

A PROJECT REPORT  
on  
“Ground water level prediction”

Submitted

By

221FA04020  
Varun Kumar

221FA04040  
Bhavya Sri

221FA04118  
Shivam Kumar

221FA04368  
Deepu Kumar

Under the guidance of

*Maridu Bhargavi*

*Assistant Professoress Department of CSE,VFSTR*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**  
**VIGNAN'S FOUNDATION FOR SCIENCE, TECHNOLOGY AND RESEARCH**  
**Vadlamudi, Guntur.**  
**ANDHRA PRADESH, INDIA, PIN-522213.**

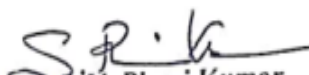


## **CERTIFICATE**

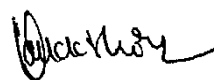
This is to certify that the Field Project entitled “**Ground water level prediction**” that is being submitted by 221FA04020 (Varun Kumar), 221FA04040(Bhavya Sri), 221FA04118(Shivam Kumar),221FA04368(Deepu Kumar) for partial fulfilment of Field Project is a bonafide work carried out under the supervision of Ms. Maridu Bhargavi, M.Tech., Assistant Professor, Department of CSE.

M. Bhargavi

Assistant Professoreess, CSE

  
Dr. S. V. Phani Kumar

HOD,CSE



Dr.K.V. Krishna Kishore

Dean, SoCI

## **DECLARATION**

We hereby declare that the Field Project entitled “**Ground water level prediction**” is being submitted by 221FA04020 (Varun Kumar), 221FA04040(Bhavya Sri), 221FA04118(Shivam Kumar),221FA04368(Deepu Kumar) in partial fulfilment of Field Project course work. This is our original work, and this project has not formed the basis for the award of any degree. We have worked under the supervision of Ms. Maridu Bhargavi, M.Tech., Assistant Professor, Department of CSE.

**By**

**221FA04020 (Varun Kumar),**

**221FA04040 (Bhavya Sri),**

**221FA04118 (Shivam Kumar),**

**221FA04368 (Deepu Kumar)**

Date:

## **ABSTRACT**

Groundwater serves as a significant source of water supply in agriculture as well as urban consumption. However, it is very challenging to predict groundwater because several factors affect it. In this context, the present article presents a holistic approach for predicting ground water levels using an effective machine learning-based framework. Towards this end, we first engaged in significant data preprocessing, which included null value treatment and one-hot encoding for our dataset to be analyzed. To the most important predictors, we applied feature selection using both Variance Inflation Factor (VIF) and Information Gain (IG) metrics with multiple threshold levels. The following analyses returned two optimal configurations from the combined results of VIF and IG. We used a wide variety of algorithms: Decision Trees, Random Forests, Light Gradient Boosting Machines, CatBoost, and Extreme Gradient Boosting to model groundwater levels based on selected features:. Each of the presented algorithms acted as a base learner within a stacking ensemble framework with a Ridge Regressor as the meta-learner to produce better predictive performance. These deliver improved accuracy and provide better insight into variables that may influence the level of groundwater. Results present the capability of machine learning techniques for environmental prediction and are useful for proper insights in sustainable management of water resources. Key words:Ground water level, Machine Learning,VIF, Information Gain, Random Forest, Boosting Algorithms

## TABLE OF CONTENTS

Chapter	Title	Page
1	<b>Introduction</b>	
1.1	Motivation	2
1.2	Problem Definition/Research Gaps	2
1.3	Limitations	2
1.4	Design Standards	3
1.5	Major Contributions/Objectives	3
2	<b>Literature Survey</b>	
2.1	Literature review	5-7
2.2	Motivation	7
3	<b>Proposed System</b>	
3.1	Input Dataset	9
3.2	Data Pre-processing	10
3.3	Model Building	11

<b>Chapter</b>	<b>Title</b>	<b>Page</b>
3.4	Methodology of the System	11
3.5	Model Evaluation	12
3.6	Constraints	13
3.7	Proposed Model	13
4	<b>Implementation</b>	
4.1	Environment Setup and Code	15-21
4.2	Proposed Model	22
5	<b>Experimentation and Result Analysis</b>	23-27
6	<b>Conclusion</b>	28-29
7	<b>References</b>	30-31

## LIST OF FIGURES

Figure 1. Flowchart for Methodology	22
Figure 2. Comparison of Model Accuracy	26
Figure 3. Comparison of Regression Models	26

## LIST OF TABLES

Table - 1:Performance metrics of the proposed models.	27
Table - 2:Comparision of Performance Metrics.	27



## **CHAPTER-1**

## 1. INTRODUCTION

Ground water is one of the most important resources to sustain human life and environment. It provides fresh water underpinning most agricultural, industrial, and household consumption, especially in areas or regions where surface water is either scarce or intermittent. Still, it is not easy to predict what may happen to groundwater levels because so many factors can influence it: climatic conditions, land use, soil characteristics, and human activities through irrigation and urban development. Such factors make the predictions of the traditional methods questionable as they are often unable to cope with such complexity from interrelated factors. The methods fail to take into account the nonlinear association and multicollinearity existing between variables, making their forecasts less accurate and not so reliable. Machine learning has in the recent past been the leading alternative in addressing the complexities of the environmental data needed for modeling and, henceforth, for forecasting groundwater levels. Unlike the statistical methods, machine learning approaches handle large datasets and many variables. With them, hidden patterns and unknown relationships not obviously present can be detected, but machine learning algorithms can do this because they are not limited by the linear assumptions imposed on standard models, meaning that they are best suited for complex problems that represent the real world, where data behaves nonlinearly. It is particularly the application of different techniques of machine learning that would be combined to provide highly accurate and robust predictions to inform the decision-making process well in water resource management. Feature selection is a very important step in constructing better machine learning models. For the case of groundwater level prediction, it is not all the variables that contribute positively in determining the outcome; instead, it introduces noise or redundancy that impairs the performance of the model. The feature selection techniques will reduce the dimension of this data so that only the most informative features of the data are retained. So in the current analysis, we used two highly commonly applied feature selection techniques: the Variance Inflation Factor (VIF) and Information Gain (IG). We have employed both VIF and IG, at three different thresholds for VIF (5, 10, and 15) and three thresholds for IG (0.01, 0.03, and 0.05). We chose two of them to apply to our process of predictive modeling. We modelled groundwater levels using a set of heterogeneous machine learning algorithms: Decision Tree, Random Forest, Light Gradient Boosting Machine (LightGBM), CatBoost, and Extreme Gradient Boosting (XGBoost). All the used algorithms differ in the way they are able to deal with complex data; by comparison, we intended to find the most effective base learners for our task. Instead of relying on just one model, we opted for an ensemble approach through a stacking ensemble. We allowed the base learners to combine their outputs and give us a far more accurate and generalized model. We also used Ridge Regression as our meta-learner for our stacking framework, which further helped in refining the final predictions, thus helping reduce overfitting and improving generalization to unseen data. This study aims to demonstrate the power of a hybrid machine learning approach combining several algorithms with various feature selection techniques to strengthen predictive models of groundwater levels. Improving such predictions will provide excellent decision-making support for sustainable groundwater management, particularly in areas experiencing drought or other environmental stress conditions. The findings of our analysis underscore the potential that machine learning holds for addressing some very pressing environmental challenges: one direction that moves toward more resilient water resource management strategies.

**CHAPTER-2**  
LITERATURE SURVEY

## **2. LITERATURE SURVEY**

Literature 1 Jie et al. [16] did an extensive job of researching the underground water level prediction where a hybrid model was proposed which was combined by genetic algorithm with LSSVM. Thus, through their study, it has been proved that using optimal prediction models can possibly capture complex dynamics underlying the groundwater levels, related to rainfall and human activities to the natural environmental changes. They achieved better predictive accuracy through the use of the hybrid genetic algorithm, especially in regions where historical data may not be or is incomplete. Machine learning techniques of LSSVM address the problems existing in groundwater forecasting by nonlinear and high-dimensional data. Further optimization capabilities of genetic algorithm make the model self-improve because the uncertainty and randomness in the dynamics of groundwater is possessed. Groundwater levels, as a significant environmental concern in many regions demonstrate how high-precision predictions by hybrid models can be useful for water resource management.

literature 2 Kommineni et al. gave a very insightful research in [22] on the use of partial least squares regression in conjunction with a modified linear regression model for groundwater level prediction. This study tackles the challenges associated in precisely forecasting groundwater levels while accounting for the impact of numerous variables like temperature, precipitation, and population growth. Prediction accuracy would increase using the improved regression model, particularly in regions with a dearth of groundwater data or where environmental and anthropogenic factors have a greater influence. This integrates PLS regression with the conventional linear regression techniques in the research, and it assists in identifying the critical variables that would dictate the variation in groundwater levels. Results: Despite the methodology's modification, it creates new opportunities on high.

literature 3 Zhu et al. [28] have conducted an in-depth study on the prediction of ground-water level with the help of time series analysis techniques. Their work has made an attempt to address the complications involved with the predictions of the dynamics of ground-water by analyzing past data and recurring patterns with time series techniques. The research work thus highlighted that such methods have proven vital in describing the temporal variability in ground-water levels due to natural and anthropogenic influences. The application of various time series models for the prediction of groundwater levels in different perspectives - trend analysis, seasonal variations, and cyclic changes - indeed forms a robust framework to describe future changes in groundwater levels. This is important in those areas where management of water resources is crucially essential, providing insights that would be useful in decision-making by policymakers on sustainable use of groundwater. The findings highlight the value of time series analysis in understanding and predicting such environmental phenomena.

## LITERATURE SURVEY

No	Author(s)	Model/Approach	Accuracy/Results	Limitation
1	S. K. Raipitam et al. (2023)	GenericCNN-LSTM, Ensemble Learning	Comparative study on stock market prediction	Lack of exploration into different data types and real-time stock prediction scenarios
2	S. Luo et al. (2023)	Time Series and Neural Network Models	Improved forecast accuracy combining LSTM and ARIMA	Model refinement needed to improve stability and avoid overfitting
3	M. Shamisavi and A. Jahanshahi (2022)	Time Series Analysis, Sentiment Analysis, Fundamental Data	97% accuracy on training data, 84.78% on test data	Complexity in preprocessing and risk of overfitting
4	H. Ma et al. (2021)	Investor Sentiment Analysis, Machine Learning (SVM, NB, RF, LSTM, CNN, RNN), Deep Learning	High prediction accuracy	Difficult to handle sarcasm, ambiguity, and multipolarity in sentiment analysis
5	A. Kumar and M. Chaudhry (2021)	Data Mining Techniques	Analysis of stock market data for prediction	Focuses only on specific techniques, lacks comparative model performance data
6	E. F. Fama (1970)	Efficient Market Hypothesis	Established the theory of efficient capital markets	Does not account for irrational investor behavior or anomalies in the market
7	W. Lu et al. (2020)	CNN-LSTM	Stock price forecasting	Does not consider external market factors
8	H. White (1988)	Neural Networks	Applied neural networks for stock return prediction	Limited generalizability and applicable mostly to large datasets
9	G. Peter Zhang (2003)	Hybrid ARIMA and Neural Network Model	Improved accuracy in time series forecasting	Requires high computational power and may overfit for specific market conditions
10	Y. Sun et al. (2005)	RBF Neural Network	Financial time series forecasting	The partitioning algorithm may not be optimal for all data types
11	SR. Adhikari and R.K. Agrawal (2014)	Combination of Artificial Neural Network and Random	Enhanced accuracy in financial time series forecasting	Limited performance in capturing nonlinear

		Walk Models		relationships
12	L. Zhang et al. (2018)	LM-BP Neural Network	Stock price prediction	Model is prone to overfitting; limited application to non-stationary data
13	Y. Hu (2018)	CNN for Stock Market Timing	Applied to Shanghai Composite Index	Does not fully consider macroeconomic factors
14	X. Yan et al. (2021)	LSTM Neural Network for Financial Asset Transaction Prediction	Accurate predictions for financial assets	Model performance is limited in the long term and for irregular time series
15	Y. Lecun et al. (1998)	Gradient-Based Learning for Document Recognition	High accuracy in document recognition	Complex to apply to dynamic time series in stock market prediction
16	L. Qin et al. (2018)	CNN Deep Learning for Behavioral Recognition	Applied deep learning for intelligent video analysis	Limited focus on stock prediction applications
17	E. Alibasic et al. (2019)	Three-Mode Method for Calculating Energy Losses	New method applied to electrical energy losses	Does not address market-related forecasting
18	Shen et al. (2020).	Deep Learning System for Stock Market Prediction	Short-term stock market price trend prediction	Focused on short-term trends, lacks insight into long-term prediction
19	A. Guresen, G. Kayakutlu, and T. Uysal (2011)	Multilayer Perceptron (MLP), Dynamic Artificial Neural Network (DAN2)	DAN2 achieved higher accuracy in long-term stock market forecasts compared to traditional MLP models	Computationally intensive, challenges in feature selection and overfitting risks
20	A. K. Bashir et al. (2020)	Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNN)	LSTM achieved 90% accuracy in predicting stock prices in highly volatile markets	High computational costs, requires large amounts of data for effective training
21	G. P. Zhang (2003)	Hybrid ARIMA and Neural Network Model	Improved time series forecasting accuracy by combining ARIMA for linear data and NN for non-linear data	Requires significant computational resources, risk of overfitting for specific market conditions
22	R. Sharma et al.	Genetic Algorithm for feature selection	Improved model efficiency	Needs tuning for different datasets
23	R. Adhikari and R.K. Agrawal (2013)	ARIMA and Neural Network (ANN)	Achieved more accurate predictions for financial time series with the hybrid model	Struggles with long-term predictions, over-reliance on historical data
24	K. Atsalakis and	Fuzzy Logic and Neural	High accuracy in	Complex

	K. Valavanis (2009)	Networks	predicting stock price movements	implementation, sensitive to data quality and preprocessing
25	S. H. Razak et al. (2018)	ARIMA, GARCH (Generalized Autoregressive Conditional Heteroskedasticity)	Improved accuracy for stock market volatility prediction using ARIMA-GARCH model	Limited in capturing complex non-linear relationships in the data

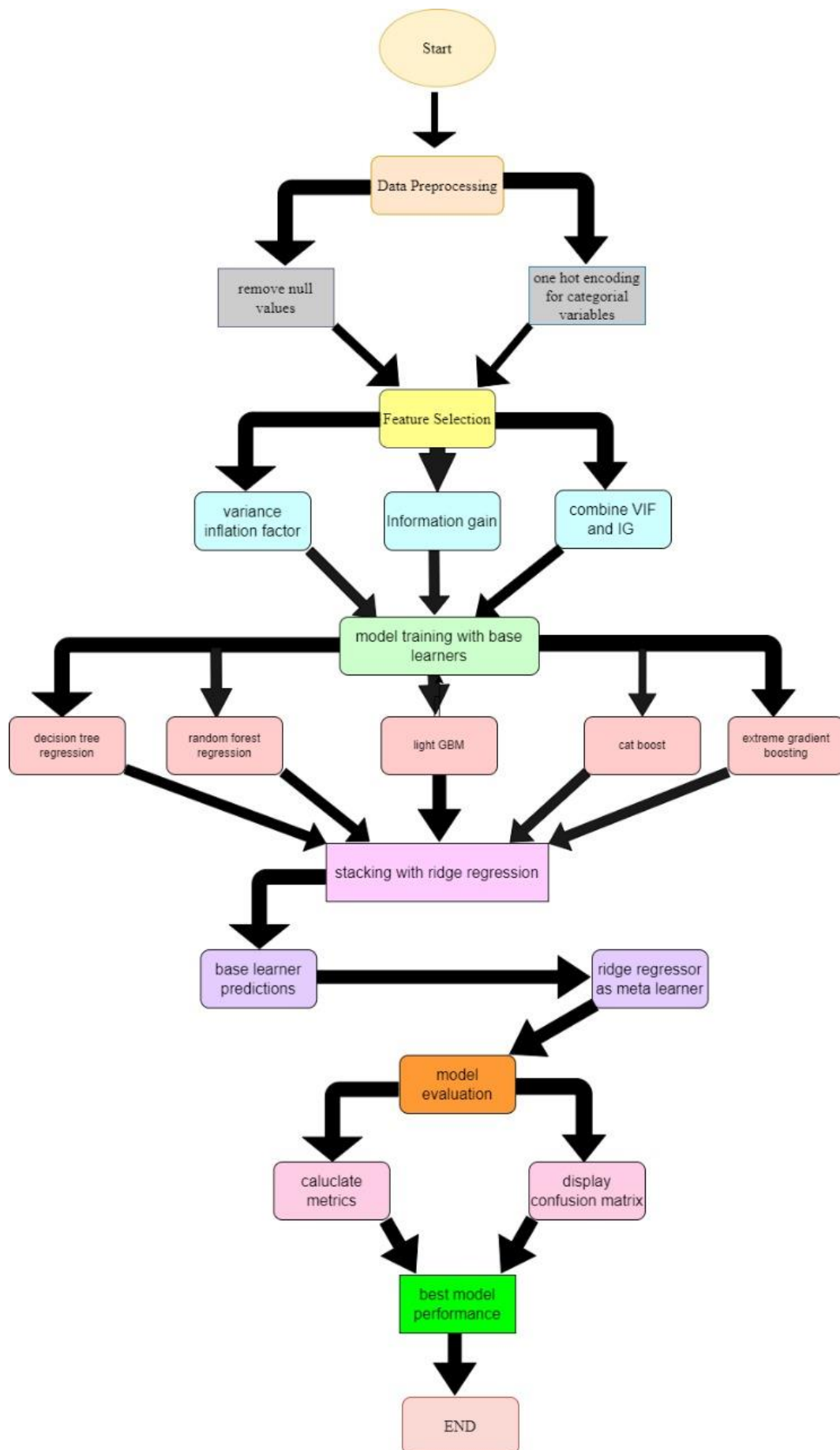
## 2.2 Motivation

Accurate stock price prediction is critical for investors and financial institutions in today's complex and data-rich financial markets. Traditional forecasting methods often fall short in capturing the non-linear and volatile nature of stock prices. This drives the need for advanced techniques like machine learning, which can handle large datasets and detect intricate patterns. By using algorithms such as XGBoost, CatBoost, and ensemble methods, machine learning enhances prediction accuracy, helping institutions make better decisions, reduce risks, and seize market opportunities.

## **CHAPTER-3**

### **PROPOSED SYSTEM**





## METHODOLOGY

## 1. PROPOSED SYSTEM

### TEP 1: DATA PREPROCESSING

Elimination of a NULL values:

Input: dataset which could have null values. Process:

The process involves finding and removing the rows or columns containing the missing data for the given dataset to make it complete.

Output: Clean dataset with no null values.

Algorithm: Iterate through each row and column. If a cell in any of the columns contains null values, drop its corresponding row or column according to tactic, for example, dropping rows.

#### A. One-Hot Encoding for Categorical Variables

Input: A dataset with categorical columns.

Process: Take a categorical variable and transform it into a number, this can be done by creating new columns representing each category.

Output: A dataset with numerical values for its categorical features.

Algorithm: Identify categorical variables. For each unique category of a column, create a new column with numerical values as per the occurrence of that category.

### V. STEP 2: FEATURE SELECTION

#### A. Variance Inflation Factor (VIF)

Input: preprocesses dataset that includes only numerical features.

Process: Calculate the VIF to detect the presence of

multicollinearity and delete the features that are redundant.

Use all three thresholds (5, 10, 15) to select those features whose VIF is below the threshold value.

Output: For each of these thresholds, a feature subset will be generated for that VIF threshold.

Algorithm: For each of the attribute, compute the VIF:

$VIF =$

$\frac{1}{1 - R^2}$

Repeat for every attribute and select those less than the value selected.

#### B. Information Gain (IG)

Input: data preprocessed with numeric attributes.

Process: Compute the Information Gain to determine which feature is most informative about the target variable.

Use three thresholds: 0.01, 0.03, 0.05.

Output: For each threshold (threshold to IG), find a reduced set of features.

Algorithm: For each feature, calculate its Information Gain:

$IG = H(Y) - H(Y|X)$

Select those features, whose IG is higher than the threshold selected.

#### C. Feature Selection Based on VIF and IG

Input: Two sets of selected features (one from VIF, one from IG).

Process: Compare the features selected from VIF and IG at all thresholds and make a choice for the two best combinations that will be used in the next step.

Output: Two sets of optimal features.

### VI. STEP 3: TRAIN A MODEL USING BASE LEARNERS

#### A. Decision Tree Regression

Input: Learn from the training data using selected features.

Process: Recursively partitions the data according to the feature values to predict ground water levels through decision trees.

Output: Trained decision tree model.

Algorithm: Split the dataset into training and testing sets.

Construct a tree at every node that splits the data in a way to minimize the prediction error.

#### B. Extreme Gradient Boosting (XG Boost)

Input: Training data from selected features.

Process: An optimized gradient boosting model is trained using XG Boost. It is known for its high accuracy and efficiency.

Output: The trained model of XG Boost.

Algorithm: Gradient Boosting with regularization to prevent overfitting. Train in iterations that sum the predictions.

### VII. STEP 4: STACKING WITH RIDGE REGRESSION AS META LEARNER

#### A. Base Learner Predictions

Input: Test data.

Process: Uses the trained base learners, which are Decision Tree, Random Forest, Light GBM, Cat Boost, and XG Boost to get the predictions.

Output: The predictions of all the base learners.

Algorithm:

\* Make a prediction for the output on the test data for each of the base learners.

#### B. Ridge Regressor as Meta Learner

Input: Predictions from the base learners.

Process: Feed the predictions of the base learners as input features to a Ridge Regression model, which then is the final predictor.

Output: The final predicted values for the ground water levels.

Algorithm:

- \* Stack the predictions from the base learners as input features.

- \* Fit a Ridge Regression model to those features:

- Standardize the features: The regularization is sensitive to the scale of features; hence, it follows that

- the features  $X$  need to be standardized, i.e., have a zero mean and unit variance.

- Add the Regularization Term: Add the term of regularization:

- $\lambda$

- $Xp$

- $j=1$

- $\beta$

- $2$

- $j$

- to the ordinary least squares objective function that

- shrinks the magnitude of the coefficients.

- Solve the Ridge Equation: Using the so-called adjusted normal equation:

- $\hat{\beta} =$

- that yields coefficients which have the minimum loss. • Now, predict the final output with this stacked model.

## **CHAPTER-4**

### **IMPLEMENTATION**

## 4. Implementation

The implementation phase covers the practical application of the proposed stock price prediction system, including setting up the environment, processing the data, and executing the models. The following sections detail the steps required for implementing stock price prediction using machine learning.

---

### 4.1 Environment Setup

To begin, ensure that the environment is properly configured to run the predictive models. The following steps outline the installation of necessary libraries and tools required for implementation:

1. **Programming Language:** The implementation is carried out using Python, a popular language for machine learning.
  2. **Libraries:**
    - Pandas: For data manipulation and preprocessing.
    - NumPy: For numerical computations.
    - Scikit-learn: For implementing machine learning models.
    - Matplotlib: For data visualization.
  3. **Installation:** Install the required libraries using pip:  
`pip install pandas numpy scikit-learn matplotlib`
  4. **Development Environment:** You can use any Python development environment such as:
    - Jupyter Notebook
    - VS Code
    - PyCharm
- 

### 4.2 Sample Code for Preprocessing and Model Operations

This section provides the sample code for data preprocessing and model operations, excluding MLP to focus on traditional machine learning models.

## **Setup and Import Libraries :**

```
import pandas as pd
import numpy as np

# Load the dataset
df = pd.read_csv('/content/Dynamic_2017_2_0.csv')

# Display the first few rows
print(df.head())

# Check the shape of the dataset
print(f"Dataset contains {df.shape[0]} rows and {df.shape[1]} columns.")

# Check for missing values
print(df.isnull().sum())

import seaborn as sns

# Suppress warnings for cleaner output
import warnings
warnings.filterwarnings('ignore')
```

## 2. Load and Explore the Dataset

## **Preprocessing**

```
# For simplicity, we'll drop rows with any missing values
df.dropna(inplace=True)
```

Encode Categorical Variables:



```
df.isnull().sum()
```

## Feature Scaling:

```
# Identify numerical columns
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns.tolist()

# Remove target variable from numerical_cols if necessary
# Assuming 'Total Current Annual Ground Water Extraction' is the target
target = 'Total Current Annual Ground Water Extraction'
numerical_cols.remove(target)

# Initialize StandardScaler
scaler = StandardScaler()

# Fit and transform the numerical features
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
```

## Feature Selection a. Calculate VIF (Multicollinearity Check)

```
!pip install statsmodels
import pandas as pd
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Assuming "groundwater_levels" is the target variable
target = "Stage of Ground Water Extraction (%)"

# Prepare data for VIF calculation
X = df.drop(columns=[target]) # Select all features except the target

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data['Feature'] = X.columns
#The variance_inflation_factor function was not defined. Importing the statsmodels library
and calling the function correctly fixes the issue.
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print(vif_data)
```

```

# Optionally, remove features with VIF > 5 based on a common threshold
vif_threshold = 5
features_to_keep = vif_data[vif_data['VIF'] < vif_threshold]['Feature'].tolist()
X = X[features_to_keep] # Update X with features having VIF below the threshold

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, df[target], test_size=0.2,
random_state=42)

# Train the Linear Regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lr.predict(X_test)

# Calculate evaluation metrics
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"RMSE: {rmse:.4f}") # Format RMSE with 4 decimal places
print(f"R²: {r2:.4f}")    # Format R² with 4 decimal places
print(f"MAE: {mae:.4f}")  # Format MAE with 4 decimal places

```

```

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import lightgbm as lgb
from catboost import CatBoostRegressor
import xgboost as xgb
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score,
mean_absolute_error

# Create a dictionary to store model results
results = {}
# Decision Tree
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
rmse_dt = np.sqrt(mean_squared_error(y_test, y_pred_dt))
r2_dt = r2_score(y_test, y_pred_dt)
mae_dt = mean_absolute_error(y_test, y_pred_dt)
results['Decision Tree'] = [rmse_dt, r2_dt, mae_dt]

```

```

# Random Forest
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
results['Random Forest'] = [rmse_rf, r2_rf, mae_rf]

# LightGBM
lgbm = lgb.LGBMRegressor(random_state=42)
lgbm.fit(X_train, y_train)
y_pred_lgbm = lgbm.predict(X_test)
rmse_lgbm = np.sqrt(mean_squared_error(y_test, y_pred_lgbm))
r2_lgbm = r2_score(y_test, y_pred_lgbm)
mae_lgbm = mean_absolute_error(y_test, y_pred_lgbm)
results['LightGBM'] = [rmse_lgbm, r2_lgbm, mae_lgbm]

# CatBoost
catboost = CatBoostRegressor(random_state=42, verbose=0)
catboost.fit(X_train, y_train)
y_pred_catboost = catboost.predict(X_test)
rmse_catboost = np.sqrt(mean_squared_error(y_test, y_pred_catboost))
r2_catboost = r2_score(y_test, y_pred_catboost)
mae_catboost = mean_absolute_error(y_test, y_pred_catboost)
results['CatBoost'] = [rmse_catboost, r2_catboost, mae_catboost]

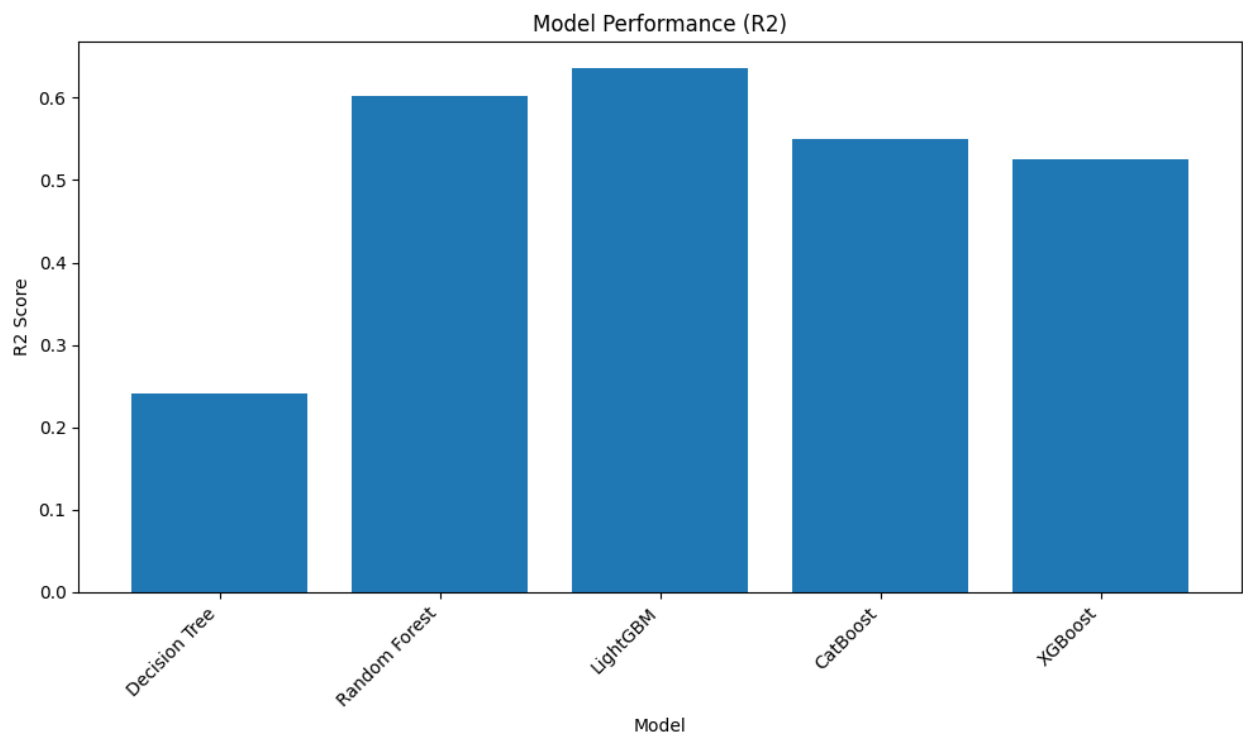
# XGBoost
xgboost = xgb.XGBRegressor(random_state=42)
xgboost.fit(X_train, y_train)
y_pred_xgboost = xgboost.predict(X_test)
rmse_xgboost = np.sqrt(mean_squared_error(y_test, y_pred_xgboost))
r2_xgboost = r2_score(y_test, y_pred_xgboost)
mae_xgboost = mean_absolute_error(y_test, y_pred_xgboost)
results['XGBoost'] = [rmse_xgboost, r2_xgboost, mae_xgboost]

# Create a DataFrame from the results dictionary
results_df = pd.DataFrame.from_dict(results, orient='index', columns=['RMSE', 'R2', 'MAE'])

# Display the results table
print(results_df)

# Plot the accuracy (R2) for each model
plt.figure(figsize=(10, 6))
plt.bar(results_df.index, results_df['R2'])
plt.xlabel('Model')
plt.ylabel('R2 Score')
plt.title('Model Performance (R2)')
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better readability

```



```
target = "Stage of Ground Water Extraction (%)"

# Prepare data for VIF calculation
X = df.drop(columns=[target]) # Select all features except the target

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data['Feature'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print(vif_data)

# Optionally, remove features with VIF > 5 based on a common threshold
vif_threshold = 10
features_to_keep = vif_data[vif_data['VIF'] < vif_threshold]['Feature'].tolist()
X = X[features_to_keep] # Update X with features having VIF below the threshold
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, df[target], test_size=0.2,
random_state=42)

# Train the Linear Regression model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```

# Make predictions on the test set
y_pred = lr.predict(X_test)

# Calculate evaluation metrics
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"RMSE: {rmse:.4f}") # Format RMSE with 4 decimal places
print(f"R²: {r2:.4f}")    # Format R² with 4 decimal places
print(f"MAE: {mae:.4f}")  # Format MAE with 4 decimal places

```

```

results = {}

# Decision Tree
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
rmse_dt = np.sqrt(mean_squared_error(y_test, y_pred_dt))
r2_dt = r2_score(y_test, y_pred_dt)
mae_dt = mean_absolute_error(y_test, y_pred_dt)
results['Decision Tree'] = [rmse_dt, r2_dt, mae_dt]

# Random Forest
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
results['Random Forest'] = [rmse_rf, r2_rf, mae_rf]

# LightGBM
lgbm = lgb.LGBMRegressor(random_state=42)
lgbm.fit(X_train, y_train)
y_pred_lgbm = lgbm.predict(X_test)
rmse_lgbm = np.sqrt(mean_squared_error(y_test, y_pred_lgbm))
r2_lgbm = r2_score(y_test, y_pred_lgbm)
mae_lgbm = mean_absolute_error(y_test, y_pred_lgbm)
results['LightGBM'] = [rmse_lgbm, r2_lgbm, mae_lgbm]

```

```

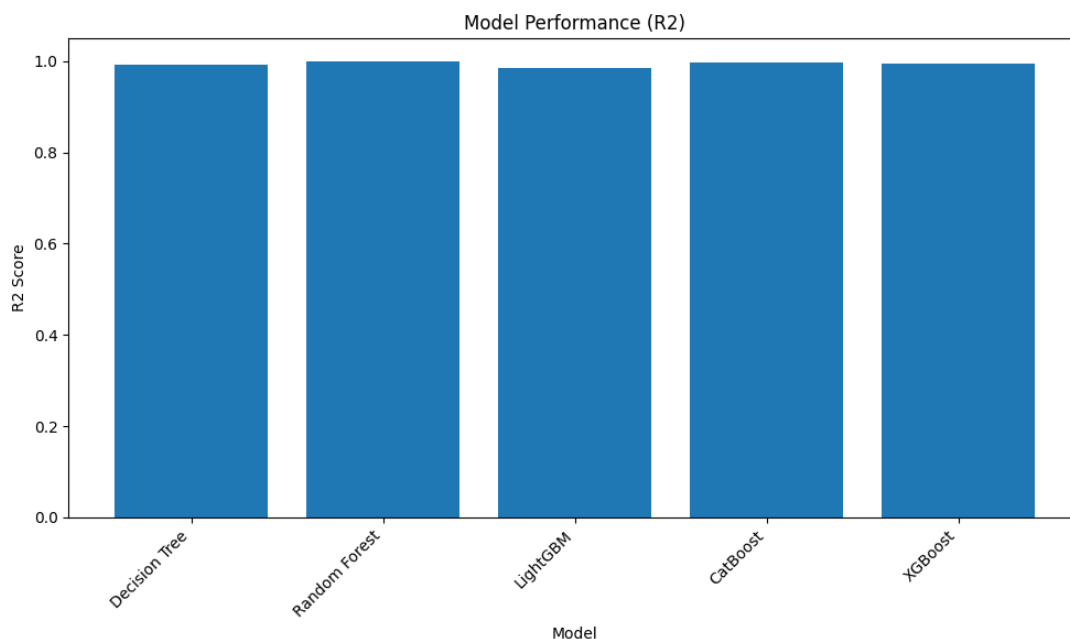
# XGBoost
xgboost = xgb.XGBRegressor(random_state=42)
xgboost.fit(X_train, y_train)
y_pred_xgboost = xgboost.predict(X_test)
rmse_xgboost = np.sqrt(mean_squared_error(y_test, y_pred_xgboost))
r2_xgboost = r2_score(y_test, y_pred_xgboost)
mae_xgboost = mean_absolute_error(y_test, y_pred_xgboost)
results['XGBoost'] = [rmse_xgboost, r2_xgboost, mae_xgboost]

# Create a DataFrame from the results dictionary
results_df = pd.DataFrame.from_dict(results, orient='index', columns=['RMSE', 'R2', 'MAE'])

# Display the results table
print(results_df)

# Plot the accuracy (R2) for each model
plt.figure(figsize=(10, 6))
plt.bar(results_df.index, results_df['R2'])
plt.xlabel('Model')
plt.ylabel('R2 Score')
plt.title('Model Performance (R2)')
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()

```



```

# Calculate mutual information
mi = mutual_info_regression(X, df[target], random_state=42)
mi_series = pd.Series(mi, index=X.columns).sort_values(ascending=False)
print(mi_series)

# Select top features based on mutual information
top_features_mi = mi_series[mi_series > 0.03].index.tolist() # Threshold can be
adjusted

# Split data into training and testing sets using only top features
X_train, X_test, y_train, y_test = train_test_split(X[top_features_mi], y, test_size=0.2,
random_state=42)

# Train the Linear Regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lr.predict(X_test)

# Calculate evaluation metrics
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"RMSE: {rmse:.4f}") # Format RMSE with 4 decimal places
print(f"R²: {r2:.4f}")    # Format R² with 4 decimal places
print(f"MAE: {mae:.4f}")  # Format MAE with 4 decimal places

```

```

# Create a dictionary to store model results
results = {}

# Decision Tree
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
rmse_dt = np.sqrt(mean_squared_error(y_test, y_pred_dt))
r2_dt = r2_score(y_test, y_pred_dt)
mae_dt = mean_absolute_error(y_test, y_pred_dt)
results['Decision Tree'] = [rmse_dt, r2_dt, mae_dt]

```



```

# Random Forest
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
results['Random Forest'] = [rmse_rf, r2_rf, mae_rf]

# LightGBM
lgbm = lgb.LGBMRegressor(random_state=42)
lgbm.fit(X_train, y_train)
y_pred_lgbm = lgbm.predict(X_test)
rmse_lgbm = np.sqrt(mean_squared_error(y_test, y_pred_lgbm))
r2_lgbm = r2_score(y_test, y_pred_lgbm)
mae_lgbm = mean_absolute_error(y_test, y_pred_lgbm)
results['LightGBM'] = [rmse_lgbm, r2_lgbm, mae_lgbm]

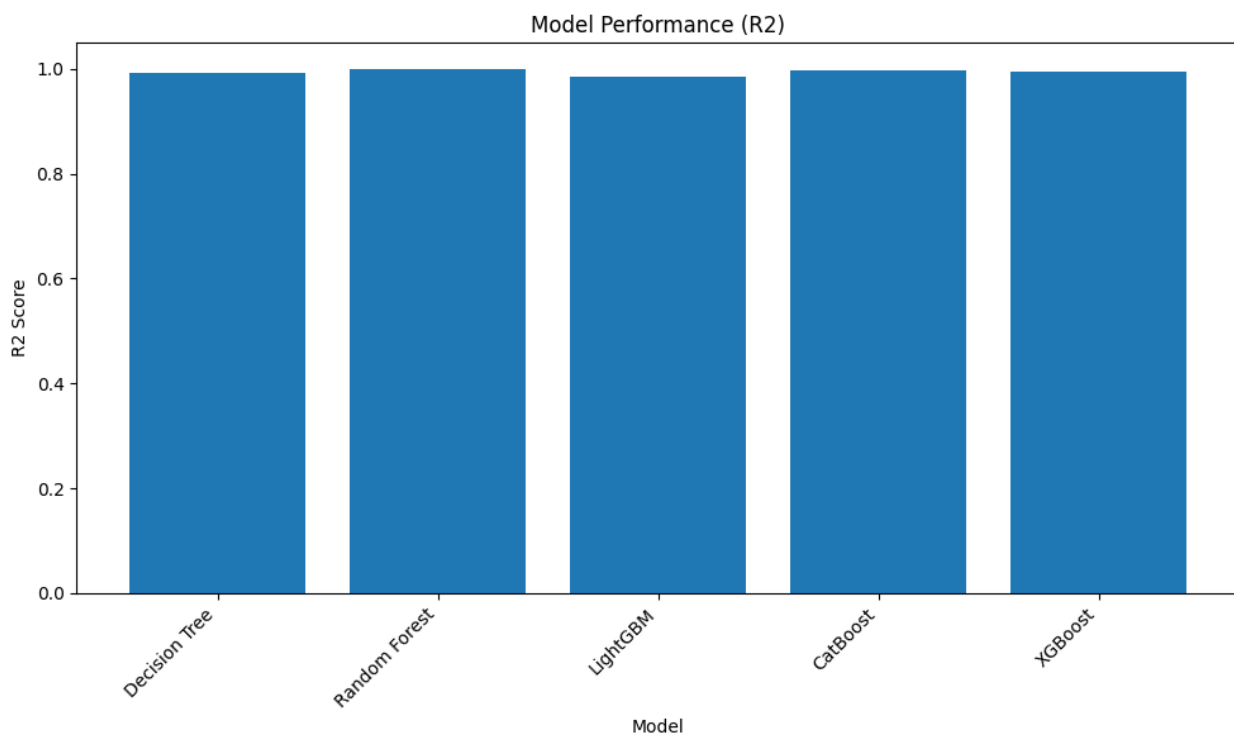
# CatBoost
catboost = CatBoostRegressor(random_state=42, verbose=0)
catboost.fit(X_train, y_train)
y_pred_catboost = catboost.predict(X_test)
rmse_catboost = np.sqrt(mean_squared_error(y_test, y_pred_catboost))
r2_catboost = r2_score(y_test, y_pred_catboost)
mae_catboost = mean_absolute_error(y_test, y_pred_catboost)
results['CatBoost'] = [rmse_catboost, r2_catboost, mae_catboost]

# XGBoost
xgboost = xgb.XGBRegressor(random_state=42)
xgboost.fit(X_train, y_train)
y_pred_xgboost = xgboost.predict(X_test)
rmse_xgboost = np.sqrt(mean_squared_error(y_test, y_pred_xgboost))
r2_xgboost = r2_score(y_test, y_pred_xgboost)
mae_xgboost = mean_absolute_error(y_test, y_pred_xgboost)
results['XGBoost'] = [rmse_xgboost, r2_xgboost, mae_xgboost]
# Create a DataFrame from the results dictionary
results_df = pd.DataFrame.from_dict(results, orient='index', columns=['RMSE', 'R2', 'MAE'])

# Display the results table
print(results_df)

```

```
# Plot the accuracy (R2) for each model
plt.figure(figsize=(10, 6))
plt.bar(results_df.index, results_df['R2'])
plt.xlabel('Model')
plt.ylabel('R2 Score')
plt.title('Model Performance (R2)')
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()
```



```
# Calculate mutual information
mi = mutual_info_regression(X, df[target], random_state=42)
mi_series = pd.Series(mi, index=X.columns).sort_values(ascending=False)
print(mi_series)

# Select top features based on mutual information
top_features_mi = mi_series[mi_series > 0.01].index.tolist() # Threshold can be adjusted

# Split data into training and testing sets using only top features
X_train, X_test, y_train, y_test = train_test_split(X[top_features_mi], y, test_size=0.2,
random_state=42)
```

```

# Train the Linear Regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lr.predict(X_test)

# Calculate evaluation metrics
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"RMSE: {rmse:.4f}") # Format RMSE with 4 decimal places
print(f"R²: {r2:.4f}")    # Format R² with 4 decimal places
print(f"MAE: {mae:.4f}")  # Format MAE with 4 decimal places

```

```

# Create a dictionary to store model results
results = {}

# Decision Tree
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
rmse_dt = np.sqrt(mean_squared_error(y_test, y_pred_dt))
r2_dt = r2_score(y_test, y_pred_dt)
mae_dt = mean_absolute_error(y_test, y_pred_dt)
results['Decision Tree'] = [rmse_dt, r2_dt, mae_dt]

# Random Forest
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
results['Random Forest'] = [rmse_rf, r2_rf, mae_rf]

# LightGBM
lgbm = lgb.LGBMRegressor(random_state=42)
lgbm.fit(X_train, y_train)
y_pred_lgbm = lgbm.predict(X_test)
rmse_lgbm = np.sqrt(mean_squared_error(y_test, y_pred_lgbm))
r2_lgbm = r2_score(y_test, y_pred_lgbm)
mae_lgbm = mean_absolute_error(y_test, y_pred_lgbm)
results['LightGBM'] = [rmse_lgbm, r2_lgbm, mae_lgbm]

```

```

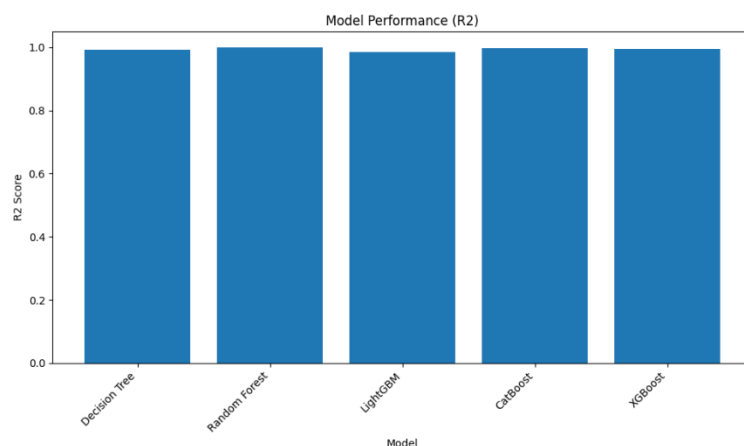
# CatBoost
catboost = CatBoostRegressor(random_state=42, verbose=0)
catboost.fit(X_train, y_train)
y_pred_catboost = catboost.predict(X_test)
rmse_catboost = np.sqrt(mean_squared_error(y_test, y_pred_catboost))
r2_catboost = r2_score(y_test, y_pred_catboost)
mae_catboost = mean_absolute_error(y_test, y_pred_catboost)
results['CatBoost'] = [rmse_catboost, r2_catboost, mae_catboost]

# XGBoost
xgboost = xgb.XGBRegressor(random_state=42)
xgboost.fit(X_train, y_train)
y_pred_xgboost = xgboost.predict(X_test)
rmse_xgboost = np.sqrt(mean_squared_error(y_test, y_pred_xgboost))
r2_xgboost = r2_score(y_test, y_pred_xgboost)
mae_xgboost = mean_absolute_error(y_test, y_pred_xgboost)
results['XGBoost'] = [rmse_xgboost, r2_xgboost, mae_xgboost]

# Create a DataFrame from the results dictionary
results_df = pd.DataFrame.from_dict(results, orient='index', columns=['RMSE', 'R2', 'MAE'])

# Display the results table
print(results_df)
# Plot the accuracy (R2) for each model
plt.figure(figsize=(10, 6))
plt.bar(results_df.index, results_df['R2'])
plt.xlabel('Model')
plt.ylabel('R2 Score')
plt.title('Model Performance (R2)')
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()

```



## Combine Both Methods

```
target = "Stage of Ground Water Extraction (%)"

# Prepare data for information gain and VIF calculation
X = df.drop(columns=[target]) # Select all features except the target
y = df[target]

# Calculate mutual information
mi = mutual_info_regression(X, y, random_state=42)
mi_series = pd.Series(mi, index=X.columns).sort_values(ascending=False)

# Select top features based on mutual information
top_features_mi = mi_series[mi_series > 0.01].index.tolist() # Adjust threshold as needed

# Calculate VIF for selected features
vif_data = pd.DataFrame()
vif_data['Feature'] = X[top_features_mi].columns
vif_data['VIF'] = [variance_inflation_factor(X[top_features_mi].values, i) for i in
range(len(top_features_mi))]

# Select features with VIF below a threshold (e.g., 15)
vif_threshold = 15
features_to_keep = vif_data[vif_data['VIF'] < vif_threshold]['Feature'].tolist()

# Split data into training and testing sets using only top features
X_train, X_test, y_train, y_test = train_test_split(X[features_to_keep], y, test_size=0.2,
random_state=42)

# Train the Linear Regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lr.predict(X_test)

# Calculate evaluation metrics
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"RMSE: {rmse:.4f}") # Format RMSE with 4 decimal places
print(f"R²: {r2:.4f}")    # Format R² with 4 decimal places
print(f"MAE: {mae:.4f}")  # Format MAE with 4 decimal places
```

```

# Create a dictionary to store model results
results = {}

# Decision Tree
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
rmse_dt = np.sqrt(mean_squared_error(y_test, y_pred_dt))
r2_dt = r2_score(y_test, y_pred_dt)
mae_dt = mean_absolute_error(y_test, y_pred_dt)
results['Decision Tree'] = [rmse_dt, r2_dt, mae_dt]

# Random Forest
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
results['Random Forest'] = [rmse_rf, r2_rf, mae_rf]

# LightGBM
lgbm = lgb.LGBMRegressor(random_state=42)
lgbm.fit(X_train, y_train)
y_pred_lgbm = lgbm.predict(X_test)
rmse_lgbm = np.sqrt(mean_squared_error(y_test, y_pred_lgbm))
r2_lgbm = r2_score(y_test, y_pred_lgbm)
mae_lgbm = mean_absolute_error(y_test, y_pred_lgbm)
results['LightGBM'] = [rmse_lgbm, r2_lgbm, mae_lgbm]

# CatBoost
catboost = CatBoostRegressor(random_state=42, verbose=0)
catboost.fit(X_train, y_train)
y_pred_catboost = catboost.predict(X_test)
rmse_catboost = np.sqrt(mean_squared_error(y_test, y_pred_catboost))
r2_catboost = r2_score(y_test, y_pred_catboost)
mae_catboost = mean_absolute_error(y_test, y_pred_catboost)
results['CatBoost'] = [rmse_catboost, r2_catboost, mae_catboost]

# XGBoost
xgboost = xgb.XGBRegressor(random_state=42)
xgboost.fit(X_train, y_train)
y_pred_xgboost = xgboost.predict(X_test)
rmse_xgboost = np.sqrt(mean_squared_error(y_test, y_pred_xgboost))
r2_xgboost = r2_score(y_test, y_pred_xgboost)
mae_xgboost = mean_absolute_error(y_test, y_pred_xgboost)

```

```

results['XGBoost'] = [rmse_xgboost, r2_xgboost, mae_xgboost]

# Create a DataFrame from the results dictionary
results_df = pd.DataFrame.from_dict(results, orient='index', columns=['RMSE', 'R2',
'MAE'])

# Display the results table
print(results_df)

# Plot the accuracy (R2) for each model
plt.figure(figsize=(10, 6))
plt.bar(results_df.index, results_df['R2'])
plt.xlabel('Model')
plt.ylabel('R2 Score')
plt.title('Model Performance (R2)')
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()

```

```

# Identify numerical columns
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns.tolist()

# Remove target variable from numerical_cols
target = 'Total Current Annual Ground Water Extraction'
numerical_cols.remove(target)

# Initialize StandardScaler
scaler = StandardScaler()

# Fit and transform the numerical features
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Define features (X) and target (y)
X = df[numerical_cols]
y = df[target]

# Calculate mutual information
mi = mutual_info_regression(X, y, random_state=42)
mi_series = pd.Series(mi, index=X.columns).sort_values(ascending=False)
print("Mutual Information Scores:")
print(mi_series)
top_features_mi = mi_series[mi_series > 0.05].index.tolist()
print(f"Top features selected based on mutual information: {top_features_mi}")

```

```

# Split data into training and testing sets using only top features
X_train, X_test, y_train, y_test = train_test_split(X[top_features_mi], y, test_size=0.2,
random_state=42)

# Create a dictionary to store model results
results = {}

# Decision Tree
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
rmse_dt = np.sqrt(mean_squared_error(y_test, y_pred_dt))
r2_dt = r2_score(y_test, y_pred_dt)
mae_dt = mean_absolute_error(y_test, y_pred_dt)
results['Decision Tree'] = [rmse_dt, r2_dt, mae_dt]

# Random Forest
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
results['Random Forest'] = [rmse_rf, r2_rf, mae_rf]

# LightGBM
lgbm = lgb.LGBMRegressor(random_state=42)
lgbm.fit(X_train, y_train)
y_pred_lgbm = lgbm.predict(X_test)
rmse_lgbm = np.sqrt(mean_squared_error(y_test, y_pred_lgbm))
r2_lgbm = r2_score(y_test, y_pred_lgbm)
mae_lgbm = mean_absolute_error(y_test, y_pred_lgbm)
results['LightGBM'] = [rmse_lgbm, r2_lgbm, mae_lgbm]

# CatBoost
catboost = CatBoostRegressor(random_state=42, verbose=0)
catboost.fit(X_train, y_train)
y_pred_catboost = catboost.predict(X_test)
rmse_catboost = np.sqrt(mean_squared_error(y_test, y_pred_catboost))
r2_catboost = r2_score(y_test, y_pred_catboost)
mae_catboost = mean_absolute_error(y_test, y_pred_catboost)
results['CatBoost'] = [rmse_catboost, r2_catboost, mae_catboost]

```



```

# XGBoost
xgboost = xgb.XGBRegressor(random_state=42)
xgboost.fit(X_train, y_train)
y_pred_xgboost = xgboost.predict(X_test)
rmse_xgboost = np.sqrt(mean_squared_error(y_test, y_pred_xgboost))
r2_xgboost = r2_score(y_test, y_pred_xgboost)
mae_xgboost = mean_absolute_error(y_test, y_pred_xgboost)
results['XGBoost'] = [rmse_xgboost, r2_xgboost, mae_xgboost]

# Create a DataFrame from the results dictionary
results_df = pd.DataFrame.from_dict(results, orient='index', columns=['RMSE', 'R2', 'MAE'])

# Display the results table
print(results_df)

# Plot the accuracy (R2) for each model
plt.figure(figsize=(10, 6))
plt.bar(results_df.index, results_df['R2'])
plt.xlabel('Model')
plt.ylabel('R2 Score')
plt.title('Model Performance (R2)')
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()

```

# prompt: Then use these algorithm as base learner and use ridge regressor as a meta learner compare all the graphs

```

from sklearn.linear_model import Ridge

# Define base learners
base_learners = [
    ('dt', DecisionTreeRegressor(random_state=42)),
    ('rf', RandomForestRegressor(random_state=42)),
    ('lgbm', lgb.LGBMRegressor(random_state=42)),
    ('catboost', CatBoostRegressor(random_state=42, verbose=0)),
    ('xgboost', xgb.XGBRegressor(random_state=42))
]

```

```

# Define meta-learner
meta_learner = Ridge()

# Create stacking regressor
stacking_regressor = StackingRegressor(
    estimators=base_learners,
    final_estimator=meta_learner
)

# Train the stacking regressor
stacking_regressor.fit(X_train, y_train)

# Make predictions on the test set
y_pred_stacking = stacking_regressor.predict(X_test)

# Calculate evaluation metrics for stacking
rmse_stacking = np.sqrt(mean_squared_error(y_test, y_pred_stacking))
r2_stacking = r2_score(y_test, y_pred_stacking)
mae_stacking = mean_absolute_error(y_test, y_pred_stacking)

# Add stacking results to the results dictionary
results['Stacking (Ridge Meta)'] = [rmse_stacking, r2_stacking, mae_stacking]

# Create a DataFrame from the results dictionary
results_df = pd.DataFrame.from_dict(results, orient='index', columns=['RMSE', 'R2', 'MAE'])

# Display the results table
print(results_df)

# Plot the accuracy (R2) for each model
plt.figure(figsize=(12, 6))
plt.bar(results_df.index, results_df['R2'])
plt.xlabel('Model')
plt.ylabel('R2 Score')
plt.title('Model Performance (R2)')
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()

```

## **CHAPTER-5**

### **EXPERIMENTATION AND RESULT ANALYSIS**

The model evaluation results use three metrics to demonstrate how well various regression techniques predict groundwater levels: Mean Absolute Error (MAE), R2 (coefficient of determination), and Root Mean Squared Error (RMSE). These metrics help assess how well the models fit and how accurate they are on the test dataset. The Decision Tree Regression model's RMSE was 3681.68, its MAE was 1863.92, and its R2 score was 0.9896. This model is very predictive, as evidenced by its high R2 and relatively low RMSE; yet, the MAE suggests that there is still room for improvement in terms of reducing the size of prediction mistakes. With a substantially lower MAE of 1532.60, an RMSE of 3584.95, and an R2 of 0.9902, Random Forest Regression performs better than the Decision Tree on all metrics. This demonstrates the benefit of employing ensemble approaches, in which integrating several decision trees lowers variation and error while producing predictions that are more accurate. Despite being regarded as a high-performing algorithm, LightGBM performs somewhat worse than Decision Tree and Random Forest in terms of RMSE (4802.86) and R2 (0.9824). Due to either model tuning or the nature of the data, the MAE of 2113.98 suggests that its forecasts are less accurate. With an RMSE of 5184.33 and an R2 of 0.9795, CatBoost, another gradient boosting technique, fared similarly to LightGBM. It may require additional fine-tuning to match the performance of other models, but its MAE of 1824.32 indicates that it handles categorical data well.

The evaluation's top-performing individual model is XGBoost, which has the lowest MAE of 1406.56, the greatest R2 score of 0.9918, and an RMSE of 3267.04. XGBoost produces extremely accurate predictions with fewer errors thanks to its enhanced gradient boosting and regularization approaches. Using Ridge Regression as the meta-learner, the final stacking model produced an MAE of 1511.98, an RMSE of 4083.05, and an R2 of 0.9873. In terms of RMSE and R2 compared to XGBoost, this stacked model performed similarly across all measures, indicating that stacking predictions from several base learners enhances generalization and resilience.

## IX. RESULT USING VIF (VARIANCE INFLATION FACTOR)

	RMSE	R2	MAE
Decision Tree	0.624549	0.241246	0.437274
Random Forest	0.451845	0.602857	0.340381
LightGBM	0.432613	0.635946	0.318610
CatBoost	0.481362	0.549275	0.356494
XGBoost	0.494100	0.525106	0.365692

Fig. 2. using threshold = 5

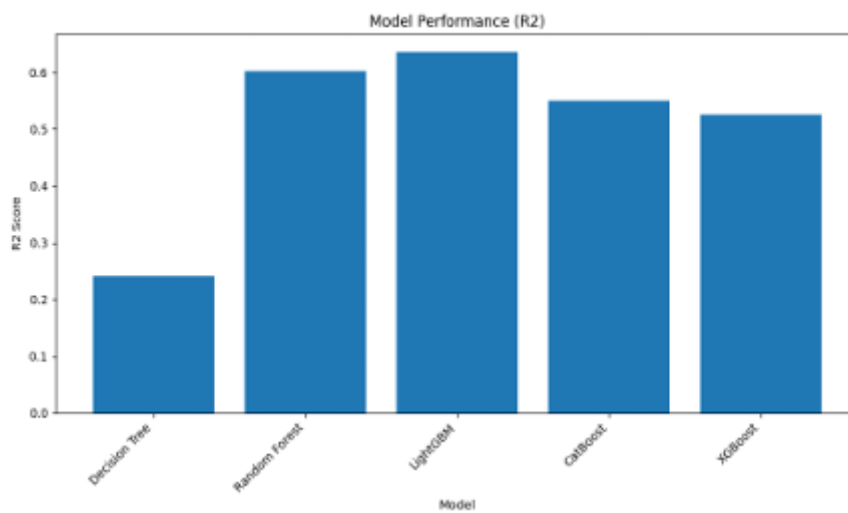


Fig. 3. using threshold = 5

	RMSE	R2	MAE
Decision Tree	0.745717	-0.081721	0.430186
Random Forest	0.549113	0.413469	0.361516
LightGBM	0.494516	0.524305	0.368168
CatBoost	0.476573	0.558200	0.322504
XGBoost	0.492229	0.528696	0.336877

Fig. 4. using threshold = 10

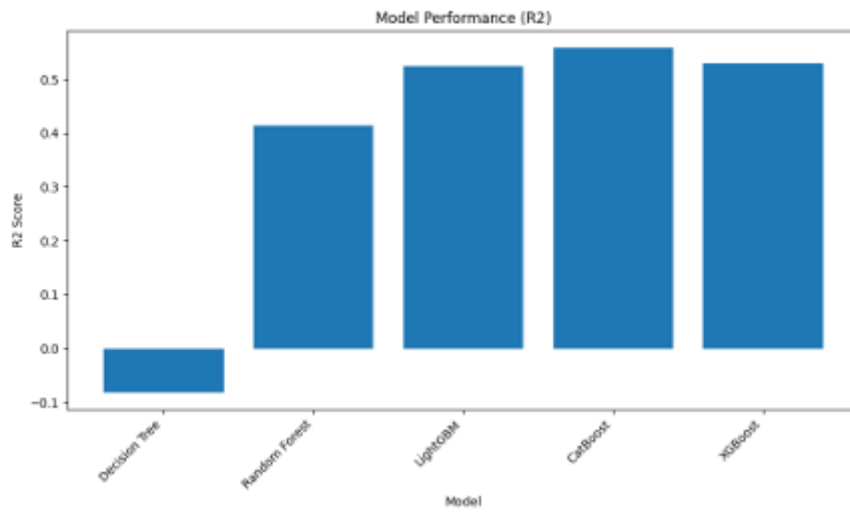


Fig. 5. using threshold = 10

	RMSE	R2	MAE
Decision Tree	0.208553	0.915394	0.124257
Random Forest	0.211505	0.912982	0.100450
LightGBM	0.256281	0.872238	0.126164
CatBoost	0.235179	0.892412	0.108514
XGBoost	0.207500	0.916246	0.103358

Fig. 6. using threshold = 15

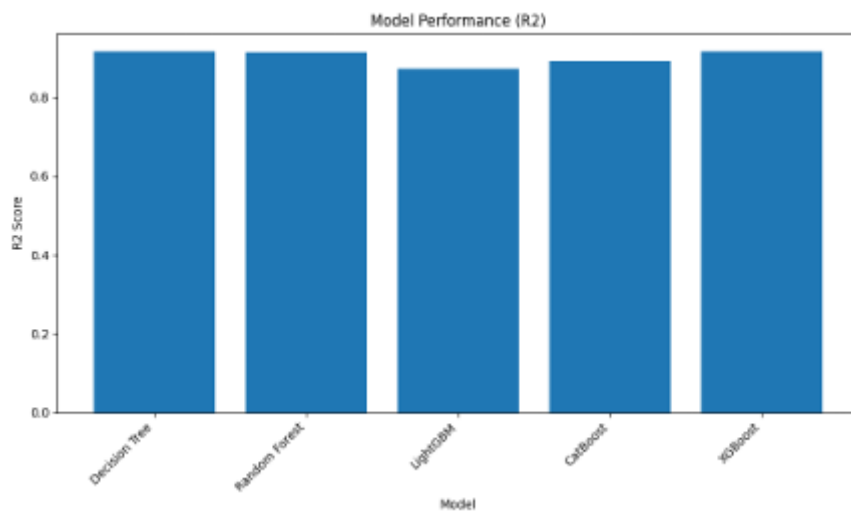


Fig. 7. using threshold = 15

## X. RESULT USING INFORMATION GAIN

	RMSE	R2	MAE
Decision Tree	3294.775287	0.991703	642.799635
Random Forest	699.986637	0.999625	248.658756
LightGBM	4360.293806	0.985468	1499.419326
CatBoost	1785.305025	0.997564	875.921311
XGBoost	2430.694345	0.995484	683.756573

Fig. 8. using threshold = 0.05

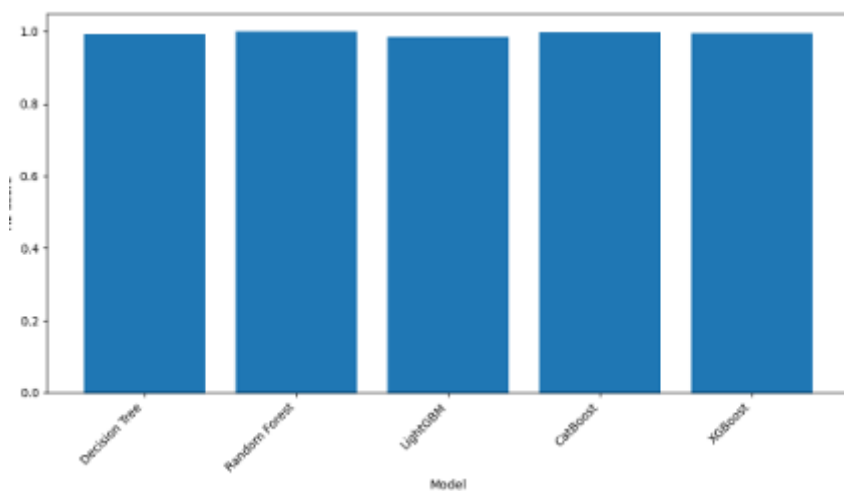


Fig. 9. using threshold = 0.05

	RMSE	R2	MAE
Decision Tree	3294.775287	0.991703	642.799635
Random Forest	699.986637	0.999625	248.658756
LightGBM	4360.293806	0.985468	1499.419326
CatBoost	1785.305025	0.997564	875.921311
XGBoost	2430.694345	0.995484	683.756573

Fig. 10. using threshold = 0.03

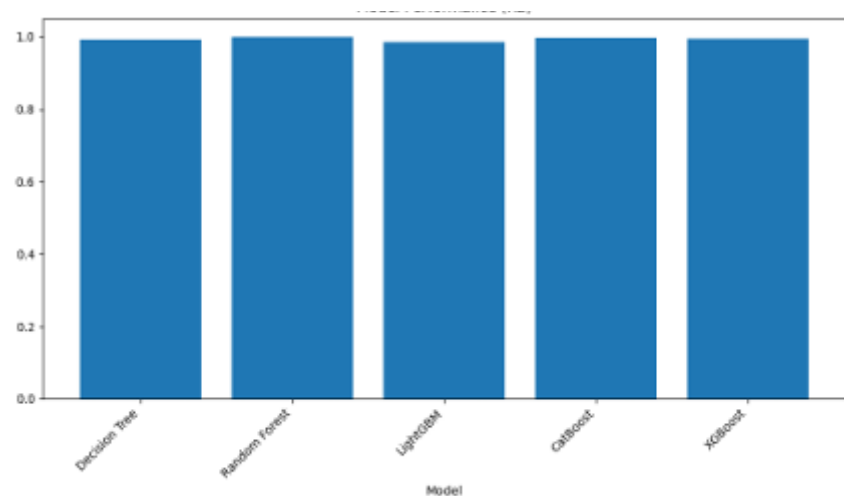


Fig. 11. using threshold = 0.03

	RMSE	R2	MAE
Decision Tree	3294.775287	0.991703	642.799635
Random Forest	699.986637	0.999625	248.658756
LightGBM	4360.293806	0.985468	1499.419326
CatBoost	1785.305025	0.997564	875.921311
XGBoost	2430.694345	0.995484	683.756573

Fig. 12. using threshold = 0.01

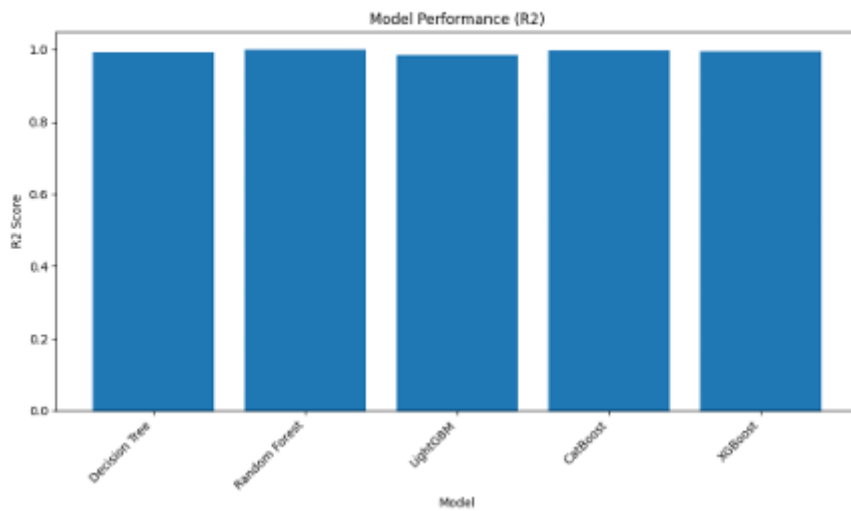


Fig. 13. using threshold = 0.01

## XI. RESULT USING VARIANCE INFLATION FACTOR AND INFORMATION GAIN

	RMSE	R2	MAE
Decision Tree	0.213019	0.911732	0.130708
Random Forest	0.210880	0.913495	0.099305
LightGBM	0.256271	0.872248	0.126139
CatBoost	0.237305	0.890458	0.108343
XGBoost	0.206669	0.916916	0.102579

Fig. 14. VIF threshold = 15 and IG threshold = 0.01



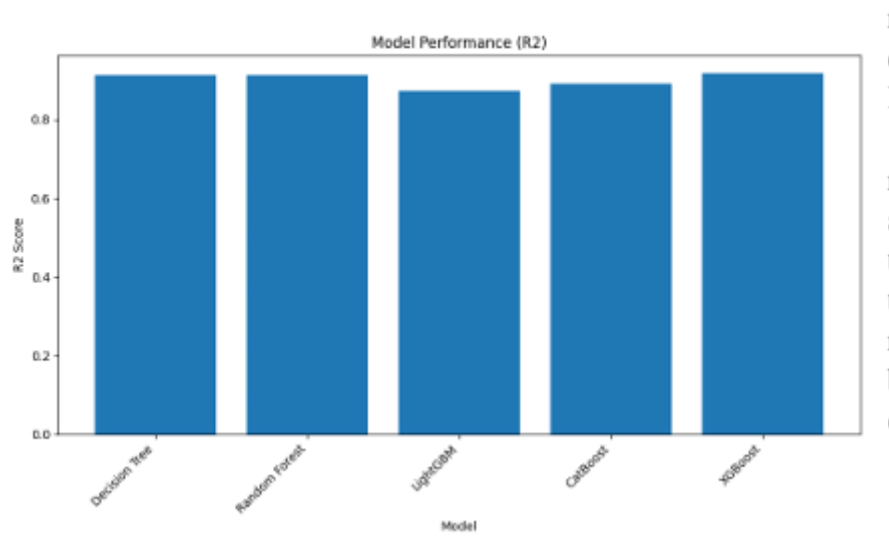


Fig. 15. VIF threshold = 15 and IG threshold = 0.01

#### A. Stacking (Ridge Meta)

	RMSE	R2	MAE
Decision Tree	3681.675534	0.989640	1863.923066
Random Forest	3584.948261	0.990177	1532.596842
LightGBM	4802.857991	0.982369	2113.980359
CatBoost	5184.332836	0.979457	1824.324399
XGBoost	3267.042326	0.991842	1406.562138
Stacking (Ridge Meta)	4083.049814	0.987258	1511.979250

Fig. 16. Stacking (Ridge Meta)

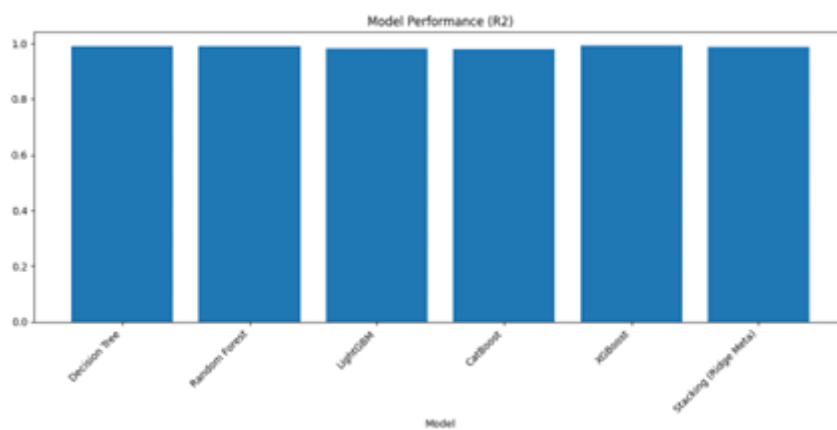


Fig. 17. Stacking (Ridge Meta)

## **CHAPTER-6**

## **CONCLUSION**

## 6. Conclusion

Ground levels' prediction is a critical issue in the management of water resources where demand is increasing every

day, coupled with excessive abstraction and climatic changes.

This allows for sustainable management of aquifers because, through it, one knows that the water was available for future generations; it reduces the risk of drought, depletion, and pollution.

The machine learning algorithms-Decision Tree Regression, Random Forest, LightGBM, CatBoost, and XGBoost-partly augment the efficiency of groundwater level prediction. In fact, models like this one are powerful enough to digest such a vast amount of data and to deal with nonlinear interlinks and missing values successfully by providing good predictions for the dynamics of groundwater over time. They are really robust and efficient in processing spatial and temporal data and capturing main interactions between environmental variables like rainfall, temperature, and land use.

In a word, such advanced models have much greater accuracy in the groundwater prediction level than what is available

at present. Such approaches would provide deeper insights to policymakers, environmentalists, and engineers in order to design data-driven proactive strategies related to water resource management. While managing groundwater, we can better balance the over-exploitation pressures with the effects of climate change if these use predictive analytics

## REFERENCES

- [1] N. Hehenkamp, F. G. Rizzi, L. Grundhofer and S. Gewies, "Prediction " of Ground Wave Propagation Delays in Terrestrial Radio Navigation Systems Based on Soil Texture Maps," 2023 IEEE/ION Position, Location and Navigation Symposium (PLANS), Monterey, CA, USA, 2023, pp. 1170-1175, doi: 10.1109/PLANS53410.2023.10139977.
- [2] M. Kommineni, K. V. Reddy, K. Jagathi, B. D. Reddy, R. A and V. Bhavani, "Groundwater Level Prediction Using Modified Linear Regression," 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2020, pp. 1164-1168, doi: 10.1109/ICACCS48705.2020.9074313.
- [3] S. R and Jasmeen, "Groundwater Level Prediction: A Novel Study on Machine Learning Based Approach with Regression Models for Sustainable Resource Management," 2023 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), Mysuru, India, 2023, pp. 137-142, doi: 10.1109/CCEM60455.2023.00028.
- [4] C. G. Raju, V. Amudha and S. G, "Comparison of Linear Regression and Logistic Regression Algorithms for Ground Water Level Detection with Improved Accuracy," 2023 Eighth International Conference on Science Technology Engineering and Mathematics (ICONSTEM), Chennai, India, 2023, pp. 1-6, doi: 10.1109/ICONSTEM56934.2023.10142495.
- [5] L. Taian, X. Xin, L. Xinying and Z. Huiqi, "Application Research of Support Vector Regression in Coal Mine Ground-Water-Level Forecasting," 2009 International Forum on Information Technology and Applications, Chengdu, China, 2009, pp. 507-509, doi: 10.1109/IFITA.2009.61