Advanced Housing Price Prediction Project Results

Technique	Description	Result
🔍 Data Wrangling	Processing data from loading, exploring, and cleaning with Pandas	Enhanced data quality and prepared for analysis
🎯 EDA (Exploratory Data Analysis)	Different advanced graphs during data exploration, accompanied by insights	Provided valuable insights into dataset patterns and trends
📊 Feature Engineering	Performing advanced feature engineering techniques, including interaction terms and polynomial features	Optimized dataset for modeling and analysis
⚙️ Modeling	Creating at least 2 types of advanced models using pipelines and hyperparameter tuning	Identified the most accurate predictive models
🧠 Model Evaluation	Determining and calculating relevant metrics for accurate model performance assessment	Ensured accurate and reliable model predictions

Project Objectives:

- Objective 1: Conduct comprehensive exploratory data analysis to understand the patterns and trends in predicting housing prices.
- Objective 2: Visualize and compare the impact of various features on housing prices to identify key predictors.
- Objective 3: Implement and evaluate machine learning models for accurate housing price prediction.
- Objective 4: Handle data imbalances and outliers to ensure robust and reliable model performance.

Data Set Description:

The dataset includes details of housing offers, with features related to property characteristics, location, and prices.

Project Steps:

Data Wrangling:

- Processed the dataset, including handling missing values, addressing outliers, and preparing the data for further analysis.
- Identified key features influencing housing prices through statistical analysis and correlation studies.

EDA (Exploratory Data Analysis):

- Created a minimum of 5 advanced graphs, including pair plots, violin plots, joint plots, cluster maps, and interactive visualizations, to provide insights into the dataset's patterns and trends.
- Analyzed geographical distribution and its impact on property prices using interactive maps and geospatial data visualization.

Feature Engineering:

- Performed advanced feature engineering, including one-hot encoding, interaction terms, and polynomial features.
- Split the data into training and testing sets for model optimization.

Modeling:

- Created advanced models using pipelines, such as Lasso Regression and Gradient Boosting Regressor, with hyperparameter tuning.
- Evaluated model performance using metrics such as mean squared error, R-squared, and adjusted R-squared.

Model Evaluation and Enhancement:

- Tested models on unseen data to assess their generalization capabilities and fine-tuned hyperparameters for optimal performance.
- Explored regularization techniques to mitigate overfitting and enhance model robustness.

Explanations:

1. **Data Wrangling (🔍):** The data wrangling phase played a crucial role in improving the overall data quality, ensuring accurate and reliable predictions from the models.

2. **EDA (Exploratory Data Analysis) (🎯):** Creating a minimum of 5 advanced insightful graphs provided valuable insights into the dataset's patterns and trends.

3. **Feature Engineering (📊):** Performing advanced feature engineering techniques optimized the dataset for modeling and analysis.

4. **Modeling (⚙️):** Creating advanced regression models identified the most accurate predictive models for housing prices.

5. **Model Evaluation (🧠):** Calculating relevant metrics ensured accurate and reliable model predictions.

Question and Answer:

1. **Q:** How did the feature scaling techniques affect the model's overall performance during the prediction process?

   A: Feature scaling significantly enhanced the model's convergence and efficiency, leading to more accurate and reliable predictions of housing prices.

2. **Q:** What were the key challenges encountered during the project?

   A: Ensuring robustness in the face of changing housing market trends and minimizing the model's sensitivity to outliers were critical challenges addressed through continuous monitoring and fine-tuning of model parameters.

3. **Q:** Which evaluation metrics were primarily used during the project?

   A: Mean squared error, R-squared, and adjusted R-squared were the primary metrics used to evaluate the model's performance and accuracy in predicting housing prices.

Importing libraries 🐍

In [138…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import r2_score,mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Lasso, Ridge
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.impute import SimpleImputer
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Loading the Dataset 📁

In [139…

```
df = pd.read_csv("./data/Housing.csv")
```

Displaying initial rows and overview of the dataset

In [140…

```
df.shape
```

Out[140]:

```
(545, 13)
```

In [141…

```
print("Initial few rows of the dataset:")
print(df.head())

print("\nOverview of the features and their types in the dataset:")
print(df.info())
```

```
Initial few rows of the dataset:
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000  7420         4          2        3      yes        no       no
1  12250000  8960         4          4        4      yes        no       no
2  12250000  9960         3          2        2      yes        no      yes
3  12215000  7500         4          2        2      yes        no      yes
4  11410000  7420         4          1        2      yes       yes      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no             yes        2      yes        furnished
1              no             yes        3       no        furnished
2              no              no        2      yes   semi-furnished
3              no             yes        3      yes        furnished
4              no             yes        2       no        furnished

Overview of the features and their types in the dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
None
```

In [142…

```
df.duplicated().sum()
```

Out[142]:

```
0
```

In [143…

```
numeric_df = df.select_dtypes(include=['number'])
correlation_matrix = numeric_df.corr()
correlation_matrix
```

Out[143]:

	price	area	bedrooms	bathrooms	stories	parking
price	1.000000	0.535997	0.366494	0.517545	0.420712	0.384394
area	0.535997	1.000000	0.151858	0.193820	0.083996	0.352980
bedrooms	0.366494	0.151858	1.000000	0.373930	0.408564	0.139270
bathrooms	0.517545	0.193820	0.373930	1.000000	0.326165	0.177496
stories	0.420712	0.083996	0.408564	0.326165	1.000000	0.045547
parking	0.384394	0.352980	0.139270	0.177496	0.045547	1.000000

In [144...

```
df.describe() # data stats
```

Out[144]:

	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

In [146...

```
df.isnull().sum() # null values check
```

Out[146]:

```
price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
prefarea            0
furnishingstatus    0
dtype: int64
```

In [147...

```
df.duplicated().sum() # duplicate values check
```

Out[147]:

```
0
```

Data Wrangling

Handling missing values

In [148…

```
imputer = SimpleImputer(strategy='mean')
housing_data = housing_data.dropna()  # Drop rows with missing values
housing_data = housing_data.reset_index(drop=True)
```

In [149…

```
data=df.copy()
```

EDA (Exploratory Data Analysis) 📊 📈 📉

Violin Plot

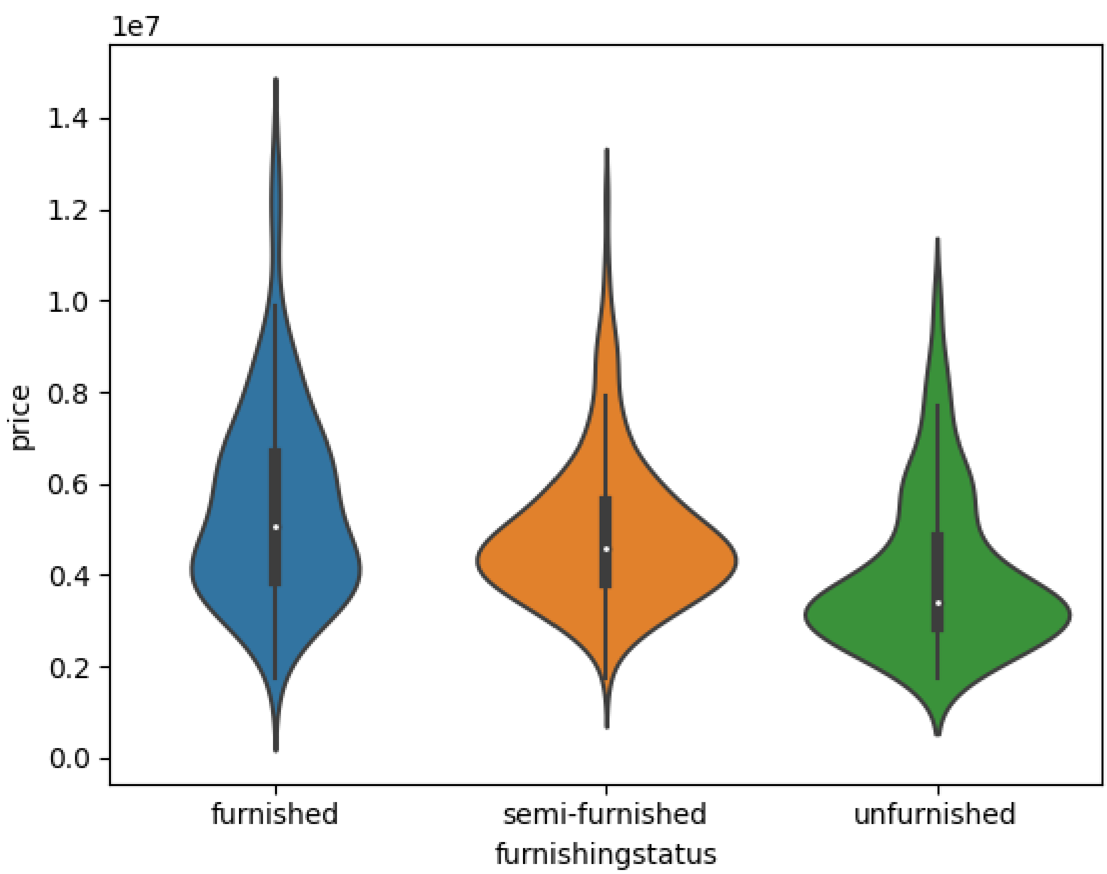
It helps in understanding the distribution of a variable and its variations across different categories. The width of the plot represents the density of data points at different values.

In [150…

```
sns.violinplot(x='furnishingstatus', y='price', data=df)
```

Out[150]:

```
<Axes: xlabel='furnishingstatus', ylabel='price'>
```



Joint Plot

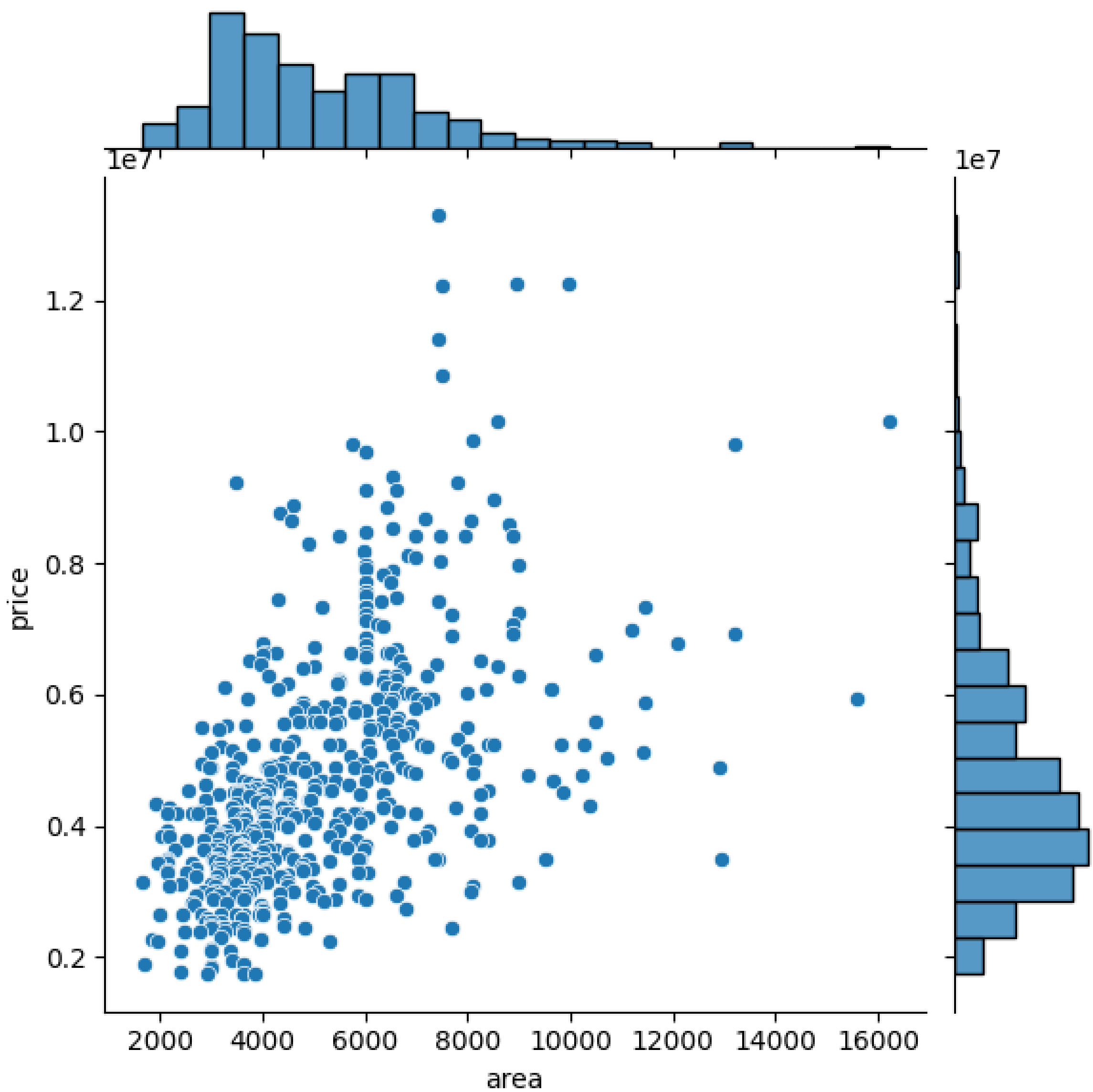
It helps visualize the joint distribution of two variables, providing information about their correlation, spread, and individual distributions.

In [151…

```
sns.jointplot(x='area', y='price', data=df, kind='scatter')
```

Out[151]:

```
<seaborn.axisgrid.JointGrid at 0x2788c245150>
```



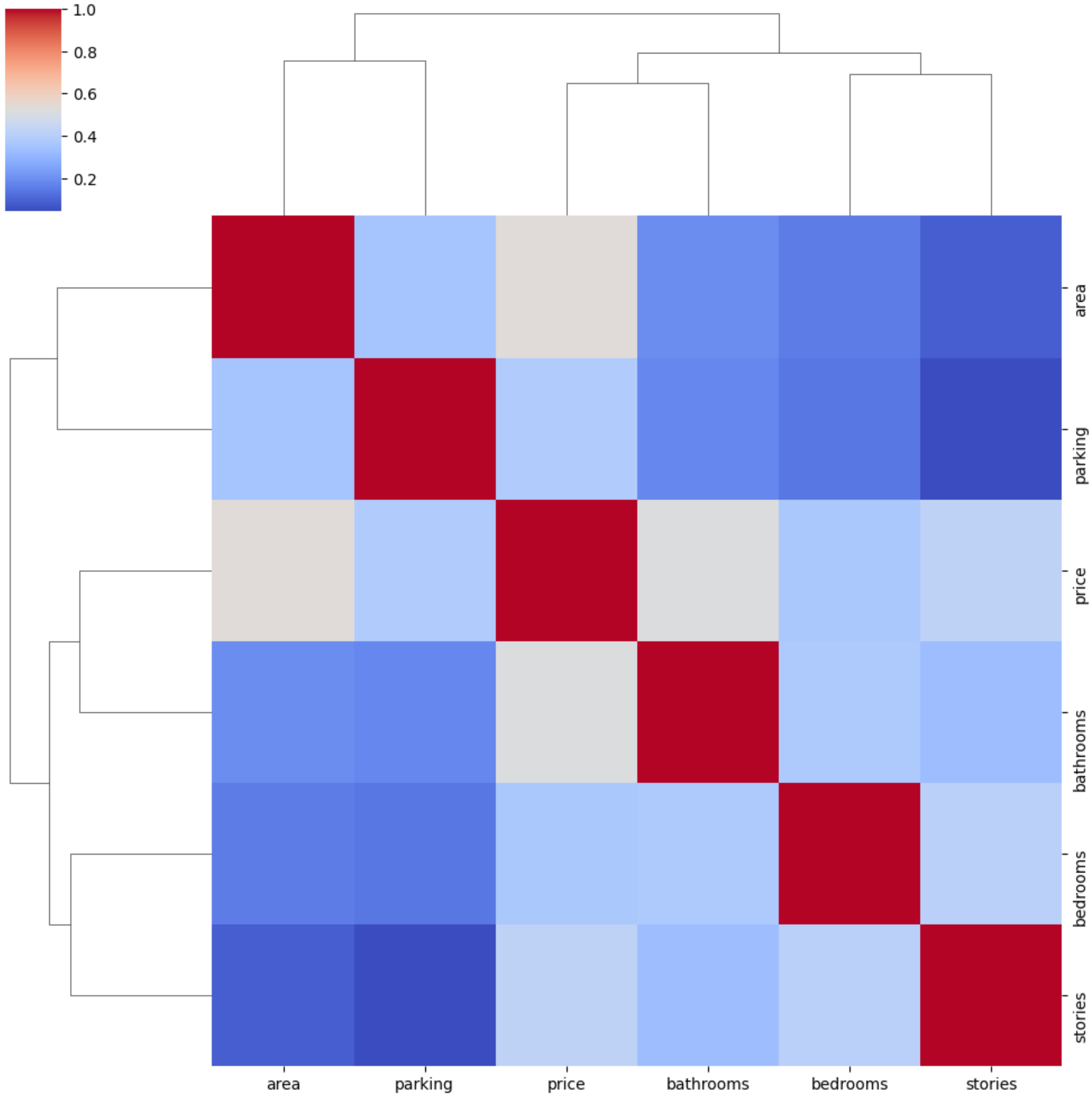
Cluster Map

By observing clusters of similar patterns, it helps identify potential subgroups or relationships between different variables, aiding in understanding underlying structures.

In [152…

```
numeric_df = df.select_dtypes(include=['number'])
correlation_matrix = numeric_df.corr()
sns.clustermap(correlation_matrix, cmap='coolwarm')
```

Out[152]: <seaborn.matrix.ClusterGrid at 0x2788c21cd50>



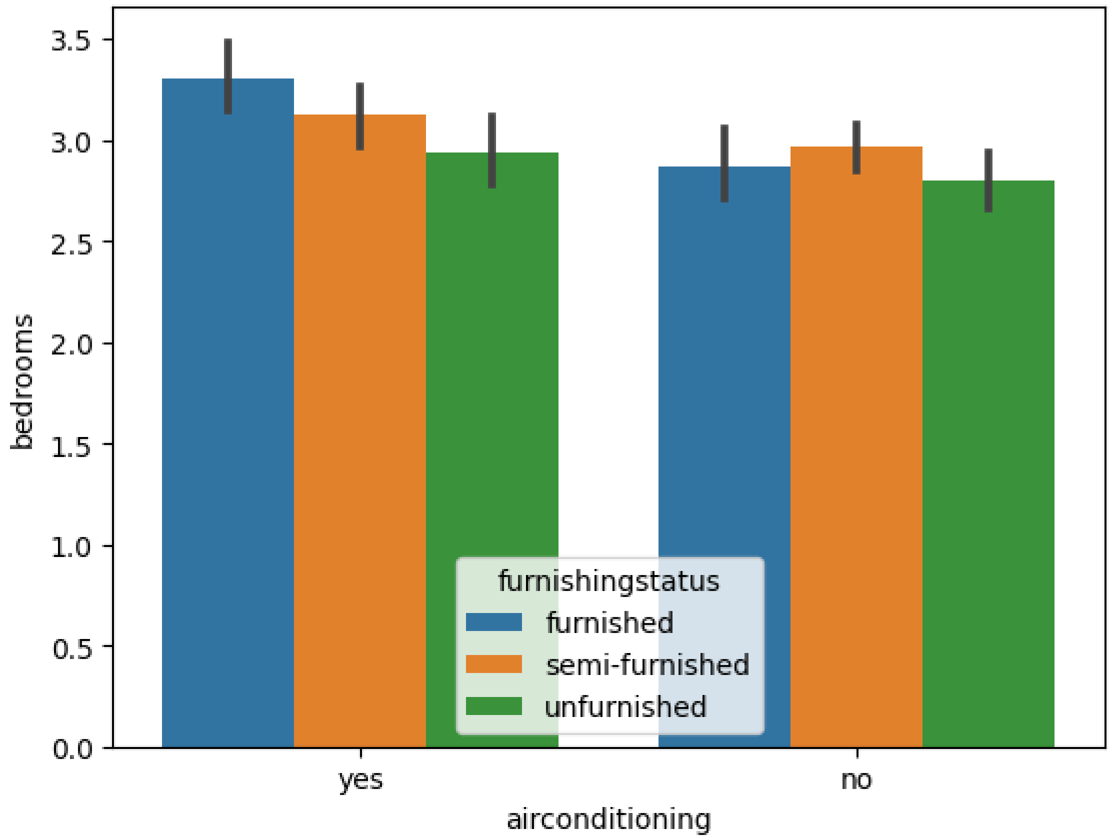
Bar Plot 📊

Insight: It provides insights into how the furnishing status may influence the relationship between air conditioning, bedrooms, and the overall pricing.

In [153…

```
sns.barplot(x=df['airconditioning'],y=df['bedrooms'],hue=df["furnishingstatus"])
```

Out[153]: <Axes: xlabel='airconditioning', ylabel='bedrooms'>

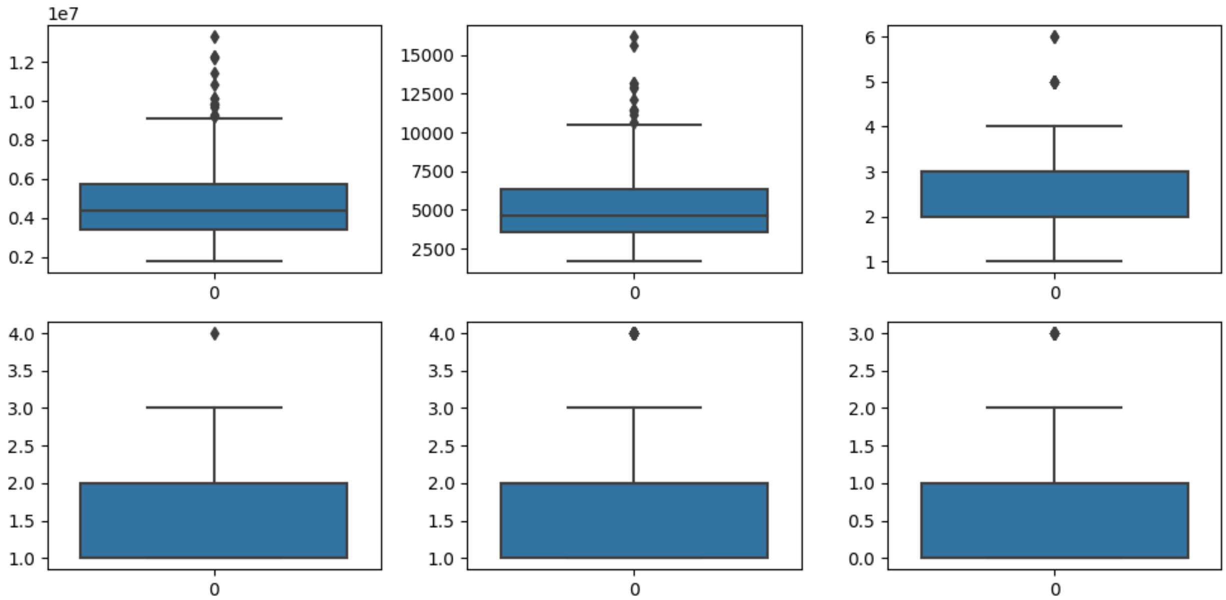


Box Plot 📊

It helps identify potential outliers in the dataset and understand the spread and central tendency of each feature.

In [155…

```
fig, axs = plt.subplots(2,3, figsize = (10,5))
plt1 = sns.boxplot(df['price'], ax = axs[0,0])
plt2 = sns.boxplot(df['area'], ax = axs[0,1])
plt3 = sns.boxplot(df['bedrooms'], ax = axs[0,2])
plt1 = sns.boxplot(df['bathrooms'], ax = axs[1,0])
plt2 = sns.boxplot(df['stories'], ax = axs[1,1])
plt3 = sns.boxplot(df['parking'], ax = axs[1,2])

plt.tight_layout()
```

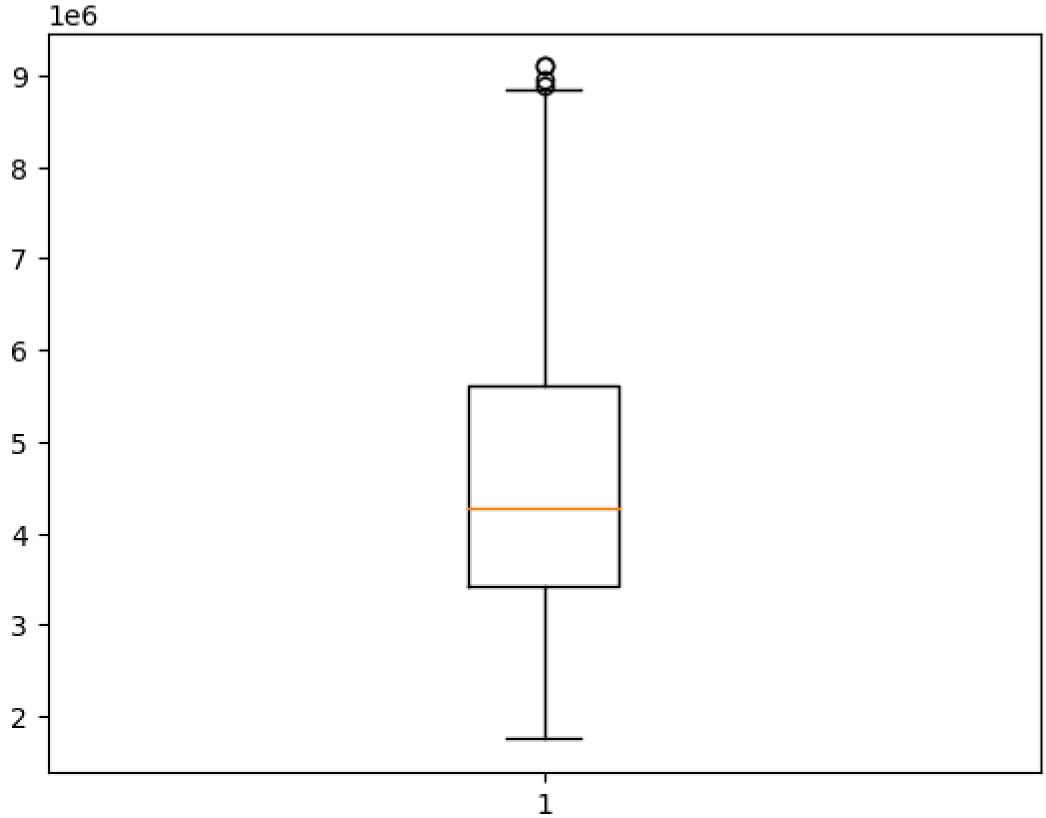


In [156…

```
# Dealing with outliers in price
Q1 = df.price.quantile(0.25)
Q3 = df.price.quantile(0.75)
IQR = Q3 - Q1
df = df[(df.price >= Q1 - 1.5*IQR) & (df.price <= Q3 + 1.5*IQR)]

plt.boxplot(df.price)
```

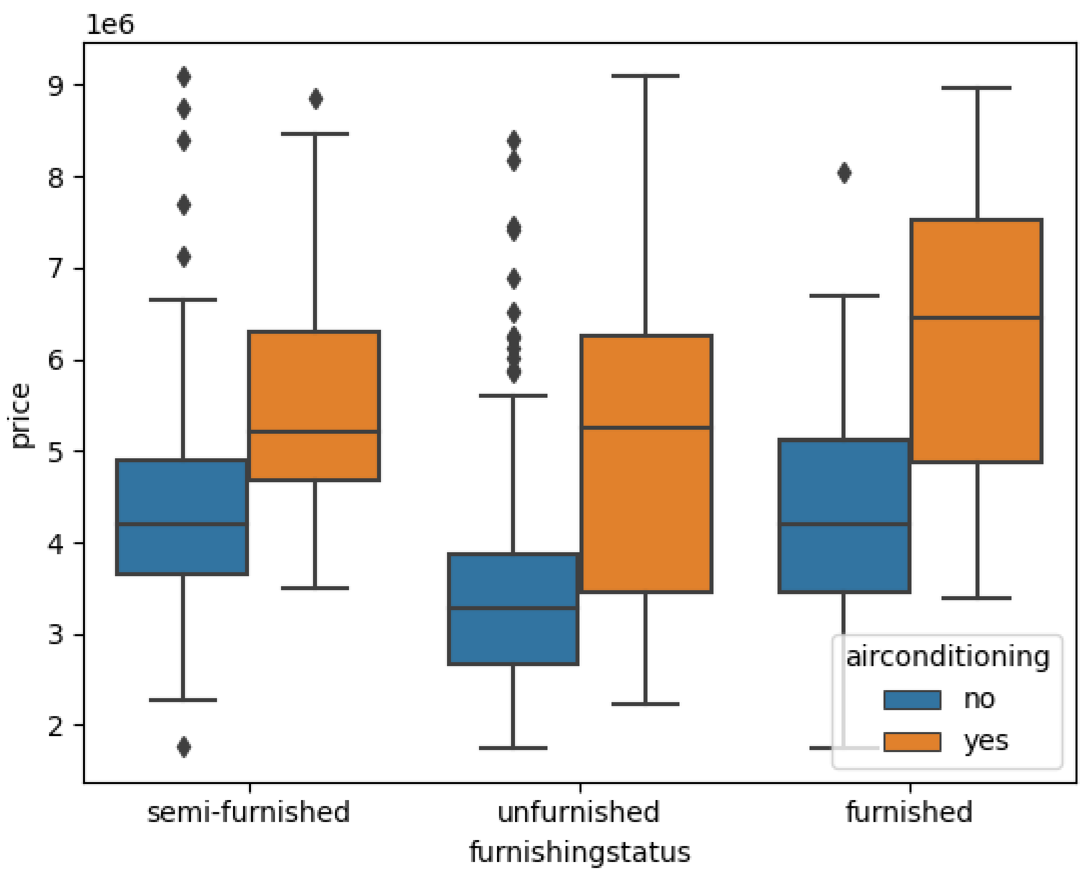
Out[156]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x2788c300c50>,
  <matplotlib.lines.Line2D at 0x2788c3034d0>],
 'caps': [<matplotlib.lines.Line2D at 0x2788c302090>,
  <matplotlib.lines.Line2D at 0x2788c300610>],
 'boxes': [<matplotlib.lines.Line2D at 0x2788c2f0950>],
 'medians': [<matplotlib.lines.Line2D at 0x2788c3022d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x27889447310>],
 'means': []}
```



In [159…
```
sns.boxplot(x = 'furnishingstatus', y = 'price', hue = 'airconditioning', data = df)
```

Out[159]: <Axes: xlabel='furnishingstatus', ylabel='price'>



Distplot 📊

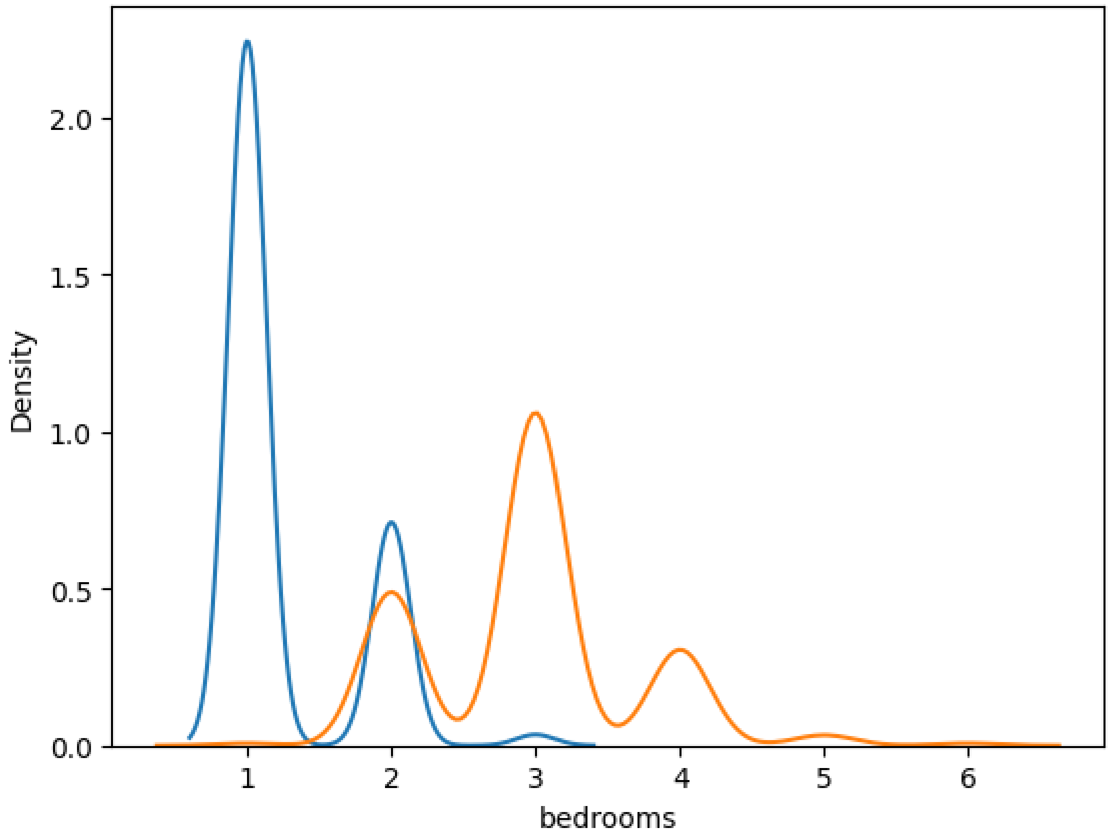
It gives an overview of the distribution patterns of these features, aiding in understanding their variations.

In [160…

```
sns.distplot(df["bathrooms"],hist=False)
sns.distplot(df["bedrooms"],hist=False)
```

Out[160]:

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



Heatmap 🌡️

It helps identify relationships and dependencies between different features, revealing which variables are strongly correlated.

In [161…

```
# Heatmap


plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, cmap='viridis', annot=True)
plt.show()
```

Scatter Plot 📊 📈

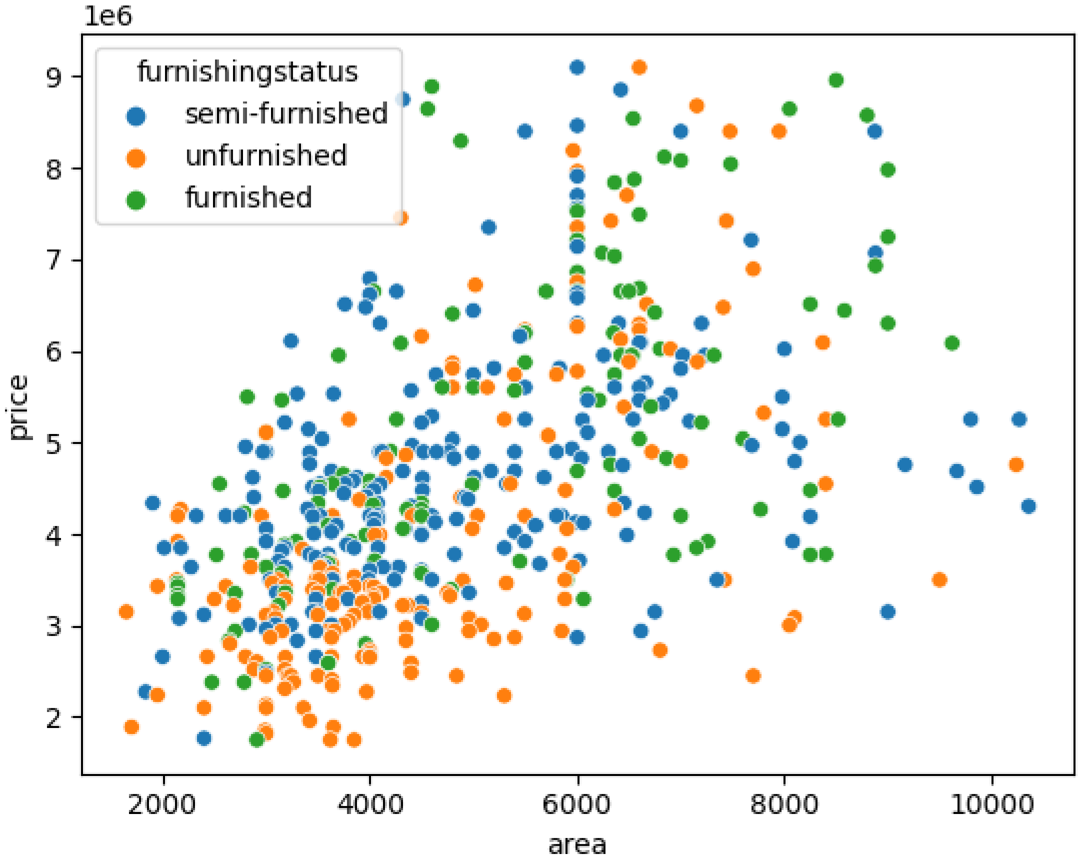
It provides insights into how the furnishing status may impact the correlation between the area and pricing.

In [162…

```
sns.scatterplot(y=df['price'],x=df['area'],hue=df['furnishingstatus'])
```

Out[162]:

```
<Axes: xlabel='area', ylabel='price'>
```



Pair Plot 📊 📈

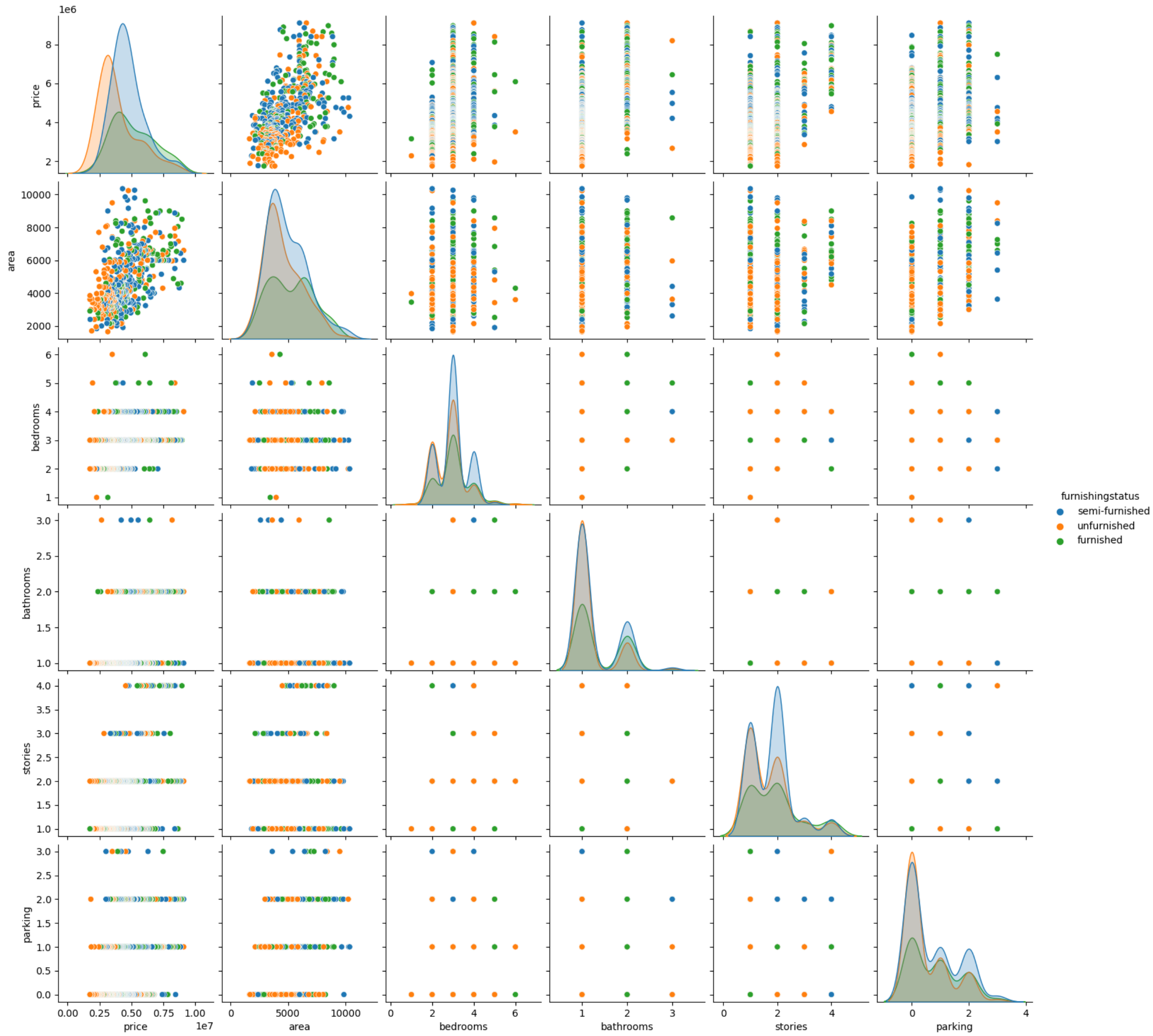
It helps visualize relationships and distributions among multiple variables simultaneously, especially focusing on the impact of furnishing status.

In [163…

```
sns.pairplot(df,hue="furnishingstatus")
```

Out[163]:

```
<seaborn.axisgrid.PairGrid at 0x2788d2210d0>
```



Feature Engineering 🛠️

In [164…

```
status = pd.get_dummies(data[['furnishingstatus','mainroad', 'guestroom', 'basement',
data = pd.concat([data, status], axis = 1)
data.drop(['furnishingstatus','mainroad', 'guestroom', 'basement', 'hotwaterheating',
data.head()
```

Out[164]:

	price	area	bedrooms	bathrooms	stories	parking	furnishingstatus_semi-furnished	furnishingstatus_u
0	13300000	7420	4	2	3	2	False	
1	12250000	8960	4	4	4	3	False	
2	12250000	9960	3	2	2	2	True	
3	12215000	7500	4	2	2	3	False	
4	11410000	7420	4	1	2	2	False	

In [165…

```
poly = PolynomialFeatures(degree=2, interaction_only=True)
X_poly = poly.fit_transform(X)
X_poly
```

Out[165]:

```
array([[1.00e+00, 7.42e+03, 4.00e+00, ..., 0.00e+00, 0.00e+00, 1.00e+00],
       [1.00e+00, 8.96e+03, 4.00e+00, ..., 0.00e+00, 0.00e+00, 0.00e+00],
       [1.00e+00, 9.96e+03, 3.00e+00, ..., 0.00e+00, 0.00e+00, 0.00e+00],
       ...,
       [1.00e+00, 3.62e+03, 2.00e+00, ..., 0.00e+00, 0.00e+00, 0.00e+00],
       [1.00e+00, 2.91e+03, 3.00e+00, ..., 0.00e+00, 0.00e+00, 0.00e+00],
       [1.00e+00, 3.85e+03, 3.00e+00, ..., 0.00e+00, 0.00e+00, 0.00e+00]])
```

In [166…

```
X = data.drop(['price'],axis=1)
y = data['price']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=42)
```

Feature Scaling ⚖️

In [177…

```
scaler = MinMaxScaler()
# scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Models Training 📈

In [168…

```
models = {
    'Random Forest Regressor': RandomForestRegressor(),
    'Gradient Boost Regressor': GradientBoostingRegressor(),
    'XGBoost': xgb.XGBRegressor(),
    'XGRF Regressor': xgb.XGBRFRegressor(),
    'Support Vector regressor': SVR(),
    'Lasso Reg': Lasso(),
    'Ridge Reg': Ridge(),
    'LGBM Reg': LGBMRegressor(),
    'Cat Boost': CatBoostRegressor()
}
```

In [169…

```
pred = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    pred[name] = y_pred
```

```
971:	learn: 335664.1679869	total: 873ms	remaining: 25.1ms
972:	learn: 335436.9188065	total: 874ms	remaining: 24.3ms
973:	learn: 335258.1541760	total: 875ms	remaining: 23.4ms
974:	learn: 334928.7610126	total: 876ms	remaining: 22.5ms
975:	learn: 334676.5605425	total: 877ms	remaining: 21.6ms
976:	learn: 334628.5918678	total: 877ms	remaining: 20.7ms
977:	learn: 334478.3811054	total: 878ms	remaining: 19.8ms
978:	learn: 333811.0750418	total: 879ms	remaining: 18.9ms
979:	learn: 333797.2630738	total: 880ms	remaining: 18ms
980:	learn: 333490.0443705	total: 881ms	remaining: 17.1ms
981:	learn: 332898.5248321	total: 882ms	remaining: 16.2ms
982:	learn: 332857.6070741	total: 883ms	remaining: 15.3ms
983:	learn: 332572.6387849	total: 884ms	remaining: 14.4ms
984:	learn: 331944.4350380	total: 885ms	remaining: 13.5ms
985:	learn: 331857.1609948	total: 886ms	remaining: 12.6ms
986:	learn: 331506.7407724	total: 887ms	remaining: 11.7ms
987:	learn: 331386.5438701	total: 888ms	remaining: 10.8ms
988:	learn: 331300.0614605	total: 889ms	remaining: 9.88ms
989:	learn: 331120.6698878	total: 889ms	remaining: 8.98ms
990:	learn: 330870.8772035	total: 890ms	remaining: 8.09ms
991:	learn: 330673.6867186	total: 891ms	remaining: 7.19ms
992:	learn: 330234.1723886	total: 892ms	remaining: 6.29ms
993:	learn: 329993.8442320	total: 893ms	remaining: 5.39ms
994:	learn: 329932.3125402	total: 894ms	remaining: 4.49ms
995:	learn: 329723.1141252	total: 895ms	remaining: 3.59ms
996:	learn: 329556.4119168	total: 896ms	remaining: 2.69ms
997:	learn: 329243.4004583	total: 896ms	remaining: 1.8ms
998:	learn: 329039.1850896	total: 897ms	remaining: 898us
999:	learn: 328818.3197745	total: 898ms	remaining: 0us
```

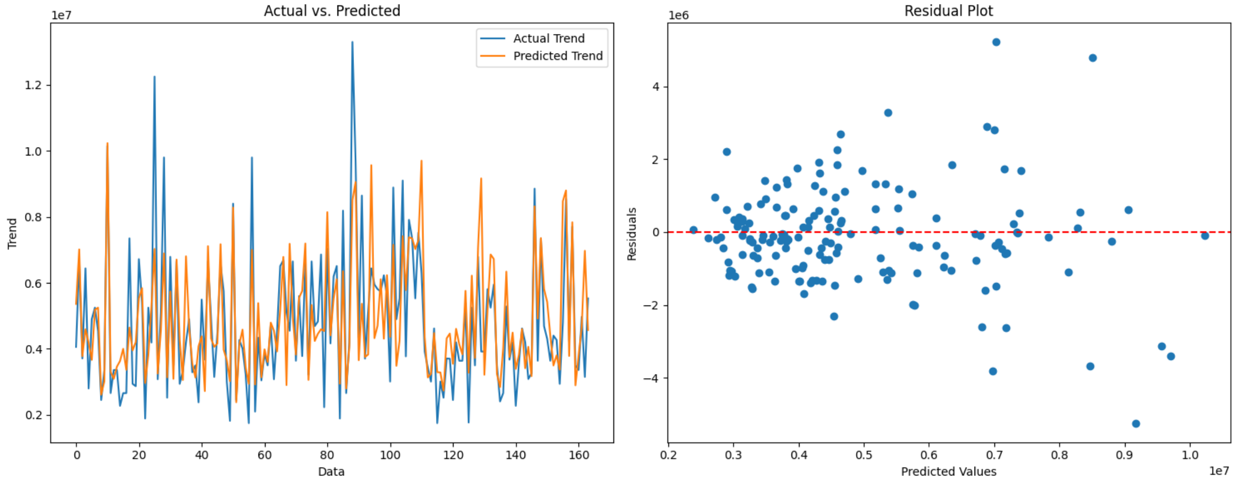
Model Analysis & Models Evaluation 🧪

In [170…

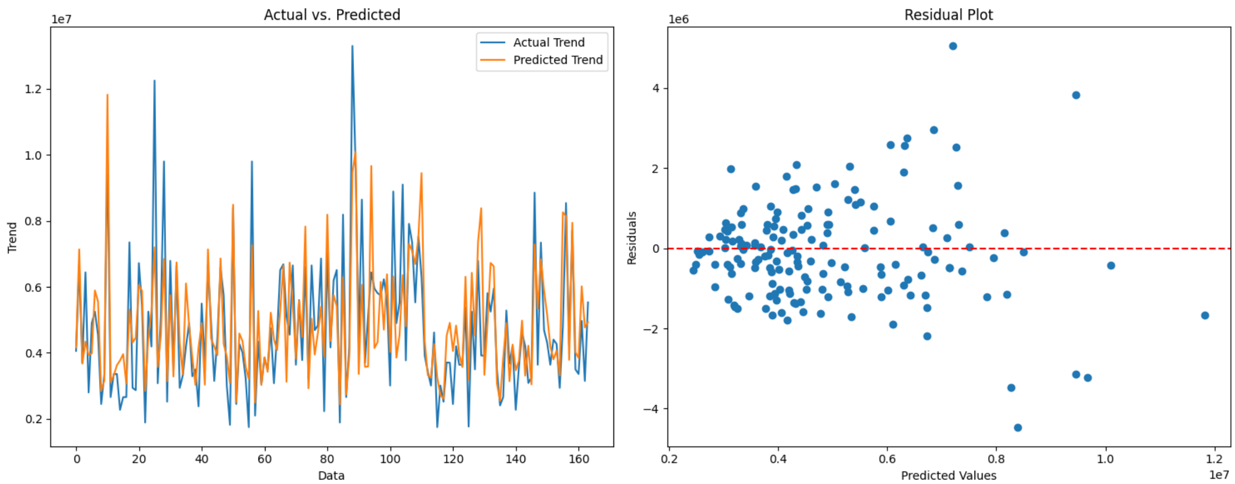
```
acc = {}
for name, y_pred in pred.items():
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    acc[name] = r2
    print(f"Results for {name} : ")
    print(f"Mean Square Error : {mse}")
    print(f"R2 Score : {r2}")
    plt.figure(figsize=(15, 6))

    # Plot Actual vs. Predicted values
    plt.subplot(1, 2, 1)
    plt.plot(np.arange(len(y_test)), y_test, label='Actual Trend')
    plt.plot(np.arange(len(y_test)), y_pred, label='Predicted Trend')
    plt.xlabel('Data')
    plt.ylabel('Trend')
    plt.legend()
    plt.title('Actual vs. Predicted')

    # Plot Residuals
    residuals = y_test - y_pred

    plt.subplot(1, 2, 2)
    plt.scatter(y_pred, residuals)
    plt.axhline(y=0, color='r', linestyle='--')
    plt.xlabel('Predicted Values')
```

```
    plt.ylabel('Residuals')
    plt.title('Residual Plot')

    plt.tight_layout()
    plt.show()
```

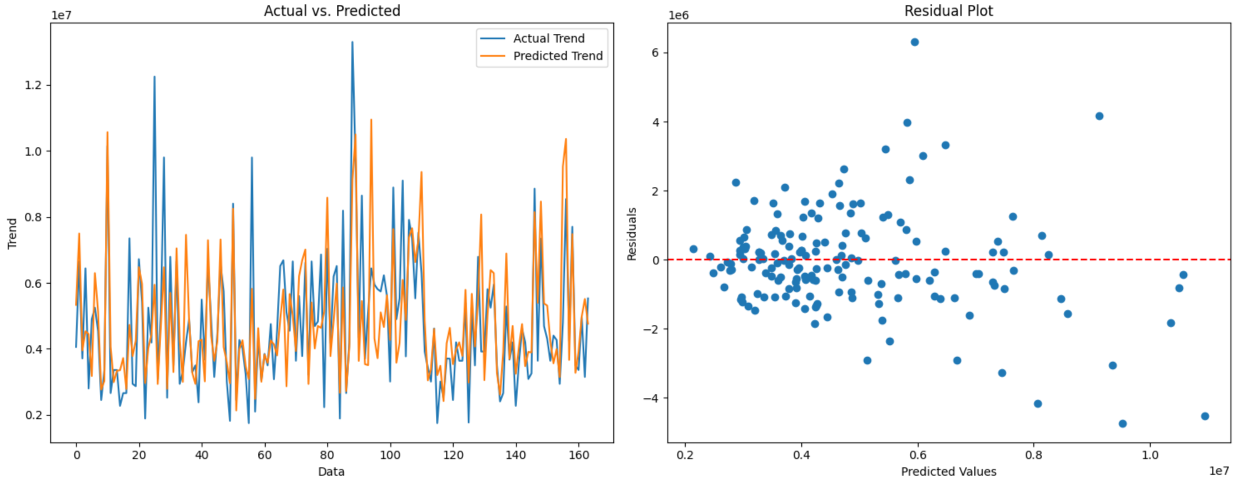
```
Results for Random Forest Regressor :
Mean Square Error : 1884243823363.0266
R2 Score : 0.5624540988944284
```



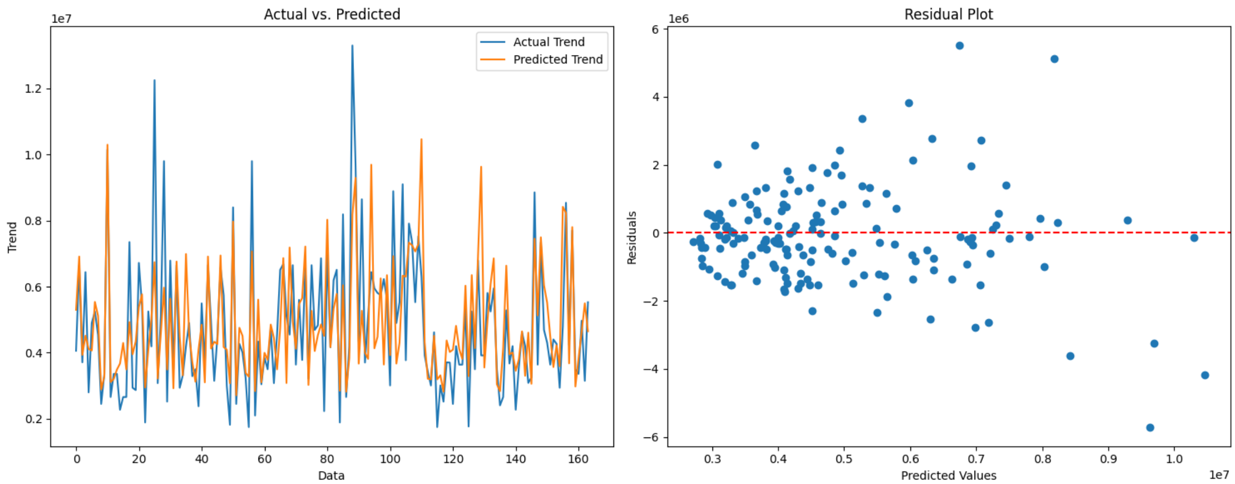

```
Results for Gradient Boost Regressor :
Mean Square Error : 1607156888035.4075
R2 Score : 0.626797285959315
```



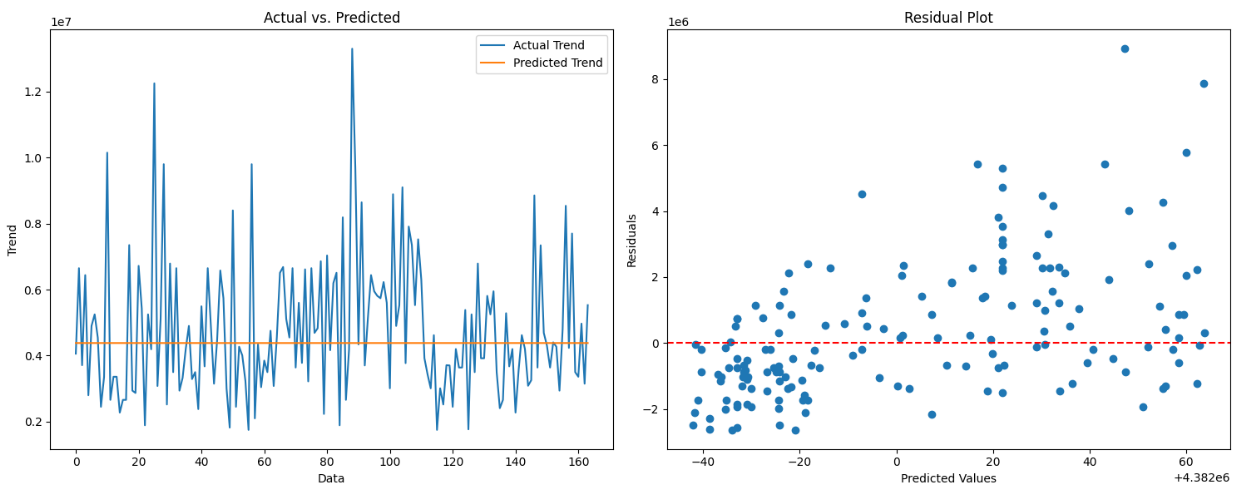

```
Results for XGBoost :
Mean Square Error : 2085041665999.2249
R2 Score : 0.5158262305119282
```



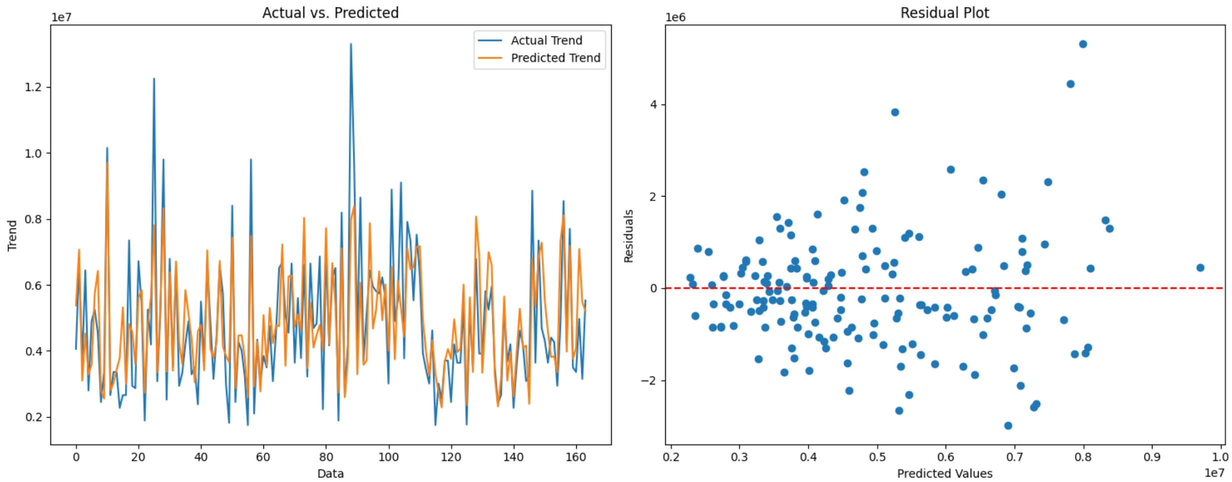
Results for XGRF Regressor :
Mean Square Error : 2097740904480.9404
R2 Score : 0.5128773023127531



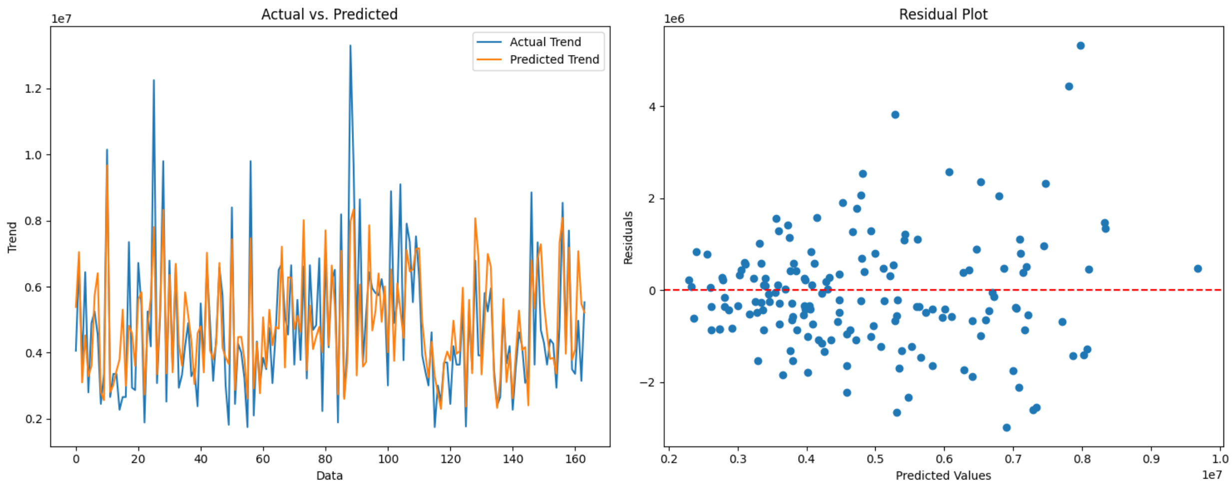
Results for Support Vector regressor :
Mean Square Error : 4462165325262.905
R2 Score : -0.03617277335133684



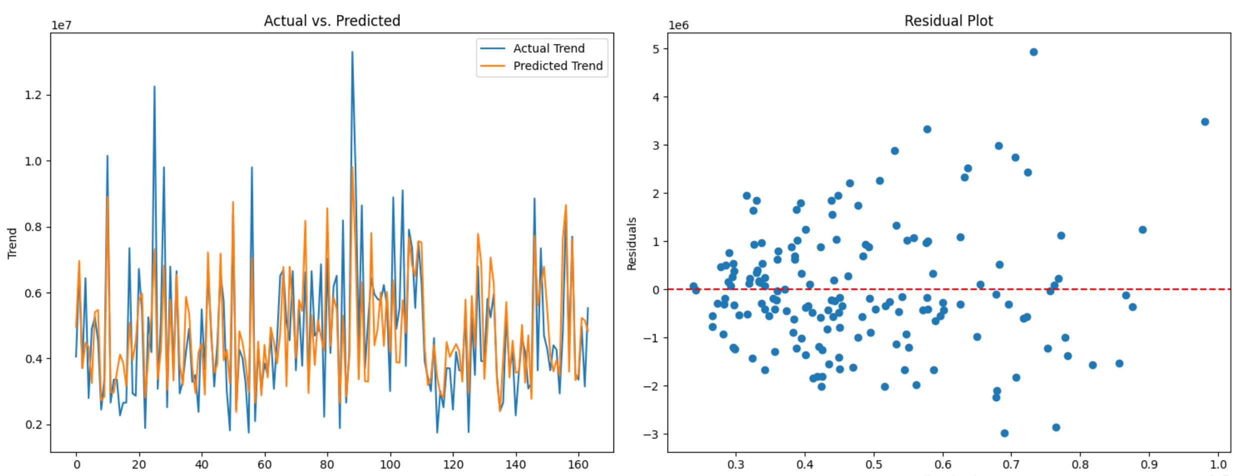
Results for Lasso Reg :
Mean Square Error : 1523021266688.3394
R2 Score : 0.6463346705594011



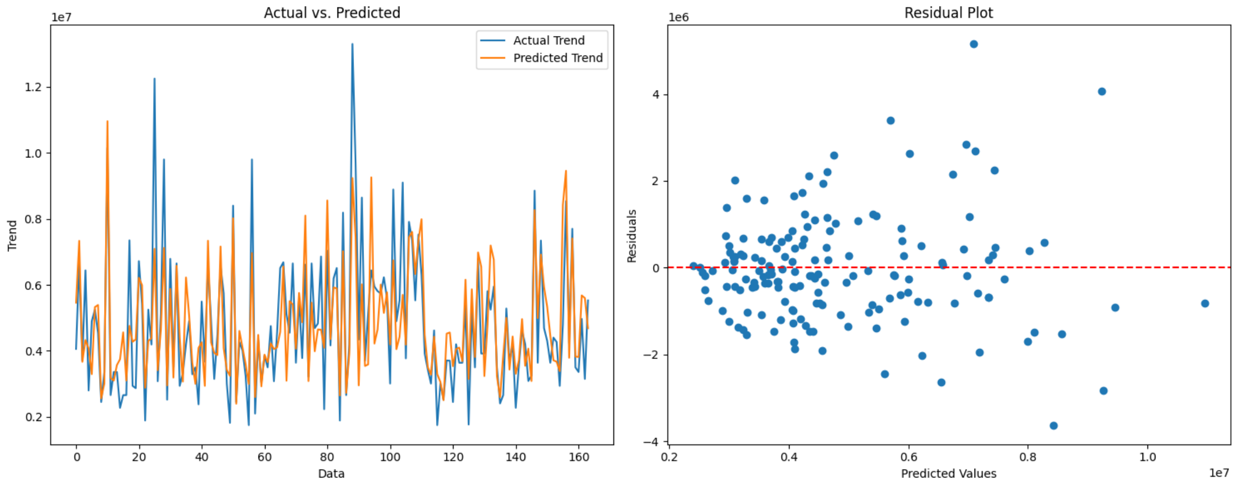
Results for Ridge Reg :
Mean Square Error : 1525354840593.371
R2 Score : 0.6457927843743896



Results for LGBM Reg :
Mean Square Error : 1602977666834.8694
R2 Score : 0.6277677554301107



Results for Cat Boost :
Mean Square Error : 1532351539182.4873
R2 Score : 0.6441680600414872



Inference:

- Lasso Reg, Ridge Reg, and Cat Boost models have relatively higher accuracy compared to other models.
- Support Vector Regressor has a negative accuracy, indicating poor performance.
- Random Forest Regressor, XGBoost, and LGBM Reg also have moderate accuracy.

Model Evaluation with Insights

Identifying the best model based on R2 score

In [125…

```
best_model_name = max(acc, key=acc.get)
best_model = models[best_model_name]
```

Train the best model on the entire dataset

In [126…

```
best_model.fit(X, y)
```

Out[126]:

```
▾ Lasso
Lasso()
```

Evaluate the best model on the test set

In [127…

```
y_pred_best = best_model.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
```

Provide insights

In [128…

```
print(f"Best Model - Name: {best_model_name}")
print(f"Best Model - Mean Squared Error: {mse_best}")
print(f"Best Model - R2 Score: {r2_best}")
```

```
Best Model - Name: Lasso Reg
Best Model - Mean Squared Error: 1406633275222.0205
Best Model - R2 Score: 0.6733614746134041
```

Accuracy 🎯

In [179…

```
data = pd.DataFrame.from_dict(acc, orient='index', columns=['Accuracy'])
data
```

Out[179]:

	Accuracy
Random Forest Regressor	0.562454
Gradient Boost Regressor	0.626797
XGBoost	0.515826
XGRF Regressor	0.512877
Support Vector regressor	-0.036173
Lasso Reg	0.646335
Ridge Reg	0.645793
LGBM Reg	0.627768
Cat Boost	0.644168

📊 Inference:

- Random Forest Regressor achieved an accuracy of 0.562454, indicating moderate performance.
- Gradient Boost Regressor performed relatively better with an accuracy of 0.626797.
- XGBoost showed comparable performance with an accuracy of 0.515826.
- XGRF Regressor had the lowest accuracy among the models at 0.512877.
- Support Vector Regressor had a negative accuracy of -0.036173, indicating poor performance on the given regression problem.
- Lasso Reg showed **good performance** with an accuracy of 0.646335.
- Ridge Reg achieved a **similar accuracy** of 0.645793, indicating its effectiveness in the regression task.
- LGBM Reg performed well with an accuracy of 0.627768.
- Cat Boost also showed **promising results** with an accuracy of 0.644168.