**Crop Yield Prediction**

Prepared for

Advanced Certification In

Data Science, Machine Learning and IoT

 E&ICT Academy, IIT Guwahati.

By

Swapnasubham Das

Senior Software Engineer

OpenText

December 9, 2024

**1. Prepare the problem**

* Load libraries and Load dataset

**2. Summarize Data**

* Descriptive statistics and Data visualizations

**3. Prepare Data**

* Data Cleaning and Feature Selection
* Feature Engineering

**4. Evaluate Algorithms**

* Split-out validation dataset
* Test options and evaluation metric
* Compare Algorithms

**5. Improve Accuracy**

* Algorithm Tuning
* Ensembles

**6. Finalize Model**

* Submit predictions

**Problem statement :-**

This dataset encompasses agricultural data for multiple crops cultivated across various states in India from the year 1997 till 2020. The dataset provides crucial features related to crop yield prediction, including crop types, crop years, cropping seasons, states, areas under cultivation, production quantities, annual rainfall, fertilizer usage, pesticide usage, and calculated yields.

**Columns Description: -**

|  |  |
| --- | --- |
| **Feature Name** | **Description** |
| Crop | The name of the crop cultivated. |
| Crop\_Year | The year in which the crop was grown. |
| Season | The specific cropping season (e.g., Kharif, Rabi, Whole Year). |
| State | The Indian state where the crop was cultivated. |
| Area | The total land area (in hectares) under cultivation for the specific crop. |
| Production | The quantity of crop production (in metric tons). |
| Annual\_Rainfall | The annual rainfall received in the crop-growing region (in mm). |
| Fertilizer | The total amount of fertilizer used for the crop (in kilograms). |
| Pesticide | The total amount of pesticide used for the crop (in kilograms). |
| Yield | The calculated crop yield (production per unit area). |

**Prepare the problem: -**

**Load libraries:-**

1. **Core Libraries:**
   * **numpy and pandas:** For numerical computations and data manipulation.
   * **scipy.stats:** To perform advanced statistical tests and analyses.
2. **Statistical Analysis Tools:**
   * **variance\_inflation\_factor from statsmodels:** Helps detect multicollinearity among features.
3. **Visualization Libraries:**
   * **matplotlib and seaborn:** For creating static visualizations with improved aesthetics.
   * **plotly.express and plotly.graph\_objects:** To create interactive visualizations.
   * **missingno:** Specially used for visualizing missing data patterns.
4. **Miscellaneous Settings:**
   * **Warnings Filtering:** Suppressed warnings using warnings.filterwarnings to improve notebook readability.
   * **Seaborn Style:** Set to darkgrid for better visual appeal.
   * **Pandas Display Options:** Adjusted to show all rows and columns, ensuring comprehensive data visibility.
   * **Matplotlib Inline Backend:** Ensures plots are displayed inline within the notebook.

These libraries and configurations form the foundational toolkit for the analysis and modeling pipeline, enhancing both efficiency and presentation quality.

**Machine Learning Models:-**

* **Linear Models:**
  + **LinearRegression:** A foundational regression model for linear relationships.
  + **Ridge, Lasso, ElasticNet:** Linear models with different regularization techniques to prevent overfitting.
* **Tree-Based Models:**
  + **DecisionTreeRegressor:** A non-linear model that splits data recursively based on feature values.
  + **RandomForestRegressor:** An ensemble method combining multiple decision trees to enhance accuracy and reduce overfitting.
* **Boosting Algorithms:**
  + **GradientBoostingRegressor, AdaBoostRegressor:** Boosting methods that sequentially refine predictions to reduce error.
  + **XGBRegressor, CatBoostRegressor, LGBMRegressor:** Highly efficient gradient boosting libraries tailored for speed and accuracy.
* **Ensemble and Non-Parametric Models:**
  + **VotingRegressor:** Combines predictions from multiple models for a consensus output.
  + **BaggingRegressor:** Averages predictions from multiple base models trained on different data subsets.
  + **KNeighborsRegressor:** A non-parametric model that predicts based on the similarity to neighboring data points.

**Data Splitting and Optimization:**

* **train\_test\_split:** Ensures unbiased evaluation by dividing the data into training and testing subsets.
* **GridSearchCV:** Automates hyperparameter tuning by exhaustively searching the best parameter combinations.

**Performance Metrics:**

* **r2\_score:** Evaluates the proportion of variance explained by the model (higher values indicate better fit).
* **mean\_squared\_error:** Measures the average squared difference between predicted and actual values.

**Preprocessing Tools:**

* **Scaling and Transformation:**
  + **StandardScaler:** Normalizes data by centering and scaling it.
  + **MinMaxScaler:** Scales data to a fixed range, typically [0, 1].
  + **PowerTransformer:** Applies transformations to stabilize variance and improve data distribution.
* **Encoding Categorical Data:**
  + **LabelEncoder:** Converts categorical labels into numerical values.
  + **OneHotEncoder:** Encodes categorical features into binary indicator variables.

**libraries version: -**

|  |  |
| --- | --- |
| **Library Name** | **Version** |
| numpy | 1.26.4 |
| pandas | 2.2.2 |
| scipy | 1.13.1 |
| statsmodels | 0.14.2 |
| matplotlib | 3.8.4 |
| seaborn | 0.13.2 |
| plotly | 5.22.0 |
| scikit-learn | 1.4.2 |
| xgboost | 2.1.1 |
| catboost | 1.2.3 |
| lightgbm | 4.5.0 |

**Loading Dataset:-**

the dataset was loaded using **pandas.read\_csv** into a DataFrame. This function is commonly used to import tabular data from a CSV file for further analysis.

* **pd.read\_csv("crop\_yield.csv"):** Reads the dataset named crop\_yield.csv and stores it in the DataFrame variable **df**. This dataset likely contains information related to various features and crop yields that will be used to build the prediction model.

**Dataset Overview:**

* **Number of Entries:** 19,689 rows.
* **Number of Columns:** 10 columns.

**Column Summary:**

1. **Crop**: Non-null, categorical data of type object.
2. **Crop\_Year**: Non-null, numerical data of type int64.
3. **Season**: Non-null, categorical data of type object.
4. **State**: Non-null, categorical data of type object.
5. **Area**: Non-null, numerical data of type float64.
6. **Production**: Non-null, numerical data of type int64.
7. **Annual\_Rainfall**: Non-null, numerical data of type float64.
8. **Fertilizer**: Non-null, numerical data of type float64.
9. **Pesticide**: Non-null, numerical data of type float64.
10. **Yield**: Non-null, numerical data of type float64.

**Data Characteristics:**

* **Data Types:** The dataset contains:
  + 5 columns with float64 data type (continuous numerical variables).
  + 2 columns with int64 data type (discrete numerical variables).
  + 3 columns with object data type (categorical variables).
* **Non-Null Counts:** All columns have 19,689 non-null values, indicating no missing data in the dataset.
* **Memory Usage:** The dataset occupies approximately **1.5 MB** in memory.

**Purpose of This Step:**

* **Understanding Data Types:** Knowing the data types helps in deciding preprocessing steps like encoding categorical variables or scaling numerical ones.
* **Missing Data Check:** Since all columns have non-null values, no immediate imputation or handling of missing data is required.
* **Memory Optimization:** Understanding memory usage helps optimize operations when working with larger datasets.

This detailed summary lays the groundwork for effective data preprocessing and feature engineering.

**Report Description: Statistical Summary of Numerical Features:-**

**Key Observations:**

1. **Crop\_Year:** The data spans from 1997 to 2020, with a mean year of 2009.
2. **Area:** The area under cultivation varies widely, from as small as 0.5 hectares to a maximum of 50.8 million hectares, with a highly skewed distribution.
3. **Production:** Production values are heavily skewed, ranging from 0 to over 6.3 billion tonnes, with a median value of ~13,804 tonnes.
4. **Annual\_Rainfall:** Rainfall ranges from 301.3 mm to 6,552.7 mm, with an average of 1,437.76 mm.
5. **Fertilizer:** Fertilizer usage exhibits a vast range, with extremely high outliers skewing the distribution.
6. **Pesticide:** Similar to fertilizer, pesticide usage shows a wide range with significant outliers.
7. **Yield:** Crop yield varies between 0 and 21,105, with a median yield of ~1.03, suggesting that most data points represent lower yields

**Insights and Next Steps:**

* **Skewness and Outliers:** Several features (e.g., Area, Production, Fertilizer, Pesticide, Yield) show significant skewness and extreme values. These outliers need to be addressed (e.g., through log transformations or robust scaling) to improve model performance.
* **Scaling Requirements:** Features like Area, Production, and Yield may require scaling due to their large magnitudes and variability.
* **Data Imbalance:** The presence of zeros in columns like Production and Yield could indicate potential issues, such as missing values or non-yielding crops, requiring further investigation.

This statistical overview forms the foundation for informed preprocessing decisions and feature engineering strategies.

**Key Observations:**

1. **Crop:**
   * There are 55 unique crop types in the dataset.
   * "Rice" is the most frequently occurring crop, appearing 1,197 times.
2. **Season:**
   * The dataset covers 6 unique seasons (e.g., Kharif, Rabi, Zaid).
   * "Kharif" is the most common season, accounting for 8,232 entries.
3. **State:**
   * Data spans across 30 unique states in India.
   * "Karnataka" is the most represented state, with 1,432 entries.

**Insights and Next Steps:**

* **Feature Encoding:** These categorical features will need encoding (e.g., one-hot or label encoding) to be used in machine learning models.
* **Dominant Categories:** The dominance of certain values (e.g., "Rice," "Kharif," and "Karnataka") might indicate class imbalances that could influence model training.
* **Exploratory Analysis:** Further exploration of less frequent categories may reveal important insights about underrepresented crops, seasons, or states.

**Checking NaN values and duplicates in our Dataset**

* **No Missing Values:** All 10 columns in the dataset have complete data, with no missing values present.
* **Data Completeness:** The absence of missing values simplifies the preprocessing pipeline, as no imputation or data removal is necessary at this stage.
* **No Duplicate Records:** The dataset contains no duplicate rows, ensuring the integrity and uniqueness of the data.
* **Data Quality:** The absence of duplicates eliminates the need for data cleaning related to redundancy.

**Correlation matrix for numeric columns:-**

**Key Observations:**

1. **Strong Positive Correlations:**
   * **Area with Fertilizer (0.97) and Pesticide (0.97):** Indicates larger agricultural areas tend to use higher amounts of fertilizers and pesticides.
   * **Fertilizer and Pesticide (0.95):** Suggests a strong relationship between the use of fertilizers and pesticides.
2. **Moderate Positive Correlation:**
   * **Production with Yield (0.57):** Indicates a moderate relationship between crop production and yield, as expected.
3. **Negligible Correlations:**
   * Features like Crop\_Year, Annual\_Rainfall, and Yield show weak or negligible correlations with other features, suggesting limited direct linear relationships.
4. **Potential Multicollinearity:**
   * The high correlations between Area, Fertilizer, and Pesticide may indicate multicollinearity, which could affect regression model performance.

**Next Steps:**

* **Address Multicollinearity:** Consider techniques like Variance Inflation Factor (VIF) analysis to handle multicollinearity if these features are used in regression models.
* **Feature Engineering:** Explore non-linear relationships and interactions, especially for weakly correlated features like Yield and Annual\_Rainfall.
* **Visualization:** Use heatmaps or pair plots to further analyze these relationships and confirm any potential insights.

**Data Preprocessing Report:-**

The following steps were carried out to preprocess the dataset:

1. **Data Overview:**
   * The dataset, initially containing 19,689 entries and 10 columns, consists of information on crop yield across different states, years, and conditions in India. The columns include attributes like crop type, year, season, area, production, annual rainfall, and agricultural inputs (fertilizer, pesticide).
2. **Data Cleaning and Transformation:**
   * **Missing Data:** No missing values were identified across any columns in the dataset, ensuring the data is complete.
   * **Duplicate Data:** No duplicate entries were found, confirming the dataset’s uniqueness.
   * **Outliers:** While outliers were not explicitly removed, the dataset contains extreme values for some numerical variables (e.g., very high production values), which should be considered when building predictive models.
3. **Feature Engineering:**
   * **Standardizing Column Names:** Column names were standardized to title case using .str.title() to maintain consistency in formatting.
   * **Removal of Data for Year 2020:** The year 2020 had insufficient records, so those rows were removed to ensure data quality for model building.
   * **Categorical Columns:**
     + The categorical columns ('Crop', 'Season', and 'State') were identified and encoded using **one-hot encoding** to convert them into numerical formats. This transformation created new binary columns for each category, making the dataset suitable for machine learning models that require numerical inputs.
4. **Data Summary:**
   * **Data Distribution:** The dataset consists of multiple crop types, seasonal variations (Kharif, Rabi, Whole Year, etc.), and data from 30 states across India. The most frequent crop in the dataset is **Rice**, followed by **Maize** and **Moong**.
   * **Descriptive Statistics:** A summary of the dataset’s numerical columns revealed:
     + **Production:** Ranges from 0 to over 6 billion, indicating significant variation.
     + **Area:** The dataset includes fields with varying areas of cultivation, from small plots to large-scale farming operations.
     + **Fertilizer and Pesticide Usage:** These values show wide-ranging usage across different states, with some areas heavily utilizing these inputs for crop growth.
     + **Annual Rainfall:** A crucial factor affecting crop yields, with values varying widely across states.

**Data After Encoding:**

* + After one-hot encoding the categorical features, the dataset was expanded to **95 columns**, where each new column corresponds to one category of the original categorical features. The dataset is now ready for machine learning model implementation.

**Data Splitting Report**

In this step, the dataset was split into training and testing sets for model development and evaluation.

1. **Features and Target Variable:**
   * **Features (X):** The dataset's features were separated from the target variable ('Yield'). The feature set includes all columns except 'Yield', and it now consists of **94 columns** after one-hot encoding.
   * **Target (Y):** The target variable is 'Yield', representing the crop yield that we aim to predict.
2. **Train-Test Split:**
   * The dataset was split into training and testing sets using train\_test\_split() from sklearn.model\_selection.
   * The training set contains 80% of the data, and the testing set contains 20%. This is a typical split ratio that ensures enough data for model training while reserving sufficient data for model evaluation.
3. **Shape of the Data:**
   * **Training Set:**
     + **Features (X\_train):** 15,721 rows and 94 columns.
     + **Target (y\_train):** 15,721 rows.
   * **Testing Set:**
     + **Features (X\_test):** 3,931 rows and 94 columns.
     + **Target (y\_test):** 3,931 rows.

**Create a function for evaluate model:-**

The function evaluate\_model\_performance() is designed to evaluate the performance of a machine learning model by calculating and printing key metrics: R² (coefficient of determination), Adjusted R², and RMSE (Root Mean Squared Error) for both the training and testing data. Here's a breakdown of the function:

**Function Parameters:**

* model: The machine learning model to be evaluated (e.g., linear regression, decision tree, etc.).
* x\_train: Training feature set (input variables).
* y\_train: Actual target values for the training set.
* x\_test: Testing feature set (input variables).
* y\_test: Actual target values for the testing set.

**Key Operations:**

1. **Model Fitting**: The model is trained on the training dataset (x\_train, y\_train).
2. **Predictions**:
   * The model makes predictions on both the training and testing datasets (x\_train and x\_test).
3. **Performance Metrics**:
   * **R² Score**: Measures the proportion of variance in the target variable that is explained by the model. Higher values indicate better performance.
   * **Adjusted R²**: Adjusted version of R² that takes the number of predictors into account, preventing overfitting with too many features.
   * **RMSE**: Measures the average magnitude of the errors between predicted and actual values, with lower values indicating better predictions.
4. **Scores Collection**: The function appends the calculated R², Adjusted R², and RMSE scores to predefined lists (training\_scores\_r2, testing\_scores\_r2, etc.), allowing for further analysis or comparison across different models.
5. **Display of Results**: The calculated metrics for both the training and testing datasets are printed out for the user, formatted for easy interpretation.

**Example Output:**

LinearRegression Performance Metrics:

Training Data: R² = 85.43%, Adjusted R² = 85.28%, RMSE = 123.4567

Testing Data : R² = 83.12%, Adjusted R² = 82.98%, RMSE = 134.5678

**Lists for Storing Results:**

* training\_scores\_r2: Stores R² scores for the training data.
* testing\_scores\_r2: Stores R² scores for the testing data.
* training\_scores\_adj\_r2: Stores Adjusted R² scores for the training data.
* testing\_scores\_adj\_r2: Stores Adjusted R² scores for the testing data.
* training\_scores\_rmse: Stores RMSE values for the training data.
* testing\_scores\_rmse: Stores RMSE values for the testing data.

This function is helpful for comparing the performance of different models or tracking model improvement during tuning. You can reuse this function by passing different models to see how they perform on the same dataset.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithms** | **Training Score R²** | **Training Score Adjusted R²** | **Training Score RMSE** | **Testing Score R²** | **Testing Score Adjusted R²** | **Testing Score RMSE** |
| Stacking Regressor | 99.12 | 99.12 | 83.44 | 98.94 | 98.91 | 85.68 |
| Random Forest | 99.58 | 99.58 | 57.82 | 98.74 | 98.7 | 93.46 |
| XGBoost | 100 | 100 | 1.58 | 98.41 | 98.37 | 104.9 |
| Gradient Boost | 99.72 | 99.71 | 47.41 | 98.39 | 98.35 | 105.38 |
| Bagging Regressor | 99.45 | 99.44 | 66.23 | 98.36 | 98.32 | 106.51 |
| CatBoost | 99.98 | 99.98 | 12.06 | 98.08 | 98.04 | 115.08 |
| Decision Tree | 100 | 100 | 0 | 97.78 | 97.72 | 123.89 |
| LGBM | 97.76 | 97.75 | 133.24 | 95.88 | 95.78 | 168.7 |
| Voting Regressor | 95.98 | 95.96 | 178.54 | 94.48 | 94.35 | 195.28 |
| KNN | 94.44 | 94.41 | 210.04 | 93.43 | 93.26 | 213.11 |
| AdaBoost | 90.64 | 90.59 | 272.43 | 89.69 | 89.44 | 266.84 |
| Ridge | 84.91 | 84.82 | 345.95 | 81.45 | 81 | 357.97 |
| Lasso | 84.91 | 84.81 | 346.04 | 81.35 | 80.89 | 358.95 |
| Linear Regression | 84.92 | 84.83 | 345.83 | 81.35 | 80.89 | 358.96 |
| ElasticNet | 43.07 | 42.72 | 672.05 | 49.18 | 47.94 | 592.52 |
| SVM | -0.08 | -0.68 | 891.03 | 0.02 | -2.43 | 831.1 |

**Based on the performance evaluation of different regression models, here's a summary of their performance metrics:**

**Key Insights:**

1. **Top Performing Models (Overall):**
   * **XGBoost**: Achieved the highest **Training R²** and **Adjusted R²** of 100%, with an incredibly low RMSE of 1.58 on the training data. It also performs well on the testing data (R² = 98.41%, RMSE = 104.90).
   * **Decision Tree**: Achieved **100% R²** on the training data with no RMSE error (0.00), but performs slightly worse on the testing data (R² = 97.78%).
   * **CatBoost**: Similar performance to Decision Tree with **99.98% R²** on the training data and **98.08% R²** on the testing data.
2. **Ensemble Models**:
   * **Stacking Regressor**: Performs excellently, with **99.12% R²** on the training data and **98.94% R²** on the testing data. The RMSE is slightly higher but still very good.
   * **Random Forest**: Shows strong performance with **99.58% R²** on the training set and **98.74% R²** on the testing set.
   * **Gradient Boosting**: Very close to Random Forest in performance, achieving **99.72% R²** on the training set and **98.39% R²** on the testing set.
3. **Model with Strong Generalization**:
   * **Random Forest**: Demonstrates solid generalization with relatively low RMSE on testing (93.46), just behind **XGBoost**.
   * **Bagging Regressor**: Performs well overall, though its RMSE on testing is a bit higher (106.51) compared to other models.
4. **Less Effective Models**:
   * **ElasticNet** and **SVM** show poor performance, especially on the training set where the **R²** is negative or extremely low. These models have high RMSE values, indicating poor predictive power.
5. **Linear Models (Ridge, Lasso, Linear Regression)**:
   * These models perform similarly, with **R²** values ranging between **84.91%** to **84.92%** on the training set and around **81.35%** on the testing set. They have relatively high RMSE values compared to tree-based models and ensemble methods.

**Conclusion:**

For strong performance with minimal error, ensemble models like **XGBoost**, **Stacking Regressor**, and **Random Forest** are highly recommended due to their excellent generalization abilities and low error rates. On the other hand, simpler models like **Linear Regression**, **ElasticNet**, and **SVM** should be reconsidered if predictive accuracy is a key requirement.

**Modelling after EDA:-**

The skewness values indicate the asymmetry in the distribution of your data for each variable.

**Key Points:**

1. **Skewness near 0**: Indicates a roughly symmetric distribution.
2. **Positive skewness**: Indicates a right-skewed distribution, where the tail is longer on the right side (more smaller values).
3. **Negative skewness**: Indicates a left-skewed distribution, where the tail is longer on the left side (more larger values).

**Analysis of the Results:**

* **Area (21.86)**:
  + Strongly positive skewness, meaning most values are concentrated on the lower side, with a few extremely large values (right tail).
* **Production (19.30)**:
  + Similarly strongly positive skewness, indicating the presence of large outliers on the higher side.
* **Annual\_Rainfall (2.13)**:
  + Moderately positive skewness, suggesting the data is right-skewed but not as extreme as Area or Production.
* **Fertilizer (13.41)** and **Pesticide (25.64)**:
  + Extremely high positive skewness, indicating a heavy concentration of smaller values with some very large outliers.

**Implications:**

1. **Data Transformation**: Variables with high skewness (Area, Pesticide) may need transformation (e.g., logarithmic or Box-Cox) for statistical analysis that assumes normality.
2. **Outliers**: Positive skewness often indicates the presence of outliers in the upper range. Check these values to understand their impact.
3. **Modeling**: Skewed variables may influence regression or machine learning models, as most methods assume normality or require data scaling.

**TransformerUsed**: PowerTransformer  
**Transformation Method**: Yeo-Johnson

**Objective:**

To normalize the distribution of the feature columns in the dataset, making the data more Gaussian-like, which can improve the performance of machine learning models, especially those sensitive to data distribution (e.g., linear regression).

**Steps Performed:**

1. **Exclusion of the Target Variable**:
   * The target variable, Yield, was excluded from the transformation to prevent unintended modification of the dependent variable.
   * Columns transformed: All columns in the dataset except Yield.
2. **Transformation Process**:
   * A PowerTransformer was initialized with the yeo-johnson method, which supports both positive and negative values in the data.
   * The transformer was fitted to the selected columns (transformed\_columns).
   * The transformation was applied, and the normalized values replaced the original data in the selected columns.
3. **Handling Feature Distribution**:
   * The transformation aimed to reduce skewness and stabilize variance across all features, ensuring that the data adheres to assumptions required by many statistical models.

**Outcome:**

* All feature columns, except Yield, were successfully transformed.
* The resulting dataset now contains features with normalized distributions, enhancing their suitability for machine learning algorithms.

**Benefits of Transformation:**

* Improved model performance and stability due to reduced skewness and better adherence to linearity assumptions.
* Greater effectiveness in feature scaling and standardization processes.
* Enhanced compatibility with algorithms sensitive to outliers or non-Gaussian features.

This step is crucial for preparing the data for machine learning models, particularly for those relying on linearity and normality assumptions.

**Performance Evaluation Summary After Power Transformation:-**

The application of the PowerTransformer using the Yeo-Johnson method resulted in the following updated model performances:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Training R² (%)** | **Training Adjusted R² (%)** | **Training RMSE** | **Testing R² (%)** | **Testing Adjusted R² (%)** | **Testing RMSE** |
| Gradient Boost | 99.69 | 99.69 | 49.7 | 98.45 | 98.41 | 103.52 |
| Bagging Regressor | 99.57 | 99.57 | 58.28 | 98.44 | 98.4 | 103.95 |
| Stacking Regressor | 99.14 | 99.13 | 82.79 | 98.4 | 98.36 | 105.03 |
| XGBoost | 100 | 100 | 2.43 | 98.4 | 98.36 | 105.06 |
| KNN | 98.33 | 98.32 | 115.25 | 98.2 | 98.16 | 111.4 |
| Random Forest | 99.6 | 99.6 | 56.42 | 97.9 | 97.85 | 120.38 |
| CatBoost | 99.97 | 99.97 | 14.2 | 97.65 | 97.59 | 127.48 |
| Decision Tree | 100 | 100 | 0 | 96.82 | 96.75 | 148.17 |
| Voting Regressor | 97.37 | 97.35 | 144.5 | 96.41 | 96.32 | 157.58 |
| LGBM | 97.32 | 97.31 | 145.69 | 95.71 | 95.6 | 172.24 |
| AdaBoost | 90.59 | 90.53 | 273.22 | 89.58 | 89.33 | 268.25 |
| Ridge | 85.37 | 85.28 | 340.69 | 82.88 | 82.46 | 343.95 |
| Linear Regression | 85.37 | 85.28 | 340.69 | 82.88 | 82.46 | 343.95 |
| Lasso | 85.32 | 85.23 | 341.29 | 82.75 | 82.33 | 345.19 |
| ElasticNet | 76.64 | 76.5 | 430.52 | 74.11 | 73.48 | 422.96 |
| SVM | 0.74 | 0.15 | 887.35 | 0.84 | -1.57 | 827.69 |

**Key Insights**

1. **Top Performers**:
   * **Gradient Boost** and **Bagging Regressor** show consistently high performance with testing R² values of 98.45% and 98.44%, respectively.
   * **XGBoost** maintains exceptional training performance (R² = 100%) and a strong generalization to the test set (R² = 98.40%).
2. **Balanced Models**:
   * **Stacking Regressor** offers a robust balance between training (R² = 99.14%) and testing (R² = 98.40%) performance.
   * **KNN** demonstrates improved performance after transformation, achieving a testing R² of 98.20%.
3. **Performance Decline**:
   * **Decision Tree** and **CatBoost** show overfitting tendencies, with perfect or near-perfect training scores but comparatively lower test performance.
4. **Marginal Models**:
   * **Ridge**, **Linear Regression**, and **Lasso** perform similarly, with modest R² scores around 82%-85% on both training and testing sets.
5. **Underperformers**:
   * **SVM** and **ElasticNet** exhibit poor performance, with low R² and high RMSE values, indicating poor fit to the data.

**Conclusion**

* **Best Overall Models**: Gradient Boost, Bagging Regressor, and Stacking Regressor stand out as the most effective models, achieving excellent testing scores and maintaining a good balance between training and testing performance.
* **Recommendation**: Based on testing R² and RMSE, Gradient Boost and Bagging Regressor are the top choices for accurate predictions on this dataset.

**Performance Evaluation Summary after hyper tuning:-**

This table summarizes the performance of various regression algorithms, evaluating their ability to predict the target variable using the transformed dataset.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Training R² (%)** | **Training Adjusted R² (%)** | **Training RMSE** | **Testing R² (%)** | **Testing Adjusted R² (%)** | **Testing RMSE** |
| Gradient Boost | 99.87 | 99.87 | 32.57 | 98.61 | 98.58 | 97.95 |
| Bagging Regressor | 99.46 | 99.46 | 65.27 | 98.39 | 98.35 | 105.46 |
| Stacking Regressor | 99.13 | 99.12 | 83.27 | 98.35 | 98.31 | 106.69 |
| Random Forest | 99.55 | 99.55 | 59.8 | 97.95 | 97.9 | 118.94 |
| CatBoost | 99.99 | 99.99 | 7.08 | 97.91 | 97.86 | 120.16 |
| KNN | 100 | 100 | 0 | 97.85 | 97.8 | 121.95 |
| XGBoost | 100 | 100 | 0.1 | 97.72 | 97.67 | 125.4 |
| Decision Tree | 99.82 | 99.81 | 38.31 | 97.48 | 97.42 | 131.83 |
| LGBM | 99.96 | 99.95 | 18.86 | 96.87 | 96.8 | 146.96 |
| Voting Regressor | 98.02 | 98.01 | 125.41 | 95.77 | 95.66 | 171 |
| AdaBoost | 85.19 | 85.1 | 342.77 | 83.32 | 82.92 | 339.46 |
| Ridge | 85.37 | 85.28 | 340.7 | 82.88 | 82.46 | 343.95 |
| Linear Regression | 85.37 | 85.28 | 340.69 | 82.88 | 82.46 | 343.95 |
| ElasticNet | 85.33 | 85.24 | 341.19 | 82.79 | 82.37 | 344.82 |
| Lasso | 85.32 | 85.23 | 341.27 | 82.76 | 82.34 | 345.15 |

**Key Insights**

1. **Top Performers**:
   * **Gradient Boost** achieved the highest testing R² (98.61%) with a testing RMSE of 97.95, making it the best-performing model.
   * **Bagging Regressor** and **Stacking Regressor** are close contenders, with testing R² values of 98.39% and 98.35%, respectively, and RMSE values under 107.
2. **High Performers**:
   * **Random Forest**, **CatBoost**, and **KNN** offer strong predictive capabilities with testing R² values above 97.80%. However, **KNN** shows overfitting tendencies, with a perfect training R² of 100.00%.
3. **Overfitting Models**:
   * **XGBoost**, **KNN**, and **CatBoost** achieved perfect or near-perfect training R² scores, suggesting overfitting despite acceptable test performance.
4. **Moderate Models**:
   * **Voting Regressor** and **LGBM** performed reasonably well but lag behind the top models, with testing R² values of 95.77% and 96.87%, respectively.
5. **Low Performers**:
   * **Linear Regression**, **Ridge**, **Lasso**, and **ElasticNet** have nearly identical and lower performance, with testing R² around 82%-83% and RMSE above 340.
6. **Poor Performers**:
   * **AdaBoost** significantly underperforms compared to other ensemble methods, with a testing R² of 83.32%.

**Conclusion**

* **Best Overall Model**: **Gradient Boost** is the top choice, offering the best balance of predictive accuracy and generalization.
* **Recommendations**:
  + Use **Bagging Regressor** or **Stacking Regressor** as alternatives for a balanced and robust performance.
  + Avoid models like **AdaBoost** and simple linear methods for this dataset, as they fail to capture the complexity adequately.

**Report on Model Performance:-**

**1. Performance Without EDA**

* **Best Models**:
  + **Stacking Regressor** achieves the highest **testing R² (98.94%)** and the lowest **RMSE (85.68)**, demonstrating excellent predictive performance.
  + **Random Forest** follows with a testing R² of **98.74%** and RMSE of **93.46**.
  + **XGBoost** and **Gradient Boost** also perform strongly with testing R² above **98.39%** but have slightly higher RMSE values.
* **Insights**:
  + **Ensemble Models** like Stacking, Random Forest, and Gradient Boost significantly outperform simpler models such as Ridge, Lasso, and Linear Regression.

**2. Performance After EDA**

* **Top Performers**:
  + **Gradient Boost** performs the best after EDA with **testing R² of 98.45%** and RMSE of **103.52**.
  + **Bagging Regressor** and **Stacking Regressor** closely follow, with testing R² values of **98.44%** and **98.40%**, respectively.
* **Insights**:
  + EDA slightly improves model performance across the top models, reducing RMSE values compared to simpler models.

**3. Performance After Hyperparameter Tuning**

* **Best Models**:
  + **Gradient Boost** achieves the best performance with **testing R² of 98.61%** and RMSE of **97.95**.
  + **Bagging Regressor** and **Stacking Regressor** remain close contenders with testing R² values of **98.39%** and **98.35%**, respectively.
* **Insights**:
  + Hyperparameter tuning consistently improves the performance of ensemble models, further validating their robustness and ability to generalize well.

**Final Verdict:-**

Based on the analysis:

1. **Gradient Boosting** is the best model overall:
   * Highest performance after hyperparameter tuning (testing R²: **98.61%**, RMSE: **97.95**).
   * Stable performance across all stages (EDA and tuning).
   * Excellent generalization capabilities.
2. **Alternative Choices**:
   * **Bagging Regressor** and **Stacking Regressor** are strong alternatives with similar performances.
   * If interpretability is a priority, consider **Random Forest**, which also performs strongly and is easier to interpret compared to Gradient Boost.

**Recommendation:-**

Use **Gradient Boosting** as the final model for deployment. Ensure proper scaling and feature transformations during deployment to maintain the model's performance consistency

**Resources: -**

Kaggle : [Kaggle: Your Home for Data Science](https://www.kaggle.com/)

Chatgpt: [ChatGPT](https://chatgpt.com/)

Claude: [Claude](https://claude.ai/new)

Perplexity : [Perplexity](https://www.perplexity.ai/)

Towardsdatascience : Towardsdatascience