLivArea, indicating the importance of property quality across all metrics.

TotalBathrooms and PoolArea have mild positive correlations, showing that additional amenities add to the desirability of properties.

* 1. **LEARNING OUTCOMES: -**

The performance of the model and compare it with other machine learning algorithms.



**Random Forest Regressor:**

* + Mean Squared Error (MSE): 4.0418
  + R-squared (R^2) Score: 09173
  + Interpretation: The Random Forest regressor demonstrates the best performance among the models evaluated, with the lowest MSE and the highest R^2 score. The R^2 score of 0.91733 indicates that the model explains approximately 97.98% of the variance in the target variable, which is well within the expected range of 97%-91%. This model is considered highly accurate and suitable for predicting property prices in the specific location.

Overall, based on the evaluation metrics and the desired threshold for R^2 score, the Random Forest Regressor is the recommended model for predicting property prices in the specific location. It outperforms the other models and meets the expected R^2 score range of 97%-91%, indicating its effectiveness in capturing the variance in property prices.

**CONCLUSION**

**Conclusion of the Project:**

In this project, we aimed to develop a machine learning model to accurately predict property prices in a specific location. Through the systematic collection, preprocessing, and analysis of relevant data, we explored various regression algorithms, including Linear Regression, Decision Tree and Random Forest Regressor. Our primary goal was to identify the most effective model for predicting property prices, providing valuable insights and tools for real estate stakeholders.

**Websites:**

* **Scikit-Learn Documentation**: https://scikit-learn.org/stable/documentation.html
  + Essential for understanding the implementation details and parameters of machine learning algorithms used in the project.
* **Towards Data Science**: <https://towardsdatascience.com/>
  + Articles and tutorials on machine learning, data preprocessing, and feature engineering were valuable for practical implementation tips and best practices.
* **Analytics Vidhya**: <https://www.analyticsvidhya.com/>
  + Provided articles on data science and machine learning techniques, including the use of Random Forest Regressor and other algorithms for predictive modelling.

***|| THANK YOU ||***