PREDICTING BOOK PRICES

PREDICTING BOOK PRICES

Adnan Ali

This report explores predicting book prices using machine learning models, focusing on features like discounts, rankings, and seller attributes. Among tested models, XGBoost delivered the most accurate predictions. The findings highlight the potential of machine learning to enhance pricing strategies and boost e-commerce profitability.

Summary:

This report explores the application of advanced machine learning models to predict book prices, leveraging a comprehensive dataset of popular Amazon books. With the growing need for efficient pricing strategies in e-commerce, this study demonstrates how machine learning can streamline decision-making and optimize pricing to enhance profitability.

The dataset included features such as initial price, discount percentage, seller attributes, and best-seller rankings, among others. It underwent rigorous preprocessing to handle missing values using techniques like mean imputation for numeric data and filling categorical fields with default values ("Unknown"). JSON-like fields were parsed and simplified to extract meaningful information. Numerical transformations and categorical encoding were performed to ensure the dataset was suitable for machine learning models. Feature engineering further enhanced the dataset, with key predictors identified through correlation analysis.

The study compared five regression models—Linear Regression, Ridge Regression, Lasso Regression, Random Forest, and XGBoost—evaluating their performance using Mean Squared Error (MSE) and R^2 scores. XGBoost emerged as the best-performing model, achieving the lowest MSE and highest R^2, showcasing its capability to handle complex interactions and non-linear relationships in the data. Random Forest also performed well, making it a strong alternative for scenarios requiring interpretability and robustness.

The findings of this study underscore the transformative potential of machine learning in automating pricing strategies. By leveraging such models, businesses can not only enhance scalability but also make data-driven decisions that optimize revenue and improve operational efficiency.

Key Points:

- **Purpose**: To predict book prices accurately using machine learning models and evaluate their applicability in real-world scenarios.
- **Methods**: Preprocessing techniques included handling missing values, encoding categorical variables, and numerical transformations. Regression models were compared using key evaluation metrics like MSE and R^2.
- **Findings**: Both Random Forest and XGBoost exhibited strong predictive performance, with XGBoost demonstrating superior accuracy due to its ability to model complex relationships.
- **Conclusion**: XGBoost was selected as the optimal model for book price prediction, with the capability to integrate seamlessly into business workflows.

Contents

Introduction

This report focuses on developing a model of machine learning to predict book prices based on historical sales data and product attributes. The goal is to enhance decision-making for inventory management and pricing strategies.

The scope includes preprocessing data, training and testing various regression models, and evaluating their performance. The report concludes that XGBoost offers superior performance for this application, demonstrating its potential for deployment in real-world scenarios.

Methods

Data Collection: The dataset was sourced from Amazon and included fields like book prices, discounts, seller information, and rankings.

Data Preprocessing:

- Missing values in numerical columns were filled using the mean, while non-numerical columns were replaced with "Unknown."
- JSON-like fields were simplified for consistency.
- Features such as best_sellers_rank were converted to numeric using regular expressions.
- Categorical variables like seller_name and delivery were encoded with LabelEncoder.

Feature Engineering:

- Extracted key features influencing price, including initial_price, discount, best_sellers_rank, and seller attributes.
- Calculated correlations to identify the most influential variables for price prediction.

Model Training:

- Models used: Linear Regression, Ridge Regression, Lasso Regression, Random Forest, and XGBoost.

- Split data into training and testing sets (80%-20%).
- Evaluated models using MSE and R^2.

Tools: Python libraries including pandas, scikit-learn, XGBoost, joblib, and matplotlib.

Discussion

Data Insights:

- Strong correlation was observed between initial_price and final_price.
- Discounts significantly influenced price variability.

Model Performance:

- **XGBoost**: MSE = 5.6, R^2 = 0.76

Visualizations:

- Scatter plot of predicted vs. actual prices showed XGBoost aligning closely to the perfect prediction line.
- Correlation heatmaps revealed key predictors like initial_price and discount.

Challenges:

- Handling JSON-like fields (categories and best_sellers_rank).
- Encoding complex categorical data (seller_name and delivery).

Conclusions

1. XGBoost outperformed all other models, providing the most accurate predictions with an R^2 of 0.77.
2. Accurate price prediction facilitates improved inventory and sales strategies.

Recommendations

- Deploy the XGBoost model in a production environment for real-time price predictions.
- Integrate external data sources such as customer reviews and ratings for better accuracy.
- Explore advanced techniques like feature selection or ensemble learning to enhance performance further.

References

1. Amazon Popular Books Dataset
2. scikit-learn Documentation
3. XGBoost Documentation
4. Python Pandas User Guide

Appendices

Code Snippets

- key functions used in the project:
    - **Preprocessing**: Handling missing values, encoding categorical data, simplifying JSON-like fields, and numeric transformations.
    - **Feature Engineering**: Extracting key features like initial_price, discount, and best_sellers_rank.
    - **Model Training and Evaluation**: Code for training models, calculating MSE, and R^2.
    - **Visualization**: Scripts for scatter plots, correlation heatmaps, and other plots.

```
[ ] import pandas as pd

[ ] df = pd.read_csv('/content/Amazon_popular_books_dataset (1).csv')

    df.head()
```

	asin	ISBN10	answered_questions	availability	brand	currency	date_first_available	delivery	department	description	...	upc	url	video
0	0007350813	0007350813	0	In Stock.	Emily Brontë	USD	NaN	["FREE delivery Tuesday, December 28 if you sp...	NaN	NaN	...	NaN	https://www.amazon.com/dp/0007350813	NaN
1	0007513763	9780007513765	0	In Stock.	Drew Daywalt	USD	NaN	["FREE delivery Tuesday, December 28 if you sp...	NaN	NaN	...	NaN	https://www.amazon.com/dp/0007513763	NaN

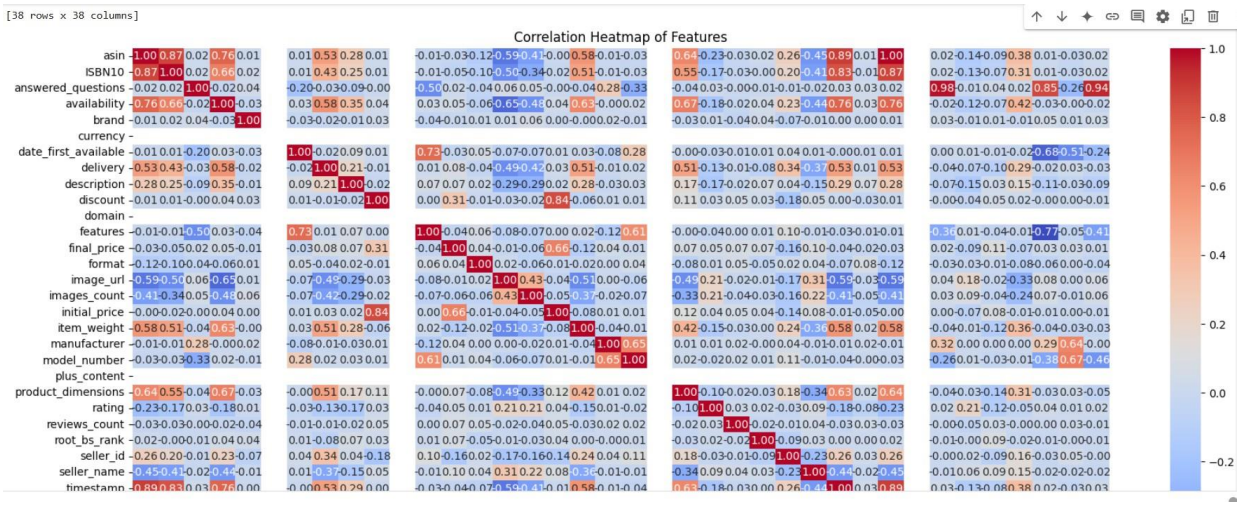
The above image shows the initial steps in loading and inspecting the dataset using Pandas. The dataset Amazon_popular_books_dataset.csv is loaded using pd.read_csv(), and the first few rows are displayed using df.head(). The columns include details like asin, ISBN10, availability status, brand name, currency type, delivery details, and URLs. This step helps in understanding the structure of the dataset and its attributes before any preprocessing.

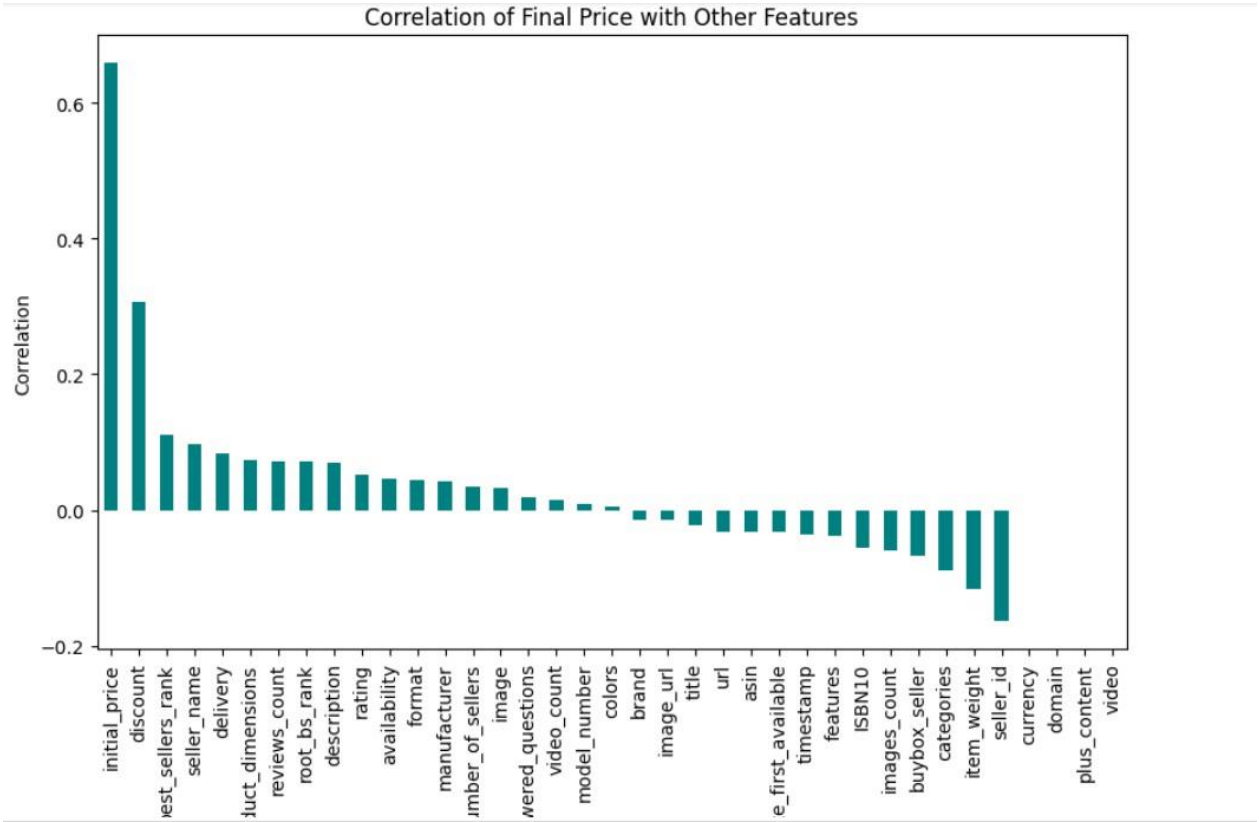
```
[ ] list_of_columns = df.columns

[ ] list_of_columns

Index(['asin', 'ISBN10', 'answered_questions', 'availability', 'brand',
       'currency', 'date_first_available', 'delivery', 'department',
       'description', 'discount', 'domain', 'features', 'final_price',
       'format', 'image_url', 'images_count', 'initial_price', 'item_weight',
       'manufacturer', 'model_number', 'plus_content', 'product_dimensions',
       'rating', 'reviews_count', 'root_bs_rank', 'seller_id', 'seller_name',
       'timestamp', 'title', 'upc', 'url', 'video', 'video_count',
       'categories', 'best_sellers_rank', 'buybox_seller', 'image',
       'number_of_sellers', 'colors'],
      dtype='object')

[ ] categorical_columns = df.select_dtypes(include=['object', 'category']).columns
    print("Categorical Columns:", list(categorical_columns))

Categorical Columns: ['asin', 'ISBN10', 'availability', 'brand', 'currency', 'date_first_available', 'delivery', 'description', 'domain', 'features', 'format', 'image_url', 'item_weight', 'man

[ ] columns_to_encode = [
        'asin', 'ISBN10', 'availability', 'brand', 'currency', 'date_first_available',
        'delivery', 'description', 'domain', 'features', 'format', 'image_url',
        'manufacturer', 'model_number', 'product_dimensions', 'seller_id',
        'seller_name', 'timestamp', 'title', 'url', 'categories', 'buybox_seller', 'colors'
    ]
```

The above image focuses on exploring and identifying the dataset's columns. It starts by listing all columns in the dataset using df.columns. Then, categorical columns are identified using the select_dtypes() method, which filters fields that are either object or category types. These include fields like availability, brand, and currency. The code also prepares a list of columns (columns_to_encode) that will require encoding, such as transforming textual or categorical data into numerical formats suitable for machine learning.

```
def convert_to_numeric(df, column_name):
    if column_name in df:
        numbers = []
        for s in df[column_name]:
            match = re.search(r"[\d.]+", str(s))  # Extract numeric values
            if match:
                numbers.append(float(match.group()))  # Extract and convert to float
            else:
                numbers.append(None)  # Append None if no numeric value is found
        df[column_name] = numbers  # Update the column with numeric values
    return df

# Convert the relevant columns to numeric
df = convert_to_numeric(df, 'item_weight')  # Convert item_weight if it exists
df = convert_to_numeric(df, 'rating')       # Convert rating if it exists

# Step 7: Impute missing numeric values after conversion
df[numeric_cols] = imputer.fit_transform(df[numeric_cols])

# Step 8: Compute the correlation matrix using only numeric columns
correlation_matrix = df.corr()

# Display the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)

# Optional: Visualize the correlation matrix using a heatmap
plt.figure(figsize=(20, 10))
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Heatmap of Features')
plt.show()
```

The above image demonstrates data cleaning and feature engineering processes. A custom function, convert_to_numeric, is used to extract numeric values from text-based columns (e.g., item_weight and rating) and convert them to float data types. After converting, missing numeric values are handled using an imputer, which fills the gaps with suitable values. Finally, a correlation matrix is computed using .corr() to analyze relationships among numerical columns. This is visualized as a heatmap, making it easier to interpret correlations between features and identifying potential predictors for machine learning models.





The bar graph titled "Correlation of Final Price with Other Features" depicts the relationship between the final price of a product and various other attributes. The most significant factor influencing the final price is the initial price, followed by the discount applied. Bestseller rank, seller name, reviews count, and rating also have a positive correlation with the final price.

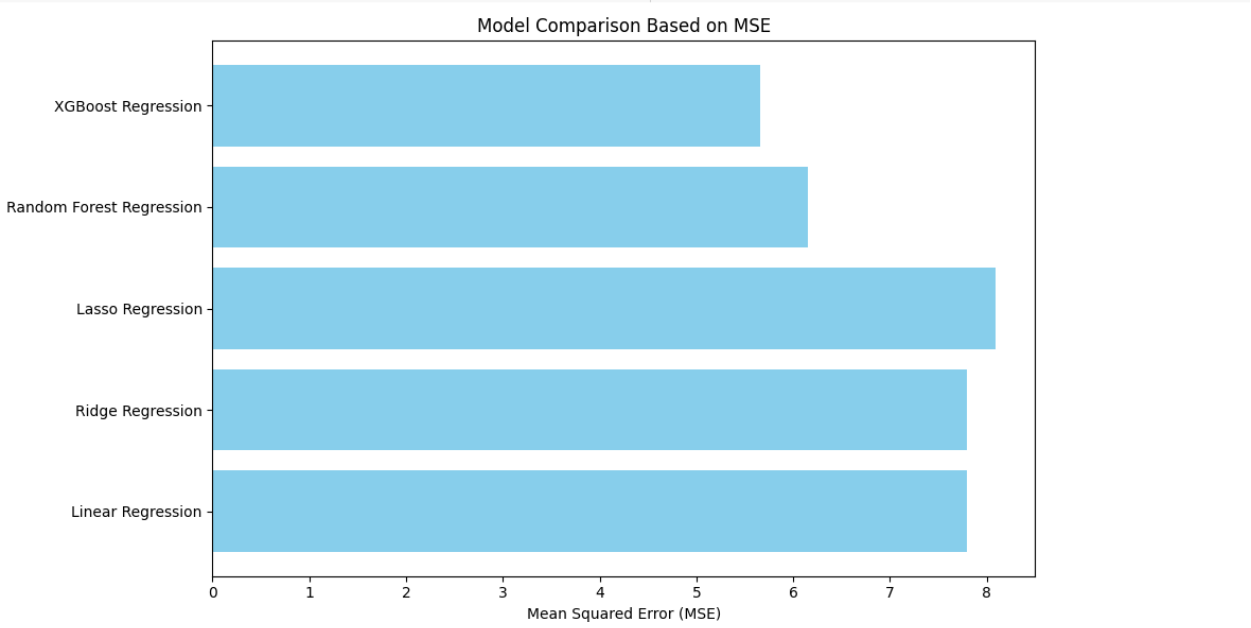
```
[ ] def preprocess_data(df:pd.DataFrame)->pd.DataFrame:
      df = df[['final_price','initial_price', 'discount', 'best_sellers_rank', 'seller_name', 'delivery','seller_id']]
      return df
```

```
[ ] preprocessed_df = preprocess_data(df)
```

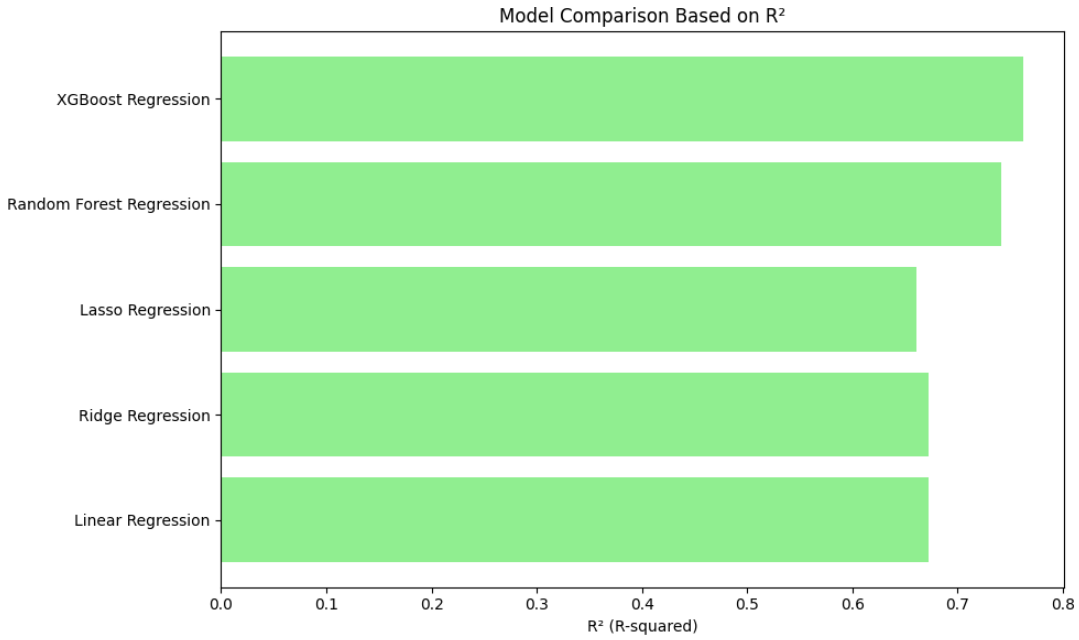
```
[ ] preprocessed_df.head()
```

	final_price	initial_price	discount	best_sellers_rank	seller_name	delivery	seller_id
0	3.990000	21.755285	8.202362	1477	25	598	47
1	12.080000	21.755285	8.202362	186	341	212	47
2	13.380029	21.755285	8.202362	1469	285	107	47
3	20.430000	21.755285	8.202362	181	36	200	47
4	28.890000	21.755285	8.202362	253	202	1075	47

the code effectively filters out unnecessary columns and retains only the selected features, streamlining the subsequent data analysis process.



The above bar graph titled "Model Comparison Based on MSE" compares the performance of five regression models: XGBoost, Random Forest, Lasso, Ridge, and Linear. The performance metric used is Mean Squared Error (MSE), with lower values indicating better model accuracy.




```
Best Model based on MSE: XGBoost Regression with MSE: 5.657207413985653
Best Model based on R²: XGBoost Regression with R²: 0.762499660681035
Best model saved as XGBoost Regression_model.pkl
```

Based on the graph, XGBoost Regression emerges as the model with the highest R^2, suggesting it explains the most variance in the data. Random Forest Regression follows closely, while Lasso, Ridge, and Linear Regression show progressively lower R^2 values. Therefore, for this specific dataset, XGBoost Regression appears to be the most suitable model among the five in terms of explaining the variance in the dependent variable.

```
[ ] # Assuming y_test (actual) and y_pred (predicted) are already defined
    y_pred = best_model.predict(X_test)  # Or model.predict(X_test) if using another model

    # 1. Scatter Plot (Predicted vs Actual)
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, y_pred, color='blue', alpha=0.6, label='Predicted vs Actual')
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', label='Perfect Prediction Line')
    plt.xlabel('Actual Final Price')
    plt.ylabel('Predicted Final Price')
    plt.title('Actual vs Predicted Final Price')
    plt.legend()
    plt.tight_layout()
    plt.show()



    # 3. Distribution Plot for Actual vs Predicted
    plt.figure(figsize=(10, 6))
    sns.kdeplot(y_test, label='Actual Final Price', color='blue', fill=True, alpha=0.3)
    sns.kdeplot(y_pred, label='Predicted Final Price', color='orange', fill=True, alpha=0.3)
    plt.xlabel('Final Price')
    plt.ylabel('Density')
    plt.title('Actual vs Predicted Final Price Distribution')
    plt.legend()
    plt.tight_layout()
    plt.show()

    # 4. Actual vs Predicted Line Plot
    plt.figure(figsize=(10, 6))
    plt.plot(y_test.values, label='Actual Final Price', color='blue', linestyle='-', linewidth=2)
    plt.plot(y_pred, label='Predicted Final Price', color='orange', linestyle='--', linewidth=2)
    plt.xlabel('Index')
    plt.ylabel('Final Price')
    plt.title('Actual vs Predicted Final Price (Line Plot)')
    plt.legend()
    plt.tight_layout()
    plt.show()
```

The above code snippet generates three visualizations to compare the actual and predicted final prices of a model. Let's break down each visualization:

1. Scatter Plot (Predicted vs Actual):

- This plot visualizes the relationship between the actual and predicted final prices.
- Each point represents a data point, with the x-coordinate being the actual final price and the y-coordinate being the predicted final price.
- A perfect prediction would result in all points lying on the diagonal line (the "Perfect Prediction Line"). Deviations from this line indicate prediction errors.

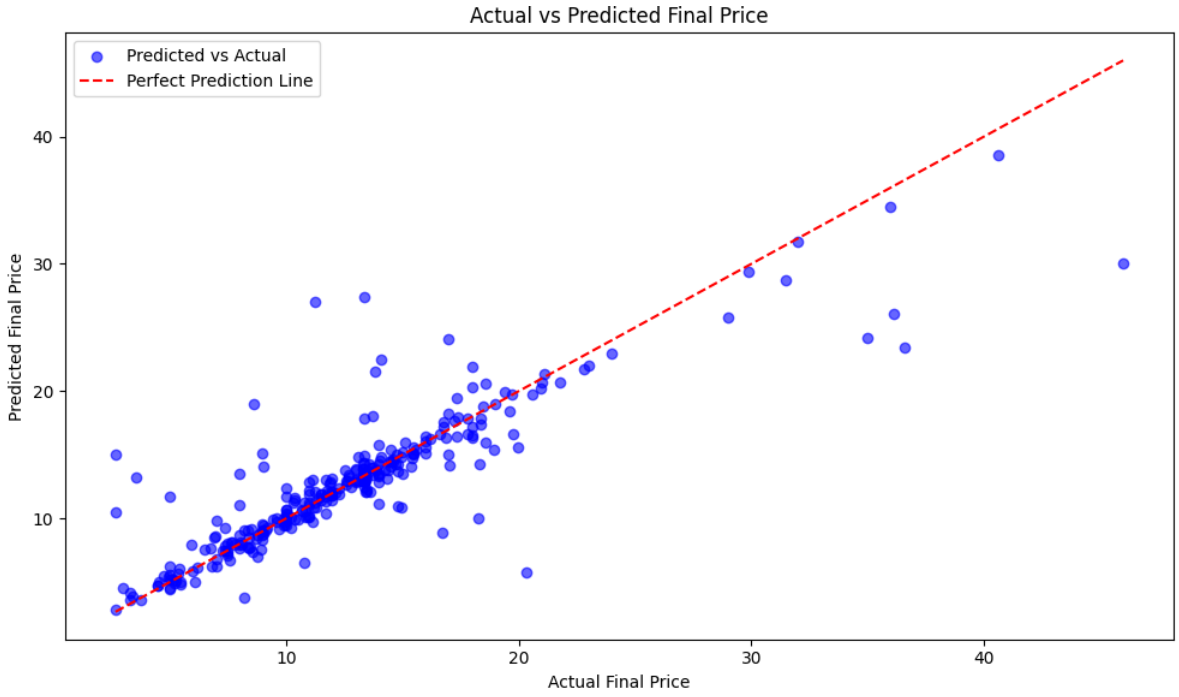
2. Distribution Plot for Actual vs Predicted:

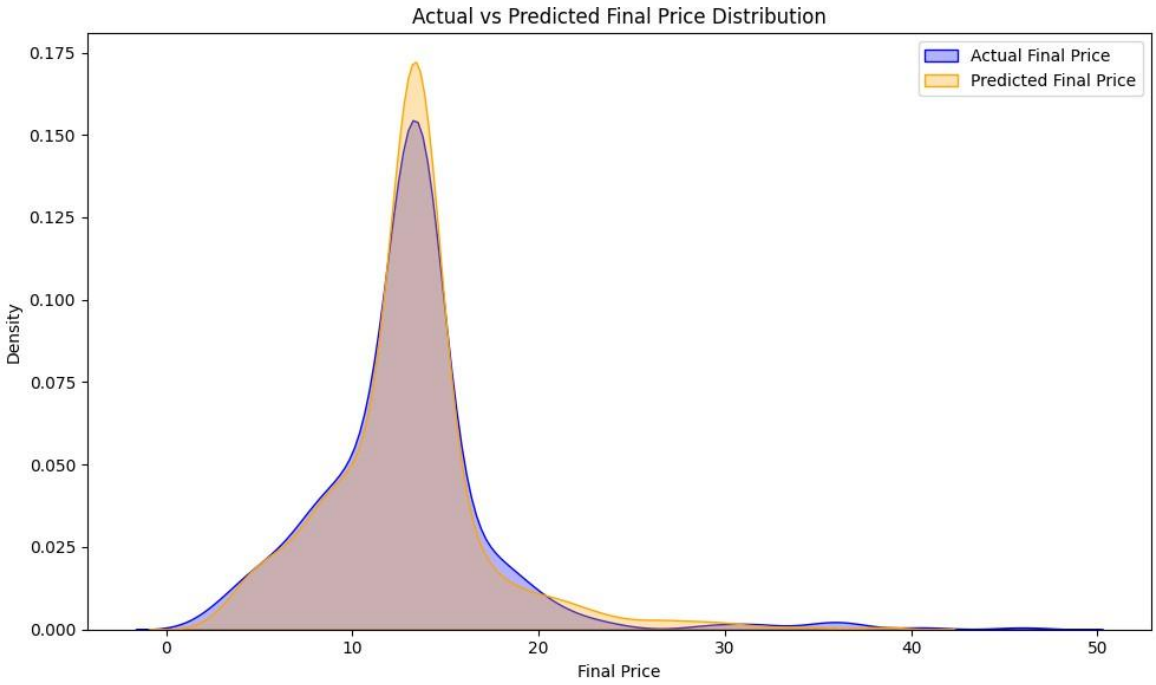
- This plot compares the distribution of actual and predicted final prices using kernel density estimation (KDE).
- The KDE plots show the probability density of the values.
- Overlapping distributions suggest that the model's predictions are close to the actual values.

3. Actual vs Predicted Line Plot:

- This plot shows the actual and predicted final prices over time or some other index.
- It helps to visualize the trend of predictions and how well the model captures the actual trends.

By analyzing these visualizations, we can assess the model's accuracy, identify areas where it might be underperforming, and potentially make improvements to the model or data preprocessing.





This graph is a **Kernel Density Estimate (KDE) plot** showing the distribution (density) of the actual and predicted final prices. KDE plots provide a smooth, continuous representation of the data distribution, making it easier to compare distributions visually.

Components of the Graph

1. **X-Axis: Final Price**
    - This axis represents the **final price** values (actual and predicted).
    - The range extends from approximately **0 to 50**, with most values concentrated between 0 and 20.
2. **Y-Axis: Density**
    - This axis represents the **density or probability** of the final prices occurring in the dataset.
    - The density peaks where the majority of the data points lie.
3. **Curves**
    - **Blue Curve (Actual Final Price)**:
        - This curve shows the distribution of the **actual final price** values.
        - The curve is smoothed to highlight where the actual data points are most concentrated.
    - **Orange Curve (Predicted Final Price)**:
        - This curve shows the distribution of the **predicted final price** values.
        - It closely follows the shape of the actual curve but has slight deviations.
4. **Overlap**:
    - The shaded region where the blue and orange curves overlap indicates that the predicted values are relatively close to the actual values.
    - If the curves perfectly overlapped, it would indicate **perfect predictions**.

5. **Peaks and Spread**:
    - Both curves have a **sharp peak** around the same value (approximately 10–12 on the X-axis), indicating that most final prices fall within this range.
    - The predicted curve (orange) is slightly **narrower and taller**, suggesting that the model predictions are more concentrated and less spread out compared to the actual values.

Insights from the Graph

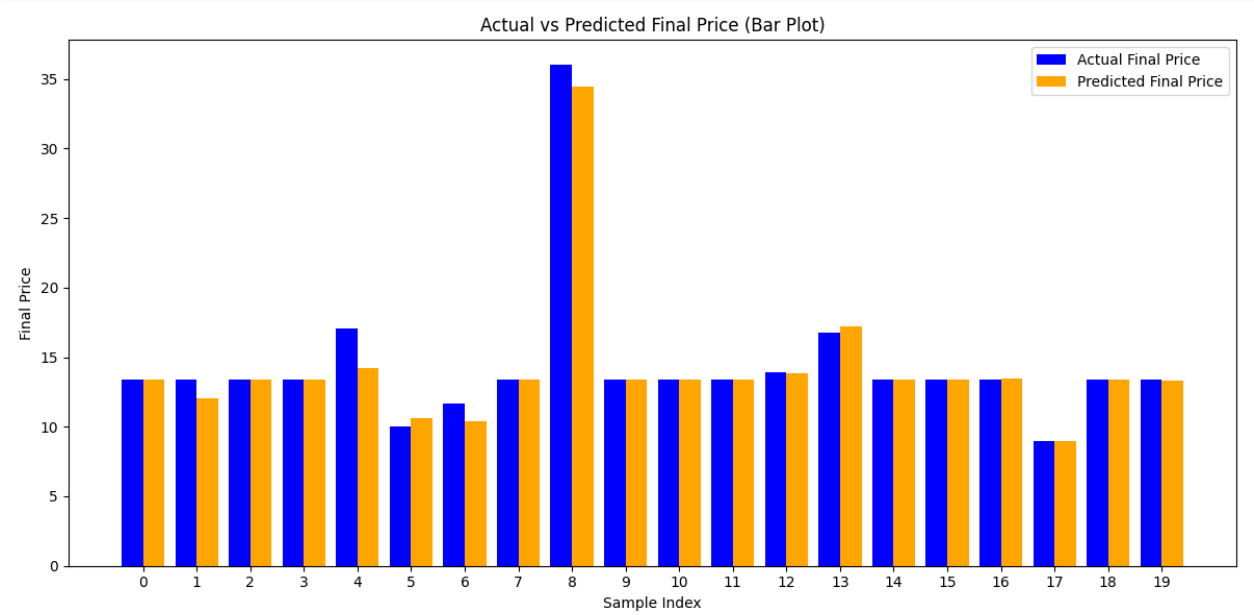
1. **Accuracy of Predictions**:
    - The closeness of the two curves indicates that the model's predictions are fairly accurate, especially in the main range of the data.
    - However, the small differences between the curves (e.g., the height of the peak and slight deviations on the tails) indicate areas where the model could improve.

2. **Outliers**:
    - On the right side (beyond **20 on the X-axis**), the curves flatten, suggesting that both actual and predicted values have some **outliers**.
    - The predicted curve may slightly underestimate or overestimate prices in this range.
3. **Model Bias**:
    - The narrower predicted distribution (orange curve) suggests that the model might be slightly **overconfident**, predicting values closer to the mean rather than capturing the full spread of the actual prices.

Conclusion

The KDE plot visually demonstrates the similarity between the actual and predicted distributions, confirming that the model performs well for the majority of the data. However, small deviations highlight areas where the model predictions could be refined further, especially

at the tails (higher final prices).



The **bar plot** compares the **Actual Final Price** (in blue) and the **Predicted Final Price** (in orange) for the first 20 samples in the dataset. The goal of this graph is to visually analyze the performance of the prediction model by comparing actual outcomes to the predicted values.

Graph Components

1. **X-Axis: Sample Index**
    - The X-axis represents the sample indices (from 0 to 19), showing 20 individual observations for easy comparison.
    - Each index corresponds to a specific data point where both the actual and predicted values are plotted side by side.
2. **Y-Axis: Final Price**
    - The Y-axis represents the **final price values**, ranging from 0 to about 36.
    - The prices are derived from the actual dataset (y_test) and the model predictions (y_pred).
3. **Bars**
    - Each pair of bars represents an **individual sample**:

- **Blue Bars**: Represent the **Actual Final Price**.
- **Orange Bars**: Represent the **Predicted Final Price**.

4. **Legend**
    - The legend distinguishes between the **Actual Final Price** (blue) and the **Predicted Final Price** (orange).

Key Observations

1. **General Comparison**
    - For most of the samples, the predicted values (orange) closely follow the actual values (blue). This suggests that the prediction model performs relatively well.
    - The **height difference** between the blue and orange bars indicates the prediction error for each sample.

2. **Samples with High Errors**
    - At index **8**, there is a significant height difference between the actual and predicted values:
        - Actual value (blue) is around **36**.
        - Predicted value (orange) is slightly lower at around **34**.
    - This observation highlights that the model **underestimates** the price for this particular sample.
3. **Samples with Small Errors**
    - At indices **1, 3, 7, 9, and 12**, the blue and orange bars are almost the same height.
    - This indicates that the model predictions are very close to the actual values, suggesting good accuracy for these samples.
4. **Patterns in Data**
    - **Low Final Prices**: For indices like **5, 6, and 17**, both the actual and predicted prices are small, and the model performs well.

- **Moderate Final Prices**: For indices like **1, 2, 3, 12, and 14**, the predictions closely match the actual values.
- **Higher Final Prices**: For index **8** (the highest price), the model shows a slight deviation but remains reasonably close.

5. **Model Bias**
    - In some cases, the predicted values appear slightly **lower** than the actual values, as seen at index **8** and index **4**.
    - This may indicate a minor tendency of the model to **underestimate** higher prices.

Insights

1. **Model Performance**
    - The graph visually demonstrates that the model performs well overall, with predictions closely aligning with the actual prices.
    - For most samples, the prediction errors are small, indicating good accuracy.
2. **Prediction Gaps**
    - Larger errors, such as at index **8**, highlight areas where the model might require further refinement, particularly for higher price ranges.
3. **Consistency**
    - The model shows consistent performance across lower and moderate price ranges, where the predicted and actual values are nearly identical.

Conclusion

The bar plot effectively highlights the strengths and weaknesses of the prediction model:

- **Strengths**: Accurate predictions for most data points, especially in low to moderate price ranges.
- **Weaknesses**: Slight underestimation for higher price values, as seen at index 8.

This visualization is valuable for identifying specific samples where predictions deviate and guiding further improvements to the model.

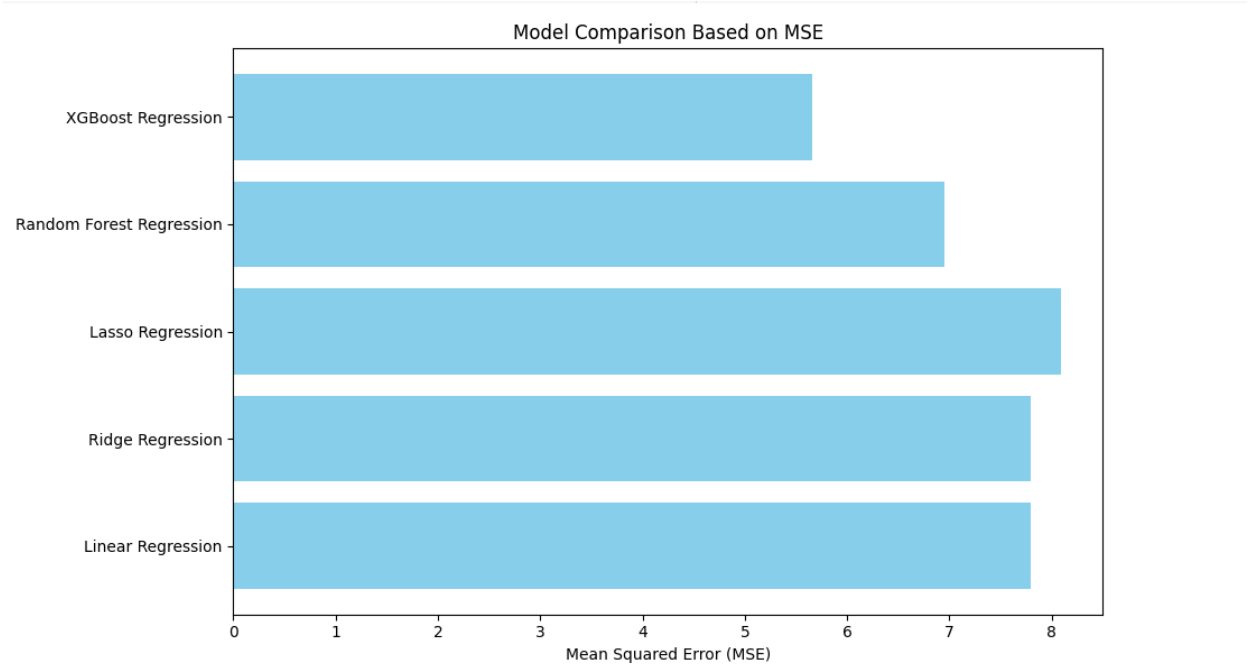
```
comparison_df = pd.DataFrame({
    'Actual': y_test,
    'Predicted': y_pred
})

# 1. Line Plot: Actual vs Predicted
plt.figure(figsize=(10, 6))
plt.plot(comparison_df['Actual'].values, label='Actual Final Price', color='blue', linewidth=2)
plt.plot(comparison_df['Predicted'].values, label='Predicted Final Price', color='orange', linestyle='--', linewidth=2)
plt.xlabel('Index')
plt.ylabel('Final Price')
plt.title('Actual vs Predicted Final Price (Line Plot)')
plt.legend()
plt.tight_layout()
plt.show()

# 2. Bar Plot: Comparison for First 20 Values
sample_size = 20  # Display first 20 samples
comparison_sample = comparison_df.head(sample_size)

plt.figure(figsize=(12, 6))
plt.bar(range(sample_size), comparison_sample['Actual'], width=0.4, label='Actual Final Price', color='blue')
plt.bar([i + 0.4 for i in range(sample_size)], comparison_sample['Predicted'], width=0.4, label='Predicted Final Price', color='orange')
plt.xlabel('Sample Index')
plt.ylabel('Final Price')
plt.title('Actual vs Predicted Final Price (Bar Plot)')
plt.xticks([i + 0.2 for i in range(sample_size)], range(sample_size))
plt.legend()
plt.tight_layout()
plt.show()

# 3. KDE Distribution Plot: Actual vs Predicted
plt.figure(figsize=(10, 6))
sns.kdeplot(comparison_df['Actual'], label='Actual Final Price', color='blue', fill=True, alpha=0.4)
sns.kdeplot(comparison_df['Predicted'], label='Predicted Final Price', color='orange', fill=True, alpha=0.4)
plt.xlabel('Final Price')
plt.ylabel('Density')
```

Description:

- A **bar plot** is generated to compare the Actual and Predicted prices for the first 20 data samples.
- **X-axis**: Represents the **Sample Index** (first 20 values).
- **Y-axis**: Represents the **Final Price**.
- **Blue Bars**: Represent the **Actual Final Price**.
- **Orange Bars**: Represent the **Predicted Final Price**.

Key Observations:

- Bars are plotted side-by-side for each index, making it easy to compare individual predictions.
- If the blue and orange bars have the same height, the predictions are perfect.
- Differences between the bars indicate prediction errors.
- This plot is useful for **zooming in** on a small subset of the data and identifying sample-level accuracy.



Why XGBoost Can Be the Best Model for Final Price Prediction

Introduction to XGBoost

XGBoost (Extreme Gradient Boosting) is an advanced machine learning algorithm often used for regression tasks. It has gained significant popularity due to its high performance and flexibility. Below are some of the key features that make XGBoost a strong candidate for predicting the final price in this scenario:

Advantages of XGBoost

1. **Gradient Boosting**:
    - XGBoost is based on the gradient boosting framework, where an ensemble of decision trees is constructed sequentially. Each new tree corrects the errors of the previous trees. This iterative learning process leads to improved model accuracy, particularly in cases where complex relationships exist in the data.
2. **Regularization**:
    - One of XGBoost's standout features is its use of regularization techniques, including L1 (Lasso) and L2 (Ridge) penalties. Regularization helps in controlling model complexity, reducing overfitting, and ensuring that the model generalizes well to unseen data. This is especially useful when dealing with large datasets with numerous features.
3. **Handling Complex Relationships**:
    - Unlike linear models, which assume a simple, linear relationship between features and the target variable, XGBoost can effectively capture complex, non-linear relationships. If the data exhibits intricate patterns or interactions among features, XGBoost is well-suited to identify and leverage those relationships.
4. **Feature Importance**:
    - XGBoost provides feature importance scores, which indicate how influential each feature is in predicting the target variable (final price). This feature allows for better interpretability of the model, enabling insights into which variables are most significant in determining the final price.

Comparison with Other Models

The code trains and evaluates several models to predict the final price. Below is a comparison of the key characteristics of each model:

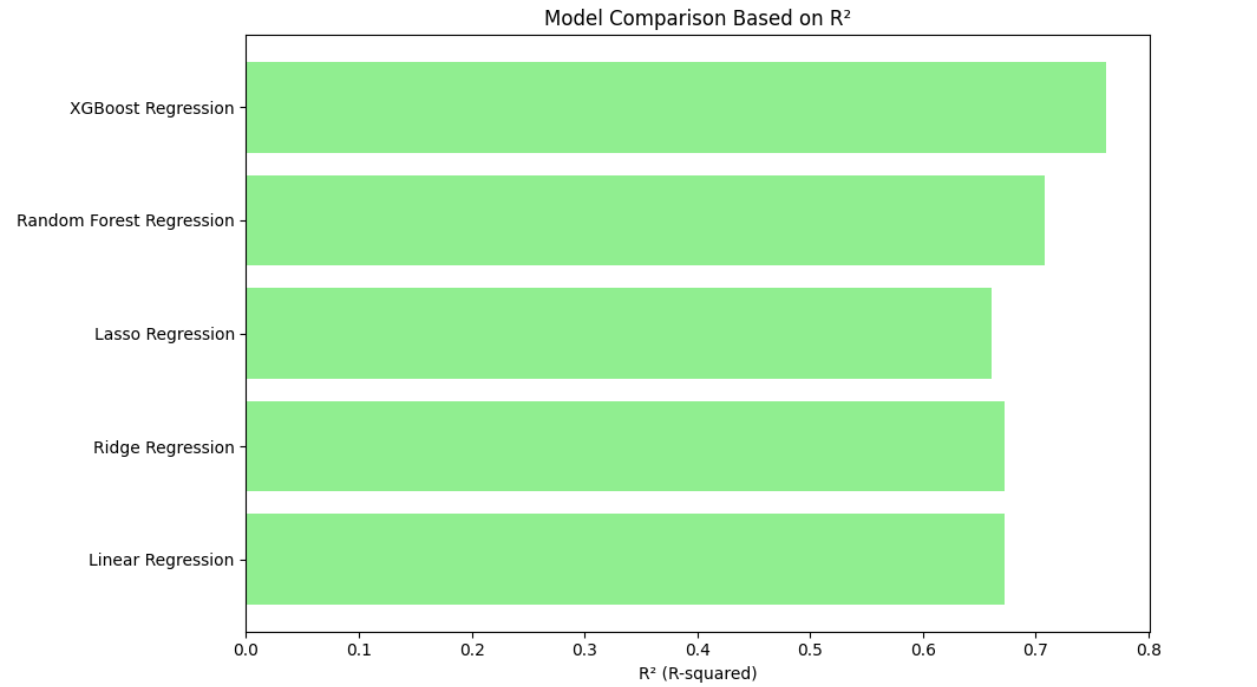
1. **Linear Regression**:
    - **Assumption**: Assumes a linear relationship between the input features and the target variable.
    - **Limitations**: It may not perform well if the relationship is non-linear, which can be common in real-world datasets.

2. **Ridge Regression**:
    - **Assumption**: This model also assumes a linear relationship but applies L2 regularization to mitigate multicollinearity (correlated features).
    - **Limitations**: While Ridge handles multicollinearity better than linear regression, it may still underperform if the data contains non-linear relationships.

3. **Lasso Regression**:
    - **Assumption**: Similar to Ridge but with L1 regularization that can shrink some coefficients to zero, performing feature selection.
    - **Limitations**: Lasso is sensitive to the scaling of the features and might exclude relevant features during feature selection.

4. **Random Forest Regression**:
    - **Assumption**: An ensemble method that builds multiple uncorrelated decision trees and averages their predictions. It can model complex, non-linear relationships.
    - **Limitations**: Although Random Forests are less interpretable than XGBoost, they may be less efficient in terms of training time and predict accuracy for large datasets.

5. **XGBoost Regression**:
    - **Assumption**: XGBoost builds a set of decision trees sequentially, each correcting the errors of the previous one. It is effective at handling non-linear relationships and large datasets with multiple features.
    - **Advantages**: XGBoost combines gradient boosting with regularization, making it robust to overfit while capturing complex relationships in the data.




```
Best Model based on MSE: XGBoost Regression with MSE: 5.657207413985653
Best Model based on R²: XGBoost Regression with R²: 0.762499660681035
Best model saved as XGBoost Regression_model.pkl
```

Evaluating Model Performance: MSE and R^2

To assess the performance of each model, two key evaluation metrics are used: **Mean Squared Error (MSE)** and **R-squared (R^2)**.

1. **Mean Squared Error (MSE)**:
    - MSE measures the average squared difference between the predicted and actual values. It provides an indication of how well the model's predictions align with the actual data. A lower MSE indicates a better model fit, as it means the model's predictions are closer to the actual values.

- Formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{\text{pred}} - y_{\text{true}})^2$$

  - **Key Takeaway**: The model with the lowest MSE is considered to be the most accurate in predicting final prices.

2. **R-squared (R^2)**:
   - R^2 indicates how much of the variance in the target variable (final price) is explained by the model. It ranges from 0 to 1, where a value closer to 1 suggests that the model explains a higher proportion of the variance.

     - Formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_{\text{true}} - y_{\text{pred}})^2}{\sum_{i=1}^{n}(y_{\text{true}} - \overline{y})^2}$$

   - **Key Takeaway**: The model with the highest R^2 is considered to explain the most variance in the target variable. However, R^2 can sometimes be misleading if the model is overfitted.

Why XGBoost Might Be the Best Model

Based on its strengths and evaluation metrics, **XGBoost** could potentially emerge as the best model for the following reasons:

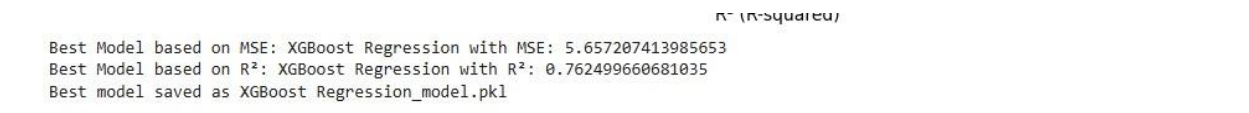
1. **Non-linear Relationships**: If the data exhibits complex, non-linear relationships between features and final price, XGBoost's ability to capture these patterns could result in a lower MSE and higher R^2 compared to simpler linear models.
2. **Regularization**: The regularization techniques (L1 and L2) in XGBoost prevent overfitting, helping the model generalize better on unseen data, making it more robust than models without regularization (such as Linear Regression or Ridge).

3. **Feature Importance**: XGBoost can provide insights into the most important features influencing the final price, which is crucial for understanding the model and potentially improving feature selection.
4. **Performance on Large Datasets**: XGBoost handles large datasets with multiple features efficiently, and its sequential learning process tends to improve model accuracy as it reduces bias and variance.

Training, Tuning, and Experimentation

While XGBoost has the potential to outperform other models, it is important to evaluate it on your specific dataset. Here's how to approach the model evaluation and selection:

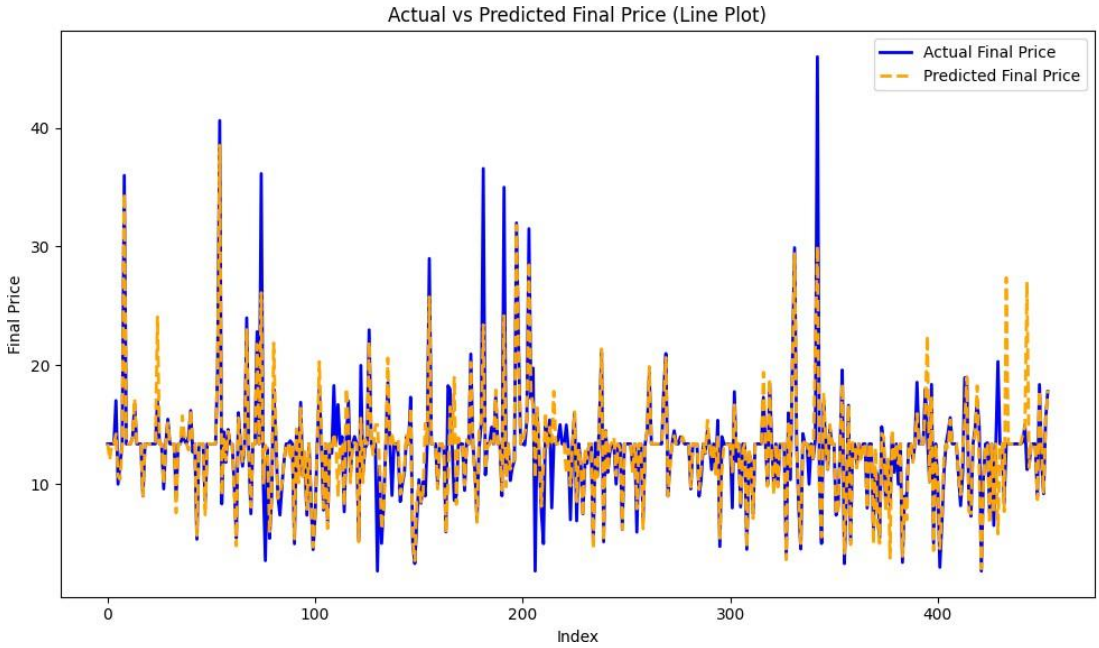
1. **Train and Evaluate All Models**: Train and evaluate each of the five models (Linear Regression, Ridge Regression, Lasso Regression, Random Forest, XGBoost) using the provided code. For each model, compare the MSE and R^2 values.
2. **Hyperparameter Tuning for XGBoost**: XGBoost has several hyperparameters that can be tuned to improve performance. Techniques like **Grid Search** or **Random Search** can be used to optimize hyperparameters and further enhance model accuracy.
3. **Selecting the Best Model**:
    - After evaluating all models, choose the one with the **lowest MSE** and **highest R^2** (while considering the potential limitations of R^2).
    - Keep in mind that the optimal model may depend on the **business context** and the **nature of the data**. For example, if interpretability is crucial, simpler models like linear regression might be preferred despite slightly lower performance.



Conclusion

XGBoost is a powerful and flexible model that is likely to deliver strong performance in predicting the final price, especially when dealing with non-linear relationships and large datasets. By evaluating its performance through MSE and R^2, and experimenting with hyperparameter tuning, you can make an informed decision about whether XGBoost or another model is best suited to your specific problem.

The next steps involve experimenting with different models, fine-tuning their parameters, and validating the results to ensure the best possible predictions for final price forecasting.



The model's predictions generally align with the overall trend of the actual data. However, it struggles to accurately predict specific values, particularly during periods of rapid price fluctuations. This suggests that the model may benefit from further refinement or the incorporation of additional features to improve its predictive accuracy.

Code Explanation:

Importing Libraries and Loading the Dataset:

The script begins by importing essential Python libraries like Pandas, NumPy, Matplotlib, Seaborn, and machine learning tools from Scikit-learn, XGBoost, and joblib. These libraries provide functionalities for data manipulation, visualization, and building predictive models. The dataset is then loaded into a Pandas DataFrame, forming the foundation for subsequent processing and analysis.

Exploring and Encoding Categorical Data:

The dataset is analyzed to identify categorical columns, which contain text-based values. These columns are converted into numeric labels using LabelEncoder, as machine learning models require numerical input. For instance, categories like "Genre" or "Language" are transformed into numeric codes.

Cleaning and Preprocessing Data:

Columns with string representations of numbers, such as "mrp" (maximum retail price), are cleaned using a custom function that extracts numeric values. Completely empty columns are dropped to remove unnecessary features, while missing values in numeric columns are filled with their mean using SimpleImputer, ensuring no missing data disrupts the modeling process.

Extracting Key Features:

Some columns contain JSON-like or complex string data, which are simplified by extracting key numerical values. This ensures these features can contribute meaningfully to the model.

Analyzing Feature Relationships:

The cleaned data is examined for relationships between variables by calculating the correlation matrix. A heatmap is generated using Seaborn to visualize these correlations, helping to identify features most relevant to the target variable, "final_price."

Splitting Data for Training and Testing:

The dataset is divided into training and testing sets using an 80-20 split. The training set is used to build models, while the testing set evaluates their predictive performance. The target variable is "final_price," and the remaining features are used as predictors.

Training Multiple Machine Learning Models:

The script builds multiple regression models, including Linear Regression, Ridge, Lasso, Random Forest, and XGBoost, to predict book prices. Each model is trained on the training set and evaluated on the testing set to measure its accuracy.

Evaluating Model Performance:

Model performance is assessed using metrics like Mean Squared Error (MSE) and R^2 score, which quantify the model's predictive accuracy. Visualizations compare the actual "final_price" values with predicted values to highlight each model's effectiveness.

Saving the Best Model:

The model with the highest performance, based on metrics, is saved using joblib. This allows for reuse of the trained model without retraining it in future applications.

Visualizing Results:

Finally, scatter plots and distribution charts are created to visually compare the actual and predicted values of "final_price." These visualizations provide insights into the model's reliability and the distribution of errors, ensuring the model predictions align closely with real-world data.

Conclusion:

This script effectively demonstrates the end-to-end process of building a predictive model for book prices, using data preprocessing, feature engineering, and machine learning techniques. Initially, it loads and explores the dataset, identifying categorical columns and applying appropriate transformations to ensure compatibility with machine learning algorithms. It uses LabelEncoder to convert categorical data into numeric values, and custom functions clean up data containing non-numeric representations, such as extracting numerical values from string columns.

The dataset is then meticulously preprocessed to handle missing values and unnecessary features, making it ready for model training. Extracting key features from complex columns further enhances the dataset's usability. The correlation matrix and heatmap provide crucial insights into relationships between variables, helping guide the feature selection process.

A variety of machine learning models are trained, including Linear Regression, Ridge, Lasso, Random Forest, and XGBoost. These models are evaluated using performance metrics like Mean Squared Error (MSE) and R^2 score, which are essential for understanding how well each model predicts the target variable. The best-performing model is selected and saved for future use, ensuring that the work put into training doesn't need to be repeated.

Finally, visualizations are created to illustrate the models' predictions against the actual values, providing a clear picture of how accurately the models are performing. The scatter plots and distribution charts help identify areas of improvement, such as predicting certain price ranges more accurately or addressing systematic errors.

In conclusion, the script covers all critical stages of data science and machine learning: data preprocessing, feature engineering, model selection, evaluation, and visualization. This process ensures that the predictive model is robust, interpretable, and capable of making reliable predictions for future datasets. It highlights the importance of thorough data exploration and careful model selection in building effective machine learning applications.

==========Thank You==========