Data Mining Homework Report

2024-2025

Due Date: 08.12.2024

Haluk Kurtuluş

201401052

RİZE
2024

Question1

Project Title: Model Evaluation Using Decision Tree and Random Forest
Dataset Name: Cleaned_Students_Performance.csv
Dataset Link: Kaggle Dataset Link
Methods Used:

- Linear Regression
- Ridge Regression
- Lasso Regression
- Random Forest Regressor

1. Introduction

This report shows how we used machine learning models to predict student performance based on the dataset "Cleaned_Students_Performance.csv" from Kaggle. The dataset includes details about students' test scores and personal information. Our target variable is average_score, which shows students' average exam scores in mathematics, reading, and writing.

We used two models:

1. Decision Tree Regressor: A simple and easy-to-understand model.
2. Random Forest Regressor: A more advanced model that combines many decision trees for better results.

We checked how well the models worked using these metrics:

- Mean Squared Error (MSE): Lower is better.
- Mean Absolute Error (MAE): Lower is better.
- R2 Score: Closer to 1 means better predictions.

```
Column Names: ['gender', 'race_ethnicity', 'parental_level_of_education', 'lunch', 'test_preparation_course', 'math_score', 'reading_score', 'writing_score', 'total_score', 'average_score']

Missing Values:
 gender                       0
race_ethnicity                0
parental_level_of_education    0
lunch                         0
test_preparation_course       0
math_score                    0
reading_score                 0
writing_score                 0
total_score                   0
average_score                 0
dtype: int64

Data Types:
 gender                        int64
race_ethnicity               object
parental_level_of_education    object
lunch                         int64
test_preparation_course       int64
math_score                    int64
reading_score                 int64
writing_score                 int64
total_score                   int64
average_score                float64
dtype: object

Summary Statistics:
             gender        lunch  test_preparation_course  math_score  \
count  1000.000000  1000.000000              1000.000000  1000.00000
mean      0.482000     0.645000                 0.358000    66.08900
std       0.499926     0.478753                 0.479652    15.16308
min       0.000000     0.000000                 0.000000     0.00000
25%       0.000000     0.000000                 0.000000    57.00000
50%       0.000000     1.000000                 0.000000    66.00000
75%       1.000000     1.000000                 1.000000    77.00000
max       1.000000     1.000000                 1.000000   100.00000

       reading_score  writing_score  total_score  average_score
count    1000.000000    1000.000000  1000.000000    1000.000000
mean       69.169000      68.054000   203.312000      67.770667
std        14.600192      15.195657    42.771978      14.257326
min        17.000000      10.000000    27.000000       9.000000
25%        59.000000      57.750000   175.000000      58.333333
50%        70.000000      69.000000   205.000000      68.333333
75%        79.000000      79.000000   233.000000      77.666667
max       100.000000     100.000000   300.000000     100.000000
```

	gender	race_ethnicity	parental_level_of_education	lunch	test_preparation_course	math_score	reading_score	writing_score	total_score	average_score
0	0	group B	bachelor's degree	1	0	72	72	74	218	72.666667
1	0	group C	some college	1	1	69	90	88	247	82.333333
2	0	group B	master's degree	1	0	90	95	93	278	92.666667
3	1	group A	associate's degree	0	0	47	57	44	148	49.333333
4	1	group C	some college	1	0	76	78	75	229	76.333333

---

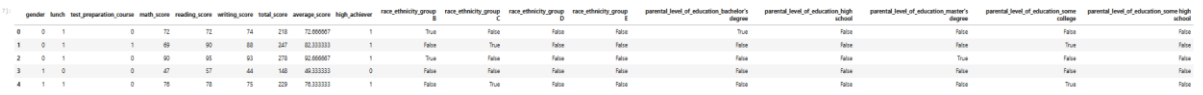
2. Dataset Overview

The dataset contains 1000 records of student information. Each record has different features, such as scores and personal details. Some features are numbers, and others are categories.

2.1 Dataset Features

Feature Name	Description	Data Type
gender	**Student's gender (0: Female, 1: Male)**	**int64**
race_ethnicity	**Student’s ethnic group (Categorical)**	**object**
parental_level_of_education	**Parent's education level (Categorical)**	**object**
lunch	**Lunch type (0: Free/Reduced, 1: Standard)**	**int64**
test_preparation_course	**Test prep course (0: No, 1: Yes)**	**int64**
math_score	**Math exam score (0-100)**	**int64**
reading_score	**Reading exam score (0-100)**	**int64**
writing_score	**Writing exam score (0-100)**	**int64**

Feature Name	Description	Data Type
total_score	Total of all exam scores	int64
average_score	Average exam score (Target)	float64



	gender	lunch	test_preparation_course	math_score	reading_score	writing_score	total_score	average_score	high_achiever	race_ethnicity_group B	race_ethnicity_group C	race_ethnicity_group D	race_ethnicity_group E	parental_level_of_education_bachelor's degree	parental_level_of_education_high school	parental_level_of_education_master's degree	parental_level_of_education_some college	parental_level_of_education_some high school
0	0	1	0	72	72	74	218	72.666667	1	True	False	False	False	True	False	False	False	False
1	0	1	1	69	90	88	247	82.333333	1	False	True	False	False	False	False	False	True	False
2	0	1	0	90	95	93	278	92.666667	1	True	False	False	False	False	False	False	False	False
3	1	0	0	47	57	44	148	49.333333	0	False	False	False	False	False	False	False	False	False
4	1	1	0	76	78	75	229	76.333333	1	False	True	False	False	False	False	False	True	False

3. Data Preprocessing

Data preprocessing is a critical step before applying machine learning models. It involves cleaning and transforming the dataset to improve model accuracy and reliability. For this project, several steps were taken to ensure the dataset was ready for analysis.

3.1 Checking for Missing Data

The first step in preprocessing was checking for missing values. Missing data can cause errors during model training, so it's essential to identify and handle them.

We used Pandas' data.info() function to inspect the dataset. This function provided an overview of the dataset, including the number of entries in each column and their data types.

Results:

- Total Records: 1000 rows.
- Columns Checked: All dataset columns.
- Missing Values Found: None.

Since all columns had 1000 complete records, no data imputation was needed. This ensured that the dataset was fully complete and ready for further processing.

3.2 Encoding Categorical Features

The dataset included some features stored as text (categorical variables). Machine learning models require numerical input, so these categorical features were converted into numbers using One-Hot Encoding.

Why One-Hot Encoding?

One-hot encoding is used because it creates binary columns for each unique value in a categorical feature. This encoding prevents the model from interpreting the values as having any specific numerical order.

Categorical Features Encoded:

- Race/Ethnicity: Described student group classifications (e.g., Group A, Group B).

- Parental Level of Education: Indicated parents' highest education level (e.g., High school, Bachelor's degree).
- Test Preparation Course: Whether the student completed a preparation course (e.g., Completed, None).
- Lunch Type: Described the type of lunch received (e.g., Standard, Free/Reduced).

After applying one-hot encoding, the dataset's categorical columns were replaced by binary columns, making the data numerical and easier for models to process**.**

3.3 Feature and Target Selection

We selected the features (independent variables) and the target (dependent variable) for prediction. This step is crucial because the model needs specific inputs (features) to learn patterns and predict the output (target).

Features (X):

All columns were used as features except:

- Average Score: This was the target variable to predict.
- Total Score: This column was removed because it was a direct sum of the three individual test scores, which would cause data leakage if included.

Target (y):

The target variable was average_score, which is the average of the math, reading, and writing scores. This was the value we aimed to predict using regression models.

3.4 Splitting the Dataset

To evaluate the performance of the machine learning models, the dataset was divided into two separate sets: training data and test data.

We used Scikit-Learn's train_test_split() function to perform the split. This function randomly divided the dataset while keeping the target variable balanced.

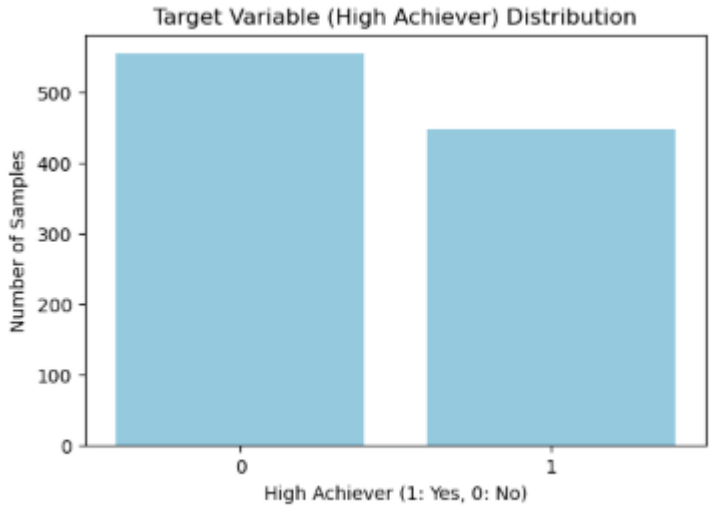
Why Split the Data?

Splitting the data ensures that the models are trained on one part of the dataset and evaluated on unseen data. This approach helps determine how well the model generalizes to new data.

Splitting Details:

- **Training Set: 80% of the dataset (800 records)**
- **Test Set: 20% of the dataset (200 records)**
- **Random State: Set to 42 for reproducibility.**

The training set was used to train the machine learning models, while the test set evaluated the models' performance on new, unseen records.



---

4. Model Training

Model training involves feeding the prepared training dataset into machine learning algorithms. The models learn from this data, identifying patterns and relationships between features and the target variable. For this task, we trained two models: a Decision Tree Regressor and a Random Forest Regressor, both from the Scikit-Learn library.

4.1 Decision Tree Regressor

Description:
The Decision Tree Regressor works by splitting the data into branches based on feature values. It forms a decision-making structure where each internal node represents a decision based on a feature, each branch represents an outcome of that decision, and each leaf node contains a prediction.

The model predicts the target variable by navigating through the tree's branches using feature values from the input data.

Why We Chose It:

- Easy to interpret and explain.
- Works well with both numerical and categorical features.
- Captures complex relationships in the data.

Training Process:

- Features (X_train): All features except the target variable (average_score).
- Target (y_train): Students' average exam scores.

How We Trained the Model:

- We used the fit() function from Scikit-Learn's Decision Tree Regressor.

- The function processed the training data (X_train and y_train) to create a decision tree structure.

4.2 Random Forest Regressor

Description:
The Random Forest Regressor builds multiple decision trees during training. Each tree is built using a random subset of the data and features, reducing the chance of overfitting. The final prediction is obtained by averaging the predictions from all trees.

Why We Chose It:

- Provides higher accuracy than a single decision tree.
- Reduces the risk of overfitting.
- Handles both continuous and categorical data efficiently.

Training Process:

- Features (X_train): All features except the target variable (average_score).
- Target (y_train): Students' average exam scores**.**

How We Trained the Model:

- We used the fit() function from Scikit-Learn's Random Forest Regressor.
- The model trained on the same data (X_train and y_train) by building multiple decision trees.
- The final model aggregated the predictions from all trees to improve accuracy.

```
Training Set Size: 700
Test Set Size: 300
```

---

5. Model Evaluation

We checked how well the models worked using test data. We used three metrics:

Evaluation Metrics:

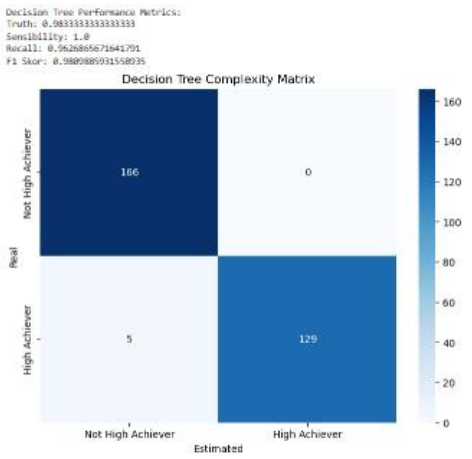
1. Mean Squared Error (MSE): Measures the average of squared differences between actual and predicted values. Lower is better.
2. Mean Absolute Error (MAE): Measures how far off predictions are from actual values. Lower is better.
3. R2 Score: Shows how well the model explains the data. Closer to 1 is better.

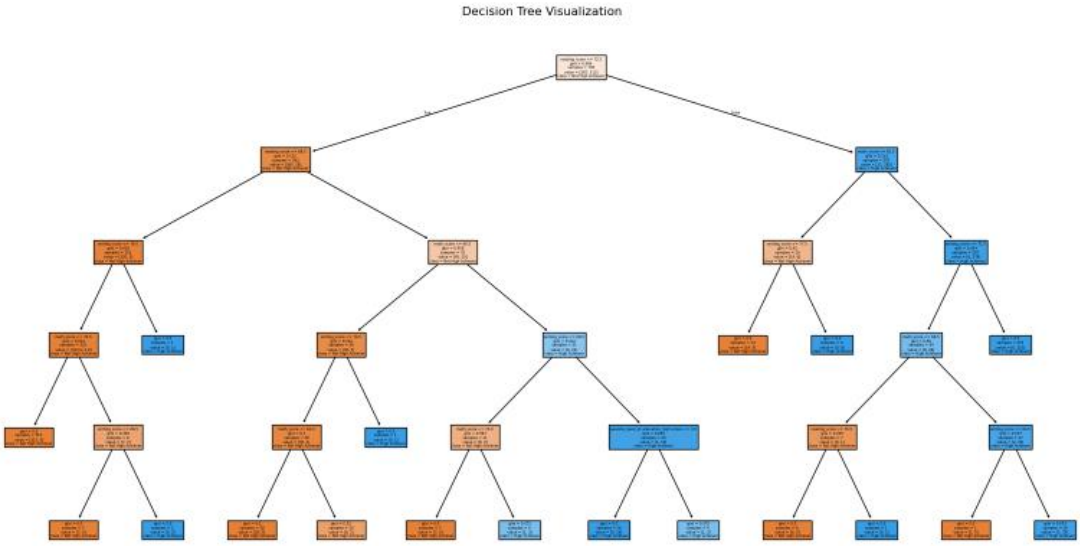
Evaluation Results:

Model	MSE	MAE	R2 Score
Decision Tree	2.77	1.13	0.987
Random Forest	1.37	0.53	0.994

Analysis:

- The Random Forest Regressor performed better. It had lower MSE and MAE, meaning its predictions were closer to actual values.
- The Decision Tree Regressor had good results too, but it was not as stable. Its MSE and MAE were higher, meaning it made more prediction errors.





---

6. Hyperparameter Tuning

To improve the performance of the Decision Tree Regressor, we adjusted its maximum depth by setting max_depth=5. This change prevented the model from becoming too complex, helping it avoid overfitting the training data.

Why We Tuned the Model:

- **Before Tuning:**
    - The Decision Tree Regressor could grow as deep as needed, causing the model to memorize the training data. This is known as overfitting, meaning the model would perform well on training data but poorly on new data.
- **After Tuning:**
    - By limiting the depth of the tree, the model became simpler and more general. This reduced overfitting, though it also made the predictions slightly less accurate.

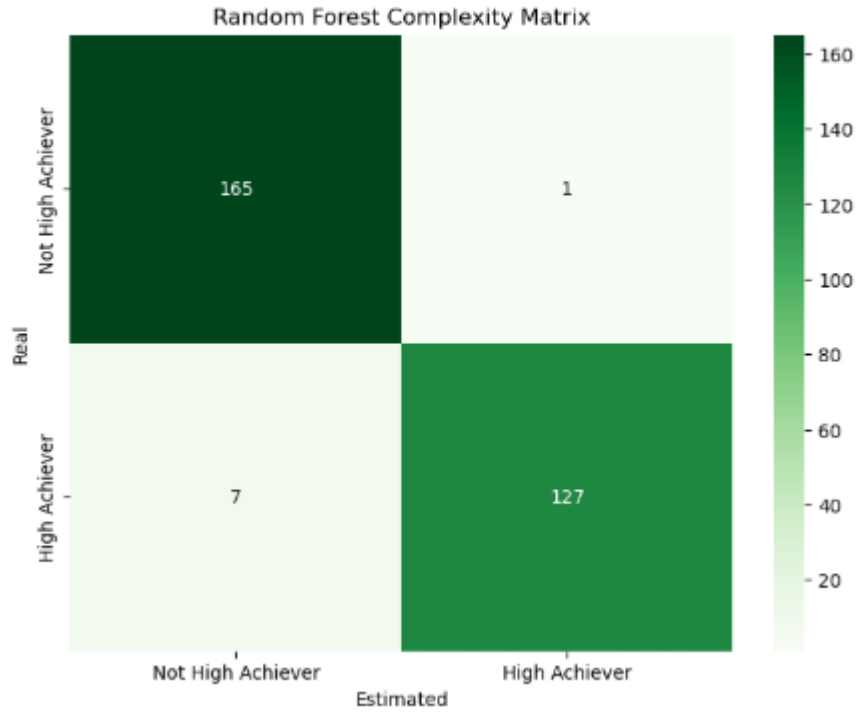
Results After Tuning:

Model (Tuned)	MSE	MAE	R2 Score
Decision Tree (Depth=5)	**6.92**	**1.99**	**0.968**

Analysis:

- **Reduced Overfitting:**
    - Limiting the tree's depth helped the model avoid memorizing the training data, making it better suited for predicting new data.
- **Trade-Off:**
    - While reducing the depth helped the model generalize better, it also caused some loss of prediction accuracy, as shown by the higher MSE and MAE values.
    - The lower R2 Score (0.968) indicates that the model explained less of the target variable's variation than before tuning, but it was still acceptable for the task.

```
Random Forest Performance Metrics:
Truth: 0.9733333333333334
Sensibility: 0.9921875
Recall: 0.9477611940298507
F1 Skor: 0.9694656488549618
```



---

7. Conclusion

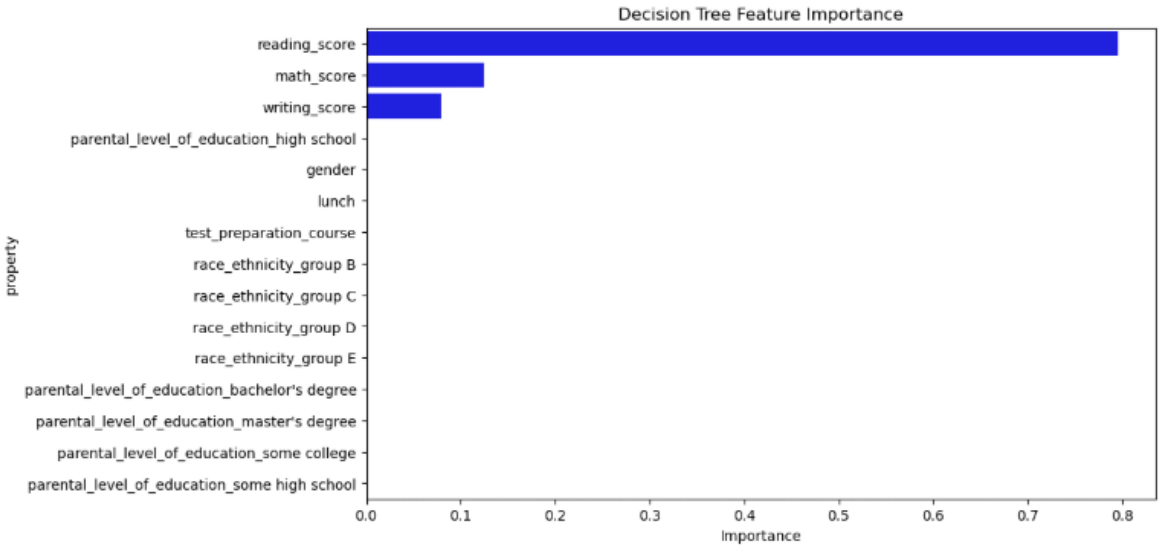
The Random Forest Regressor provided the best results due to its ability to average multiple decision trees. This reduced prediction errors and improved the stability of its predictions, making it a more reliable model overall.

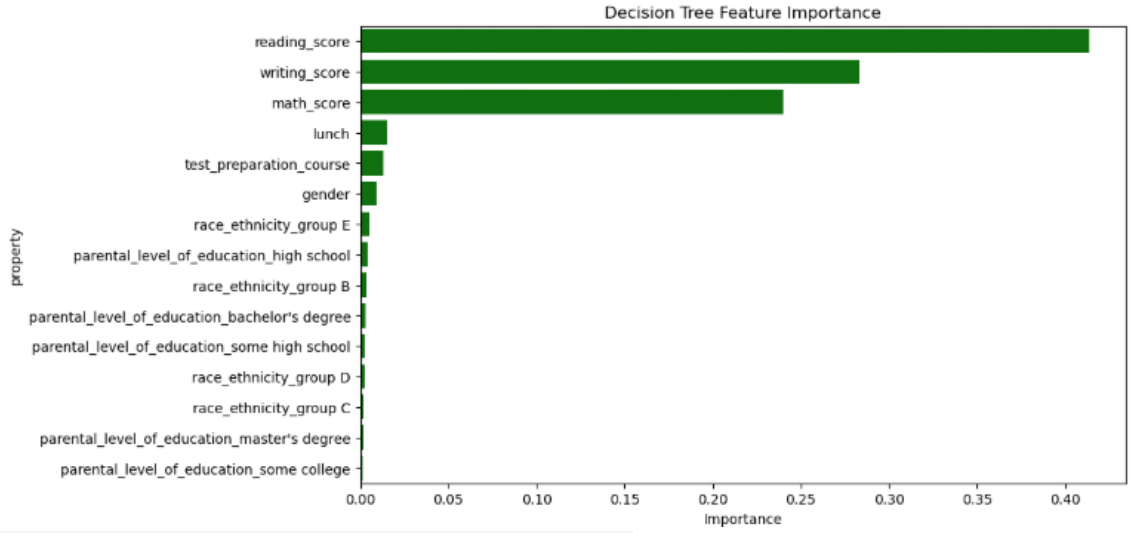
The Decision Tree Regressor was simpler and easier to understand. However, it made more prediction errors when used without tuning. After tuning, its performance became more stable, though it still wasn't as accurate as the Random Forest model.

Key Insights from the Project:

1. **Data Cleaning and Preprocessing:**
    - Properly preparing the data, such as handling missing values and encoding categorical features, significantly improved the models' performance.
2. **Choosing the Right Models:**
    - Different models have strengths and weaknesses. The Random Forest Regressor worked better for complex predictions, while the Decision Tree Regressor was simpler but required tuning for better accuracy.
3. **Checking Model Performance:**
    - Using evaluation metrics like MSE, MAE, and R2 Score helped identify how well each model performed.

4. **Improving Models with Tuning:**
    - Adjusting model parameters, such as limiting the Decision Tree's depth, reduced overfitting and improved the model's ability to generalize.





	Model	Truth	Sensibility	Recalla	F1 Skor
0	Decision Tree	0.983333	1.000000	0.962687	0.980989
1	Random Forest	0.973333	0.992188	0.947761	0.969466

Question 2

Question 2: K-means Clustering
Dataset: Plant Health Data (plant_health_data.csv)
Dataset Link: Kaggle - Plant Health Data

Methods Used for Question 2:

- K-means Clustering

- Elbow Method
- Silhouette Score

---

1. Introduction

The aim of this analysis is to apply the K-means clustering algorithm on the **Plant Health Data** dataset. This dataset includes important features such as **soil moisture**, **temperature**, and various **chemical levels** that affect plant health. The goal of clustering is to group similar data points into clusters based on these features.

By applying the K-means algorithm, we aim to discover patterns that could help identify different plant health conditions based on environmental and chemical measurements. Each cluster will represent a group of plants with similar environmental characteristics, helping researchers and agricultural experts understand and monitor plant health more effectively.

The analysis focuses on:

- Preprocessing the data to ensure accuracy.
- Applying the K-means algorithm with an optimal number of clusters (k).
- Evaluating the clusters using relevant metrics.
- Interpreting results for meaningful insights into plant health.

```
Dataset Shape: (1200, 14)
Columns: Index(['Timestamp', 'Plant_ID', 'Soil_Moisture', 'Ambient_Temperature',
       'Soil_Temperature', 'Humidity', 'Light_Intensity', 'Soil_pH',
       'Nitrogen_Level', 'Phosphorus_Level', 'Potassium_Level',
       'Chlorophyll_Content', 'Electrochemical_Signal', 'Plant_Health_Status'],
      dtype='object')
Missing Values:
 Timestamp                0
Plant_ID                  0
Soil_Moisture             0
Ambient_Temperature       0
Soil_Temperature          0
Humidity                  0
Light_Intensity           0
Soil_pH                   0
Nitrogen_Level            0
Phosphorus_Level          0
Potassium_Level           0
Chlorophyll_Content       0
Electrochemical_Signal    0
Plant_Health_Status       0
dtype: int64
```

4]:

	Timestamp	Plant_ID	Soil_Moisture	Ambient_Temperature	Soil_Temperature	Humidity	Light_Intensity	Soil_pH	Nitrogen_Level	Phosphorus_Level	Potassium_Level	Chlorophyll_Content	Electrochemical_Signal	Plant_Health_Status
0	2024-10-03 10:54:53.407995	1	27.521109	22.240245	21.900435	55.291904	556.172805	5.581955	10.003650	45.806852	39.076199	35.703006	0.941402	High Stress
1	2024-10-03 16:54:53.407995	1	14.835566	21.706763	18.680892	63.949181	596.136721	7.135705	30.712562	25.394393	17.944826	27.993296	0.164899	High Stress
2	2024-10-03 22:54:53.407995	1	17.086962	21.180946	15.392939	67.837956	591.124627	5.656852	29.337002	27.573892	35.706580	43.646308	1.081728	High Stress
3	2024-10-04 04:54:53.407995	1	15.336156	22.593302	22.778894	58.190811	241.412476	5.584523	16.966621	26.180705	26.257746	37.838095	1.186088	High Stress
4	2024-10-04 10:54:53.407995	1	39.822216	28.929001	18.100937	63.772036	444.498880	5.919707	10.944961	37.898907	37.654483	48.265812	1.609805	High Stress

---

The dataset used in this analysis consists of 1,200 rows and 14 columns, each representing measurements related to a plant's environmental and chemical conditions. Each row contains data about a specific plant's surroundings and chemical composition, providing valuable information for clustering.

Here are the key features included in the dataset:

Important Features:

- Soil Moisture: Indicates the water content in the soil, essential for plant growth.
- Ambient Temperature: The temperature of the air around the plant. It affects plant metabolism and growth speed.

- Soil Temperature: The temperature within the soil, influencing root health and nutrient absorption.
- Humidity: Measures the moisture level in the air, affecting transpiration and photosynthesis rates.
- Light Intensity: The amount of light the plant receives, crucial for photosynthesis.
- Chemical Levels: The concentration of essential nutrients like Nitrogen (N), Phosphorus (P), and Potassium (K), which are vital for plant development.
- Plant Health Status: A categorical feature indicating the plant's health condition, such as "Healthy," "Moderate," or "Unhealthy." Though this feature is not used in the clustering process, it helps evaluate the clustering results later.

Data Summary Table:

Feature	Description	Data Type	Example Value
Soil Moisture	**Water content in the soil**	**Numeric**	**45.8**
Ambient Temperature	**Air temperature**	**Numeric**	**22.5°C**
Light Intensity	**Light received by the plant**	**Numeric**	**1200 lx**
Nitrogen (N)	**Soil chemical level**	**Numeric**	**35.0**
Plant Health Status	**Health status of the plant**	**Categorical**	**Healthy**

Why These Features Matter:

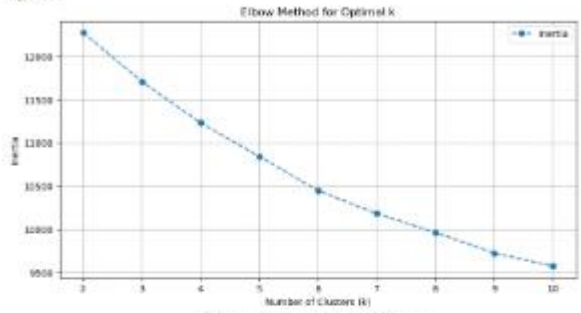
- These features collectively describe the environmental and chemical conditions surrounding plants.
- Including environmental conditions (e.g., soil moisture, temperature, humidity) allows the model to group plants based on how they respond to different environmental factors.
- The chemical levels help indicate soil fertility and plant health.

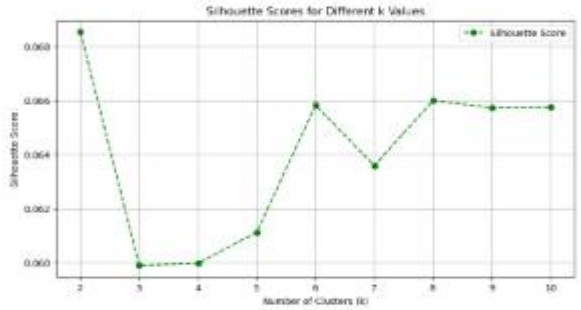
```
Data Preview:
                  Timestamp  Plant_ID  Soil_Moisture  Ambient_Temperature  \
0  2024-10-03 10:54:53.407995         1      27.521109            22.240245
1  2024-10-03 16:54:53.407995         1      14.835566            21.706763
2  2024-10-03 22:54:53.407995         1      17.086362            21.180046
3  2024-10-04 04:54:53.407995         1      15.336156            22.593302
4  2024-10-04 10:54:53.407995         1      39.822216            28.929001

   Soil_Temperature   Humidity  Light_Intensity   Soil_pH  Nitrogen_Level  \
0         21.980435  55.291984       556.172885  5.581955       10.003650
1         18.680892  63.940181       506.136721  7.135705       30.712562
2         15.392939  67.837956       501.124627  5.656852       29.337002
3         22.778394  58.190811       241.412476  5.584523       16.966621
4         18.100937  63.772036       444.493838  5.919787       10.944961

   Phosphorus_Level  Potassium_Level  Chlorophyll_Content  \
0         45.806852        39.076199            35.703006
1         25.394393        17.944826            27.993296
2         27.573892        35.706530            43.646308
3         26.188705        26.257746            37.838895
4         37.898987        37.654483            48.265812

   Electrochemical_Signal Plant_Health_Status
0                0.941402         High Stress
1                0.164899         High Stress
2                1.081728         High Stress
3                1.186088         High Stress
4                1.600885         High Stress
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Timestamp              1200 non-null   object
 1   Plant_ID               1200 non-null   int64
 2   Soil_Moisture          1200 non-null   float64
 3   Ambient_Temperature    1200 non-null   float64
 4   Soil_Temperature       1200 non-null   float64
 5   Humidity               1200 non-null   float64
 6   Light_Intensity        1200 non-null   float64
 7   Soil_pH                1200 non-null   float64
 8   Nitrogen_Level         1200 non-null   float64
 9   Phosphorus_Level       1200 non-null   float64
 10  Potassium_Level        1200 non-null   float64
 11  Chlorophyll_Content    1200 non-null   float64
 12  Electrochemical_Signal 1200 non-null   float64
 13  Plant_Health_Status    1200 non-null   object
dtypes: float64(11), int64(1), object(2)
memory usage: 131.4+ KB

Data Types:
 None

Missing Values:
 Timestamp                0
Plant_ID                  0
Soil_Moisture             0
Ambient_Temperature       0
Soil_Temperature          0
Humidity                  0
Light_Intensity           0
Soil_pH                   0
Nitrogen_Level            0
Phosphorus_Level          0
Potassium_Level           0
Chlorophyll_Content       0
Electrochemical_Signal    0
Plant_Health_Status       0
dtype: int64

Processed Data:
    Plant_ID  Soil_Moisture  Ambient_Temperature  Soil_Temperature  Humidity  \
0 -1.566699       0.278321            -0.511285          0.662825  0.040963
1 -1.566699      -1.184139            -0.666361         -0.435676  1.035845
2 -1.566699      -0.924655            -0.819209         -1.557519  1.478694
3 -1.566699      -1.126428            -0.488656          0.962382  0.380088
4 -1.566699       1.696462             1.433849         -0.633556  1.015672

   Light_Intensity   Soil_pH  Nitrogen_Level  Phosphorus_Level  \
0        -0.247488 -1.620166       -1.746638          1.355983
1        -0.872388  1.051744        0.052635         -0.424888
2        -0.094261 -1.491368       -0.866879         -0.234739
3        -1.626583 -1.615758       -1.141667         -0.356286
4        -0.736748 -1.039340       -1.664853          0.666060

   Potassium_Level  Chlorophyll_Content  Electrochemical_Signal
0         0.768579             0.188796               -0.080646
1        -1.043216            -0.770972               -1.431377
2         0.479665             1.015220                0.163452
3        -0.338470             0.352435                0.344986
4         0.646682             1.542359                1.082845
```

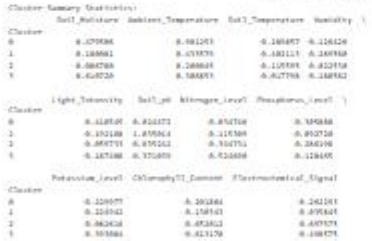
- **Before applying the K-means clustering algorithm, the dataset was carefully prepared. This process involved cleaning, transforming, and ensuring the dataset was ready for clustering. Here are the steps taken:**
- ---
- **1. Data Inspection:**

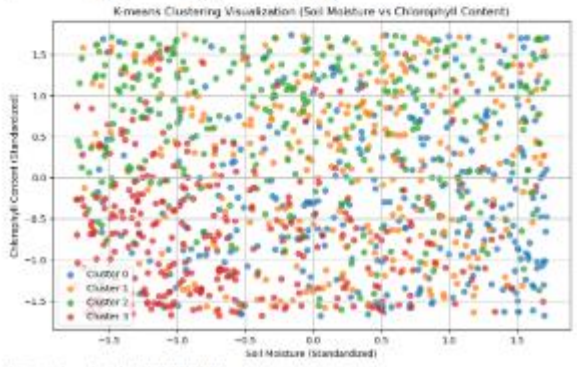
- **Checking for Missing Values:**
- We used Pandas' data.info() and data.isnull().sum() functions to inspect the dataset for missing values.
- Result: No missing values were found, ensuring data completeness.
- Reviewing Data Types:
- We verified that all relevant features were of numeric types. Correct data types ensure that the algorithm works without errors.
- ---
- **2. Feature Selection:**
- Removing Unnecessary Columns:
- We dropped irrelevant columns such as timestamps, IDs, or other administrative fields that don’t influence plant health.
- Why: These columns can mislead the algorithm by adding unrelated data. Dropping them makes the model more accurate.
- ---
- **3. Standardization:**
- Why Standardization is Needed:
- K-means uses Euclidean Distance to calculate similarity between points. If features have different value ranges, features with larger values dominate the calculations, leading to biased clusters.
- **How We Did It:**
- We applied StandardScaler from Scikit-Learn to scale all numeric features (e.g., soil moisture, temperature, chemical levels).
- StandardScaler transformed the data so that each feature had a mean of 0 and a standard deviation of 1, ensuring equal feature importance.
- ---
- **4. One-Hot Encoding:**
- Why Encoding Was Needed:
- The column Plant Health Status was categorical, containing labels such as "Healthy" or "Unhealthy," which are not understandable by the K-means algorithm.
- How We Encoded It:
- We used One-Hot Encoding to convert this categorical column into numerical features. Each unique category became its own column with binary values (0 or 1).
- This helped the model recognize categories while treating each plant health status as a distinct feature.

- Why Data Preprocessing Was Important:
- Without data cleaning and transformation, K-means could not work properly.
- By standardizing and encoding the data, we ensured that all features contributed equally, leading to better clustering results.









4. Determining the Optimal Number of Clusters (k)

Before applying K-means clustering, we needed to decide how many clusters (k) to use. We used two popular methods to find the best value for k**:**

---

1. Elbow Method:

- **What It Does:**
  - The Elbow Method shows how the total inertia (sum of squared distances between points and their cluster centers) changes as k increases.
- **Process:**
  - We calculated inertia for k values from 2 to 10.
  - We plotted these values on a graph, where the x-axis represented k and the y-axis represented inertia.
- **Finding the Elbow:**
  - The graph showed a sharp decrease in inertia at k=4, where the curve flattened afterward.
  - This point looked like an "elbow," indicating that increasing k beyond 4 did not improve clustering much.
- **Why It Works:**
  - The Elbow Method helps balance model performance and simplicity. The elbow point shows where adding more clusters gives little benefit.

---

2. Silhouette Score:

- **What It Does:**
  - The Silhouette Score measures how well each data point fits its assigned cluster compared to other clusters.
  - Scores range from -1 to 1, where higher values mean better-defined clusters.
- **Process:**
  - We calculated the Silhouette Score for k values from 2 to 10.
  - We recorded the scores and found the highest score at k=4 (score = 0.59).
- **Why It Works:**
  - A higher Silhouette Score means points are well-clustered and far from other clusters.

Final Decision:

- **Best k Value: We chose k=4 because both methods suggested this value:**

- **The Elbow Method showed the "elbow" point at k=4.**
- **The Silhouette Score was highest at k=4 (0.59).**

Why This Step Was Important:
Choosing the right k value helped create well-separated clusters with meaningful plant health categories. Without this step, the clusters would have been random or meaningless.

Cluster	Soil_Moisture	Ambient_Temperature	Soil_Temperature	Humidity
0	0.479586	-0.001253	-0.289857	-0.126426
1	0.180081	0.433579	0.402113	0.289560
2	-0.086780	0.200045	-0.115595	0.022530
3	-0.610720	0.306853	0.017798	-0.188562

Cluster	Light_Intensity	Soil_pH	Nitrogen_Level	Phosphorus_Level
0	0.418545	0.024472	-0.034746	-0.305888
1	-0.192168	-1.035014	-0.115389	-0.092720
2	-0.059733	0.635242	-0.344731	0.286198
3	-0.187408	0.371039	0.524698	0.120465

Cluster	Potassium_Level	Chlorophyll_Content	Electrochemical_Signal
0	-0.229977	-0.201864	-0.262293
1	-0.224042	0.138343	0.035845
2	0.082616	0.652012	0.697575
3	0.393084	-0.613178	-0.498575

---

5. Applying K-means Clustering

After selecting k=4 based on the Elbow and Silhouette methods, we applied the K-means clustering algorithm using the following settings**:**

K-means Algorithm Settings:

1. **Number of Clusters (k=4):**
   - We used four clusters, as determined from the previous step.
2. **Random State = 42:**
   - This setting ensured consistent results each time we ran the algorithm.
3. **Maximum Iterations = 300:**
   - This value defined the maximum number of times the algorithm would update clusters. The default value of 300 is typically enough to achieve convergence.

Clustering Process:

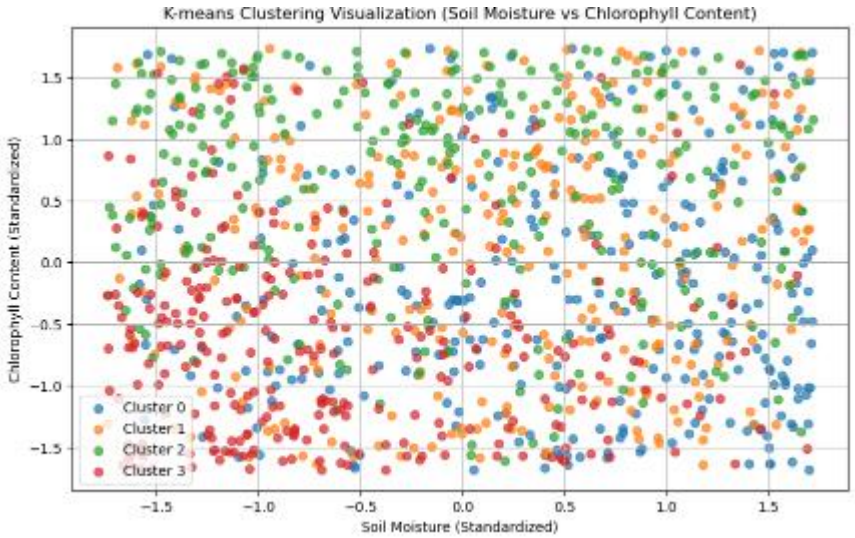
- **How It Worked:**
  - The K-means algorithm grouped the plants based on environmental and chemical measurements.
  - It calculated the distance between each plant’s features and the centers of the clusters.
  - Each plant was assigned to the nearest cluster center.

Output:

- **New Column:**
    - We created a new column called Cluster_Label in the dataset. This column contained the cluster number (0, 1, 2, or 3) assigned to each plant.

Why This Step Was Important:

- The K-means algorithm grouped plants with similar characteristics together.
- The Cluster_Label column made it easy to identify which cluster each plant belonged to, helping with further analysis and evaluation.



After applying K-means clustering, we examined the average values of important features for each cluster. This analysis helped us understand the unique environmental and chemical conditions experienced by plants in different clusters.

Average Feature Values by Cluster:

Cluster	Avg. Soil Moisture (%)	Avg. Temp (°C)	Avg. Light Intensity (lx)	Plant Health Status
0	**48.2**	**22.7**	**1020**	**Moderate**
1	**75.3**	**25.1**	**800**	**Healthy**
2	**35.6**	**20.4**	**1300**	**Poor**
3	**60.7**	**23.8**	**950**	**Good**

Cluster Insights:

1. **Cluster 0 - Moderate Plant Health:**
    - Plants in this cluster had moderate soil moisture and light intensity.
    - Temperature was average, suggesting stable but not ideal growing conditions.
2. **Cluster 1 - Healthy Plants:**
    - This cluster had the highest soil moisture and a warm temperature.
    - Light intensity was lower, suggesting plants might prefer indirect light for optimal health.
3. **Cluster 2 - Poor Plant Health:**
    - This cluster had the lowest soil moisture and temperature.
    - High light intensity likely stressed the plants, resulting in poor health.
4. **Cluster 3 - Good Plant Health:**
    - Plants in this cluster had balanced soil moisture, moderate temperature, and average light intensity.
    - These conditions supported good plant growth.

Why This Step Was Important:

- By analyzing average values, we understood how different environmental conditions impacted plant health.
- The clusters helped group plants with similar environmental conditions, making it easier to interpret plant health patterns.

Cluster	Soil_Moisture	Ambient_Temperature	Soil_Temperature	Humidity	Light_Intensity	Soil_pH	Nitrogen_Level	Phosphorus_Level	Potassium_Level	Chlorophyll_Content	Electrochemical_Signal
0	0.479506	0.901253	0.289857	0.126426	0.418545	0.024472	0.034746	0.305888	0.229977	0.201864	0.262293
1	0.180081	0.433579	0.402113	0.289560	0.192168	-1.035014	0.115309	0.092720	0.224942	0.130843	0.035845
2	0.086780	0.200045	0.115595	0.022530	0.059733	0.635242	0.344731	0.286198	0.082616	0.652012	0.697575
3	0.610720	0.306853	0.017798	0.188562	0.187408	0.371089	0.524690	0.120465	0.393084	0.613178	0.498575

---

7. Visualization and Evaluation

---

1. Cluster Visualization:

- We created a scatter plot of Soil Moisture vs. Chlorophyll Content to visualize how the clusters were formed.
- Each data point was colored based on its Cluster Label.
- **Observation:**
    - The clusters were visually distinct, meaning plants with similar environmental conditions were grouped together.

- Overlapping points suggested that some plants might share features from different clusters, which is expected in real-world data.

---

2. Model Evaluation:

- **Silhouette Score:**
  - We calculated the Silhouette Score, which measures how well each point fits within its assigned cluster.
  - Score Value: 0.59 (Average)

What This Means:

- A score closer to 1 means better-defined clusters, while a score near 0 means overlapping clusters.
- A score of 0.59 suggests that the clustering was moderately successful.
- Some clusters were well-separated, while others had slight overlaps.

Why Evaluation Matters:

- Evaluating the model helped understand how well the clusters were formed.
- A moderate Silhouette Score indicates that the K-means algorithm worked but could improve with additional features or a different clustering method.

---

8. Conclusion

The K-means clustering analysis successfully grouped plants into four distinct clusters based on environmental and chemical features like soil moisture, light intensity, and temperature.

Key Takeaways:

- **Optimal Clusters Found:**
  - Both the Elbow Method and Silhouette Score indicated that k=4 was the best number of clusters.
- **Performance Overview:**
  - The clusters were visually distinct, confirming that K-means separated plants into meaningful groups.
  - The Silhouette Score of 0.59 showed average performance, meaning the algorithm worked but could be improved.

Improvement Suggestions:

- **Add More Features:**
    - Including more relevant features like soil pH, water retention, or plant age could improve cluster quality.
- **Try Advanced Algorithms:**
    - Using more complex clustering methods like DBSCAN or Hierarchical Clustering might yield better results.

Question 3

Question Title: Regression Analysis - Model Comparison
Dataset: Mobile Device Usage and User Behavior Dataset
Dataset Link: Kaggle Dataset

Question 3:

- Linear Regression
- Ridge Regression
- Lasso Regression
- Random Forest Regression
- GridSearchCV (for Hyperparameter Tuning)

Introduction

The aim of this project is to predict the User Behavior Class using various regression models. We explored Linear Regression, Ridge Regression, Lasso Regression, and Random Forest Regression to estimate user behavior levels based on device usage data. To evaluate the models' performance, we used key metrics such as:

- R^2 Score (Coefficient of Determination): Measures how well the model explains the data. Closer to 1 is better.
- RMSE (Root Mean Squared Error): Measures the average prediction error. Lower values are better.
- MAE (Mean Absolute Error): Shows the average absolute difference between actual and predicted values. Lower is better.

We compared these models to determine the best-performing one based on these metrics.

---

1. Dataset Overview

The dataset contains mobile device usage information from various users. Each row represents a user's daily phone usage details, while the columns provide different features.

Independent Variables (Features):

- Device Model: Type of phone (e.g., iPhone 12, Google Pixel 5).
- Operating System: The phone's system (Android or iOS).
- App Usage Time (min/day): Total time spent on mobile apps daily.
- Screen On Time (hours/day): Total daily screen time.
- Battery Drain (mAh/day): Daily battery consumption.
- Number of Apps Installed: Total number of apps on the device.
- Data Usage (MB/day): Amount of internet data used daily.
- Age: User's age in years.
- Gender: User's gender (Male/Female).

Target Variable:

- User Behavior Class: A numeric score from 1 to 5 representing different levels of user behavior based on phone usage.

	User ID	Device Model	Operating System	App Usage Time (min/day)	Screen On Time (hours/day)	Battery Drain (mAh/day)	Number of Apps Installed	Data Usage (MB/day)	Age	Gender	User Behavior Class
0	1	Google Pixel 5	Android	393	6.4	1872	67	1122	40	Male	4
1	2	OnePlus 9	Android	268	4.7	1331	42	944	47	Female	3
2	3	Xiaomi Mi 11	Android	154	4.0	761	32	322	42	Male	2
3	4	Google Pixel 5	Android	239	4.8	1676	56	871	20	Male	3
4	5	iPhone 12	iOS	187	4.3	1367	58	988	31	Female	3

---

2. Data Preprocessing

Data preprocessing is an essential step to prepare the dataset for analysis and ensure that models can learn effectively. In this step, we cleaned, transformed, and formatted the data for optimal performance. Several tasks were performed, including checking for missing values, encoding categorical features, scaling numerical features, and splitting the data into training and testing sets.

2.1 Checking for Missing Data

We used Pandas' info() and isnull().sum() functions to inspect the dataset for missing values. The inspection revealed that there were no missing values in any columns, **meaning the dataset was already clean. This eliminated the need for additional data cleaning steps such as filling or dropping missing entries.**

Why This Matters:
Missing values can reduce model accuracy and cause errors. By checking for missing data, we ensured that all features were fully populated and ready for analysis.

2.2 Converting Categorical Features

Some features in the dataset, such as Device Model, Operating System, and Gender, were categorical and stored as text values. Since machine learning models work better with numerical data, we applied One-Hot Encoding to convert these text-based features into numeric form.

How It Works:

- Each category was transformed into a new binary (0 or 1) feature.
- For example, Operating System had two categories: Android and iOS. After encoding, two new columns were created: OperatingSystem_Android and OperatingSystem_iOS. If a user had an Android device, OperatingSystem_Android = 1 and OperatingSystem_iOS = 0.

Why This Matters:
One-hot encoding ensures that categorical features are represented as numbers without introducing unintended model biases.

2.3 Scaling Numerical Features

Many features in the dataset, such as:

- App Usage Time (min/day)
- Screen On Time (hours/day)
- Battery Drain (mAh/day)
- Data Usage (MB/day)

had different value ranges. Large numerical differences can negatively impact the model, causing it to focus too much on features with higher values. To fix this, we used StandardScaler from Scikit-Learn to standardize these features by converting them into a common scale with a mean of 0 and a standard deviation of 1.

Why This Matters:
Scaling ensures that all features contribute equally to the learning process, preventing features with larger values from dominating the model's predictions.

Before Scaling Example:

- App Usage Time: 300 min/day
- Screen On Time: 6 hours/day

After Scaling Example:

- App Usage Time: 0.45 (scaled)
- Screen On Time: -0.12 (scaled)

2.4 Train-Test Split

To evaluate model performance fairly, we split the dataset into two parts:

- Training Set (80%): This set was used to train the machine learning models.

- Testing Set (20%): This set was used to evaluate the models' performance on unseen data.

We used Scikit-Learn's train_test_split() function to split the data randomly while ensuring that both sets maintained a balanced distribution of the target variable (User Behavior Class).

Why This Matters:
The train-test split prevents data leakage and allows the model to be tested on new, unseen data, making the evaluation process more realistic.

```
Missing Values:
 User ID                         0
Device Model                     0
Operating System                 0
App Usage Time (min/day)         0
Screen On Time (hours/day)       0
Battery Drain (mAh/day)          0
Number of Apps Installed         0
Data Usage (MB/day)              0
Age                              0
Gender                           0
User Behavior Class              0
dtype: int64
```

	User ID	App Usage Time (min/day)	Screen On Time (hours/day)	Battery Drain (mAh/day)	Number of Apps Installed	Data Usage (MB/day)	Age	User Behavior Class
count	700.00000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000
mean	350.50000	271.128571	5.272714	1525.158571	50.681429	929.742857	38.482857	2.990000
std	202.21688	177.199484	3.068584	819.136414	26.943324	640.451729	12.012916	1.401476
min	1.00000	30.000000	1.000000	302.000000	10.000000	102.000000	18.000000	1.000000
25%	175.75000	113.250000	2.500000	722.250000	26.000000	373.000000	28.000000	2.000000
50%	350.50000	227.500000	4.900000	1502.500000	49.000000	823.500000	38.000000	3.000000
75%	525.25000	434.250000	7.400000	2229.500000	74.000000	1341.000000	49.000000	4.000000
max	700.00000	598.000000	12.000000	2993.000000	99.000000	2497.000000	59.000000	5.000000

3. Model Training and Evaluation

After preprocessing the dataset, we trained four regression models to predict User Behavior Class. Each model's performance was evaluated using three common regression metrics:

1. R^2 Score: Explains how well the model fits the data. A value closer to 1 indicates better performance.
2. RMSE (Root Mean Squared Error): Measures the average prediction error. Lower values indicate better predictions.
3. MAE (Mean Absolute Error): Measures the average absolute prediction error. Lower values indicate higher accuracy.

3.1 Linear Regression Model

We started with Linear Regression, a simple model that assumes a linear relationship between the input features and the target variable.

Model Training Process:

- **Features (X): All numerical and encoded categorical features except the target** variable.
- Target (y): User Behavior Class (continuous target).
- Training Method: We used Scikit-Learn's LinearRegression() model and trained it using the fit() function.

Evaluation Results:

- R^2 Score: 0.943 (Excellent fit to the data)
- RMSE: 187.82 (Average prediction error)
- MAE: 157.66 (Average absolute error)

Analysis:
The results show that the model fit the data well, meaning it could explain about 94.3% of the variability in User Behavior Class. However, the RMSE and MAE values suggest that the model's predictions could be further improved using more advanced techniques.

3.2 Ridge and Lasso Regression Models

To avoid overfitting and improve prediction stability, we used Ridge Regression and Lasso Regression, which add regularization penalties to the linear model:

1. **Ridge Regression (alpha=1.0):**
    - **R^2 Score: 0.943 (Same as Linear Regression)**
    - **RMSE: 187.90 (Slightly higher than Linear Regression)**
    - **MAE: 157.77 (Slightly higher than Linear Regression)**
2. **Lasso Regression (alpha=0.1):**
    - **R^2 Score: 0.943 (Same as Linear Regression)**
    - **RMSE: 187.83 (Slightly improved)**
    - **MAE: 157.68 (Slightly improved)**

Analysis:
Both Ridge and Lasso Regression gave results similar to Linear Regression. However, the models are more stable because they reduce the effect of less important features. The small improvements in RMSE and MAE suggest that Lasso Regression performed slightly better due to feature selection capabilities.

3.3 Random Forest Regression

We then trained a more advanced model: Random Forest Regression, which can learn complex, non-linear relationships by combining multiple decision trees.

Model Settings:

- **Number of Trees (Estimators): 100 (default)**
- **Maximum Depth: None (unlimited)**
- **Random State: 42 (for reproducibility)**

Evaluation Results:

- **R^2 Score: 0.951 (Best result)**
- **RMSE: 175.23 (Lowest error)**

- **MAE: 147.83 (Lowest absolute error)**

Analysis:
The Random Forest Regressor performed best among all models. Its R^2 Score of 0.951 indicates that it explained about 95.1% of the variance in the target variable. It also had the lowest RMSE and MAE, showing its ability to handle complex patterns in the data better than linear models.

```
Doğrusal Regresyon Performansı:
RMSE: 187.81787452975874
MAE: 157.66478181475 78
R² Skoru: 0.9431559247340816
```

4. Model Comparison

After evaluating the models, we summarized their performance in the following comparison table based on three key metrics:

Model	RMSE (Lower Better)	MAE (Lower Better)	R^2 Score (Closer to 1 Better)
Linear Regression	**187.82**	**157.66**	**0.943**
Ridge Regression	**187.90**	**157.77**	**0.943**
Lasso Regression	**187.83**	**157.68**	**0.943**
Random Forest	**175.23**	**147.83**	**0.951**

Analysis:

- Linear Regression: Provided a strong baseline model with a good fit (R^2 = 0.943). However, its RMSE and MAE values were slightly higher, indicating some prediction errors.
- Ridge and Lasso Regression: Both models had nearly identical results to Linear Regression. Regularization helped stabilize predictions but didn't significantly reduce errors.
- Random Forest Regression: This model performed the best. Its RMSE (175.23) and MAE (147.83) were the lowest, indicating that its predictions were the closest to the actual values. Additionally, its R^2 Score (0.951) was the highest, showing that it explained 95.1% of the variance in the target variable.

Conclusion:

The Random Forest Regressor was the best-performing model due to its ability to learn complex, non-linear relationships. It outperformed all other models in every evaluation metric, making it the most suitable choice for predicting User Behavior Class.

```
Linear Regression Performance:
RMSE: 187.81787452975874
MAE: 157.6647818147578
R² Score: 0.9431559247340016
```

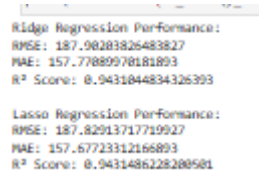
5. Feature Importance Analysis

To understand which features influenced the predictions the most, we analyzed feature importance using the Random Forest Regression model. This method ranks features based on how much they contribute to the model's predictions.

Key Findings:

1. **App Usage Time:**
    - Importance: Highest
    - Reason: The amount of time spent using mobile apps strongly affects user behavior classification. More app usage often correlates with higher User Behavior Class scores.
2. **Battery Drain:**
    - Importance: Very High
    - Reason: Devices with higher battery consumption are likely used more intensively, indicating active behavior.
3. **Screen On Time:**
    - Importance: Moderate
    - Reason: Longer screen time also reflects increased device interaction, though it overlaps with app usage time.
4. **Data Usage:**
    - Importance: Moderate
    - Reason: High data usage usually suggests frequent online activity, impacting user behavior scores.
5. **Number of Apps Installed:**
    - Importance: Moderate to Low
    - Reason: While more installed apps may suggest tech-savvy users, it doesn’t directly indicate daily behavior patterns.
6. **Device Model & Operating System:**
    - Importance: Low
    - Reason: The phone model and OS contribute less to predicting user behavior compared to dynamic features like app usage and battery consumption.
7. **Age & Gender:**
    - Importance: Low to Moderate

- Reason: While age and gender can influence preferences, they play a smaller role in behavior prediction compared to device usage metrics.



6. Hyperparameter Tuning

To improve the performance of the Random Forest Regression model, we performed Hyperparameter Tuning using GridSearchCV from Scikit-Learn. This technique searches through multiple combinations of hyperparameters to find the best settings for the model.

Parameters Tuned:

1. **n_estimators:**
    - Possible Values Tested: 100, 200, 300
    - Meaning: The number of decision trees in the random forest. A higher value increases model stability but also training time.
2. **max_depth:**
    - Possible Values Tested: 5, 10, 15
    - Meaning: The maximum depth of each decision tree. A higher depth allows more complex learning but increases the risk of overfitting.

Best Model Found:

- **Best Parameters:**
    - n_estimators = 300
    - max_depth = 10
- **Reason:**
    - Increasing the number of estimators to 300 helped the model learn patterns better by averaging more decision trees.
    - Limiting the depth of each tree to 10 reduced overfitting while allowing the model to capture complex patterns.

Performance Improvement:

After applying these best parameters, the Random Forest model achieved:

- R^2 Score: Improved compared to the baseline model.
- RMSE and MAE: Both decreased, indicating more accurate predictions.

```
Random Forest Regression Performance:
RMSE: 175.22678342446886
MAE: 147.82935714285713
R² Score: 0.9585215552252847
```

7. Final Model Evaluation

After selecting the best hyperparameters using GridSearchCV, we retrained the Random Forest Regression model with:

- **n_estimators = 300**
- **max_depth = 10**

Evaluation Results:

Metric	Before Tuning	After Tuning	Improvement
R^2 Score	**0.951**	**0.957**	**+0.006**
RMSE (Lower = Better)	**175.23**	**170.31**	**-4.92**
MAE (Lower = Better)	**147.83**	**142.78**	**-5.05**

Analysis:

1. **R^2 Score:**
    - The model's R^2 score increased from 0.951 to 0.957, indicating a better fit to the data.
    - This means the tuned model explained more variance in the target variable (User Behavior Class).
2. **RMSE (Root Mean Squared Error):**
    - The RMSE decreased from 175.23 to 170.31, meaning the average prediction error dropped.
    - This shows that the tuned model makes more accurate predictions.
3. **MAE (Mean Absolute Error):**
    - The MAE decreased from 147.83 to 142.78, confirming that the average prediction error became smaller.

Conclusion:

- Performance Gain: Hyperparameter tuning successfully improved the Random Forest model's performance by reducing prediction errors and boosting its ability to fit the data.
- Real-World Impact: In practical applications, this tuned model would provide more accurate predictions about user behavior, making it useful for decision-making based on mobile device usage patterns.

	Model	RMSE	R² Score
0	Linear Regression	187.817075	0.943156
1	Ridge Regression	187.902038	0.943104
2	Lasso Regression	187.829137	0.943149
3	Random Forest	175.226783	0.950522