APCS109A – DATA SCIENCE I: INTRODUCTION TO DATA SCIENCE

Oil & Gas project

12/11/2022

Group 90

Mofei Wang

Shaoyi Li

Sun Kim

Christoph Wippel

Benito D Isaac

IACS

INSTITUTE FOR APPLIED COMPUTATIONAL SCIENCE

AT HARVARD UNIVERSITY

I. Motivation, context & framing of the problem

The oil & gas industry is a crucial part of the global economy, with revenues of over $7 trillion in 2022 and a growth rate of 16.4% from the previous year. Making decisions about drilling and exploitation of oil and its derivatives is challenging due to the many factors that can affect production, such as the geophysical characteristics of oil deposits. As a result, investors and operators often seek to maximize oil output while minimizing costs to improve the profitability of their investments.

To support these efforts, predictive analysis of oil production can be a valuable tool. It can help investors and operators make more informed decisions about drilling and exploitation, and it can improve overall production of the commodity and its derivatives. More specifically, our dataset is only focused on the drilling project in the fracturing business[1].

The aim of the Oil & Gas case project is to develop a predictive model for cumulative oil output over a 12-month period based on data from 6000+ wells (observations) operated by 65 operators. The data includes characteristics such as well depth, oil in place, porosity, maturity, and structure derivative.

The project is divided into several steps:

1. Initial exploratory data analysis: This step involves analyzing the data to identify patterns and relationships between the different variables. We also spoke with a domain expert with years of experience in the fracturing business.
2. Pre-processing of the data: The data is cleaned and prepared for modeling, including imputing missing values using the Multivariate Imputation with Chained Equations (MICE) method.
3. Training and evaluating an initial baseline model: A multivariate linear model is trained and evaluated using one-hot encoding of categorical variables and scaled numerical variables.
4. Feature elimination using Lasso regularization: This step involves using Lasso regularization to identify and eliminate features that do not contribute significantly to the model's performance.
5. Developing further tree-based models: Additional tree-based models, such as random forests, extra-trees, and XGboost, are trained and evaluated, and the hyperparameters of these models are fine-tuned for improved performance.

II. Project aim

The goal is to develop a model that best predicts the amount of oil produced per well within the first 12 months of the well's productive life ("CumOil12Month") from the given predictors using the given data, which can help investors and operators make more informed decisions about drilling and exploitation of oil and its derivatives, leading to improved production and profitability in the industry.

III. Data description & Pre-processing

III.1. Description of data

The original Oil & Gas dataset consists of 6,098 observations and 23 variables, including the response variable, CumOil12Month. Since the response variable is quantitative, our prediction task is a regression problem. The independent variables can be grouped into three categories based on the information they provide: completion, geology, and well spacing.

1. Completion: This group includes variables related to the completion of the well, such as the operator, completion date, amount of proppant and fluid used, and the percent of different components of the reservoir. It also includes the total cost of the horizontal well in millions of dollars.
2. Geology: This group includes variables related to the geology of the reservoir and the quality of its rock samples, such as the layer of reservoir, depth, porosity, and pressure.
3. Well spacing: This group includes variables related to the spacing of the well, such as the horizontal and vertical distances to the nearest offset well in feet.

Overall, the dataset includes both categorical and quantitative variables, with 65 unique values for the operator/company name and nine unique numbers for the reservoir (see **Table 1**).

Table 1. Description of data

Variable code name	Type	Category	Description
Operator	categorical	predictor	Company that operates well (0- 64)
Reservoir	categorical	predictor	Geological formation targeted by well (varies from 1-8)
CompletionDate	datetime	predictor	Completion date /start date of well operations
LateralLength_FT	numerical	predictor	Completed length of horizontal well
ProppantIntensity_LBSPerFT	numerical	predictor	Amount of proppant per lateral foot used to complete well
FluidIntensity_BBLPerFT	numerical	predictor	Amount of fluid per lateral foot used to complete well
HzDistanceToNearestOffsetAtDrill	numerical	predictor	Horizontal distance to nearest offset well at completion date (in feet).
HzDistanceToNearestOffsetCurrent	numerical	predictor	Horizontal distance to nearest offset well at completion date (in feet).
VtDistanceToNearestOffsetCurrent	numerical	predictor	Vertical distance to nearest offset well measured at completion date (in feet).
VtDistanceToNearestOffsetAtDrill	numerical	predictor	Vertical distance to nearest offset well measured at completion date (in feet).
WellDepth	numerical	predictor	The depth of the well (in feet).
ReservoirThickness	numerical	predictor	Thickness of reservoir (in percent)
OilInPlace	numerical	predictor	Amount of oil in reservoir (in millions of barrels/square mile)
Porosity	numerical	predictor	Porosity of reservoir (in percent)

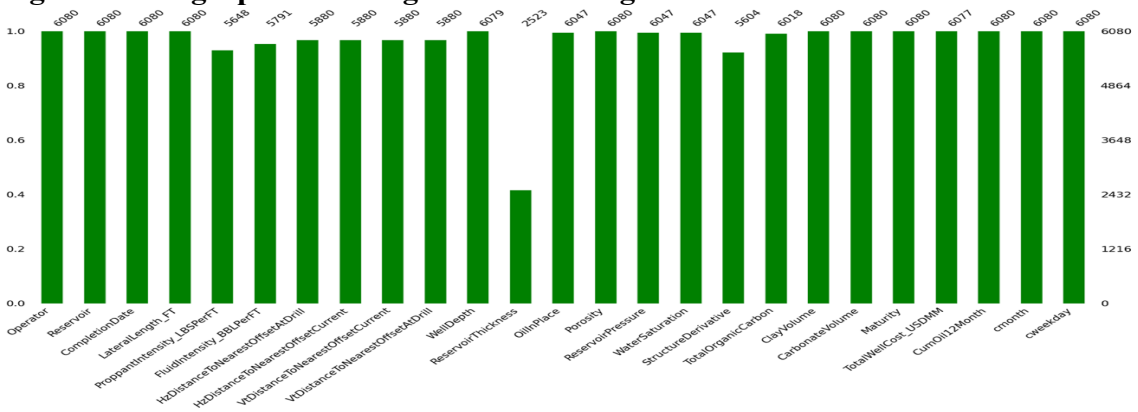
ReservoirPressure	numerical	predictor	Pressure of reservoir (in PSI)
WaterSaturation	numerical	predictor	% of water saturation in target reservoir fluid (in percent).
StructureDerivative	numerical	predictor	% of change in depth of target formation (in percent)
TotalOrganicCarbon	numerical	predictor	% of total organic carbon in target geologic formation (in percent).
ClayVolume	numerical	predictor	% of clay in target reservoir (in percent).
CarbonateVolume	numerical	predictor	% of carbonate in target reservoir (in percent)
Maturity	numerical	predictor	Maturity of target reservoir (in percent)
TotalWellCost_USDMM	numerical	predictor	Total cost of horizontal well (in millions of USD).
CumOil12Month	numerical	Dependent variable	Amount of oil produced in 12-month period of production (in barrels)

III.2. Data Processing: Missingness, Imputation and Variables transformation

Missingness in the dataset

As shown in **Figure 1**, Initial exploration of the raw dataset has shown that a few data points are missing for variables such as LateralLength_FT, ResoirThickness, TotalOrganicCarbon, StructureDerivative, PropaneIntensity_LBSPerFT. We dropped the predictor ReservoirThickness because more than 57% of data points were missing for this predictor. Our initial task was to find a robust method for the imputation of the missing data points. Based on existing literature and our knowledge of available tools, we decided to use the Multiple Imputation with Chained Equations (MICE) method which is described in a later section.

Figure 1. Bar graph of missing data in the original dataset



Other variables transformation

We added indictor variables (0/1) for missing values in addition to the imputation. We dropped TotalWellCost_USDMM since majority of the cost information is already included in other predictors. More importantly the variable is usually only observed after the well is built, which is not practically useful to decide if a well is worth drilling.

We only kept the month of year (categorical dummies) information of the CompletionDate as the predictors. Row wise we remove the observations whose CompletionDate is before Jan 2011 as the fracturing technology used in the earlier years can be hugely different.

We added indicator variables where HzDistanceToNearestOffset=2000 and VtDistanceToNearestOffset=500 which seem like the capped values for extreme large recordings, based on our data exploration process. We also dropped the rows where these distances are under 1 foot which can have bad data quality.

We transformed the response variable, CumOil12Month, by using log (base 2), so that we weighed the errors from big wells and small wells more evenly. For example, we see the errors the same in [true: 500, predicted: 600] and in [true: 50000, predicted: 60000]. In order to apply log transformation, we double checked the close-to-zero values in the response variable and dropped these observations.

Imputation with Multivariate Imputation with Chained Equations (MICE) method

The MICE imputation is a very innovative and advanced method of handling missing values, in that it accounts for uncertainty by iteratively replacing each missing data points by multiple possible values and finally combine them together to have a final one, instead of doing a single imputation as other methods such as KNN, mean, mode imputation techniques. Moreover, there is no necessity to tune any hyperparameter in MICE which enables data analysts to spend more time on the prediction models. The MICE imputation technique is applicable for both categorical and continuous variables (only the latter needs imputation in our case). Here is a description of the procedure:

1- Imputation step: Missing values are identified and imputed by a random sample of multiple plausible values, creating multiple completed datasets in the process.
2- Estimation step: Datasets generated during the initial steps are analyzed separately.
3- Pooling step: Results obtained from the previous step are finally combined into one final imputation value for each missing data point to give the final imputed dataset[2].

The above operations are done iteratively, re-imputing round after round, until the re-imputed values are close enough to the previous round or a max number if iteration is reached. Many programming software have already incorporated a MICE package that handles the complex process in background: Python, R, Stata etc. We used the implementation from sklearn in this project.One fundamental assumption of the MICE technique is that data are missing at least at random.

IV. Exploratory Data Analysis (EDA)

Initial explorations

As an exploratory analysis, we plotted scatter plots, histograms to investigate distributions of each variable, created a correlogram to identify any correlation across variables, and time series plot for investigating trend (based on completion dates)[1].

V. MODELS

V.1. Baseline model

Rationale:

We chose a linear model as a baseline model, since it can provide a simple and effective starting point for modeling a dataset, and can be useful for comparison and evaluation purposes. In particular, linear models can be quickly and easily trained on a dataset. In addition, linear models can often provide a good starting point for more complex models, and the regularized ones (next section) like lasso can be used to identify important features and other patterns in the data.

Using a linear model as a baseline can also be useful for comparison purposes. As we are trying to develop a more complex model to improve upon the performance of a linear model, we are able to use the linear model as a baseline to see how much improvement our new model is able to achieve. This helps us determine whether the added complexity of your new model is justified, or whether a simpler model may be more effective.

Method & result:

First, we used the Standard Scaler to fit/transform our predictor features. Then we used multi-linear regression to train with our processed data. As a result, we got our training R-squared at 0.4972 and our test R-squared at 0.4115.

V.2. Regularized models

Rationale:

Other than the baseline model, we wanted to investigate some other linear-style models. We chose regularized models to minimize out-of-sample errors and reach a good balance between overfitting and underfitting. Also, we wanted to make sure our models do have the opportunity to train on multiple train-test splits; therefore, we implemented both Ridge regression and lasso regression with cross-validation after the baseline model. More specifically, Lasso regression would automatically unselect the unimportant features so we could have a better idea of the 110 features in our dataset.

Method & result:

First, we put the processed data into Lasso/Ridge with 10 folds cross-validation and different alphas to check which lambdas would return to us the best mean R-squared. These two

[1] Please refer to the Jupyter Notebook and Milestone2 regarding the EDA visualizations, as we try to save space for the prediction models in this report

regressions gave us a remarkably similar result in accuracy scores and their top 8 important features are identical. For Feature Importance, we will discuss the details later in the report and compare the linear-style models with other models.

As a result, the ridge regression gave us a testing R-squared of 0.4170 at optimal alpha equal to 0.00001 and the lasso regression gave us extremely similar prediction performance with the same optimal alpha (see **Table 2**).

Table 2: Score comparison of linear models

model	training score	testing score
multi-linear	0.4972	0.4115
ridge	0.4991	0.4170
lasso	0.4991	0.4170

V.3. Tree-based models

Rationale:

There are several considerations for switching from linear-style models to tree-based models: One of the main advantages of tree models is that they can model non-linear relationships between the predictors and response variables. This is because tree models use multiple decision boundaries, each of which splits the data in a way that allows the model to learn the complex relationships in the data. In contrast, linear-style models can only model linear relationships, which can limit its ability to accurately capture the underlying structure of the data.

Another advantage of tree-based models is that they will by design capture the interactions between the predictors. This is especially useful in our case, since we introduced multiple missing indicators and max indicators (for distance measurements). We expect the interaction between these variables and the other predictors could improve the prediction power.

Moreover, as Professor Protopapas mentioned in class, XGBoost, one of the tree-based models, is usually the best performer for tabular data, which suits the format of our input dataset. In addition to the above considerations, tree-based models don't rely on the assumption that variables are normally distributed and are relatively easy to interpret.

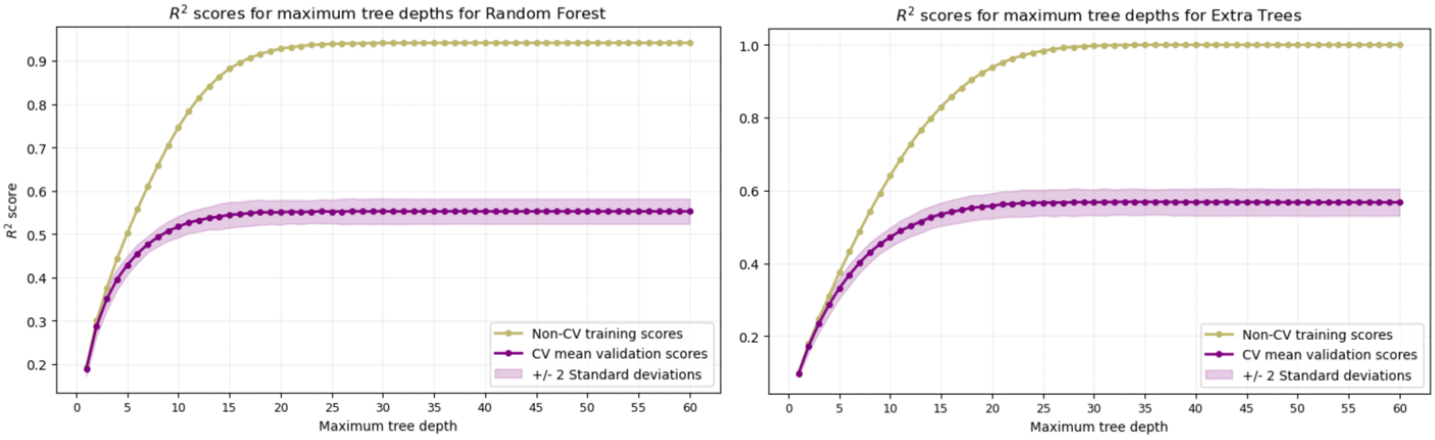
Overall, tree models can be a powerful tool for modeling complex, non-linear relationships in data and can provide accurate and interpretable predictions, especially given the characteristics of our specific dataset.

Method & result:

First, we fitted a single Decision Tree (DT) model on our data and identified the optimal tree depth with cross-validation (CV). The best identified tree depth was 6, beyond which the validation score progressively declined. The single DT model with tree depth 6 achieved a training score of 0.5267 and a testing score of 0.3794.

In an attempt to improve the model performance, we fitted a Random Tree (RT) algorithm and again identified the optimal tree depth with CV. We chose N/3 as the maximum number of features for the subset of predictors to be considered at each split, as this is the parameter for tree-based ensemble algorithms for regression problems recommended by long-term practitioners[2]. We chose 1000 as the number of estimators, which was large enough to achieve good results while still being reasonably computationally expensive. The best identified tree depth was 30, beyond which the validation score plateaued (**Figure 2**). The RT model with tree depth of 30 and 1000 estimators achieved a training score of 0.9406 and a testing score of 0.4956.

Figure 2. CV results for maximum tree depth for Random Forest and Extra Trees



We further tried to improve our model, using Extremely Randomized Trees, also known as Extra Trees (ET). Since ET are computationally more efficient than RF, we were able to increase the number of estimators to 10,000 to achieve better results. Similarly, we identified the optimal tree depth with CV, again using N/3 as the maximum number of features to be considered at each split and 1. The best identified tree-depth was 33, beyond which the validation score plateaued. The ET model with tree-depth of 33 and 10,000 estimators achieved a training score of 0.9992 and a testing score of 0.5038.

Lastly, we used the GridsearchCV() function to further fine-tune the hyperparameters, including the tree depth, the minimum number of samples required to split an internal node, the minimum number of samples required to be at a leaf node, whether or not bootstrap samples were used when building trees, and whether to re-use the solution of the previous call to fit and add more estimators to the ensemble, or just fit a whole new forest (warm_start). The optimal parameters identified with GridSearch were max_depth = 32, max_features = 1.0 (all features), min_samples_leaf = 2, min_samples_split = 2, warm_start = True, and bootstrap = False. We

[2] CS109a Lecture 20, Slide13

then fitted a final ET model with these parameters, which achieved slightly higher results, with a training score of 0.9630 and a testing score of 0.5042. Based on the results we can see ET has the best performance on the test set and RF is a little worse while single DT is much worse than ET and RF (**Table 3**). Moreover, we can see single DT cannot be too deep in order not to overfit while RF can be much deeper, and ET can be even a bit deeper than RF thanks to the benefit of variance reduction.

Table 3. Score comparison of three Tree-based Models with lasso

model	training score	testing score	tree depth
lasso	0.4991	0.4170	
DecisionTreeRegressor	0.5267	0.3796	6
RandomForestRegressor	0.9406	0.4956	30
ExtraTreesRegressor	0.9630	0.5042	32

Additional description on ExtraTrees

Extra Trees (ET), also known as Extremely Randomized Trees, are a variant of the RF algorithm that can be used for both classification and regression problems. Both algorithms are ensemble methods that construct a large number of decision trees and use them to make predictions. However, there are some key differences between the two algorithms that are worth noting.

The main difference is how both algorithms select the cut points for splitting nodes. RF chooses the optimum split while ET chooses it randomly. Though, once the split points are decided, both algorithms choose the best one from the whole subset of features. Therefore, ET adds additional randomization while still retaining some optimization in the fitting process[3]. We expect ET to have lower variance which can be useful in our dataset – we have 110 predictors and 4000+ observations (relatively wide) in the training set. The second difference is that RF uses bootstrap samples, meaning it subsamples the input data with replacement, whereas ET uses the whole original dataset but with the option to use subsample as well.

In addition to the better variance reduction, one advantage of ET is that they are faster to train than RF. ET chooses the cut points randomly instead of picking the optimal by evaluating each potential split which dramatically reduces the amount of computation.

Another interesting fact is that in classification tasks (not our case though), ET often have better performance on imbalanced data sets compared to RF. This is because the randomness in the training process can help reduce the bias towards the majority class that is often present in RF models. Overall, ET can be a useful alternative to RF in situations where the training data is noisy or wide (too many variables), the training time is limited, or the data is imbalanced in classification[4].

There are three main hyperparameters to tune in the algorithm: 1) the number of decision trees in the ensemble, 2) the number of input features to randomly select and consider for each split point, and 3) the minimum number of samples required in a node to create a new split point[3].

It is important to note that while bootstrapping is not implemented in the default ET structure, it can be adjusted in the hyper-parameters, which we took into account in the tuning of the model. This may reduce variance even more because bootstrapping makes it more diversified – however the Gridsearch process selected the optimal hyperparameter with boostrap=False for ET.

Gradient Boosting

We used the XGBoost (XGB) implementation of Gradient Boosting in our study. As mentioned by Professor Protopapas, it is usually the best performer in predictions on tabular data. The fact the algorithm sequentially fit trees based on the previous failures or errors (residuals for regression) makes this type of models very powerful in prediction.

However, it is difficult to find the optimal hyperparameters for XGB. Unlike RF and ET which hardly overfit when the n_estimators are enough, XGB is more prone to overfitting. Hyperparameters should be chosen more carefully, so that we have a reasonable balance between bias and variance. Given the limited time for this project, we had a strategic plan for tuning.

First, we decided the hyperparameters priority list to tune in the GridSearchCV process: max_depth (most important), learning_rate, subsample (subsample proportion of the data points for each tree, to get similar benefit of RF), colsample_bynode (subsample proportion of the features at each node, to get similar benefit of RF)[5]. Since we expect the max_depth to be the most important dimension to explore, we also tried a few pre-tuning rounds to estimate the range of optimal max_depth, before getting into the fine tuning. Furthermore, we use 3-fold CV in the grid search instead of 5-fold CV default setup.

We also gave the models a large n_estimator together with early_stopping_rounds=150 so that the optimal number of iterations can be figured in the sequential tree training process. Therefore, we just reduced one explicit hyperparameter to tune. To make each XGB fitting process stop early we also used 20% of the original training set as the validation set and used the rest 80% to train each tree (iteration) in XGB.

Second, we implemented parallel computing and GPU to expedite the tuning process. Similar to Deep Learning models in tensorflow or pytorch, XGB offers the option to train with a GPU, or even with multiple GPUs. Meanwhile each XGB training in the GridSearchCV process can be done independently which gives us another option to parallelize the computation. After a few benchmarked tests on an 8-CPU 8-GPU online machine. We concluded that parallelizing with all CPUs is the fastest followed by with single GPU, then by with single CPU (default) and last by with multiple GPUs (GPU parallelization may still be premature for XGB).

Best XGB at max_depth=7 gave us training score of 0.9261, CV (mean) score of 0.5579 and test score 0.5106, with other hyperparameters at learning_rate=0.004, colsample_bynode=0.12,

subsample=0.75 (**Table 4**). Compared to the other tree-based models above, we can see XGB made it to achieve the best test score but with a small edge over ET. Besides, the optimal depth for XGB is far smaller than RF and ET which are much less prone to overfitting. The actual optimal depth is close to a single DT. **Figure 3** illustrates the scores at different depths. Training scores are plotted on a separate chart as its scores are in different scales compared to the others.

Figure 3. CV results for XGB

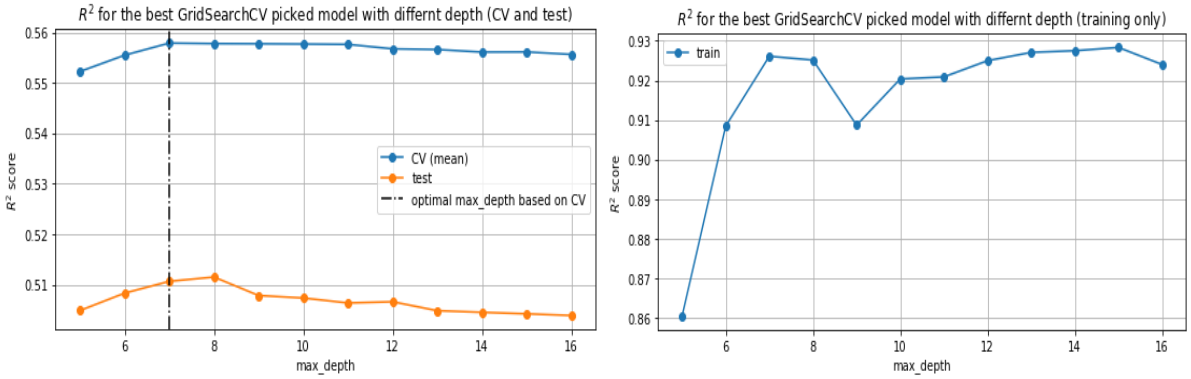


Figure 4 illustrates the learning curve (validation loss vs iteration) for the best XGB model. We can see how the hyperparameter early_stopping_rounds works in finding the optimal number of iterations (sub-trees).Comparing with other tree-based models, XGB outperforms on the testing set (smaller edge over ET).

Figure 4. Learning curve for XGB

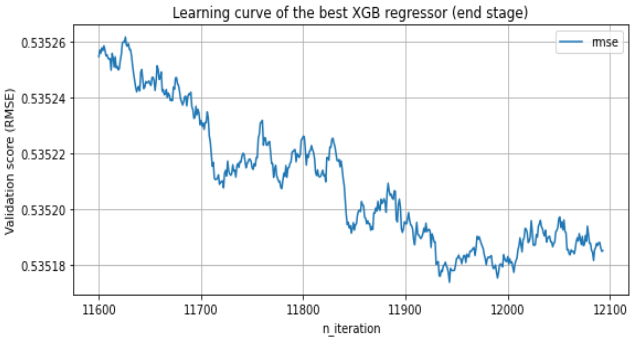
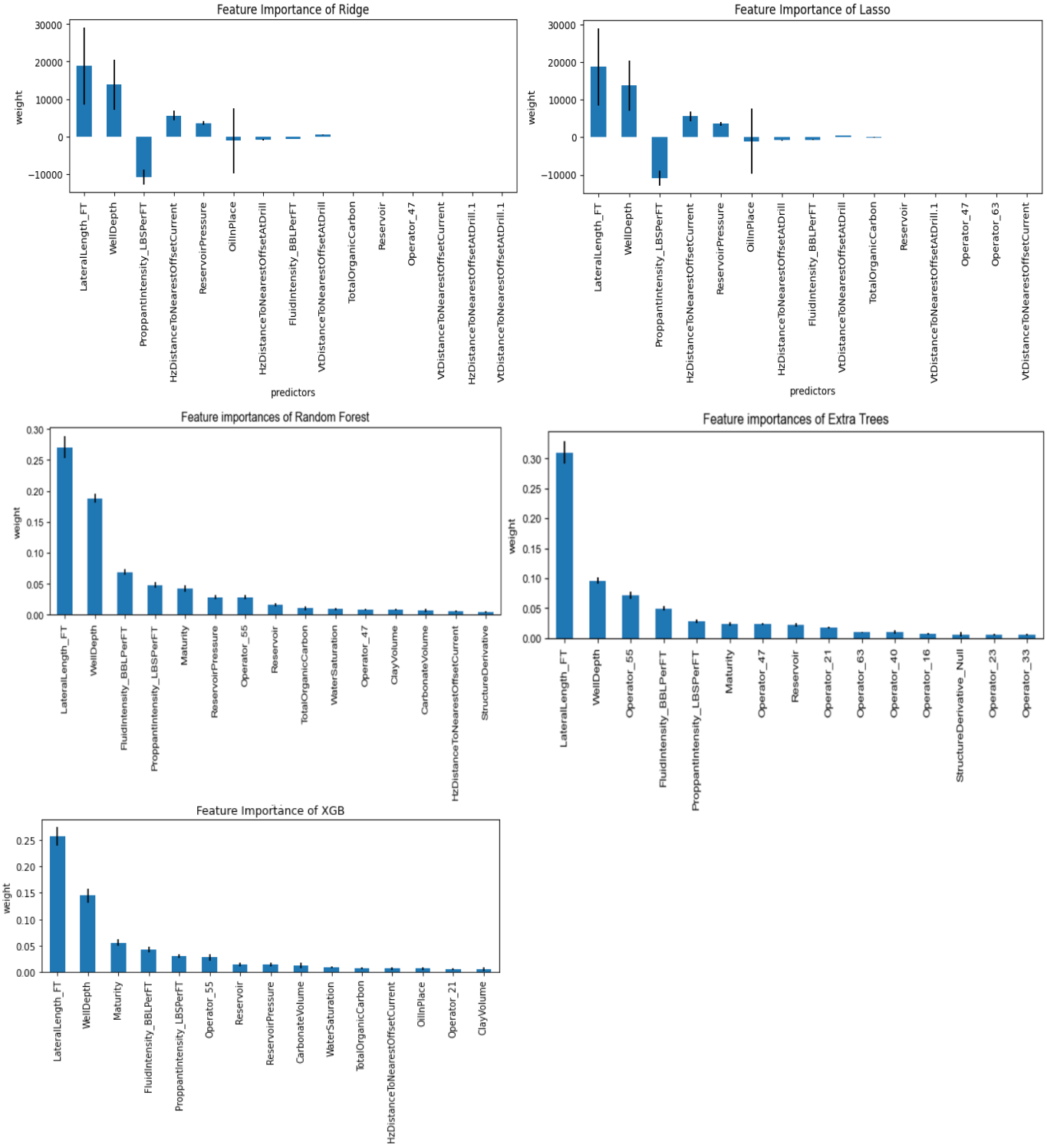


Table 4: Score comparison of four Tree-based Models with lasso

model	training score	testing score	tree depth
lasso	0.4991	0.4170	
DecisionTreeRegressor	0.5267	0.3796	6
RandomForestRegressor	0.9406	0.4956	30
ExtraTreesRegressor	0.9630	0.5042	32
XGBRgressor	0.9261	0.5106	7

Aside from obtaining a reasonable model, we also noticed fact that the results produced by XGB implementation, with random_state and other hyperparameters staying constant, generated different model fitting results. This characteristic is less preferred as it is impossible to reproduce the same model instance with the same setup/hyperparameters.

Figure 5. Permutation Feature Importance



From the feature importance plots shown above (**Figure 5**), we can easily see the differences between lasso and ridge are very small while the differences between the ensemble tree-based models are small too. Compared to the tree-based models, linear models have big negative importance in ProppantDensity which is counter-intuitive to have importance absolute values decreasing very fast after the top 5 features and have big error terms. We can see all models have the same top 2 features. Among the tree-based models, 4 out of the top 5 features are the same and they have different 3rd important features and this could be the main difference between their prediction scores. Moreover, an interesting finding here is the operator #55 seem to have their unique secret sauce in producing oils.

VI. CONCLUSION

The Oil & Gas case project aimed to develop a predictive model for cumulative oil output over the first 12-month period based on data from 6000+ wells operated by 65 operators. To start with, to understand the underlying data and findings from initial exploratory data analysis, we talked to domain experts and decided how to preprocess data for this task.

Our final model (XGBoost) was chosen based on the performance comparison on the testing set, starting from multi linear regression as a baseline model followed by Ridge/Lasso regression, single DecisionTree, RandomForest, ExtraTrees and XGBoost. XGBoost has the best performance on the test set with ExtraTrees being a competitive second.

The innovation in this study is the use of MICE imputation to fill the missing values and the use of ExtraTrees since it is better at variance reduction than RandomForest. In addition, we explored hyperparameter tuning in XGBoost by prioritizing hyperparameters and using the best computation resources available to speed up the gridsearch process. This allowed for a more thorough exploration of XGBoost and improved its performance.

While this project has introduced some interesting innovations, there is still room for improvements. One of the limitations of our final models is that in reality, the ensembled tree models are exceptionally large in space which makes the models very difficult to save and share without re-training. Meanwhile re-training may not reproduce the exact same model instance even if we use the same hyperparameter set. In the future, we also think of implementing feature elimination using tools like mRMR with cross-validation before fitting tree-based models, using shapley to visualize the tree-based models, and tuning more hyperparameters in tree-based models. The follow up study can be done along with the development of more robust multi-GPU implementations and investigate the possibility of combining multiple XGBoost models with a larger hyperparameter set to generate better predictions. Additionally, incorporating time variables in the training and prediction would be of use as technology continues to evolve.

Finally, our model developed in this project will assist investors and operators make more informed decisions about drilling and exploitation, and it can improve overall production of oil and its derivatives.

REFERENCES

1. National Geographic Society. How Hydraulic Fracturing Works | National Geographic Society. https://education.nationalgeographic.org/resource/how-hydraulic-fracturing-works.
2. Jakobsen, J. C., Gluud, C., Wetterslev, J. & Winkel, P. When and how should multiple imputation be used for handling missing data in randomised clinical trials – a practical guide with flowcharts. *BMC Medical Research Methodology* **17**, 162 (2017).
3. Geurts, P., Ernst, D. & Wehenkel, L. Extremely randomized trees. *Mach Learn* **63**, 3–42 (2006).
4. Budu, E. Random Forest Vs. Extremely Randomized Trees | Baeldung on Computer Science. https://www.baeldung.com/cs/random-forest-vs-extremely-randomized-trees (2022).
5. Martins, D. XGBoost: A Complete Guide to Fine-Tune and Optimize your Model. *Medium* https://towardsdatascience.com/xgboost-fine-tune-and-optimize-your-model-23d996fab663 (2021).