Introduction: Ensemble learning techniques have gained prominence in machine learning due to their ability to combine multiple models to improve predictive performance. Stacking, a popular ensemble method, leverages the strengths of diverse base models and a meta-model to achieve superior predictive accuracy. In this code, we implement a stacking ensemble with Random Forest and k-Nearest Neighbors (KNN) as base models, and XGBoost as the meta-model. The stacking ensemble approach involves training multiple base models on the training data and then combining their predictions as input features for a meta-model, which learns to make the final predictions. Each base model captures different aspects of the data, and the meta-model integrates these diverse perspectives to make more accurate predictions than any individual model. To ensure optimal performance, hyperparameters of the base models and the meta-model are tuned using RandomizedSearchCV, a technique that efficiently explores a hyperparameter search space to find the best configuration. The Random Forest and KNN base models are tuned for parameters such as the number of estimators, maximum depth, and minimum samples split, while XGBoost is tuned for parameters like the number of estimators, maximum depth, learning rate, and regularization terms. The stacking classifier is then instantiated with the tuned base models and the meta-model. During training, the base models learn from the data and generate predictions, which are used as input features for the meta-model. The meta-model combines these predictions to produce the final output. Finally, the performance of the stacking ensemble is evaluated using nested cross-validation, a robust technique for estimating the model's generalization performance. The mean accuracy score from nested cross-validation provides an unbiased estimate of the stacking ensemble's predictive accuracy across different folds of the dataset.

Data Description: The data set consists of the drilling data gathered from one of the oil and gas operators in the Eagleford shale region in Texas (USA). The collected data consists of 17 different parameters named as Depth, Hook Load, Bit Weight, Block Height, Rate of Penetration (ROP), Top Drive RPM, Top Drive Torque, Differential Pressure, Flow In Rate, Pump SPM (Strokes per minute), Flow Out Percent, Bit Size, Gamma Ray, Mud Weight In, Pump Pressure, D-exponent and formation. With the advancement of new technology, the drilling industry is now gifted with state of art equipment known as Auto-driller. The auto driller utilizes only three parameters as input named as Bit weight, Top Drive RPM and Flow In Rate to predict the next Rate of penetration set point. Hence, the aim of this study is to predict the formation by using only the Auto-driller parameters as an input.

Experimental Setup: The programming language used for this experiment will be python. We begin with importing a few libraries named as pandas, numpy, matplotlib.pyplot and seaborn. Pandas library helps in data manipulation and data analysis, numpy helps in performing operations on arrays, matplotlib.pyplot and seaborn are used for plotting. We begin the experiment by uploading our dataset using the pd.read_csv function. After this we divided the dataset into input and target variables in terms of X and y. We then used the .info() function to

get the information of the dataset. After this we used the .isnull.sum() function to find out the null values in the dataset. After this we removed all the unnecessary data columns from our dataset using drop() function. After this we used the corr() function to find thecorrelation between the 3 input parameters namely Bit weight, Top Drive RPM and Flow In Rate. Now we have used value_counts() function to find out the value counts of the different 'formations' present in the dataset. As the dataset was found out to be imbalanced, we have now used Synthetic Minority Oversampling Technique, or SMOTE to balance the data. We have now used MinMaxScaler to normalize the data.

Now we begin with our prediction using different algorithms:

a) **Random Forest:** For making predictions using Random Forest Algorithm we began by importing necessary libraries as RandomForestClassifier from sklearn.ensemble, cross_val_score , KFold and RandomizedSearchCV from sklearn.model_selection, accuracy_score from sklearn.metrics, and randint from scipy.stats. After this we have defined the inner and outer splits for the nested resampling. Here both inner and outer splits were equal to 5. After this we defined the search space for different hyperparameters as 'n_estimators': randint(50, 200), 'max_depth': [None, 10, 20, 30], 'min_samples_split': randint(2, 20). After defining the hyperparameter search space we performed an outer cross validation loop for nested resampling and then inner cross validation loop for hyperparameter tuning using Randomized Search. The number of iterations tried for random search was 50. Then we have finally reported the mean accuracy score for this algorithm.

b) **K-Nearest Neighbors (KNN):** For making predictions using KNN Algorithm we began by importing necessary libraries as KNeighborsClassifier from sklearn.ensemble, cross_val_score , KFold and RandomizedSearchCV from sklearn.model_selection, accuracy_score from sklearn.metrics, and randint from scipy.stats. After this we have defined the inner and outer splits for the nested resampling. Here both inner and outer splits were equal to 5. After this we defined the search space for different hyperparameters as 'n_neighbors': randint(1, 20),'weights': ['uniform', 'distance'], and 'p': [1, 2]. After defining the hyperparameter search space we performed an outer cross validation loop for nested resampling and then inner cross validation loop for hyperparameter tuning using Randomized Search. The number of iterations tried for random search was 50. Then we finally reported the mean accuracy score for this algorithm.

c) **XGBoost:** For making predictions using XGBoost Algorithm we began by importing necessary libraries as XGBClassifier from sklearn.ensemble, cross_val_score , KFold and RandomizedSearchCV from sklearn.model_selection, accuracy_score from sklearn.metrics, and randint from scipy.stats. After this we have defined the inner and

outer splits for the nested resampling. Here both inner and outer splits were equal to 5. After this we defined the search space for different hyperparameters as 'n_estimators': randint(50, 200), 'max_depth': randint(3, 10), 'learning_rate': [0.001, 0.01, 0.1, 0.2, 0.3], 'subsample': [0.6, 0.7, 0.8, 0.9, 1.0], 'colsample_bytree': [0.6, 0.7, 0.8, 0.9, 1.0], 'gamma': [0, 1, 5], 'reg_alpha': [0, 0.1, 0.5, 1.0], and 'reg_lambda': [0, 1, 5, 10]. After defining the hyperparameter search space we performed an outer cross validation loop for nested resampling and then inner cross validation loop for hyperparameter tuning using Randomized Search. The number of iterations tried for random search was 50. Then we finally reported the mean accuracy score for this algorithm.

d) **Stacking or Stacked Generalization:** For making predictions using Stacking we began by importing necessary libraries as StackingClassifier, RandomForestClassifier, KNeighborsClassifier , XGBClassifier from sklearn.ensemble, cross_val_score , KFold and RandomizedSearchCV from sklearn.model_selection, accuracy_score from sklearn.metrics, and randint from scipy.stats. After this we have defined the inner and outer splits for the nested resampling. Here both inner and outer splits were equal to 5. After this we defined the search space for different hyperparameters for all the algorithms used as for Random Forest ('n_estimators': randint(50, 200), 'max_depth': [None, 10, 20, 30], 'min_samples_split': randint(2, 20)), KNN ('n_neighbors': randint(1, 20),'weights': ['uniform', 'distance'], and 'p': [1, 2]), and for XGBoost ('n_estimators': randint(50, 200), 'max_depth': randint(3, 10), 'learning_rate': [0.001, 0.01, 0.1, 0.2, 0.3], 'subsample': [0.6, 0.7, 0.8, 0.9, 1.0], 'colsample_bytree': [0.6, 0.7, 0.8, 0.9, 1.0], 'gamma': [0, 1, 5], 'reg_alpha': [0, 0.1, 0.5, 1.0], and 'reg_lambda': [0, 1, 5, 10]). After defining the hyperparameter search space for all the algorithms we have now initialized our base learners which are Random Forest and KNN and have performed hyperparameter tuning for these base learners using random search. The number of iterations was 50 for both the base learners. After this we initialized our meta learner which is XGBoost in this case and have performed hyperparameter tuning for the meta learner as well using random search. The number of iterations for the meta learner was also 50. After this we defined our stacking classifier were random forest and KNN were base estimators and XGBoost was final estimator. After this we performed an outer cross validation loop for nested resampling. After this we fitted the base learners on the training data. We then found out the best base model and then fitted the stacking classifier on the training data. Then we finally reported the mean accuracy score for this algorithm.

Results: The value_counts() function showed that both the datasets were imbalanced, and the values have been displayed in the appendix section. The mean accuracy scores obtained using different algorithms is reported in the table below:

ML Algorithm	Mean Accuracy Score
Random Forest	0.95
K-Nearest Neighbors (KNN)	0.95
XGBoost	0.94
Stacking	0.96

References:

1) "How to use seaborn plotting", https://www.geeksforgeeks.org/python-seaborn-tutorial/
2) "Normalizing the data by use of scaler", https://www.geeksforgeeks.org/data-pre-processing-wit-sklearn-using-standard-and-minmax-scaler/
3) Sklearn API Reference. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble
4) How to Implement Stacked Generalization (Stacking) From Scratch With Python https://machinelearningmastery.com/implementing-stacking-scratch-python/#:~:text=Ensemble%20methods%20are%20an%20excellent,models%20trained%20on%20your%20dataset.
5) Cross Validation Explained: Evaluating estimator performance. Improve your ML model using cross validation. https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85
6) Nested Cross-Validation for Machine Learning with Python https://machinelearningmastery.com/nested-cross-validation-for-machine-learning-with-python/

Appendix:

Value count for different formations in the dataset.

```
formation
4    5393
0    1766
2     829
1     307
3     256
Name: count, dtype: int64
```

Correlation between the final input parameters: