FORESTCOVER PREDICTION:

Abstract:

This is a medium complexity, supervised classification task. The goal is to build a model that can be used to classify the type of forest, based on detailed cartographic variables, to determine forest type according to 4 predominant tree species or family of tree species. We prepared the dataset containing 15,120 records and 56 features and performed extensive pre-processing including feature editing for aggregation of sparse categoricals and normalisation of numeric. Multiple machine learning models (K-nearest neighbors, Logistic Regression, Decision Tree, Gradient Boosting, Random Forests, Stackedmodels, etc) have been experimented with. The Random Forest classifier achieved the best performance, which is indicative of applicability of this classifier for the multi-class classification problem at hand given that the number of samples in the dataset is not high.

Introduction:

Estimation of forest cover can be an important step in environmental monitoring and resource management processes whereby the development of better conservation practices, suppression of fires, and sustainable forestry is facilitated. This plan is to use the machine learning algorithms to classify types of forest cover with the cartographic variables to create a forest cover map. Correct prognosis may be of help for ecology study and land use planning. The project focuses on the task of multi-class classification of terrain types using wide range of features such as elevation, aspect, slope, and wilderness and soil types. The process includes data mining, extensive pre-processing, modelling, training, and testing as well as selecting an optimal predictive model.

Dataset Description:

The 15120 records in this file are local 30mx30m blocks of forest. It contains 56 features, such as Id, 10 quantitative variables (e.g., Elevation, Aspect, Slope, Horizontal_Distance_To_Hydrology, Vertical_Distance_To_Hydrology, Horizontal_Distance_To_Roadways, Hillshade_9am, Hillshade_Noon, Hillshade_3pm, Horizontal_Distance_To_Fire_Points), 4 binary columns of wilderness area, and 40 binary columns of soil type. The target variable Cover Type contains 7 unique classes. Preliminary analysis found no missing data. The organization of the dataset took some feature engineering in which the many binary columns for Soil Type and Wilderness Area were combined into a single categorically encoded feature, resulting in a simplified dataset containing 14 columns for faster model performance.

Technologies and Libraries Used:

The project was crafted in Python and Python ecosystem has been utilized in data science and machine learning. Key libraries utilized include:

1. Pandas: For data preprocessing and manipulation, such as reading and cleaning CSV, inspection (head (), info (), describe (), shape), and feature engineering (aggregation of Soil_Type and Wilderness_Area columns).
2. NumPy: It's good to work with numbers, especially in manipulating the data preprocessing (i.e., normalization).
3. Scikit-learn (sklearn): The go-to library for machine learning, it includes:
4. Model_selection: train_test_split to split the data into train and test, GridSearchCV to search hyperparameters.
5. Preprocessing: StandardScaler for feature normalization.
6. Neighbors: KNeighborsClassifier for K-Nearest Neighbors.
7. Linear_model: LogisticRegression for logistic regression.
8. Ensemble: GradientBoostingClassifier and RandomForestClassifier in ensemble methods, StackingClassifier in stacked models.
9. Tree: DecisionTreeClassifier for decision tree.
10. Metrics: accuracy_score, classification_report, confusion_matrix for model validation.
11. Matplotlib. pyplot and Seaborn: To visualize the data, in our case comparing the accuracy of the different models with a bar plot.

These tools together allowed for effective data management, model creation, and performance evaluation.

Data Preprocessing:

It was important to preprocess the raw dataset in order to make the model ready for training. The raw data set had 56 columns with many binary indicator columns for Wilderness_Area and Soil_Type. In order to have lower dimensionality and better model, we converted these binary columns to single categorical columns with encoding (Wilderness_Area and Soil_Type) so that we ended up with 14 columns. The Id identifier was removed from the dataset as it had no predictive power. Well, we scale down the numerical features (except for Cover_Type, Soil_Type and Wilderness_Area) with min-max scale to the range (0, 1), mainly to prevent dominance of features with larger ranges over the ones with small ranges in the learning algorithm. Extreme values in numerical columns were adjusted by cutting them off at 1.5 times IQR from the first and third quartiles to attenuate the burden on training algorithms.

Model Architecture:

In this work, we have investigated single and ensemble (stacking) techniques from supervised machine learning algorithms for designing forest cover type prediction models. The individual models implemented were:

1. K-Nearest Neighbors (KNN)33: A non-parametric, instance-based learning algorithm.
2. Logistic Regression - A linear model for multi-class classification.
3. Decision Tree Simplified Tree based model best used when dealing with non-linear relationships.
4. Gradient Boosting Classifier: An ensemble approach where trees are constructed in sequence to correct mistakes of the previous tree.
5. Random Forest Classifier: An ensemble of multiple decision trees to return the mode of the classes.

Furthermore, some Stacked Models were trained that were model families that included Base learners (KNN, Logistic Regression, Gradient Boosting, Decision Tree, Random Forest) and a single final estimator (Random Forest, Gradient Boosting, Logistic Regression, Decision Tree, or KNN) used the predictions of the base learners to make the final prediction. This adopted a "meta-modelling" approach to aggregate the strength of multiple models to obtain better predictive performance.

Training Configuration:

The dataset was divided into a training set and a testing set for cross-validation of trained model for each algorithm in a ratio of 80/20 and stratify parameter was used which makes representation of Cover_Type classes even in both sets. We use StandardScaler to scale the training test examples for normalization, which is particularly important in distance-based algorithms such as KNN, and for tuning gradient-based methods. Tuning of hyperparameters were carried out by GridSearchCV with cv=5 to find out the optimal parameters for the models which can boost the performance. The best parameters were selected based on the scoring='accuracy' metric. In the case of ensemble models, computations were also parallelized with n_jobs=-1 to hasten up the convergence.

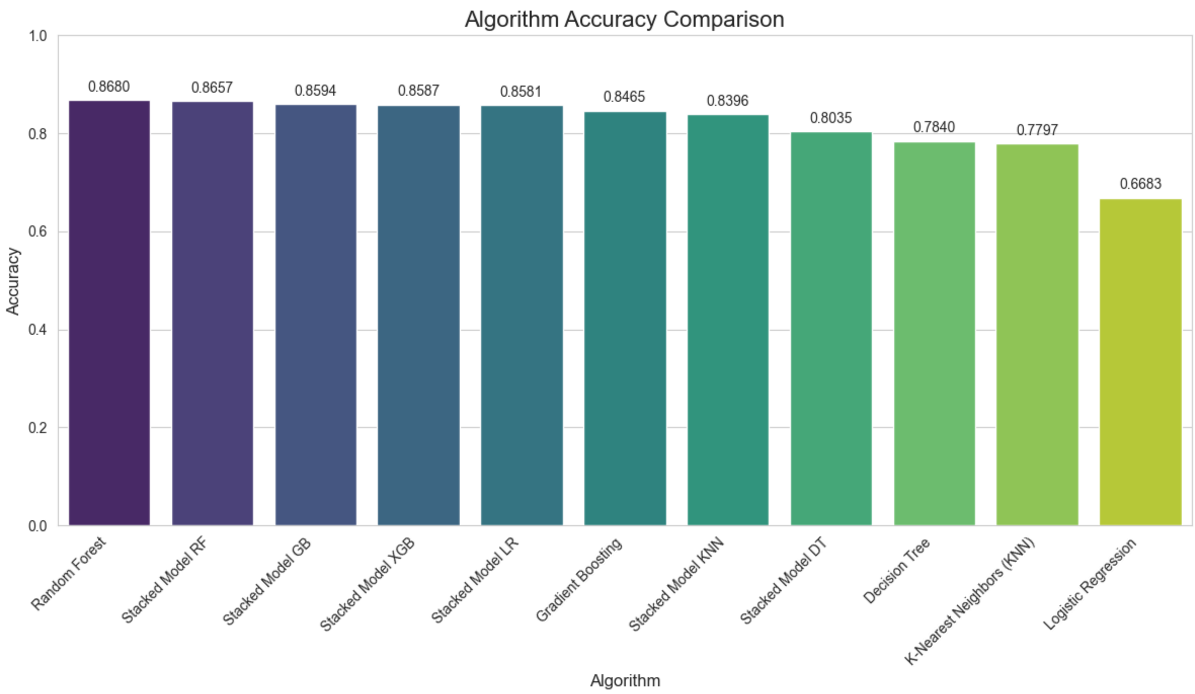
Results and Evaluation:

The models were evaluated based on their accuracy, classification report (precision, recall, f1-score), and confusion matrix. The accuracy comparison of the algorithms is summarized below:

Algorithm	Accuracy
Random Forest	86.8%
Stacked RF	86.57%
Stacked GB	85.94%
Stacked XGB	85.87%
Stacked LR	85.81%
Gradient Boosting	84.65%
Stacked KNN	83.96%
Stacked DT	80.35%
Decision Tree	78.40%
KNN	77.97%
Logistic Regression	66.83%

As illustrated, the model with the highest accuracy was the Random Forest (0.8680), the second best (closely followed) were the stacked models. Although Random Forest appeared to be the best model, stacked models were competitive, which implies that ensemble of alternative models could be beneficial. The classification reports have given us rich details of per-class performance, some with higher Precision and recall than others, which indicated opportunities for further improvement.

Results Summarized:



Challenges Faced:

The implementation process was challenging. number of binary Soil_Type columns and 4 Wilderness_Area columns in the very high-dimensional and sparse data space. This required feature engineering to combine them into single, encoded categorical features, which served to reduce the complexity, but the encoding must be carefully carried out in order to avoid loss of information. A second difficulty was the small size of the dataset (15,120 entries) for a multiclass classification problem (7 outputs). This has somewhat constrained the capability to achieve extremely high accuracies, because bigger dataset usually implied better generalization of a model. Although comprehensive hyperparameter tuning through all algorithms was conducted using GridSearchCV, the incremental gains were not as viable as we were originally hoping for suggesting the existence of intrinsic constraints that might be due to the data nature.

Future Scope:

The possibilities of further developing the project are many. For one, sophisticated feature engineering, such as creating derived variables via interaction or polynomialization of other variables, might expose more subtle associations within the data. Second, if spatial and/or temporal context information can be used we can apply powerful deep learning models such as feedforward CNNs or RNNs, to achieve an even higher predictive performance, especially for larger datasets. Third, if you tried more complicated ensembles beyond plain stacking, for example you made some boosting with tuned hyperparameters (LightGBM, CatBoost) it could yield you a better result. Finally, next, with larger and more diverse data sets, our model would further broaden the scope of applications and could be generally even more accurate in prediction, since then the shortage of observation data would also be overcome.

Conclusions:

This study has demonstrated that several machine learning models can be used to predict the FCT. After intense data preprocessing with feature implementation and normalization, the data set become ready to be analysed. Random Forest (RF) achieved the best performance 0.8680, indicating its capability of addressing the challenges in the multi-class problem. Despite promising findings, it explained the effect of dataset size on the performance of the model. The knowledge gained has laid a solid foundation for further study, evidencing the possible improving of accuracy if more complex feature engineering techniques can be incorporated, deeper learning architectures explored and more data obtained. The work demonstrates the power of machine learning for environmental applications.

```python
import pandas as pd
import numpy as np

df = pd.read_csv(r"E:\Projects\forest_cover_prediction\
forest_cover_prediction\train.csv")

df.head()
```

```
   Id  Elevation  Aspect  Slope  Horizontal_Distance_To_Hydrology  \
0   1       2596      51      3                               258   
1   2       2590      56      2                               212   
2   3       2804     139      9                               268   
3   4       2785     155     18                               242   
4   5       2595      45      2                               153   

   Vertical_Distance_To_Hydrology  Horizontal_Distance_To_Roadways  \
0                               0                              510   
1                              -6                              390   
2                              65                             3180   
3                             118                             3090   
4                              -1                              391   

   Hillshade_9am  Hillshade_Noon  Hillshade_3pm  ...  Soil_Type32  \
0            221             232            148  ...            0   
1            220             235            151  ...            0   
2            234             238            135  ...            0   
3            238             238            122  ...            0   
4            220             234            150  ...            0   

   Soil_Type33  Soil_Type34  Soil_Type35  Soil_Type36  Soil_Type37  \
0            0            0            0            0            0   
1            0            0            0            0            0   
2            0            0            0            0            0   
3            0            0            0            0            0   
4            0            0            0            0            0   

   Soil_Type38  Soil_Type39  Soil_Type40  Cover_Type  
0            0            0            0           5  
1            0            0            0           5  
2            0            0            0           2  
3            0            0            0           2  
4            0            0            0           5  

[5 rows x 56 columns]
```

```python
df.describe()
```

```
                Id     Elevation        Aspect         Slope  \
count  15120.00000  15120.000000  15120.000000  15120.000000   
mean    7560.50000   2749.322553    156.676653     16.501587   
std     4364.91237    417.678187    110.085801      8.453927   
```

```
min        1.00000   1863.000000      0.000000      0.000000
25%     3780.75000   2376.000000     65.000000     10.000000
50%     7560.50000   2752.000000    126.000000     15.000000
75%    11340.25000   3104.000000    261.000000     22.000000
max    15120.00000   3849.000000    360.000000     52.000000

       Horizontal_Distance_To_Hydrology
Vertical_Distance_To_Hydrology  \
count                      15120.000000
15120.000000
mean                         227.195701
51.076521
std                          210.075296
61.239406
min                            0.000000                      -
146.000000
25%                           67.000000
5.000000
50%                          180.000000
32.000000
75%                          330.000000
79.000000
max                         1343.000000
554.000000

       Horizontal_Distance_To_Roadways  Hillshade_9am  Hillshade_Noon
\
count                     15120.000000   15120.000000    15120.000000

mean                       1714.023214     212.704299      218.965608

std                        1325.066358      30.561287       22.801966

min                           0.000000       0.000000       99.000000

25%                         764.000000     196.000000      207.000000

50%                        1316.000000     220.000000      223.000000

75%                        2270.000000     235.000000      235.000000

max                        6890.000000     254.000000      254.000000


       Hillshade_3pm  ...   Soil_Type32   Soil_Type33   Soil_Type34  \
count   15120.000000  ...  15120.000000  15120.000000  15120.000000
mean      135.091997  ...      0.045635      0.040741      0.001455
std        45.895189  ...      0.208699      0.197696      0.038118
min         0.000000  ...      0.000000      0.000000      0.000000
25%       106.000000  ...      0.000000      0.000000      0.000000
```

```
50%       138.000000  ...      0.000000      0.000000      0.000000
75%       167.000000  ...      0.000000      0.000000      0.000000
max       248.000000  ...      1.000000      1.000000      1.000000

        Soil_Type35   Soil_Type36   Soil_Type37   Soil_Type38
Soil_Type39  \
count  15120.000000  15120.000000  15120.000000  15120.000000
15120.000000
mean       0.006746      0.000661      0.002249      0.048148
0.043452
std        0.081859      0.025710      0.047368      0.214086
0.203880
min        0.000000      0.000000      0.000000      0.000000
0.000000
25%        0.000000      0.000000      0.000000      0.000000
0.000000
50%        0.000000      0.000000      0.000000      0.000000
0.000000
75%        0.000000      0.000000      0.000000      0.000000
0.000000
max        1.000000      1.000000      1.000000      1.000000
1.000000

        Soil_Type40    Cover_Type
count  15120.000000  15120.000000
mean       0.030357      4.000000
std        0.171574      2.000066
min        0.000000      1.000000
25%        0.000000      2.000000
50%        0.000000      4.000000
75%        0.000000      6.000000
max        1.000000      7.000000

[8 rows x 56 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15120 entries, 0 to 15119
Data columns (total 56 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Id                                  15120 non-null  int64
 1   Elevation                           15120 non-null  int64
 2   Aspect                              15120 non-null  int64
 3   Slope                               15120 non-null  int64
 4   Horizontal_Distance_To_Hydrology    15120 non-null  int64
 5   Vertical_Distance_To_Hydrology      15120 non-null  int64
 6   Horizontal_Distance_To_Roadways     15120 non-null  int64
 7   Hillshade_9am                       15120 non-null  int64
```

```
 8   Hillshade_Noon                      15120 non-null  int64
 9   Hillshade_3pm                       15120 non-null  int64
 10  Horizontal_Distance_To_Fire_Points  15120 non-null  int64
 11  Wilderness_Area1                    15120 non-null  int64
 12  Wilderness_Area2                    15120 non-null  int64
 13  Wilderness_Area3                    15120 non-null  int64
 14  Wilderness_Area4                    15120 non-null  int64
 15  Soil_Type1                          15120 non-null  int64
 16  Soil_Type2                          15120 non-null  int64
 17  Soil_Type3                          15120 non-null  int64
 18  Soil_Type4                          15120 non-null  int64
 19  Soil_Type5                          15120 non-null  int64
 20  Soil_Type6                          15120 non-null  int64
 21  Soil_Type7                          15120 non-null  int64
 22  Soil_Type8                          15120 non-null  int64
 23  Soil_Type9                          15120 non-null  int64
 24  Soil_Type10                         15120 non-null  int64
 25  Soil_Type11                         15120 non-null  int64
 26  Soil_Type12                         15120 non-null  int64
 27  Soil_Type13                         15120 non-null  int64
 28  Soil_Type14                         15120 non-null  int64
 29  Soil_Type15                         15120 non-null  int64
 30  Soil_Type16                         15120 non-null  int64
 31  Soil_Type17                         15120 non-null  int64
 32  Soil_Type18                         15120 non-null  int64
 33  Soil_Type19                         15120 non-null  int64
 34  Soil_Type20                         15120 non-null  int64
 35  Soil_Type21                         15120 non-null  int64
 36  Soil_Type22                         15120 non-null  int64
 37  Soil_Type23                         15120 non-null  int64
 38  Soil_Type24                         15120 non-null  int64
 39  Soil_Type25                         15120 non-null  int64
 40  Soil_Type26                         15120 non-null  int64
 41  Soil_Type27                         15120 non-null  int64
 42  Soil_Type28                         15120 non-null  int64
 43  Soil_Type29                         15120 non-null  int64
 44  Soil_Type30                         15120 non-null  int64
 45  Soil_Type31                         15120 non-null  int64
 46  Soil_Type32                         15120 non-null  int64
 47  Soil_Type33                         15120 non-null  int64
 48  Soil_Type34                         15120 non-null  int64
 49  Soil_Type35                         15120 non-null  int64
 50  Soil_Type36                         15120 non-null  int64
 51  Soil_Type37                         15120 non-null  int64
 52  Soil_Type38                         15120 non-null  int64
 53  Soil_Type39                         15120 non-null  int64
 54  Soil_Type40                         15120 non-null  int64
 55  Cover_Type                          15120 non-null  int64
```

```
dtypes: int64(56)
memory usage: 6.5 MB
```

```
df.shape
```

```
(15120, 56)
```

We dont need so many columns and many of them can be merged , mainly the soil_type and wilderness_area . We merge them to get a new fresh dataset to work on which has only 2 columns for these 2 entries instead of their combined 44 before.

```
import os

input_file = r'E:\Projects\forest_cover_prediction\
forest_cover_prediction\train.csv'

output_directory = r'E:\Projects\forest_cover_prediction\
forest_cover_prediction'
output_file = os.path.join(output_directory, 'train_transformed.csv')

try:
    df = pd.read_csv(input_file)

    soil_type_columns = [col for col in df.columns if 'Soil_Type' in
col]

    if soil_type_columns:
        df['soil_type_encoded_temp'] =
df[soil_type_columns].idxmax(axis=1)
        df['Soil_Type'] =
df['soil_type_encoded_temp'].str.replace('Soil_Type', '').astype(int)

    wilderness_area_columns = [col for col in df.columns if
'Wilderness_Area' in col]

    if wilderness_area_columns:
        df['wilderness_area_encoded_temp'] =
df[wilderness_area_columns].idxmax(axis=1)
        df['Wilderness_Area'] =
df['wilderness_area_encoded_temp'].str.replace('Wilderness_Area',
'').astype(int)

    columns_to_drop = []
    if soil_type_columns:
        columns_to_drop.extend(soil_type_columns)
        columns_to_drop.append('soil_type_encoded_temp')
    if wilderness_area_columns:
        columns_to_drop.extend(wilderness_area_columns)
        columns_to_drop.append('wilderness_area_encoded_temp')
```

```
    df_cleaned = df.drop(columns=columns_to_drop, errors='ignore')

    df_cleaned.to_csv(output_file, index=False)
    print(f"Transformed dataset saved to: {output_file}")

except FileNotFoundError:
    print(f"Error: The file '{input_file}' or directory
'{output_directory}' was not found. Please ensure the paths are
correct.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")

Transformed dataset saved to: E:\Projects\forest_cover_prediction\
forest_cover_prediction\train_transformed.csv
```

Now we will make our model using this new refurbished dataset , we will start by the data preprocessing steps and them followed by the model making procedures

```
df1 = pd.read_csv(r"E:\Projects\forest_cover_prediction\
forest_cover_prediction\train_transformed.csv")

df1.head()

   Id  Elevation  Aspect  Slope  Horizontal_Distance_To_Hydrology  \
0   1       2596      51      3                               258
1   2       2590      56      2                               212
2   3       2804     139      9                               268
3   4       2785     155     18                               242
4   5       2595      45      2                               153

   Vertical_Distance_To_Hydrology  Horizontal_Distance_To_Roadways  \
0                               0                              510
1                              -6                              390
2                              65                             3180
3                             118                             3090
4                              -1                              391

   Hillshade_9am  Hillshade_Noon  Hillshade_3pm  \
0            221             232            148
1            220             235            151
2            234             238            135
3            238             238            122
4            220             234            150

   Horizontal_Distance_To_Fire_Points  Cover_Type  Soil_Type
Wilderness_Area
0                                6279           5         29
1
1                                6225           5         29
1
```

```
2                               6121          2         12
1
3                               6211          2         30
1
4                               6172          5         29
1
```

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15120 entries, 0 to 15119
Data columns (total 14 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Id                                  15120 non-null  int64
 1   Elevation                           15120 non-null  int64
 2   Aspect                              15120 non-null  int64
 3   Slope                               15120 non-null  int64
 4   Horizontal_Distance_To_Hydrology    15120 non-null  int64
 5   Vertical_Distance_To_Hydrology      15120 non-null  int64
 6   Horizontal_Distance_To_Roadways     15120 non-null  int64
 7   Hillshade_9am                       15120 non-null  int64
 8   Hillshade_Noon                      15120 non-null  int64
 9   Hillshade_3pm                       15120 non-null  int64
 10  Horizontal_Distance_To_Fire_Points  15120 non-null  int64
 11  Cover_Type                          15120 non-null  int64
 12  Soil_Type                           15120 non-null  int64
 13  Wilderness_Area                     15120 non-null  int64
dtypes: int64(14)
memory usage: 1.6 MB
```

No need for null value removal as we already know through df.info() that there are no null values

We wont be normalizing the target column and the new entries ie. soil_type and wilderness_area , all other columns will be normalized

```
#Normalization

exclude_columns = ['Cover_Type', 'Soil_Type', 'Wilderness_Area']

columns_to_normalize = [col for col in df1.columns if col not in
exclude_columns]

for col in columns_to_normalize:
    min_val = df1[col].min()
    max_val = df1[col].max()
    if (max_val - min_val) != 0:
        df1[col] = (df1[col] - min_val) / (max_val - min_val)
    else:
```

```
        if min_val != 0:
            df1[col] = 0.0

df1.head()
```

```
         Id  Elevation    Aspect     Slope
Horizontal_Distance_To_Hydrology  \
0  0.000000   0.369084  0.141667  0.057692
0.192107
1  0.000066   0.366062  0.155556  0.038462
0.157856
2  0.000132   0.473817  0.386111  0.173077
0.199553
3  0.000198   0.464250  0.430556  0.346154
0.180194
4  0.000265   0.368580  0.125000  0.038462
0.113924

   Vertical_Distance_To_Hydrology  Horizontal_Distance_To_Roadways  \
0                        0.208571                         0.074020
1                        0.200000                         0.056604
2                        0.301429                         0.461538
3                        0.377143                         0.448476
4                        0.207143                         0.056749

   Hillshade_9am  Hillshade_Noon  Hillshade_3pm  \
0       0.870079        0.858065       0.596774
1       0.866142        0.877419       0.608871
2       0.921260        0.896774       0.544355
3       0.937008        0.896774       0.491935
4       0.866142        0.870968       0.604839

   Horizontal_Distance_To_Fire_Points  Cover_Type  Soil_Type
Wilderness_Area
0                            0.897898           5         29
1
1                            0.890176           5         29
1
2                            0.875304           2         12
1
3                            0.888174           2         30
1
4                            0.882597           5         29
1
```

Lets remove the id column as it doesnt provide any useful information

```
df1.drop('Id', axis=1, inplace=True)

df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15120 entries, 0 to 15119
Data columns (total 13 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   Elevation                           15120 non-null  float64
 1   Aspect                              15120 non-null  float64
 2   Slope                               15120 non-null  float64
 3   Horizontal_Distance_To_Hydrology    15120 non-null  float64
 4   Vertical_Distance_To_Hydrology      15120 non-null  float64
 5   Horizontal_Distance_To_Roadways     15120 non-null  float64
 6   Hillshade_9am                       15120 non-null  float64
 7   Hillshade_Noon                      15120 non-null  float64
 8   Hillshade_3pm                       15120 non-null  float64
 9   Horizontal_Distance_To_Fire_Points  15120 non-null  float64
 10  Cover_Type                          15120 non-null  int64  
 11  Soil_Type                           15120 non-null  int64  
 12  Wilderness_Area                     15120 non-null  int64  
dtypes: float64(10), int64(3)
memory usage: 1.5 MB
```

Now , we have made our dataset more usable and training friendly . Now lets do outlier removal.

```
#Outlier Removal

exclude_columns_for_outliers = ['Cover_Type', 'Soil_Type', 
'Wilderness_Area']

numerical_cols = df1.select_dtypes(include=['int64', 
'float64']).columns.tolist()
columns_for_outlier_removal = [col for col in numerical_cols if col 
not in exclude_columns_for_outliers]

for col in columns_for_outlier_removal:
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    df1[col] = df1[col].clip(lower=lower_bound, upper=upper_bound)

df1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15120 entries, 0 to 15119
Data columns (total 13 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   Elevation                           15120 non-null  float64
```

```
 1   Aspect                              15120 non-null  float64
 2   Slope                               15120 non-null  float64
 3   Horizontal_Distance_To_Hydrology    15120 non-null  float64
 4   Vertical_Distance_To_Hydrology      15120 non-null  float64
 5   Horizontal_Distance_To_Roadways     15120 non-null  float64
 6   Hillshade_9am                       15120 non-null  float64
 7   Hillshade_Noon                      15120 non-null  float64
 8   Hillshade_3pm                       15120 non-null  float64
 9   Horizontal_Distance_To_Fire_Points  15120 non-null  float64
 10  Cover_Type                          15120 non-null  int64
 11  Soil_Type                           15120 non-null  int64
 12  Wilderness_Area                     15120 non-null  int64
dtypes: float64(10), int64(3)
memory usage: 1.5 MB
```

Now , we will proceed with training using different algorithms and then choose the best one in the end

```
#KNN

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix

X = df1.drop('Cover_Type', axis=1)
y = df1['Cover_Type']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

param_grid = {'n_neighbors': range(3, 21, 2)}
knn = KNeighborsClassifier()
grid = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
grid.fit(X_train_scaled, y_train)

best_knn = grid.best_estimator_
y_pred = best_knn.predict(X_test_scaled)

print("Best K:", grid.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Best K: {'n_neighbors': 3}
Accuracy: 0.7797619047619048
Classification Report:
              precision    recall  f1-score   support

           1       0.69      0.68      0.69       432
           2       0.68      0.52      0.59       432
           3       0.72      0.69      0.71       432
           4       0.88      0.93      0.91       432
           5       0.80      0.92      0.85       432
           6       0.76      0.76      0.76       432
           7       0.89      0.95      0.92       432

    accuracy                           0.78      3024
   macro avg       0.77      0.78      0.77      3024
weighted avg       0.77      0.78      0.77      3024

Confusion Matrix:
 [[295  78   1   0  17   2  39]
 [100 226  14   1  63  15  13]
 [  0   8 299  32  16  77   0]
 [  0   0  24 401   0   7   0]
 [  8  11  12   0 398   3   0]
 [  4   8  66  20   6 328   0]
 [ 20   1   0   0   0   0 411]]

#Logistic Regression

from sklearn.linear_model import LogisticRegression

X_train, X_test, y_train, y_test = train_test_split(X, y, 
test_size=0.2, random_state=42, stratify=y)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l2'],
    'solver': ['lbfgs', 'saga'],
    'max_iter': [1000]
}

logreg = LogisticRegression(multi_class='multinomial')
grid = GridSearchCV(logreg, param_grid, cv=5, scoring='accuracy')
grid.fit(X_train_scaled, y_train)

best_logreg = grid.best_estimator_
y_pred = best_logreg.predict(X_test_scaled)
```

```
print("Best Params:", grid.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

Best Params: {'C': 10, 'max_iter': 1000, 'penalty': 'l2', 'solver':
'saga'}
Accuracy: 0.6683201058201058
Classification Report:
               precision    recall  f1-score   support

           1       0.65      0.65      0.65       432
           2       0.53      0.44      0.48       432
           3       0.57      0.47      0.51       432
           4       0.79      0.88      0.83       432
           5       0.65      0.73      0.69       432
           6       0.58      0.62      0.60       432
           7       0.85      0.89      0.87       432

    accuracy                           0.67      3024
   macro avg       0.66      0.67      0.66      3024
weighted avg       0.66      0.67      0.66      3024

Confusion Matrix:
 [[280  66   1   0  23   0  62]
 [103 189  11   0 104  19   6]
 [  0   0 203  64  27 138   0]
 [  0   0  27 381   0  24   0]
 [  0  90  12   0 316  14   0]
 [  0   8 105  36  16 267   0]
 [ 45   1   0   0   1   0 385]]

#Gradient Boosting

from sklearn.ensemble import GradientBoostingClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.05, 0.1],
    'max_depth': [3, 5],
    'subsample': [0.8, 1.0]
```

```
}

gb = GradientBoostingClassifier(random_state=42)
grid = GridSearchCV(gb, param_grid, cv=5, scoring='accuracy', n_jobs=-
1)
grid.fit(X_train_scaled, y_train)

best_gb = grid.best_estimator_
y_pred = best_gb.predict(X_test_scaled)

print("Best Params:", grid.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Best Params: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators':
200, 'subsample': 0.8}
Accuracy: 0.8465608465608465
Classification Report:
               precision    recall  f1-score   support

           1       0.75      0.73      0.74       432
           2       0.75      0.62      0.68       432
           3       0.81      0.83      0.82       432
           4       0.95      0.97      0.96       432
           5       0.87      0.95      0.91       432
           6       0.84      0.84      0.84       432
           7       0.93      0.97      0.95       432

    accuracy                           0.85      3024
   macro avg       0.84      0.85      0.84      3024
weighted avg       0.84      0.85      0.84      3024

Confusion Matrix:
 [[317  78   1   0   6   2  28]
 [ 92 269  15   0  36  16   4]
 [  0   1 359  17  10  45   0]
 [  0   0   9 421   0   2   0]
 [  2   7   9   0 411   3   0]
 [  0   4  49   5   9 365   0]
 [ 14   0   0   0   0   0 418]]
```

```
#Decision Tree

from sklearn.tree import DecisionTreeClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

param_grid = {
    'max_depth': [5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

tree = DecisionTreeClassifier(random_state=42)
grid = GridSearchCV(tree, param_grid, cv=5, scoring='accuracy',
n_jobs=-1)
grid.fit(X_train_scaled, y_train)

best_tree = grid.best_estimator_
y_pred = best_tree.predict(X_test_scaled)

print("Best Params:", grid.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Best Params: {'criterion': 'entropy', 'max_depth': 20,
'min_samples_leaf': 1, 'min_samples_split': 2}
Accuracy: 0.7840608465608465
Classification Report:
               precision    recall  f1-score   support

           1       0.70      0.69      0.69       432
           2       0.64      0.60      0.62       432
           3       0.71      0.72      0.72       432
           4       0.92      0.93      0.92       432
           5       0.87      0.90      0.88       432
           6       0.71      0.72      0.72       432
           7       0.92      0.94      0.93       432

    accuracy                           0.78      3024
   macro avg       0.78      0.78      0.78      3024
weighted avg       0.78      0.78      0.78      3024

Confusion Matrix:
 [[296  97   0   0  10   3  26]
 [102 258  14   0  36  15   7]
 [  0   4 311  25   6  86   0]
 [  0   0  16 400   0  16   0]
 [  2  29   8   0 387   6   0]
 [  0  14  88  11   6 313   0]
 [ 25   1   0   0   0   0 406]]
```

```python
#Random Forest

from sklearn.ensemble import RandomForestClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'bootstrap': [True, False]
}

rf = RandomForestClassifier(random_state=42)
grid = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy', n_jobs=-
1)
grid.fit(X_train_scaled, y_train)

best_rf = grid.best_estimator_
y_pred = best_rf.predict(X_test_scaled)

print("Best Params:", grid.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Best Params: {'bootstrap': False, 'max_depth': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Accuracy: 0.8680555555555556
Classification Report:
               precision    recall  f1-score   support

           1       0.79      0.79      0.79       432
           2       0.79      0.67      0.72       432
           3       0.85      0.84      0.84       432
           4       0.95      0.97      0.96       432
           5       0.89      0.95      0.92       432
           6       0.85      0.89      0.87       432
           7       0.95      0.97      0.96       432

    accuracy                           0.87      3024
   macro avg       0.87      0.87      0.87      3024
weighted avg       0.87      0.87      0.87      3024
```

```
Confusion Matrix:
 [[340  65   1   0   5   1  20]
 [ 79 289  12   0  34  15   3]
 [  0   2 362  15   9  44   0]
 [  0   0   8 419   0   5   0]
 [  0   7   9   0 411   5   0]
 [  0   3  34   5   4 386   0]
 [ 13   1   0   0   0   0 418]]

#Stacked Model RF

from sklearn.ensemble import StackingClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

base_learners = [
    ('knn', KNeighborsClassifier(n_neighbors=7)),
    ('logreg', LogisticRegression(multi_class='multinomial',
solver='saga', C=1, max_iter=1000)),
    ('gb', GradientBoostingClassifier(n_estimators=100,
learning_rate=0.1, max_depth=3)),
    ('dt', DecisionTreeClassifier(max_depth=10)),
    ('rf', RandomForestClassifier(n_estimators=200, max_depth=20))
]

final_estimator = RandomForestClassifier()

stacked_model = StackingClassifier(
    estimators=base_learners,
    final_estimator=final_estimator,
    cv=5,
    n_jobs=-1
)

stacked_model.fit(X_train_scaled, y_train)
y_pred = stacked_model.predict(X_test_scaled)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

Accuracy: 0.8657407407407407
Classification Report:
               precision    recall  f1-score   support
```

```
           1       0.79      0.78      0.79       432
           2       0.76      0.69      0.72       432
           3       0.82      0.84      0.83       432
           4       0.96      0.96      0.96       432
           5       0.92      0.94      0.93       432
           6       0.85      0.88      0.86       432
           7       0.96      0.97      0.96       432

    accuracy                           0.87      3024
   macro avg       0.86      0.87      0.86      3024
weighted avg       0.86      0.87      0.86      3024

Confusion Matrix:
 [[339  74   1   0   2   1  15]
 [ 77 300  15   0  24  12   4]
 [  0   5 362  13   8  44   0]
 [  0   0  10 414   0   8   0]
 [  0  12   9   0 407   4   0]
 [  0   6  42   4   2 378   0]
 [ 14   0   0   0   0   0 418]]
```

```
#Stacked Model GB

from sklearn.ensemble import StackingClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

base_learners = [
    ('knn', KNeighborsClassifier(n_neighbors=7)),
    ('logreg', LogisticRegression(multi_class='multinomial',
solver='saga', C=1, max_iter=1000)),
    ('gb', GradientBoostingClassifier(n_estimators=100,
learning_rate=0.1, max_depth=3)),
    ('dt', DecisionTreeClassifier(max_depth=10)),
    ('rf', RandomForestClassifier(n_estimators=200, max_depth=20))
]

final_estimator = GradientBoostingClassifier()

stacked_model = StackingClassifier(
    estimators=base_learners,
    final_estimator=final_estimator,
    cv=5,
    n_jobs=-1
```

```
)

stacked_model.fit(X_train_scaled, y_train)
y_pred = stacked_model.predict(X_test_scaled)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.859457671957672
Classification Report:
               precision    recall  f1-score   support

           1       0.78      0.77      0.77       432
           2       0.75      0.70      0.72       432
           3       0.82      0.82      0.82       432
           4       0.96      0.94      0.95       432
           5       0.91      0.95      0.93       432
           6       0.84      0.89      0.87       432
           7       0.95      0.95      0.95       432

    accuracy                           0.86      3024
   macro avg       0.86      0.86      0.86      3024
weighted avg       0.86      0.86      0.86      3024

Confusion Matrix:
 [[331  80   1   0   3   0  17]
 [ 73 301  16   0  28  10   4]
 [  0   5 356  12   9  50   0]
 [  0   0  17 407   0   8   0]
 [  0  10   8   0 409   5   0]
 [  0   4  36   5   2 385   0]
 [ 22   0   0   0   0   0 410]]
```

```
#Stacked Model LR

from sklearn.ensemble import StackingClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

base_learners = [
    ('knn', KNeighborsClassifier(n_neighbors=7)),
    ('logreg', LogisticRegression(multi_class='multinomial',
solver='saga', C=1, max_iter=1000)),
    ('gb', GradientBoostingClassifier(n_estimators=100,
```

```
learning_rate=0.1, max_depth=3)),
    ('dt', DecisionTreeClassifier(max_depth=10)),
    ('rf', RandomForestClassifier(n_estimators=200, max_depth=20))
]

final_estimator = LogisticRegression()

stacked_model = StackingClassifier(
    estimators=base_learners,
    final_estimator=final_estimator,
    cv=5,
    n_jobs=-1
)

stacked_model.fit(X_train_scaled, y_train)
y_pred = stacked_model.predict(X_test_scaled)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.8581349206349206
Classification Report:
               precision    recall  f1-score   support

           1       0.77      0.78      0.78       432
           2       0.76      0.68      0.72       432
           3       0.82      0.82      0.82       432
           4       0.96      0.95      0.95       432
           5       0.90      0.94      0.92       432
           6       0.83      0.88      0.85       432
           7       0.95      0.96      0.96       432

    accuracy                           0.86      3024
   macro avg       0.86      0.86      0.86      3024
weighted avg       0.86      0.86      0.86      3024

Confusion Matrix:
 [[337  69   1   0   6   1  18]
 [ 81 294  13   0  28  13   3]
 [  0   3 354  13  11  51   0]
 [  0   0  17 409   0   6   0]
 [  0  12   8   0 407   5   0]
 [  0   7  39   5   2 379   0]
 [ 17   0   0   0   0   0 415]]
```

```
#Stacked Model DT

from sklearn.ensemble import StackingClassifier
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

base_learners = [
    ('knn', KNeighborsClassifier(n_neighbors=7)),
    ('logreg', LogisticRegression(multi_class='multinomial',
solver='saga', C=1, max_iter=1000)),
    ('gb', GradientBoostingClassifier(n_estimators=100,
learning_rate=0.1, max_depth=3)),
    ('dt', DecisionTreeClassifier(max_depth=10)),
    ('rf', RandomForestClassifier(n_estimators=200, max_depth=20))
]

final_estimator = DecisionTreeClassifier()

stacked_model = StackingClassifier(
    estimators=base_learners,
    final_estimator=final_estimator,
    cv=5,
    n_jobs=-1
)

stacked_model.fit(X_train_scaled, y_train)
y_pred = stacked_model.predict(X_test_scaled)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.8035714285714286
Classification Report:
               precision    recall  f1-score   support

           1       0.67      0.67      0.67       432
           2       0.65      0.63      0.64       432
           3       0.76      0.77      0.77       432
           4       0.95      0.91      0.93       432
           5       0.89      0.88      0.89       432
           6       0.78      0.83      0.81       432
           7       0.93      0.92      0.93       432

    accuracy                           0.80      3024
   macro avg       0.80      0.80      0.80      3024
weighted avg       0.80      0.80      0.80      3024
```

```
Confusion Matrix:
 [[291 110   1   0   6   1  23]
 [104 272  13   1  27  10   5]
 [  1   8 334  17  10  62   0]
 [  0   0  23 392   0  17   0]
 [  3  26  11   0 382  10   0]
 [  2   3  59   4   4 360   0]
 [ 32   1   0   0   0   0 399]]
```

```python
#Stacked Model KNN

from sklearn.ensemble import StackingClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

base_learners = [
    ('knn', KNeighborsClassifier(n_neighbors=7)),
    ('logreg', LogisticRegression(multi_class='multinomial',
solver='saga', C=1, max_iter=1000)),
    ('gb', GradientBoostingClassifier(n_estimators=100,
learning_rate=0.1, max_depth=3)),
    ('dt', DecisionTreeClassifier(max_depth=10)),
    ('rf', RandomForestClassifier(n_estimators=200, max_depth=20))
]

final_estimator = KNeighborsClassifier()

stacked_model = StackingClassifier(
    estimators=base_learners,
    final_estimator=final_estimator,
    cv=5,
    n_jobs=-1
)

stacked_model.fit(X_train_scaled, y_train)
y_pred = stacked_model.predict(X_test_scaled)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.8396164021164021
Classification Report:
               precision    recall  f1-score   support
```

```
              1       0.76      0.74      0.75       432
              2       0.71      0.66      0.69       432
              3       0.79      0.80      0.79       432
              4       0.95      0.94      0.95       432
              5       0.87      0.92      0.90       432
              6       0.84      0.86      0.85       432
              7       0.95      0.95      0.95       432

    accuracy                          0.84      3024
   macro avg       0.84      0.84      0.84      3024
weighted avg       0.84      0.84      0.84      3024

Confusion Matrix:
 [[318  86   1   0   7   1  19]
 [ 80 286  15   0  36  11   4]
 [  0   7 347  16  10  52   0]
 [  0   0  22 406   0   4   0]
 [  2  18  11   0 398   3   0]
 [  0   4  46   4   6 372   0]
 [ 19   1   0   0   0   0 412]]
```

```
#Stacked Model XGB

from sklearn.ensemble import StackingClassifier
from xgboost import XGBClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

base_learners = [
    ('knn', KNeighborsClassifier(n_neighbors=7)),
    ('logreg', LogisticRegression(multi_class='multinomial',
solver='saga', C=1, max_iter=1000)),
    ('gb', GradientBoostingClassifier(n_estimators=100,
learning_rate=0.1, max_depth=3)),
    ('dt', DecisionTreeClassifier(max_depth=10)),
    ('rf', RandomForestClassifier(n_estimators=200, max_depth=20))
]

final_estimator = XGBClassifier()

stacked_model = StackingClassifier(
    estimators=base_learners,
    final_estimator=final_estimator,
    cv=5,
    n_jobs=-1
```

```
)

stacked_model.fit(X_train_scaled, y_train)
y_pred = stacked_model.predict(X_test_scaled)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

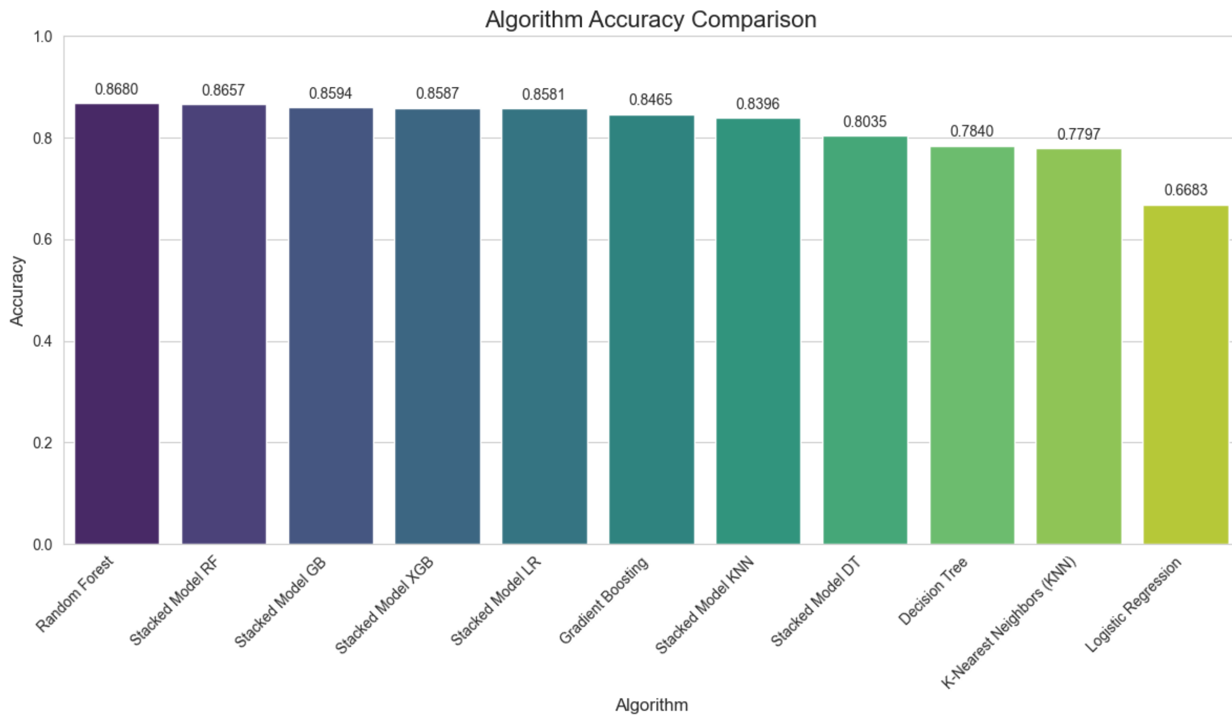
```
Accuracy: 0.8587962962962963
Classification Report:
               precision    recall  f1-score   support

           1       0.79      0.75      0.77       432
           2       0.76      0.71      0.73       432
           3       0.81      0.82      0.82       432
           4       0.96      0.95      0.96       432
           5       0.91      0.95      0.93       432
           6       0.83      0.87      0.85       432
           7       0.94      0.96      0.95       432

    accuracy                           0.86      3024
   macro avg       0.86      0.86      0.86      3024
weighted avg       0.86      0.86      0.86      3024

Confusion Matrix:
 [[324  78   1   0   4   2  23]
 [ 68 305  16   0  28  12   3]
 [  0   4 356  12   6  54   0]
 [  0   0  15 411   0   6   0]
 [  0  10   8   0 410   4   0]
 [  0   6  44   4   3 375   0]
 [ 16   0   0   0   0   0 416]]
```

Now , lets make a bar graph for comparing all these different models

```
import matplotlib.pyplot as plt
import seaborn as sns

algorithm_accuracies = {
    'Decision Tree': 0.7840,
    'K-Nearest Neighbors (KNN)': 0.7797,
    'Logistic Regression': 0.6683,
    'Random Forest': 0.8680,
    'Gradient Boosting': 0.8465,
    'Stacked Model RF': 0.8657,
    'Stacked Model GB': 0.8594,
    'Stacked Model LR': 0.8581,
    'Stacked Model DT': 0.8035,
```

```
    'Stacked Model KNN': 0.8396,
    'Stacked Model XGB': 0.8587
}

accuracies_series =
pd.Series(algorithm_accuracies).sort_values(ascending=False)

sns.set_style("whitegrid")

plt.figure(figsize=(12, 7))
ax = sns.barplot(x=accuracies_series.index,
y=accuracies_series.values, palette='viridis')

for p in ax.patches:
    ax.annotate(f'{p.get_height():.4f}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center',
                xytext=(0, 9),
                textcoords='offset points')

plt.title('Algorithm Accuracy Comparison', fontsize=16)
plt.xlabel('Algorithm', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.ylim(0.0, 1.0)

plt.xticks(rotation=45, ha='right')

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

```
C:\Users\tanis\AppData\Local\Temp\ipykernel_14008\1837040399.py:23:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  ax = sns.barplot(x=accuracies_series.index,
y=accuracies_series.values, palette='viridis')
```



Hence , we can conclude Random Forest did the best for our dataset although even its performance wasnt exceptionally great but it was surely the standout among our chosen algorithms , the dataset had around 15120 entries which isnt that big considering there can be possibly 7 outputs and hence maybe a larger dataset might help significantly with our work going ahead. Even after applying hyperparameter tuning with every algorithm the results were not exceptional and hence there still remains a lot more potential for improvement.