Predicting Housing Market

Team 2 - Project Checkpoint 3

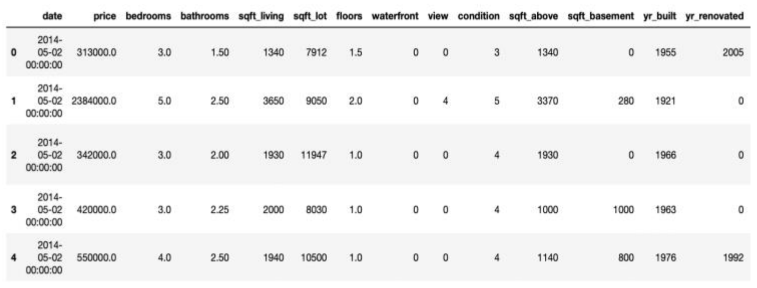
1. Introduction

In the realm of real estate, accurate prediction of house prices is crucial for various stakeholders, including buyers, sellers, and investors. Data-driven approaches, including machine learning are becoming essential for understanding and predicting market behavior. This project embarks on a data-driven journey to explore and implement machine learning models for house price prediction. Leveraging a comprehensive dataset, we aim to assess the performance of prominent regression algorithms, such as Random Forest, XGBoost, Decision Tree, Gradient Boosting, and k-Nearest Neighbors. Our evaluation metric of choice is Mean Squared Error (MSE) and R-square, allowing us to quantitatively compare the predictive capabilities of each model.

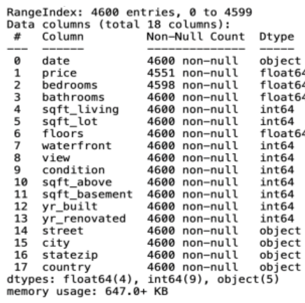
2. Dataset Preprocessing

2.1 Dataset Information

The housing market dataset that we have chosen contains a total of 18 columns and 4600 rows. Columns are date, price, square feet, statezip, bedrooms, bathrooms, sqft_living, country, sqft_lot, floors, waterfront, view, condition, sqft_above, sqft_basement, yr_built, yr_renovated, street and city with a combination of both qualitative and quantitative data types. We improved the dataset's quality by addressing missing values with suitable replacements and removing redundant columns, ensuring data accuracy, and maintaining its overall quality. The following figure illustrates the content and datatypes of the dataset.



	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992



```
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           4600 non-null   object
 1   price          4551 non-null   float64
 2   bedrooms       4598 non-null   float64
 3   bathrooms      4600 non-null   float64
 4   sqft_living    4600 non-null   int64
 5   sqft_lot       4600 non-null   int64
 6   floors         4600 non-null   float64
 7   waterfront     4600 non-null   int64
 8   view           4600 non-null   int64
 9   condition      4600 non-null   int64
 10  sqft_above     4600 non-null   int64
 11  sqft_basement  4600 non-null   int64
 12  yr_built       4600 non-null   int64
 13  yr_renovated   4600 non-null   int64
 14  street         4600 non-null   object
 15  city           4600 non-null   object
 16  statezip       4600 non-null   object
 17  country        4600 non-null   object
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB
```

Figure 2 : Dataset information

2.2 Data Cleaning

After conducting a thorough evaluation of the dataset, we identified 49 null values in the 'price' column and 2 in the 'bedrooms' column. Given the limited prevalence of these null values, we proceeded to remove them to ensure data integrity. To further refine the dataset, we applied the Z-score method to detect and eliminate outliers. The Z-score aids in identifying data points that deviate significantly from the mean, contributing to a more robust dataset by mitigating the impact of outliers. Recognizing that the data pertains exclusively to a single state and country, rendering certain columns redundant, we opted to drop the ‘country', 'state zip', 'street', and 'city' columns. Moreover, the 'date' column, was excluded to streamline the dataset for more focused and meaningful analyses in subsequent stages of the project.

2.3 Data Visualization

Leveraging the dataset, we conducted insightful data visualizations and analyses to gain valuable insights. First, to understand the evolution of housing prices over time, we generated a price vs. year-built trend plot. This visualization allowed us to discern patterns and variations in housing prices across different years, providing a temporal perspective on the real estate market. Furthermore, we delved into a univariate analysis of the 'bedrooms' variable, creating a histogram to explore the distribution of bedroom counts within the dataset. This histogram offers a comprehensive view of the frequency distribution, shedding light on the prevalent bedroom configurations in the housing data. Additionally, to unveil potential relationships between different features, we implemented a correlation matrix. This matrix provides a quantitative measure of the strength and direction of relationships between variables, offering valuable insights into potential correlations within the dataset. These visualizations and analyses collectively contribute to a comprehensive understanding of the dataset.

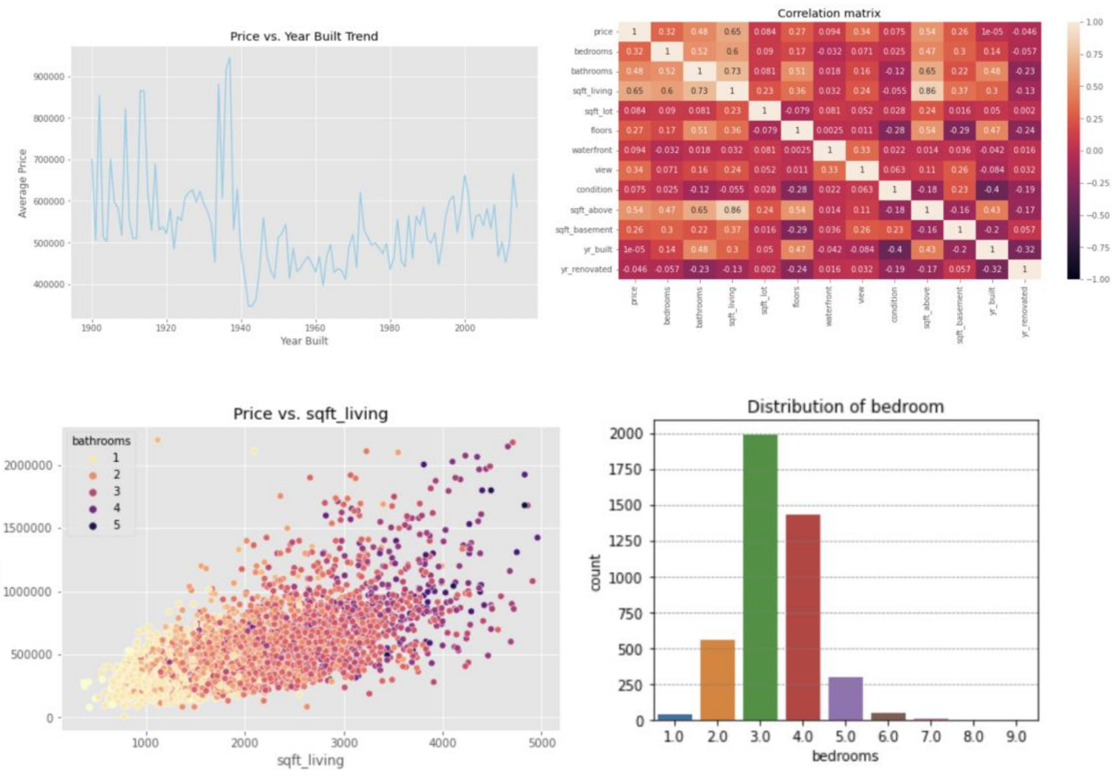
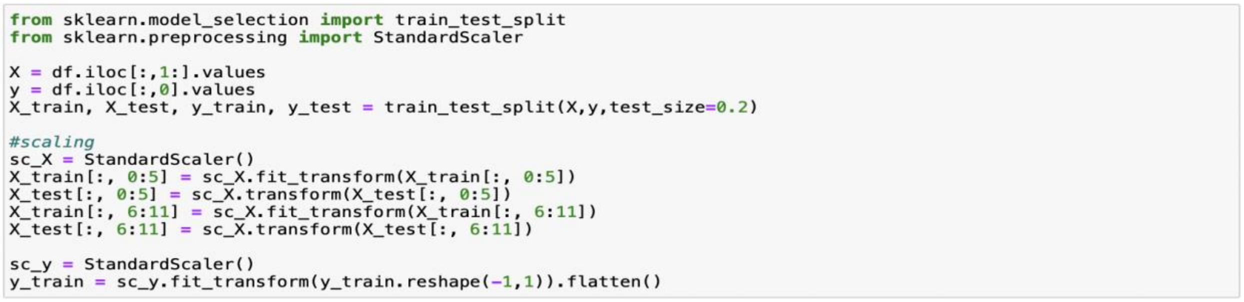


Figure 2 : Dataset Visualization

3. Implementation

The dataset is divided into training and testing sets using the train_test_split function from sklearn.model_selection. This ensures that the model is trained on a subset of the data and evaluated on a separate, unseen portion, providing a reliable assessment of its generalization performance.

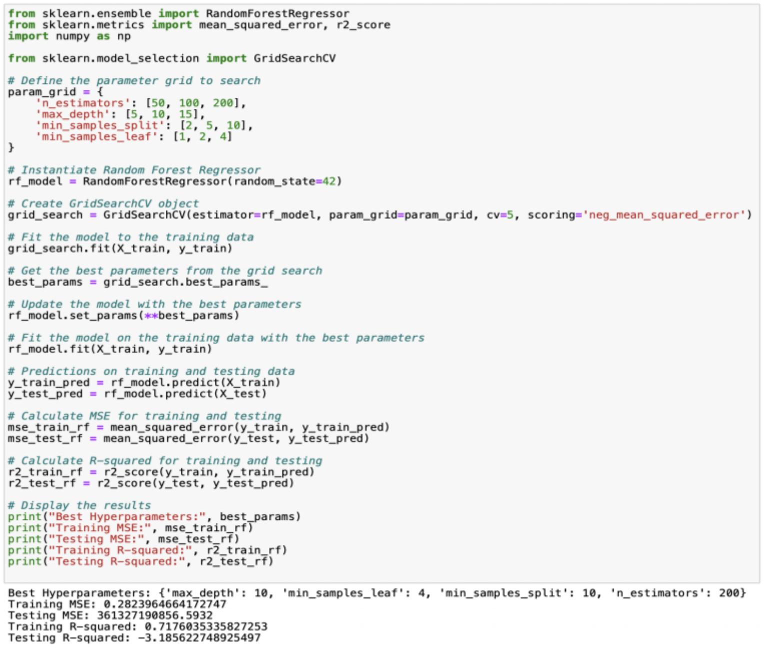
Furthermore, to enhance the effectiveness of the models, we employ feature scaling using StandardScaler from sklearn.preprocessing. Standardization transforms the numerical features of the dataset to a standard scale, with a mean of 0 and a standard deviation of 1. This step is essential as it ensures that all features contribute equally to the model training process, preventing certain features from dominating due to differences in their scales. Both the input features (X) and the target variable (y) undergo standardization to facilitate a more robust and accurate model training process. Overall, these preprocessing steps lay the groundwork for the subsequent application of machine learning models, improving their performance and interpretability in the context of house price prediction.



```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = df.iloc[:,1:].values
y = df.iloc[:,0].values
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)

#scaling
sc_X = StandardScaler()
X_train[:, 0:5] = sc_X.fit_transform(X_train[:, 0:5])
X_test[:, 0:5] = sc_X.transform(X_test[:, 0:5])
X_train[:, 6:11] = sc_X.fit_transform(X_train[:, 6:11])
X_test[:, 6:11] = sc_X.transform(X_test[:, 6:11])

sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train.reshape(-1,1)).flatten()
```

Fig 3 : Data Preprocessing

We have selected random forest as our core algorithm. We have also implemented GridSearchCV to automate the process of hyperparameter tuning. GridSearchCV systematically tests different combinations of hyperparameters for a Random Forest Regressor. It uses cross-validation to assess performance and identifies the hyperparameter set that minimizes mean squared error. The model is then updated with the optimal hyperparameters for improved performance on the given dataset.

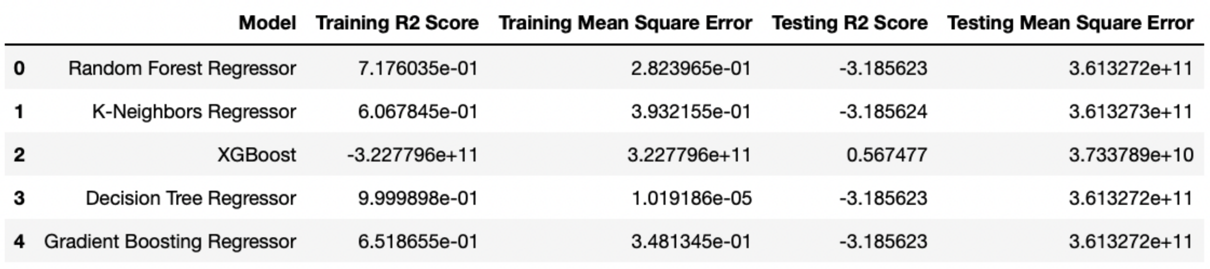


```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Instantiate Random Forest Regressor
rf_model = RandomForestRegressor(random_state=42)

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error')

# Fit the model to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_

# Update the model with the best parameters
rf_model.set_params(**best_params)

# Fit the model on the training data with the best parameters
rf_model.fit(X_train, y_train)

# Predictions on training and testing data
y_train_pred = rf_model.predict(X_train)
y_test_pred = rf_model.predict(X_test)

# Calculate MSE for training and testing
mse_train_rf = mean_squared_error(y_train, y_train_pred)
mse_test_rf = mean_squared_error(y_test, y_test_pred)

# Calculate R-squared for training and testing
r2_train_rf = r2_score(y_train, y_train_pred)
r2_test_rf = r2_score(y_test, y_test_pred)

# Display the results
print("Best Hyperparameters:", best_params)
print("Training MSE:", mse_train_rf)
print("Testing MSE:", mse_test_rf)
print("Training R-squared:", r2_train_rf)
print("Testing R-squared:", r2_test_rf)
```

```
Best Hyperparameters: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 200}
Training MSE: 0.2823964664172747
Testing MSE: 361327190856.5932
Training R-squared: 0.7176035335827253
Testing R-squared: -3.185622748925497
```

Fig 4 : Implementation of Random Forest Regressor

In addition to the Random Forest Regressor, our analysis incorporates four other diverse models, namely the 'K-Neighbors Regressor,' 'XGBoost,' 'Decision Tree Regressor,' and 'Gradient Boosting Regressor.' To comprehensively assess and compare the performance of these models, we have employed key evaluation metrics, specifically the Mean Squared Error (MSE) and R-squared. These metrics provide valuable insights into the accuracy and goodness-of-fit of each model, allowing us to gauge their effectiveness in capturing and predicting the underlying patterns within the given dataset.



	Model	Training R2 Score	Training Mean Square Error	Testing R2 Score	Testing Mean Square Error
0	Random Forest Regressor	7.176035e-01	2.823965e-01	-3.185623	3.613272e+11
1	K-Neighbors Regressor	6.067845e-01	3.932155e-01	-3.185624	3.613273e+11
2	XGBoost	-3.227796e+11	3.227796e+11	0.567477	3.733789e+10
3	Decision Tree Regressor	9.999898e-01	1.019186e-05	-3.185623	3.613272e+11
4	Gradient Boosting Regressor	6.518655e-01	3.481345e-01	-3.185623	3.613272e+11

Fig 5 : Results

The model evaluation results reveal insights into their performance. While the Random Forest Regressor and K-Neighbors Regressor show reasonable training fit, they struggle to generalize to unseen data, evident in negative R2 Scores and high Mean Square Errors on the testing dataset. XGBoost's results indicate potential issues with the model configuration, showing extremely high negative Mean Square Errors. The Decision Tree Regressor displays overfitting with near-perfect training results but poor generalization. The Gradient Boosting Regressor performs similarly to Random Forest and K-Neighbors models.

4. Conclusion

To conclude, our project focused on predicting house prices through a data-driven approach using machine learning models. The dataset underwent thorough preprocessing, including handling missing values, removing outliers, and refining features. We employed five regression models - Random Forest, K-Neighbors, XGBoost, Decision Tree, and Gradient Boosting, evaluated using Mean Squared Error (MSE) and R-squared. The results revealed that while Random Forest and K-Neighbors models showed reasonable training fit, they struggled to generalize to new data. XGBoost exhibited potential configuration issues, and the Decision Tree model displayed overfitting. The Gradient Boosting model performed similarly to Random Forest and K-Neighbors. Our findings emphasize the importance of assessing both training and testing performance. Continuous refinement of model configurations is essential for improving generalization and predictive accuracy in the dynamic real estate market.