12/1/2017

Predicting Oil Production Based on Oil Wells Fracking Quality

GUIDE TO ENGINEERING DATA SCIENCE FINAL PROJECT

Olabode Alamu
UNIVERSITY OF HOUSTON

Table of Contents

SUMMARY

Hydraulic fracking provides a well stimulation method to extract oil from unconventional reservoirs using high pressure fluid to fracture the rocks and allow the flow of fluid from the rock formation. Hydraulic fracking process produces lots of data which can be optimized to bring out better insights and optimize production.

In this project, the data was provided in two spreadsheets- one for training and another to test the trained model. The data contained drilling parameters for various wells in the Permian basin. It was required to predict the oil produced from wells in the test model. The dataset in the training spreadsheet was imported and cleaned using various techniques. The dataset was also aggregated using the groupby method in pandas and data enrichment carried out on the dataset. The mean value was calculated for some columns within the dataset while the sum was calculated for the other columns as part of the enrichment process.

Feature selection is of utmost importance in a multivariate linear regression problem because selecting the wrong features can hurt the accuracy of your models. For this analysis, 12 features were selected. The recursive feature extraction was used to select features out of all the available features in the dataset based on their importance to the regression model.

To obtain improved results, the feature matrix was standardized using the StandardScaler module in sci-kit learn. Four regression models were investigated in this project for accuracy and the mean average error was used as a baseline to measure the performance of each model. The models investigated were Linear Regression, Support Vector Regression, Decision Tree Regression, and Multilayer perceptron regression.

The Multilayer Perceptron Regression gave the least error and was selected to make the predictions in the test model spreadsheet.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This chapter introduces the problem statement of this project, the objective of this project was to use different machine learning algorithm to predict the oil production from various wells located in the Permian basin in Texas. The python programming language was used to carry out data analysis of the dataset. The dataset was provided in two spreadsheets- one for training and the other for testing the selected model that was trained.

1.1 OVERVIEW

Hydraulic fracturing is a stimulation technique in which rock is fractured by a pressurized liquid. The process involves the high-pressure injection of 'fracking fluid' (primarily water, containing sand or other proppants suspended with the aid of thickening agents) into a wellbore to create cracks in the deep-rock formations through which natural gas, petroleum, and brine will flow more freely. When the hydraulic pressure is removed from the well, small grains of hydraulic fracturing proppants (either sand or aluminum oxide) hold the fractures open.

Hydraulic fracturing has helped to produce unconventional oil and gas from reservoirs with low permeability which would have been difficult to extract without the use of this technique.

The drilling process produces tons of information, from pressure readings to flowrates, penetration rates, temperature etc. In recent times, exploration companies have turned to this wealth of data to derive insights into how their processes can be improved.

1.2 METHODOLOGY

The python programming language has lots of libraries which are perfect for data analysis. Notable amongst them is the Pandas library which is used for data manipulation and cleaning. The data cleaning step prepares the dataset for machine learning algorithm. This project is intended to be used to predict the oil production from wells, given the drilling data from various regions in the Permian basin.

This prediction requires a regression model and not a classification problem. Different regression algorithms can be used to predict a multivariate problem, in this case, four models were used-

Linear regression, Support Vector Regression, Decision tree regression and Multilayer perceptron model. The mean average error was used as a metric to measure the performance of the different regression models.

The model with the least amount of error was then selected to predict the oil production from the data contained within the test model spreadsheet. The detailed code for this analysis can be found in the python file with this submission along with the printed codes found in the appendix of this report.

CHAPTER 2

METHODOLOGY

This section covers the method which was used in carrying out the design project. The data for this project was available in two excel files. The software tool used was Python with its Pandas library, Numpy for algebra calculations, Matplotlib for data visualization and Sci-kit learn for machine learning.

2.1 DATA IMPORT AND CLEANING

The training data set was imported using the Pandas library in python, this dataset had 1179 rows and 28 columns of data of varying types such as texts and numbers. Missing values were checked for and only 7 missing values were found in a column. These rows were removed since they had missing values which would have affected the analysis.

Next, the columns that had texts in them were removed from the dataset since they would not be able to be processed in a machine learning regression algorithm.

2.2 AGGREGATION AND DATA ENRICHMENT

The dataset contained repetitive values since it was the data for different wells and it showed various stages in the fracking process. The stages of the fracking process weren't easily extracted from the dataset, so the column which contained the stages information was removed.

The aggregation process was done using the 'groupby' method in Pandas, and the Well ID was chosen as the reference to group the dataset by. The aggregation function chosen was summation. The aggregation process reduced the dataset to 20 rows, each row representing each well ID in the dataset.

Since the summation of columns such as Pressure, True Vertical Depth, Fracture gradient, Horizontal length etc. didn't make much sense as a summation, the mean values of these columns were computed based on the number of entries of each Well ID and the pre-calculated summation. The number of entries of each Well ID was found using a custom function called 'rowcount'.

The snapshot of the code is shown below.

```
"""
This function was created to count the number of rows in the dataset
which partains to a particular well ID number.
it takes in the dataframe of interest as input, counts the number of rows per well id
and returns a dataframe with the number of rows per well ID as output
"""
def rowcount(dataframe):
    Unique = dataframe['WELL_ID'].unique() # Checks for the unique well IDs
    length_list = []
    # slices through the dataframe till only unique Well ids are found and counted
    for i in Unique:
        length=len(dataframe[dataframe['WELL_ID']== i])
        length_list.append(length) # appends the count to the list

    # pass into a dataframe
    Count = pd.DataFrame(data= length_list, columns = ['No of rows'])
    Count['Well ID']= Unique
    return Count
```

Figure 1 Snapshot of code for counting rows.

The mean value was calculation by dividing the summation of the column of interest by the number of rows of each Well ID. This was achieved using another function shown below.

```
# This would be achieved using a custom function
```

```
def mean_calculator(dataframe,new_column_names,old_column_names):
    """
    This function takes in a dataframe and creates new columns based
    on the calculated mean of the previous columns. it requires a list of the new column names and
    a list of the old column names which require a mean to be computed.
    """

    for i,j in zip(new_column_names,old_column_names):
        dataframe[i]= dataframe[j]/dataframe['Count']
```

Figure 2 Snapshot of function to calculate the mean.

2.3 FEATURE EXTRACTION AND STANDARDIZATION

The key features of the dataset which have the most significant influence on regression were calculated using a sci-kit learn model called Recursive Feature Elimination. This model selects the number of important parameters. For this study, 12 features were selected.

The selected features are:

1. Mass of proppant used.
2. Number of production days.
3. Mean Horizontal well length.
4. Mean lower perforation length.
5. Mean maximum pressure.
6. Mean upper perforation.
7. Mean minimum pressure.
8. Total well length.
9. Mean Fracture gradient.
10. Mean middle perforation length.
11. Average pressure of fracking.
12. Mean True Vertical Depth.

From a domain standpoint, these features have a strong influence on the drilling process. The dataset was then used to create the X matrix based on these features and the matrix standardized using the StandardScaler library in sci-kit learn.

2.4 REGRESSION ANALYSIS

The standardized and cleaned X matrix was split into training and testing sets using a 70:30 ratio. The training data was then fed into different regression models.

Four regression models were considered in this project. They are:

1. Linear Regression.
2. Support Vector Regression.
3. Decision Tree Regression.
4. Multilayer Perceptron Regression.

Each regression model was fed with the training data and the test data was used to predict the outcome based on the previous training.

The metric used to compare the efficiency of the different regressors are:

1. Mean Absolute Error.
2. Mean Squared Error.
3. Mean Squared Error.

The regressor which gave the least values for these metrics was chosen for prediction purposes.

2.5 PREDICTION

The predictions were made based on the chosen regressor. The test model spreadsheet was imported and grouped following the same process as for the first spreadsheet such that the same input features in the training phase was used for the test phase.

The results were then exported as an excel file called 'FinalResult.csv contained in the submission zip file.

CHAPTER 3

RESULTS

This section of the report contains the results obtained from the exploratory data analysis performed on the dataset along with the results obtained from comparing different machine learning algorithms and their relative effectiveness in predicting the oil production rate from test dataset. The Mean Average Error, Squared Average Error and the Root Mean error were used as the metrics in gauging the effectiveness of each model.

This section is going to be in three parts. The first part would show the visualization of the different parameters and their possible explanations, the second part would show the results of the regression analysis while the last part contained the conclusion based on the results obtained.

3.1 DATA VISULIZATION

Data visualization helps to show the relationship between different variables in a data analysis study. In this project, the visualization was done using the Matplotlib library in Python. An extensive use of scatter plots was employed to show the relative correlations between pairs of variables.

Plot of Volume of proppant against liquid produced

There seems to be a direct relationship between volume of proppant and the amount of liquid produced from a well. This makes sense considering the fracking process and its dependence on using appropriate proppants.

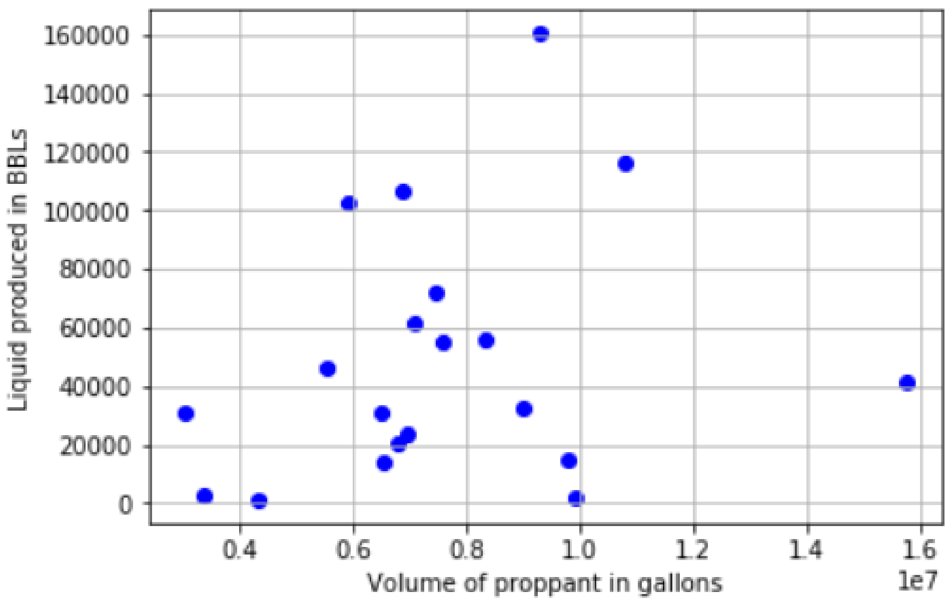


Figure 3 Plot of volume of proppant against liquid produced.

Plot of Volume of proppant against gas produced

A similar linear trend is seen with gas production from a fracked well. This shouldn't come as a surprise since gas is usually produced with liquid in an oil/gas well.

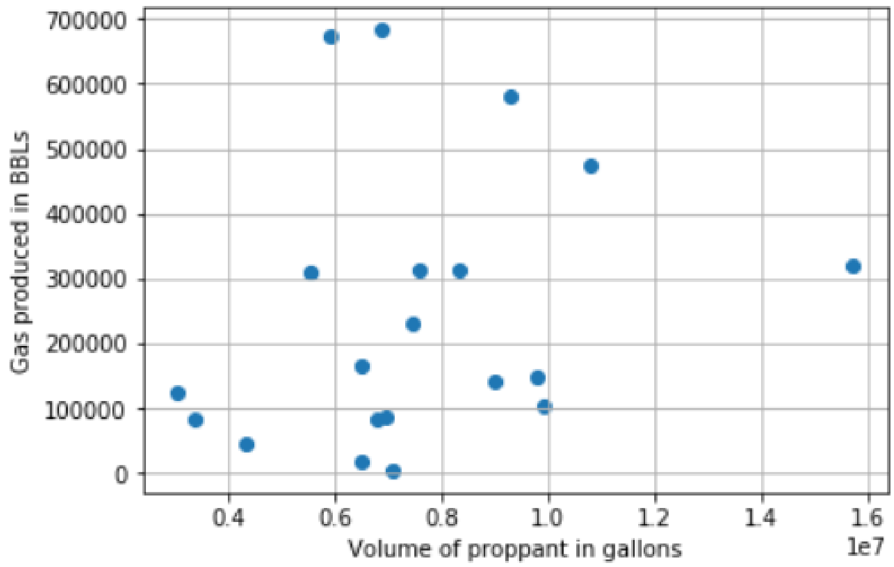


Figure 4 Plot of volume of proppant against gas produced.

Plot of True vertical depth against liquid produced

The True vertical depth of a well is the vertical distance a well is drilled before the commencement of directional drilling. It has a linear relationship with the amount of liquid produced.

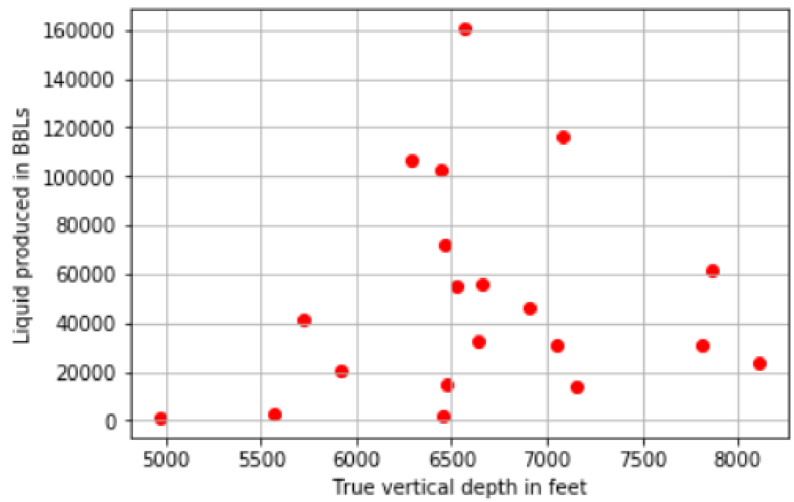


Figure 5 Plot of TVD against liquid produced.

Plot of Upper penetration length against liquid produced

There exists a linear relationship between the upper penetration depth and the amount of liquid produced.

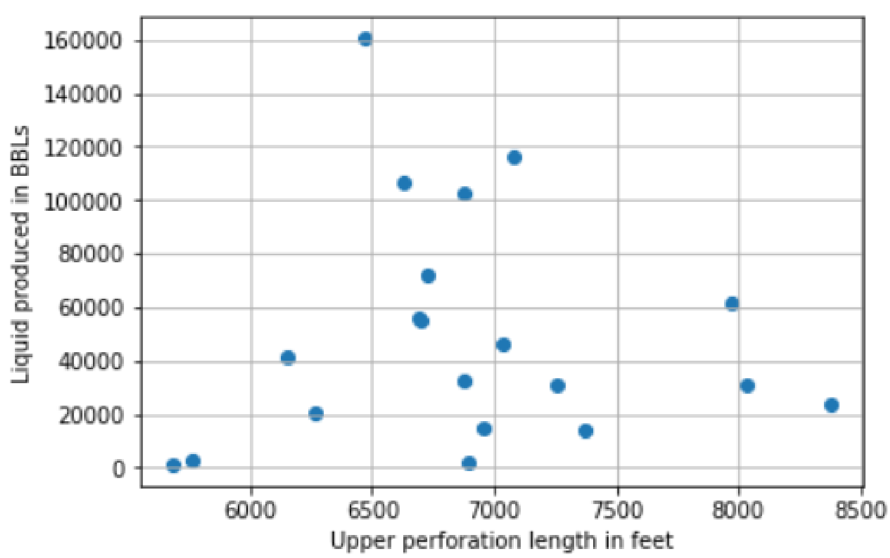


Figure 6 Plot of upper penetration against liquid produced.

Plot of Average fracking pressure against liquid produced

A linear relationship is seen between the average fracking pressure and the amount of liquid produced. A possible explanation for this could be seen in the fact that the greater the pressure, the more readily the rocks open, leading to flow of oil and gas.

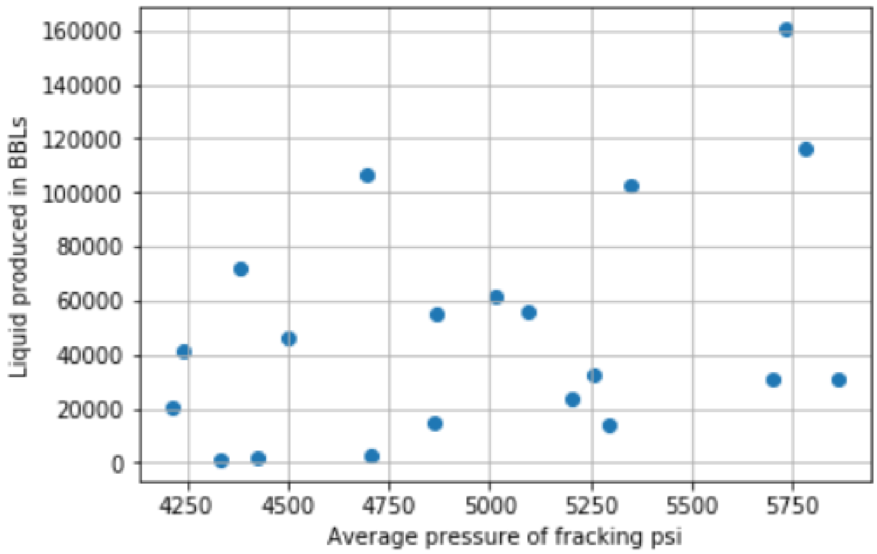


Figure 7 Plot of average fracking pressure against liquid produced.

Plot of Well horizontal length against liquid produced

The longer the horizontal well, which possibly translates to more frack stages, the more the liquid produced.

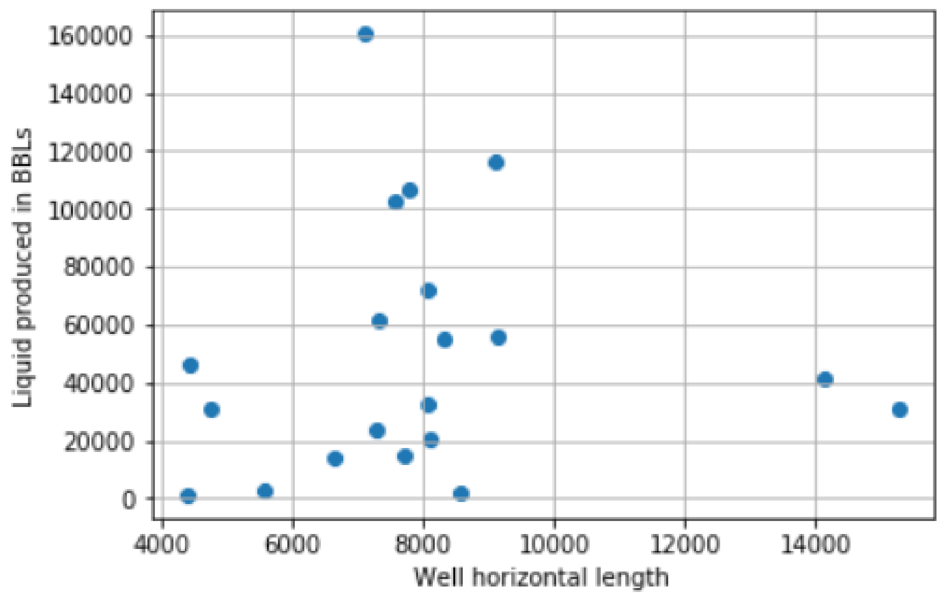


Figure 8 Plot of well horizontal length against liquid produced.

Plot of Mass of proppant against liquid produced

This plot shows a negative relationship between the mass of proppant used and the amount of liquid produced, even though it isn't a strong negative correlation.

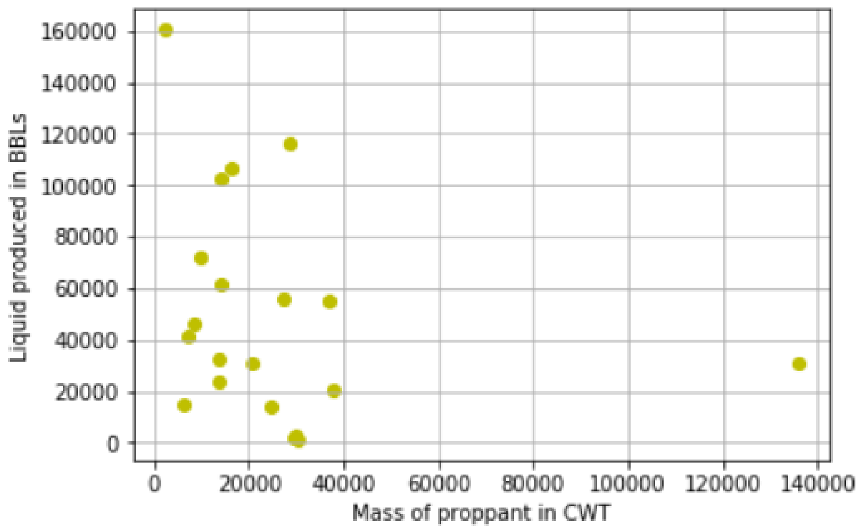


Figure 9 Plot of Mass of proppant against liquid produced.

3.2 REGRESSION ANALYSIS

The objective of this project was to predict the amount of oil produced from a well developed through multistage fracking. This prompted the need for regression tasks since we are trying to predict a future occurrence based on past data.

Four regression models were tested in this project- Linear regression, Support Vector Regression, Decision Tree Regression, and Multilayer perceptron regression.

The tables below show the performance of the different regression algorithms on the dataset.

LINEAR REGRESSION

MAE	**60307.87**
MSE	**6557027948.81**
RMSE	**80975.47**

SUPPORT VECTOR MACHINES REGRESSION

MAE	**55086.97**
MSE	**4429221587.65**
RMSE	**66552.39**

DECISION TREE REGRESSION

MAE	**56939.0**
MSE	**5503521880.7**
RMSE	**74185.725**

MULTILAYER PERCEPTRON REGRESSION

MAE	**36307.77**
MSE	**1822337212.23**
RMSE	**42688.84**

The following plots were obtained based on the predictions,

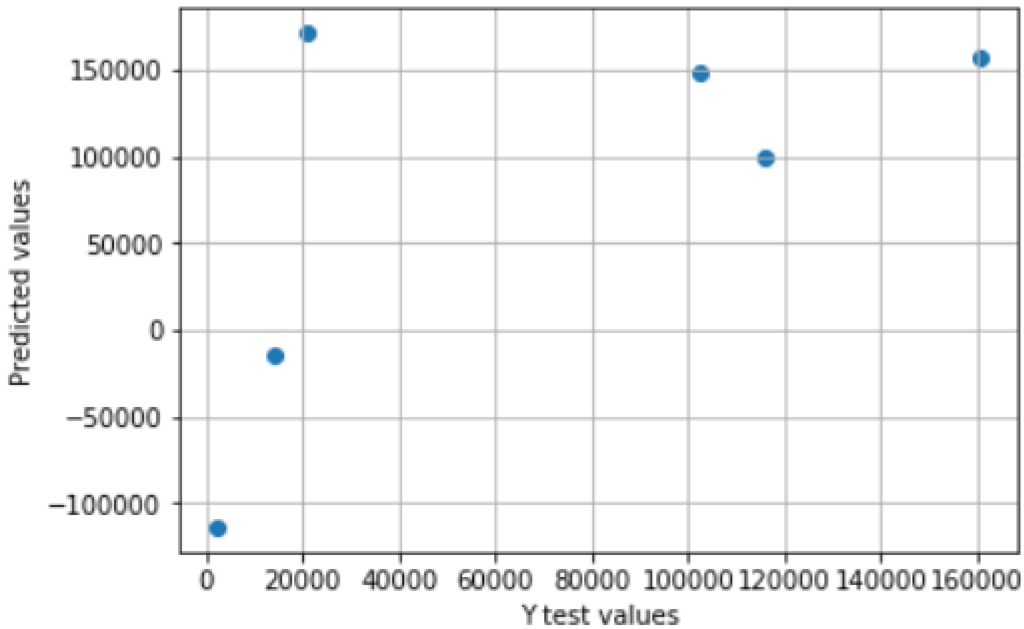


Figure 10 Linear regression plot.

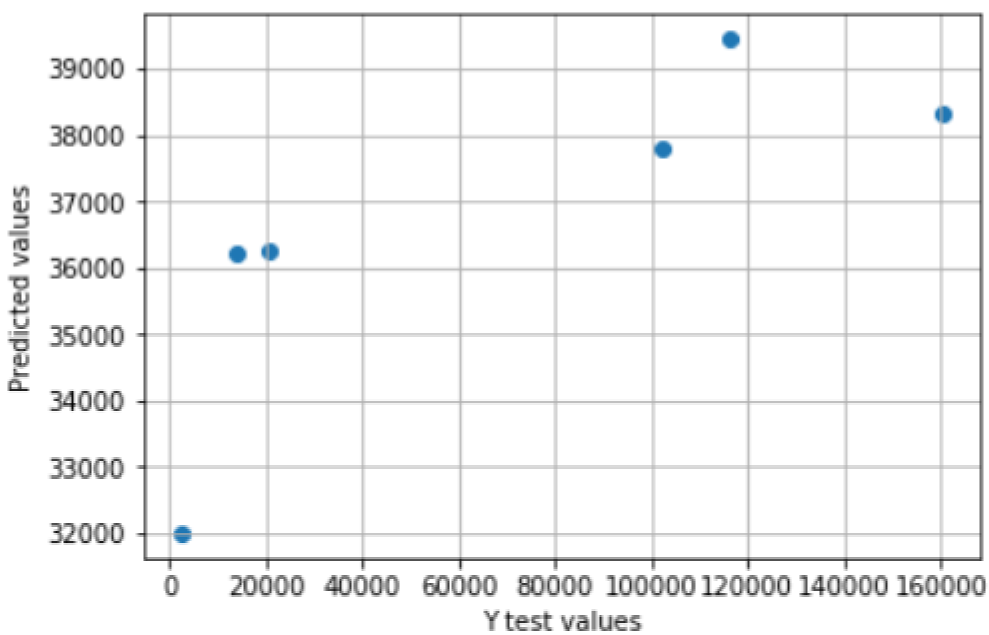


Figure 11 Support Vector Regreession plot

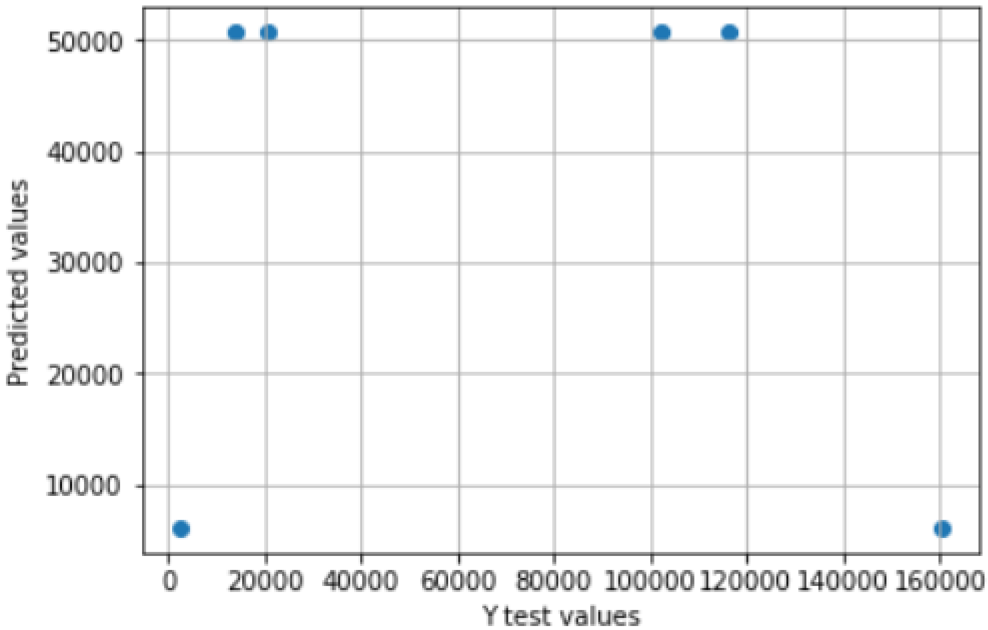


Figure 12 Decision tree Regression plot

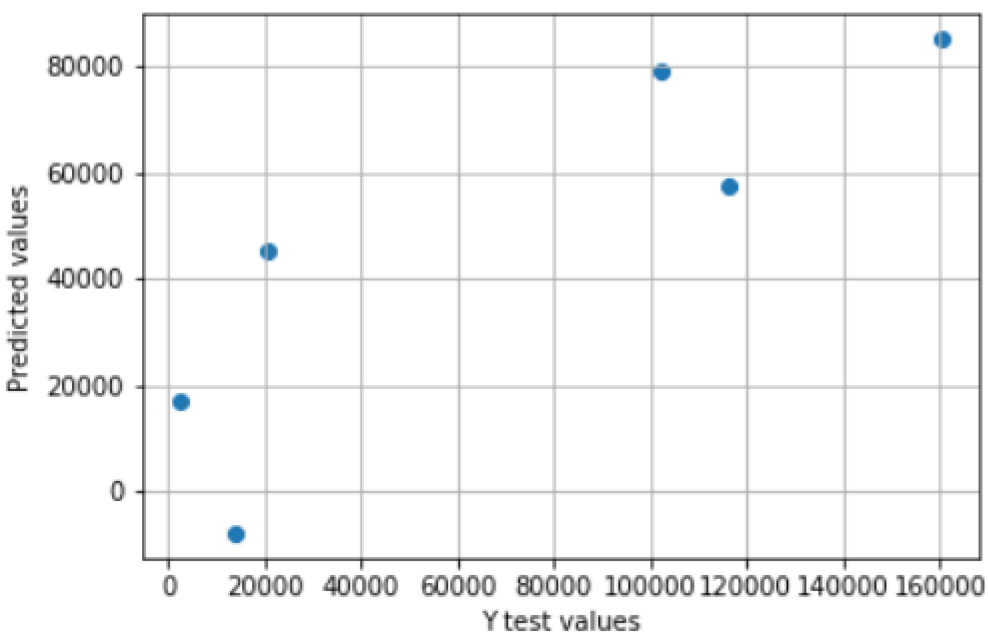


Figure 13 Multilayer Perceptron Regression plot

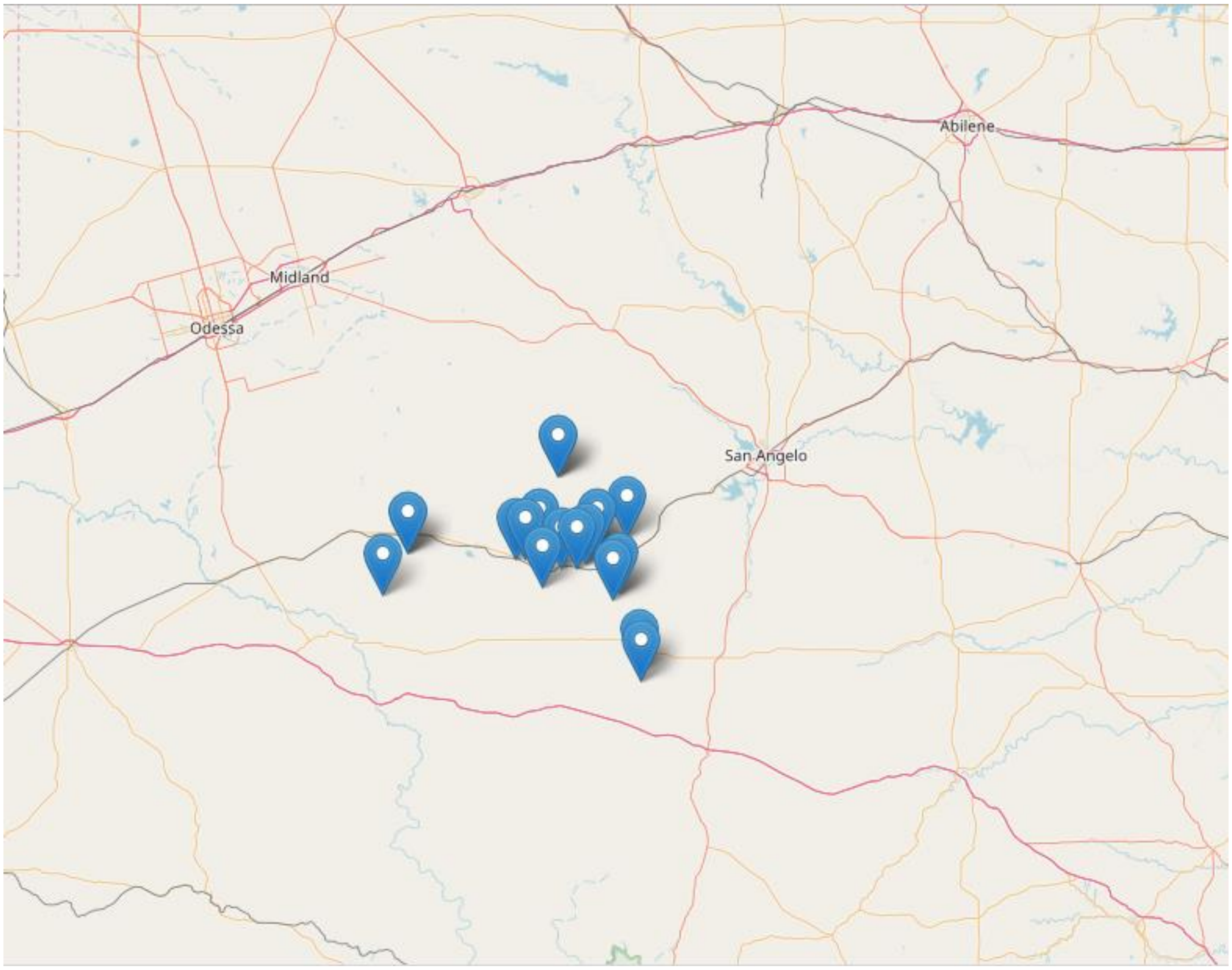
3.3 CONCLUSION

Based on the mean average error gotten for the different regression algorithms, the multilayer perceptron was selected as the model for the prediction part of the exercise because it gave the least amount of error.

The multilayer perceptron was used to predict the oil production and the result is shown below.

LIQ_CUM_BBLS	Well ID
75964.97344	1
218326.8083	5
29174.45362	9
42287.29408	13
63675.48475	17
35751.01585	20
25599.43731	27

APPENDIX WELL LOCATION MAP



Wekks are located in the Permian basin near Midland Texas.

APPENDIX – PYTHON CODE

In [144]:
```
# Import the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from mpl_toolkits.mplot3d import Axes3D # Creating 3D plots
from sklearn.feature_selection import RFE # Recursive feature extraction
from sklearn.linear_model import LinearRegression # Linear regression
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler # Scaling data
from sklearn import metrics # Calculating accuracy metrics
from sklearn.svm import SVR # Support vector regressor
from sklearn.tree import DecisionTreeRegressor # Decision tree regressor

from sklearn.neural_network import MLPRegressor # Multilayer perceptron
```

In [145]:
```
# Import the training dataset
data = pd.read_excel('IntroEngDataScienceFinalProjectTrainingData.xlsx')
```

In [146]:
```
print(data.head())
print('-------------------------------------------------------------')
print(data.isnull().sum()) # Check for missing values
```

```
   WELL_ID         JOB_DESC_STAGING     PROPPANT_MESH_SIZE  \
0        2   Wolfcamp Day 1 Stage 5                  40/70
1        2   Wolfcamp Day 2 Stage 7                  40/70
2        2  Wolfcamp Day 4 Stage 16  Sand, White, 100 mesh
3        2   Wolfcamp Day 2 Stage 6                  40/70
4        2   Wolfcamp Day 2 Stage 7  Sand, White, 100 mesh

  PROPPANT_MESH_DESCRIPTION  PROPPANT_MASS_USED PROPPANT_MASS_UOM  \
0        Sand, White, 40/70                  72       CWT=132 lbs
1        Sand, White, 40/70                  78               CWT
2     Sand, White, 100 mesh                  99               CWT
3        Sand, White, 40/70                  75               CWT
4     Sand, White, 100 mesh                  77               CWT

   VOLUME_PUMPED_GALLONS  AVERAGE_STP AVERAGE_STP_UOM  FRACTURE_GRADIENT  \
0                 356493       4393.0             PSI               0.76
1                 483451       2287.0             PSI               0.76
2                 126599       3835.0             PSI               0.76
3                 356122       4417.0             PSI               0.76
4                 127084       2287.0             PSI               0.76

    ...    MIN_STP_UOM  MAX_STP MAX_STP_UOM  UPPER_PERF  LOWER_PERF  \
0   ...            PSI     7565         PSI        6151       14009
1   ...            PSI     5129         PSI        6151       14009
2   ...            PSI     6101         PSI        6151       14009
3   ...            PSI     5250         PSI        6151       14009
4   ...            PSI     5129         PSI        6151       14009

   TRUE_VERTICAL_DEPTH  WELL_HORZ_LENGTH  NET_PROD_DAYS LIQ_CUM_BBLS  GAS_CUM
0                 6888             14136            670        41307   320872
1                 6888             14136            670        41307   320872
2                 6888             14136            670        41307   320872
3                 6888             14136            670        41307   320872
4                 6888             14136            670        41307   320872

[5 rows x 28 columns]
---------------------------------------------------------------
WELL_ID                      0
JOB_DESC_STAGING             0
PROPPANT_MESH_SIZE           0
PROPPANT_MESH_DESCRIPTION    0
PROPPANT_MASS_USED           0
PROPPANT_MASS_UOM            0
```

```
VOLUME_PUMPED_GALLONS        0
AVERAGE_STP                  0
AVERAGE_STP_UOM              0
FRACTURE_GRADIENT            0
FRACTURE_GRADIENT_UOM        0
MD_MIDDLE_PERFORATION        0
MD_MIDDLE_PERFORATION_UOM    0
TVD_DEPTH                    0
TOP_DEPTH                    0
WELL_LATITUDE                0
WELL_LONGITUDE               0
MIN_STP                      7
MIN_STP_UOM                  0
MAX_STP                      0
MAX_STP_UOM                  0
UPPER_PERF                   0
LOWER_PERF                   0
TRUE_VERTICAL_DEPTH          0
WELL_HORZ_LENGTH             0
NET_PROD_DAYS                0
LIQ_CUM_BBLS                 0
GAS_CUM                      0
dtype: int64
```

Basic Statistics and data cleaning.

In [147]:
```
# show statistics
print(data.describe())
```

	WELL_ID	PROPPANT_MASS_USED	VOLUME_PUMPED_GALLONS	AVERAGE_STP	\
count	1179.000000	1179.000000	1179.000000	1179.000000	
mean	13.569126	432.751484	128396.463953	4982.507634	
std	7.421526	465.101241	88212.514668	853.011956	
min	2.000000	1.000000	528.000000	76.000000	
25%	7.000000	189.000000	50259.500000	4431.000000	
50%	14.000000	330.000000	100477.000000	4942.000000	
75%	21.000000	538.500000	187370.000000	5497.000000	
max	26.000000	2292.000000	483451.000000	7778.000000	

	FRACTURE_GRADIENT	MD_MIDDLE_PERFORATION	TVD_DEPTH	TOP_DEPTH	\
count	1179.000000	1179.000000	1179.000000	1179.000000	
mean	0.760814	10468.521628	6642.436811	14176.510602	
std	0.019168	2289.698857	727.062446	1543.329280	
min	0.720000	5528.000000	4972.000000	9395.000000	
25%	0.750000	8662.000000	6439.000000	14027.000000	
50%	0.760000	10142.000000	6526.000000	14182.000000	
75%	0.760000	12303.500000	7083.000000	15216.000000	
max	0.800000	16011.000000	8108.000000	16186.000000	

	WELL_LATITUDE	WELL_LONGITUDE	MIN_STP	MAX_STP	UPPER_PERF	\
count	1179.000000	1179.000000	1172.000000	1179.000000	1179.000000	
mean	31.125462	-101.196234	3759.523891	6148.922816	6894.980492	
std	0.132159	0.255270	1622.044311	1293.874605	655.841823	
min	30.780850	-101.790470	9.000000	5.000000	5684.000000	
25%	31.060160	-101.240560	2957.000000	5259.500000	6628.000000	
50%	31.146660	-101.161500	3660.000000	6133.000000	6879.000000	
75%	31.173380	-101.034220	4490.000000	7162.500000	7082.000000	
max	31.396560	-100.884410	32641.000000	9160.000000	8374.000000	

	LOWER_PERF	TRUE_VERTICAL_DEPTH	WELL_HORZ_LENGTH	NET_PROD_DAYS	\
count	1179.000000	1179.000000	1179.000000	1179.000000	
mean	14022.507209	6676.299406	8236.301103	830.296862	
std	1492.878692	646.244689	2565.688023	294.704759	
min	9338.000000	5002.000000	4393.000000	549.000000	
25%	13887.000000	6475.000000	7263.000000	608.000000	
50%	14082.000000	6564.000000	7780.000000	731.000000	
75%	14984.000000	6916.000000	8579.000000	944.000000	
max	16090.000000	8119.000000	15264.000000	1522.000000	

	LIQ_CUM_BBLS	GAS_CUM
count	1179.000000	1179.000000
mean	52588.503817	258631.452926

```
std     43678.050335  211083.185502
min      1243.000000    4801.000000
25%     20888.000000   85831.000000
50%     41307.000000  165820.000000
75%     72357.000000  320872.000000
max    160458.000000  682957.000000
```

In [148]:

```python
# drop rows with missing values
data.dropna(inplace = True,axis = 0 )
# Check for missing values
print(data.isnull().sum())
```

```
WELL_ID                        0
JOB_DESC_STAGING               0
PROPPANT_MESH_SIZE             0
PROPPANT_MESH_DESCRIPTION      0
PROPPANT_MASS_USED             0
PROPPANT_MASS_UOM              0
VOLUME_PUMPED_GALLONS          0
AVERAGE_STP                    0
AVERAGE_STP_UOM                0
FRACTURE_GRADIENT              0
FRACTURE_GRADIENT_UOM          0
MD_MIDDLE_PERFORATION          0
MD_MIDDLE_PERFORATION_UOM      0
TVD_DEPTH                      0
TOP_DEPTH                      0
WELL_LATITUDE                  0
WELL_LONGITUDE                 0
MIN_STP                        0
MIN_STP_UOM                    0
MAX_STP                        0
MAX_STP_UOM                    0
UPPER_PERF                     0
LOWER_PERF                     0
TRUE_VERTICAL_DEPTH            0
WELL_HORZ_LENGTH               0
NET_PROD_DAYS                  0
LIQ_CUM_BBLS                   0
GAS_CUM                        0
dtype: int64
```

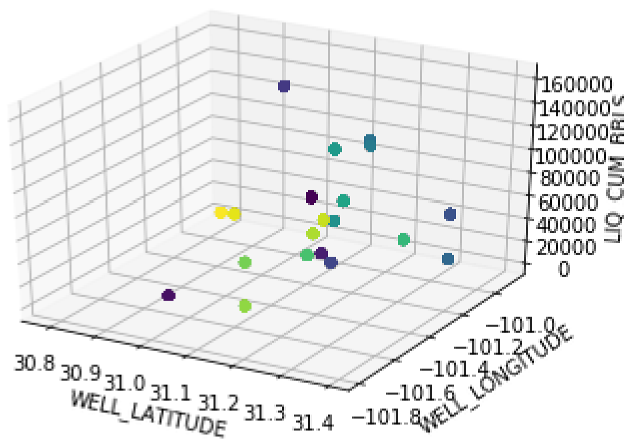
In [149]:
```
print('-------------------------------------------------------------------------------------------')
# print a list of column names
print(data.columns.tolist())
```

```
-------------------------------------------------------------------------------------
['WELL_ID', 'JOB_DESC_STAGING', 'PROPPANT_MESH_SIZE', 'PROPPANT_MESH_DESCRIPTION', 'PROPPANT_MASS_USED', 'PRO
PPANT_MASS_UOM', 'VOLUME_PUMPED_GALLONS', 'AVERAGE_STP', 'AVERAGE_STP_UOM', 'FRACTURE_GRADIENT', 'FRACTURE_GR
ADIENT_UOM', 'MD_MIDDLE_PERFORATION', 'MD_MIDDLE_PERFORATION_UOM', 'TVD_DEPTH', 'TOP_DEPTH', 'WELL_LATITUDE',
'WELL_LONGITUDE', 'MIN_STP', 'MIN_STP_UOM', 'MAX_STP', 'MAX_STP_UOM', 'UPPER_PERF', 'LOWER_PERF', 'TRUE_VERTI
CAL_DEPTH', 'WELL_HORZ_LENGTH', 'NET_PROD_DAYS', 'LIQ_CUM_BBLS', 'GAS_CUM']
```

In [150]:
```
# Remove all text columns from the dataset
data=data.drop(labels=['JOB_DESC_STAGING','PROPPANT_MESH_DESCRIPTION','PROPPANT_MASS_UOM',
          'AVERAGE_STP_UOM','FRACTURE_GRADIENT_UOM','MD_MIDDLE_PERFORATION_UOM','MIN_STP_UOM',
          'MAX_STP_UOM'], axis = 1)
```

Some visualization before aggregation

In [151]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x =data['WELL_LATITUDE']
y =data['WELL_LONGITUDE']
z =data['LIQ_CUM_BBLS']

ax.scatter(x, y, z, c=data['WELL_ID'], marker='o')

ax.set_xlabel('WELL_LATITUDE')
ax.set_ylabel('WELL_LONGITUDE')
ax.set_zlabel('LIQ_CUM_BBLS')
#plt.savefig(fname = '3Dimage')
plt.show()
```



Aggregate the data by the well ID using the 'groupby' function in pandas

In [152]:
```
# group the data by the well ID
Grouped_data =data.groupby('WELL_ID', as_index = False)
# sums all the properties by well ID
Grouped_data=Grouped_data.sum()
print(Grouped_data.head())
print('----------------------------------------------------------------------------------')
```

```
   WELL_ID  PROPPANT_MASS_USED  VOLUME_PUMPED_GALLONS  AVERAGE_STP  \
0        2                7103               15727612     279788.0   
1        3              135750                6519843     342126.0   
2        4                6022                9786752     330502.0   
3        6                2346                9280987     389917.5   
4        7               29268                9916132     318670.0   

   FRACTURE_GRADIENT  MD_MIDDLE_PERFORATION  TVD_DEPTH  TOP_DEPTH  \
0              50.16               666938.0     377982     932976   
1              45.60               696122.0     468540     915840   
2              51.00               700620.0     440300     964376   
3              51.00               682175.0     446420     929152   
4              54.00               782251.0     464832    1082520   

   WELL_LATITUDE  WELL_LONGITUDE   MIN_STP  MAX_STP  UPPER_PERF  LOWER_PERF  \
0     2047.64076     -6664.56714  233178.0   427016      405966      924594   
1     1862.19480     -6107.42820  204082.0   425172      481860      906780   
2     2115.99952     -6878.98200  258226.0   412936      473212      957576   
3     2112.09088     -6883.36392  356618.0   423198      439960      921536   
4     2240.61408     -7279.88256  241876.0   391446      496224     1068264   

   TRUE_VERTICAL_DEPTH  WELL_HORZ_LENGTH  NET_PROD_DAYS  LIQ_CUM_BBLS  \
0               454608            932976          44220       2726262   
1               413280            915840          36480       1866480   
2               440300            524076         103496       1026800   
3               446352            482800          49708      10911144   
4               464832            617688         100800        146808   

    GAS_CUM  
0  21177552  
1   1047180  
2  10210540  
3  39474476  
4   7535880  
----------------------------------------------------------------------------------
```

In [153]:
```
print(Grouped_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20 entries, 0 to 19
Data columns (total 19 columns):
WELL_ID                  20 non-null int64
PROPPANT_MASS_USED       20 non-null int64
VOLUME_PUMPED_GALLONS    20 non-null int64
AVERAGE_STP              20 non-null float64
FRACTURE_GRADIENT        20 non-null float64
MD_MIDDLE_PERFORATION    20 non-null float64
TVD_DEPTH                20 non-null int64
TOP_DEPTH                20 non-null int64
WELL_LATITUDE            20 non-null float64
WELL_LONGITUDE           20 non-null float64
MIN_STP                  20 non-null float64
MAX_STP                  20 non-null int64
UPPER_PERF               20 non-null int64
LOWER_PERF               20 non-null int64
TRUE_VERTICAL_DEPTH      20 non-null int64
WELL_HORZ_LENGTH         20 non-null int64
NET_PROD_DAYS            20 non-null int64
LIQ_CUM_BBLS             20 non-null int64
GAS_CUM                  20 non-null int64
dtypes: float64(6), int64(13)
memory usage: 3.1 KB
None
```

The entire dataset has been grouped into 20 wells based on the well IDS present. This aggregation was done by summing all the columns per well id. This process is valid for quantities such as Volume of proppant, mass of proppant etc but is meaningless for quantities such as pressure, longitude, fracture gradient etc. In order to solve this, the mean values would be calculated for some other columns instead while the sum would be used for others on a case by case basis.

In [154]:
```
# The number of rows per well id needs to be computed,
#this value would then be used to compute the mean value.
```

The function below was created by Olabode Alamu to count the number of rows that have info for a well id.

In [155]:
```
"""
This function was created to count the number of rows in the dataset
which partains to a particular well ID number.
it takes in the dataframe of interest as input, counts the number of rows per well id
and returns a dataframe with the number of rows per well ID as output
"""
def rowcount(dataframe):
    Unique = dataframe['WELL_ID'].unique() # Checks for the unique well IDs
    length_list = []
    # slices through the dataframe till only unique Well ids are found and counted
    for i in Unique:
        length=len(dataframe[dataframe['WELL_ID']== i])
        length_list.append(length) # appends the count to the list

    # pass into a dataframe
    Count = pd.DataFrame(data= length_list, columns = ['No of rows'])
    Count['Well ID']= Unique
    return Count
```

In [156]:
```
# the dataframe was passed into the function
Count=rowcount(data)
```

In [157]:
```
print(Count)
```

```
    No of rows  Well ID
0           66        2
1           60        3
2           68        4
3           68        6
4           72        7
5           52        8
6           51       10
7           65       11
8           61       12
9           70       14
10          76       15
11          66       16
12          54       18
13          38       19
14          60       21
15          58       22
16          35       23
17          64       24
18          44       25
19          44       26
```

Next we compute the mean value for some of the columns

In [158]:
```
# Create a column in the Grouped data with the number of rows
Grouped_data['Count'] = Count['No of rows']
```

In [159]:
```
print(Grouped_data.head())
```

```
   WELL_ID  PROPPANT_MASS_USED  VOLUME_PUMPED_GALLONS  AVERAGE_STP  \
0        2                7103               15727612     279788.0
1        3              135750                6519843     342126.0
2        4                6022                9786752     330502.0
3        6                2346                9280987     389917.5
4        7               29268                9916132     318670.0

   FRACTURE_GRADIENT  MD_MIDDLE_PERFORATION  TVD_DEPTH  TOP_DEPTH  \
0              50.16               666938.0     377982     932976
1              45.60               696122.0     468540     915840
2              51.00               700620.0     440300     964376
3              51.00               682175.0     446420     929152
4              54.00               782251.0     464832    1082520

   WELL_LATITUDE  WELL_LONGITUDE   MIN_STP  MAX_STP  UPPER_PERF  LOWER_PERF  \
0     2047.64076     -6664.56714  233178.0   427016      405966      924594
1     1862.19480     -6107.42820  204082.0   425172      481860      906780
2     2115.99952     -6878.98200  258226.0   412936      473212      957576
3     2112.09088     -6883.36392  356618.0   423198      439960      921536
4     2240.61408     -7279.88256  241876.0   391446      496224     1068264

   TRUE_VERTICAL_DEPTH  WELL_HORZ_LENGTH  NET_PROD_DAYS  LIQ_CUM_BBLS  \
0               454608            932976          44220       2726262
1               413280            915840          36480       1866480
2               440300            524076         103496       1026800
3               446352            482800          49708      10911144
4               464832            617688         100800        146808

    GAS_CUM  Count
0  21177552     66
1   1047180     60
2  10210540     68
3  39474476     68
4   7535880     72
```

The count column would then be used to compute the mean values for ['GAS_CUM','LIQ_CUM_BBLS','NET_PROD_DAYS','WELL_HORZ_LENGTH', 'TRUE_VERTICAL_DEPTH','LOWER_PERF','UPPER_PERF','MAX_STP', 'MIN_STP','WELL_LONGITUDE','WELL_LATITUDE', 'TOP_DEPTH','TVD_DEPTH','MD_MIDDLE_PERFORATION', 'FRACTURE_GRADIENT','AVERAGE_STP']

In [160]:
```
# This would be achieved using a custom function
```

In [161]:
```
def mean_calculator(dataframe,new_column_names,old_column_names):
    """
    This function takes in a dataframe and creates new columns based
    on the calculated mean of the previous columns. it requires a list of the new column names and
    a list of the old column names which require a mean to be computed.
    """

    for i,j in zip(new_column_names,old_column_names):
        dataframe[i]= dataframe[j]/dataframe['Count']
```

In [162]:
```
# list of columns which the mean to be computed
old_column_names = ['GAS_CUM','LIQ_CUM_BBLS','NET_PROD_DAYS','WELL_HORZ_LENGTH','TRUE_VERTICAL_DEPTH'
                   ,'LOWER_PERF','UPPER_PERF','MAX_STP','MIN_STP','WELL_LONGITUDE','WELL_LATITUDE',
                   'TOP_DEPTH','TVD_DEPTH','MD_MIDDLE_PERFORATION','FRACTURE_GRADIENT','AVERAGE_STP']
```

In [163]:
```
# new column names that would be created on the dataframe
new_column_names=['Mean Gas_cum','Mean Liquid produced','Mean Production days','Mean Horizontal length',
 'Mean True Vertical Distance','Mean Lower perforation','Mean Upper perforation','Mean Maximum STP',
'Mean Minimum STP','Longitude','Latitude','Mean TOP Depth','Mean TVD depth','Mean Mid perforation',
 'Mean Fracture Gradient','Mean STP']
```

In [164]:
```
# call the mean calculator custom function
mean_calculator(dataframe=Grouped_data,
                new_column_names=new_column_names,old_column_names=old_column_names)
```

In [165]:
```
print(Grouped_data.head())
```

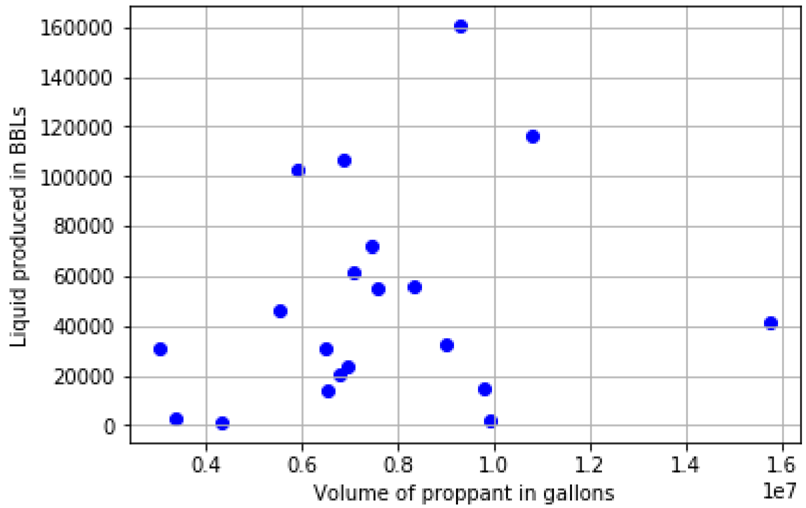
```
   WELL_ID  PROPPANT_MASS_USED  VOLUME_PUMPED_GALLONS  AVERAGE_STP  \
0        2                7103               15727612     279788.0   
1        3              135750                6519843     342126.0   
2        4                6022                9786752     330502.0   
3        6                2346                9280987     389917.5   
4        7               29268                9916132     318670.0   

   FRACTURE_GRADIENT  MD_MIDDLE_PERFORATION  TVD_DEPTH  TOP_DEPTH  \
0              50.16               666938.0     377982     932976   
1              45.60               696122.0     468540     915840   
2              51.00               700620.0     440300     964376   
3              51.00               682175.0     446420     929152   
4              54.00               782251.0     464832    1082520   

   WELL_LATITUDE  WELL_LONGITUDE     ...       Mean Upper perforation  \
0     2047.64076     -6664.56714     ...                       6151.0   
1     1862.19480     -6107.42820     ...                       8031.0   
2     2115.99952     -6878.98200     ...                       6959.0   
3     2112.09088     -6883.36392     ...                       6470.0   
4     2240.61408     -7279.88256     ...                       6892.0   

   Mean Maximum STP  Mean Minimum STP  Longitude  Latitude  Mean TOP Depth  \
0       6469.939394       3533.000000 -100.97829  31.02486         14136.0   
1       7086.200000       3401.366667 -101.79047  31.03658         15264.0   
2       6072.588235       3797.441176 -101.16150  31.11764         14182.0   
3       6223.500000       5244.382353 -101.22594  31.06016         13664.0   
4       5436.750000       3359.388889 -101.10948  31.11964         15035.0   

   Mean TVD depth  Mean Mid perforation  Mean Fracture Gradient     Mean STP  
0          5727.0          10105.121212                    0.76  4239.212121  
1          7809.0          11602.033333                    0.76  5702.100000  
2          6475.0          10303.235294                    0.75  4860.323529  
3          6565.0          10031.985294                    0.75  5734.080882  
4          6456.0          10864.597222                    0.75  4425.972222  

[5 rows x 36 columns]
```

In [166]:
```
# Create list of columns that were the sum of properties, remove them and replace with the mean property
remove_columns = ['AVERAGE_STP','FRACTURE_GRADIENT','MD_MIDDLE_PERFORATION','TVD_DEPTH',
                  'TOP_DEPTH','WELL_LATITUDE','WELL_LONGITUDE','MIN_STP','MAX_STP','UPPER_PERF',
                  'LOWER_PERF','TRUE_VERTICAL_DEPTH','WELL_HORZ_LENGTH','NET_PROD_DAYS','LIQ_CUM_BBLS'
                  ,'GAS_CUM']

# Drop some more columns
Grouped_data=Grouped_data.drop(labels=remove_columns, axis = 1)
```

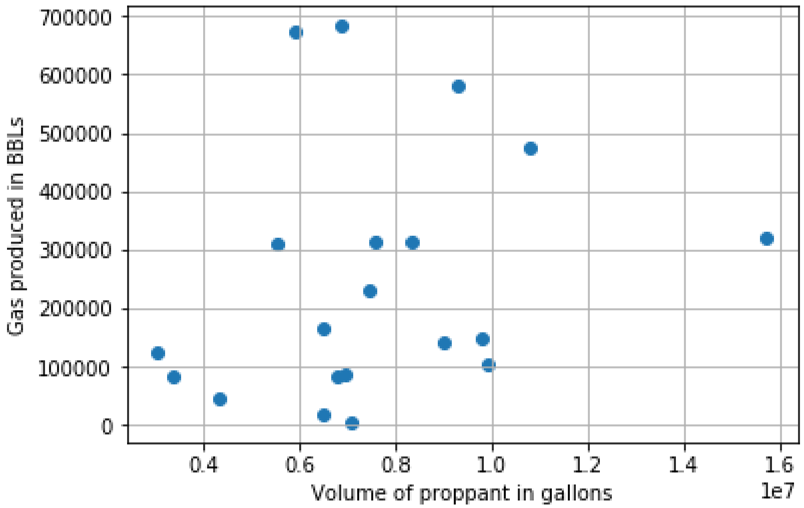
In [167]:
```
print(Grouped_data.head())
```

```
   WELL_ID  PROPPANT_MASS_USED  VOLUME_PUMPED_GALLONS  Count  Mean Gas_cum  \
0        2                7103               15727612     66      320872.0
1        3              135750                6519843     60       17453.0
2        4                6022                9786752     68      150155.0
3        6                2346                9280987     68      580507.0
4        7               29268                9916132     72      104665.0

   Mean Liquid produced  Mean Production days  Mean Horizontal length  \
0               41307.0                 670.0                 14136.0
1               31108.0                 608.0                 15264.0
2               15100.0                1522.0                  7707.0
3              160458.0                 731.0                  7100.0
4                2039.0                1400.0                  8579.0

   Mean True Vertical Distance  Mean Lower perforation  \
0                       6888.0                 14009.0
1                       6888.0                 15113.0
2                       6475.0                 14082.0
3                       6564.0                 13552.0
4                       6456.0                 14837.0

   Mean Upper perforation  Mean Maximum STP  Mean Minimum STP  Longitude  \
0                  6151.0       6469.939394       3533.000000 -100.97829
1                  8031.0       7086.200000       3401.366667 -101.79047
2                  6959.0       6072.588235       3797.441176 -101.16150
3                  6470.0       6223.500000       5244.382353 -101.22594
4                  6892.0       5436.750000       3359.388889 -101.10948

   Latitude  Mean TOP Depth  Mean TVD depth  Mean Mid perforation  \
0  31.02486         14136.0          5727.0          10105.121212
1  31.03658         15264.0          7809.0          11602.033333
2  31.11764         14182.0          6475.0          10303.235294
3  31.06016         13664.0          6565.0          10031.985294
4  31.11964         15035.0          6456.0          10864.597222

   Mean Fracture Gradient     Mean STP
0                    0.76  4239.212121
1                    0.76  5702.100000
2                    0.75  4860.323529
3                    0.75  5734.080882
4                    0.75  4425.972222
```

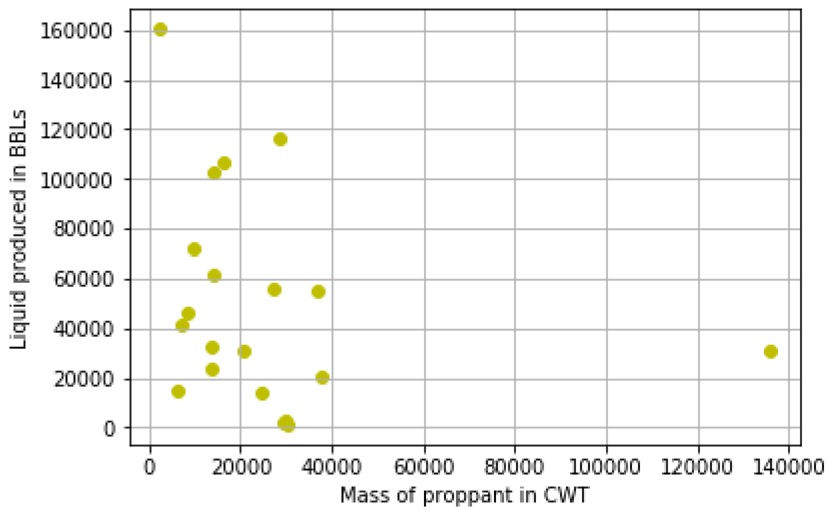
In [168]:
```
plt.scatter(x= Grouped_data['VOLUME_PUMPED_GALLONS'], y = Grouped_data['Mean Liquid produced'],c='b')
plt.ylabel('Liquid produced in BBLs')
plt.xlabel('Volume of proppant in gallons')
plt.grid()
plt.show()
```



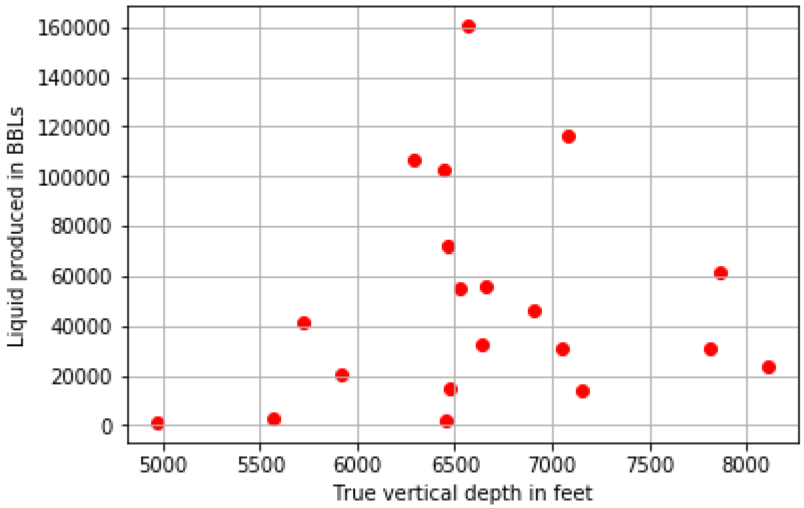
In [169]:
```
plt.scatter(x= Grouped_data['VOLUME_PUMPED_GALLONS'], y = Grouped_data['Mean Gas_cum'])
plt.ylabel('Gas produced in BBLs')
plt.xlabel('Volume of proppant in gallons')
plt.grid()
plt.show()
```



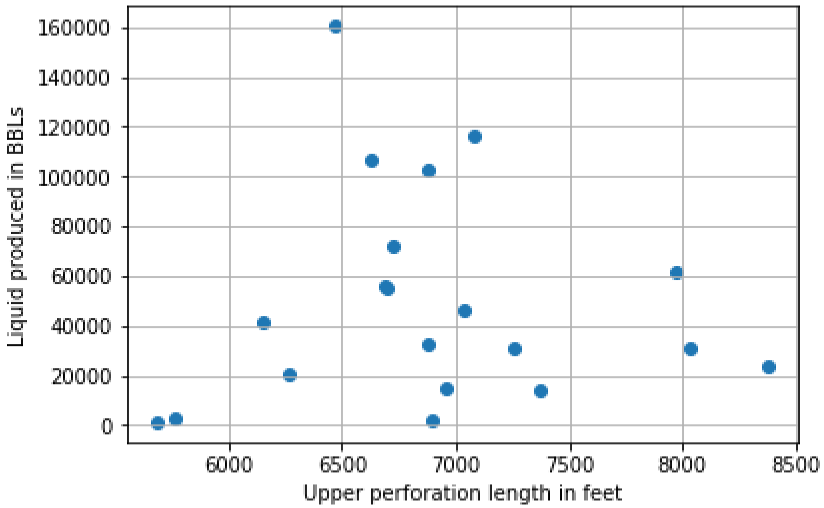
In [170]:
```
plt.scatter(x= Grouped_data['PROPPANT_MASS_USED'], y = Grouped_data['Mean Liquid produced'], c='y')
plt.ylabel('Liquid produced in BBLs')
plt.xlabel('Mass of proppant in CWT')
plt.grid()
plt.show()
```



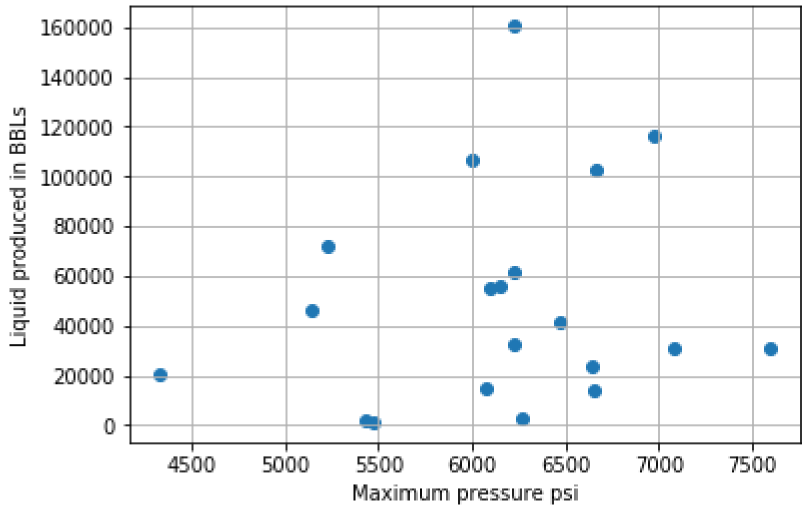
In [171]:
```
plt.scatter(x= Grouped_data['Mean TVD depth'], y = Grouped_data['Mean Liquid produced'], c = 'r')
plt.ylabel('Liquid produced in BBLs')
plt.xlabel('True vertical depth in feet')
plt.grid()
plt.show()
```



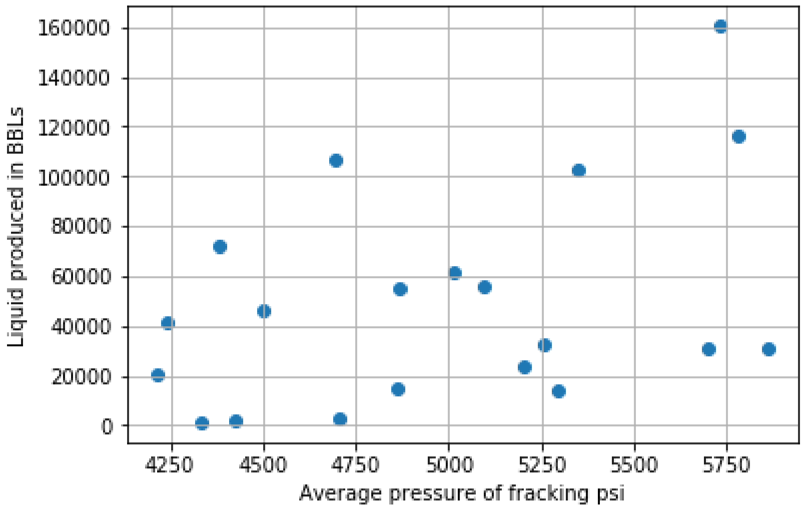
In [172]:
```
plt.scatter(x= Grouped_data['Mean Upper perforation'], y = Grouped_data['Mean Liquid produced'])
plt.ylabel('Liquid produced in BBLs')
plt.xlabel('Upper perforation length in feet')
plt.grid()
plt.show()
```



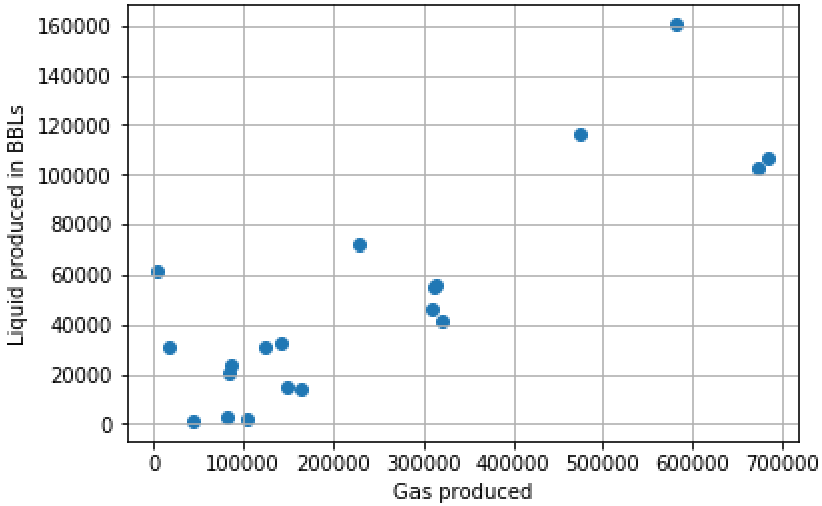
In [173]:
```
plt.scatter(x= Grouped_data['Mean Maximum STP'], y = Grouped_data['Mean Liquid produced'])
plt.ylabel('Liquid produced in BBLs')
plt.xlabel('Maximum pressure psi')
plt.grid()
plt.show()
```



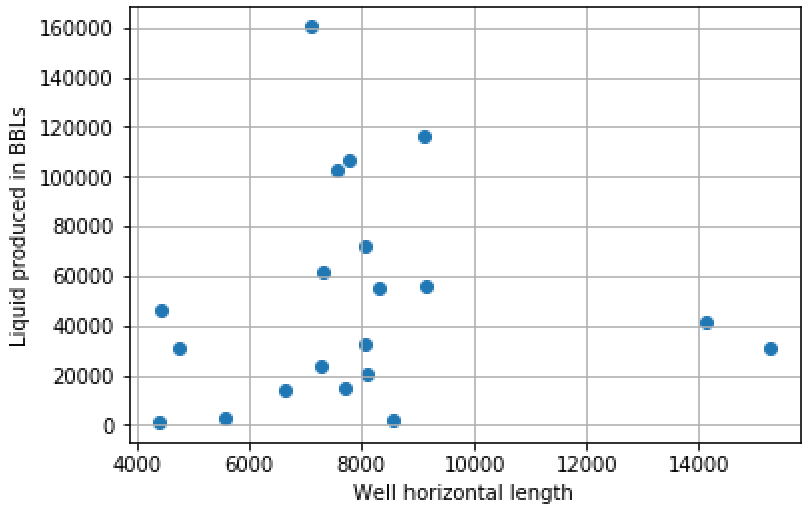
In [174]:
```
plt.scatter(x= Grouped_data['Mean STP'], y = Grouped_data['Mean Liquid produced'])
plt.ylabel('Liquid produced in BBLs')
plt.xlabel('Average pressure of fracking psi')
plt.grid()
plt.show()
```



In [175]:
```
plt.scatter(x= Grouped_data['Mean Gas_cum'], y = Grouped_data['Mean Liquid produced'])
plt.ylabel('Liquid produced in BBLs')
plt.xlabel('Gas produced')
plt.grid()
plt.show()
```



In [176]:
```
plt.scatter(x= Grouped_data['Mean Horizontal length'], y = Grouped_data['Mean Liquid produced'])
plt.ylabel('Liquid produced in BBLs')
plt.xlabel('Well horizontal length')
plt.grid()
plt.show()
```



In [ ]:

The dataframe is in the required format, we can continue with the analysis.

The exported lat and long data would be used to create the basin location map

In [177]:
```
#Export the longitude and latitude data
long_lat =Grouped_data[['WELL_ID','Latitude','Longitude',
                        'Mean Liquid produced','Mean Production days']]
# export to a csv
long_lat.to_csv('long_lat.csv')
```

In [178]:
```
# Create target variable
y = Grouped_data['Mean Liquid produced']

# These features are dropped because they are repetitve
dropoff = ['Mean Gas_cum','Count','WELL_ID','Latitude','Longitude',
           'Mean Liquid produced','Mean True Vertical Distance']

# Create input features
X=Grouped_data.drop(labels=dropoff, axis = 1)
```

Feature selection was perfromed using the recursive feature extraction model in scikit learn

The recursive feature extraction model requires an estimator, in this case, the linear regression model was chosen as the estimator since the objective of this project is that of a regression problem.

In [179]:
```
lm = LinearRegression()
# SPlit the data into training and test data

X_train_rfe, X_test_rfe, y_train_rfe, y_test_rfe = train_test_split(X,y, test_size = 0.3)
# create the RFE model and select 12 attributes out of 13 possible
rfe = RFE(estimator = lm, n_features_to_select=12, verbose=3)
rfe = rfe.fit(X_train_rfe, y_train_rfe)
# summarize the selection of the attributes
print(rfe.support_) # the parameters with True are selected
print('----------------------------------------------------------------')
print(rfe.ranking_)
```

```
Fitting estimator with 13 features.
[ True False  True  True  True  True  True  True  True  True  True  True
  True]
----------------------------------------------------------------
[1 2 1 1 1 1 1 1 1 1 1 1 1]
```

Display the selected features

In [180]:
```
# display important features
print(X.columns[rfe.support_])
```

```
Index(['PROPPANT_MASS_USED', 'Mean Production days', 'Mean Horizontal length',
       'Mean Lower perforation', 'Mean Upper perforation', 'Mean Maximum STP',
       'Mean Minimum STP', 'Mean TOP Depth', 'Mean TVD depth',
       'Mean Mid perforation', 'Mean Fracture Gradient', 'Mean STP'],
      dtype='object')
```

The selected features are ['PROPPANT_MASS_USED', 'Mean Production days', 'Mean Horizontal length', 'Mean Lower perforation', 'Mean Upper perforation', 'Mean Maximum STP', 'Mean Minimum STP', 'Mean TOP Depth', 'Mean TVD depth', 'Mean Mid perforation', 'Mean Fracture Gradient', 'Mean STP']

The 'VOLUME_PUMPED_GALLONS' was dropped since it is the least important feature acording to the analysis.

In [181]:
```
# The top 12 important features were selected
# They would be used to form the new X matrix
X = X[['PROPPANT_MASS_USED', 'Mean STP', 'Mean Fracture Gradient',
       'Mean Mid perforation', 'Mean TVD depth', 'Mean TOP Depth',
       'Mean Minimum STP', 'Mean Maximum STP', 'Mean Upper perforation',
       'Mean Lower perforation', 'Mean Horizontal length',
       'Mean Production days']]
```

Prepare the X matrix by Standardizing it.

In [182]:
```
# Standardize the data

Scaled = StandardScaler()
Scaled.fit(X)
Scaled.transform(X)
```

```
Out[182]: array([[ -6.65735159e-01,  -1.40243760e+00,  -2.45514308e-02,
                   -2.39941981e-01,  -1.20709006e+00,   1.17346359e-01,
                   -2.86831615e-01,   4.41713888e-01,  -1.09417273e+00,
                    1.30328633e-01,   2.33665043e+00,  -5.49961765e-01],
                 [  4.03865482e+00,   1.38202616e+00,  -2.45514308e-02,
                    1.19628562e+00,   1.56421887e+00,   7.87896983e-01,
                   -4.80285977e-01,   1.28504958e+00,   1.68937900e+00,
                    8.05834784e-01,   2.76370558e+00,  -7.71590815e-01],
                 [ -7.05265389e-01,  -2.20212884e-01,  -5.15580047e-01,
                   -4.98594043e-02,  -2.11442087e-01,   1.44691509e-01,
                    1.01803221e-01,  -1.02050245e-01,   1.02162271e-01,
                    1.74995253e-01,  -9.73367722e-02,   2.49565035e+00],
                 [ -8.39690114e-01,   1.44289864e+00,  -5.15580047e-01,
                   -3.10112984e-01,  -9.16448704e-02,  -1.63238654e-01,
                    2.22829409e+00,   1.04468355e-01,  -6.21857302e-01,
                   -1.49296649e-01,  -3.27143932e-01,  -3.31907376e-01],
                 [  1.44799125e-01,  -1.04695805e+00,  -5.15580047e-01,
                    4.88745024e-01,  -2.36732610e-01,   6.51765695e-01,
                   -5.41978437e-01,  -9.72177384e-01,   2.96122525e-03,
                    6.36958246e-01,   2.32798060e-01,   2.05954157e+00],
                 [ -5.67037571e-01,  -1.13350997e+00,   1.93956303e+00,
                    2.29523684e-01,  -2.26083969e-01,   3.69397303e-01,
                   -1.17551506e+00,  -1.25547751e+00,  -2.44301083e-01,
                    4.19743859e-01,   3.97146146e-02,   3.22255789e-01],
                 [ -4.32100891e-01,   5.40121586e-01,   1.93956303e+00,
                    3.19626838e-01,   5.52398275e-03,   4.58566269e-01,
                   -1.34123036e+00,   1.03354794e-01,  -1.48061262e-02,
                    5.17643301e-01,   4.08503996e-02,   2.15015926e-01],
                 [ -3.21591773e-01,  -5.38784680e-01,  -2.45514308e-02,
                    1.20905409e-01,  -4.61685161e-01,   9.77291866e-02,
                    6.85061040e-02,  -2.01911584e-01,  -3.87920507e-01,
                    1.06465643e-01,  -6.96993379e-02,  -3.31907376e-01],
                 [ -4.14730993e-01,   7.13928929e-01,  -2.45514308e-02,
                    1.32511127e-01,  -2.59360973e-01,   6.91951175e-02,
                    9.47887601e-01,   7.01268186e-01,  -1.62867389e-02,
                    1.00346928e-01,  -1.51854451e-01,  -3.31907376e-01],
                 [  7.49172834e-02,   2.28319807e-01,  -2.45514308e-02,
                    4.57880484e-01,   3.48077467e-02,   1.10058282e+00,
                    6.74442639e-01,   1.04151952e-02,  -2.94641912e-01,
                    7.26903359e-01,   4.41403900e-01,  -9.82495880e-01],
                 [  1.21614882e-01,   1.53340673e+00,  -2.45514308e-02,
                    1.17809964e+00,   5.97854663e-01,   1.33598889e+00,
                    8.13778311e-01,   1.12891407e+00,   2.84277624e-01,
```

```
         1.40363325e+00,   4.31181835e-01,  -9.82495880e-01],
       [  4.19097237e-01,  -2.03813430e-01,  -2.45514308e-02,
          4.70349313e-01,  -1.43556997e-01,   5.36440499e-01,
         -4.54950176e-01,  -7.13803610e-02,  -2.84277624e-01,
          5.47013134e-01,   1.27927248e-01,  -3.31907376e-01],
       [ -2.20615986e-02,   6.12473671e-01,  -2.45514308e-02,
          2.18243876e-01,   6.99016757e-01,  -4.49411588e-02,
         -1.35368553e+00,   6.87158877e-01,   7.12174672e-01,
         -1.65205309e-02,  -4.99026058e-01,  -9.82495880e-01],
       [ -1.73600244e-01,   1.68327460e+00,  -2.45514308e-02,
         -8.16618113e-01,   5.59253338e-01,  -1.23861638e+00,
          1.94246620e+00,   1.97477412e+00,   5.37462382e-01,
         -1.24699415e+00,  -1.21646357e+00,  -2.24667513e-01],
       [ -4.12866015e-01,   7.11531816e-02,  -2.45514308e-02,
          1.14353130e+00,   1.63343504e+00,   7.59362914e-01,
          9.03434433e-01,   1.09987861e-01,   1.60794531e+00,
          8.10729756e-01,  -2.41960059e-01,  -6.60776290e-01],
       [ -4.30528458e-01,   4.28100135e-01,  -2.45514308e-02,
          1.41024635e+00,   1.96221185e+00,   8.58043236e-01,
         -4.44742511e-01,   6.75758435e-01,   2.19722913e+00,
          9.06793584e-01,  -2.65432948e-01,  -5.49961765e-01],
       [ -6.23206335e-01,  -9.09262016e-01,  -2.45514308e-02,
         -1.10949370e+00,   3.68908872e-01,  -1.53584627e+00,
         -1.07576830e-01,  -1.37549555e+00,   2.08766380e-01,
         -1.55048242e+00,  -1.33458520e+00,  -8.71681354e-01],
       [  4.51825781e-01,  -1.44786871e+00,   1.93956303e+00,
         -2.69321815e-01,  -9.58178068e-01,   5.25502439e-02,
          5.45952303e-01,  -2.48557967e+00,  -9.28344115e-01,
          5.56803078e-02,   5.41012242e-02,   1.73782198e+00],
       [  1.74711917e-01,  -5.10275578e-01,  -1.98866590e+00,
         -1.83346000e+00,  -1.41740073e+00,  -1.65592714e+00,
         -1.01467365e+00,   1.63437079e-01,  -1.67013104e+00,
         -1.65205309e+00,  -9.12830383e-01,   5.36735515e-01],
       [  1.82793491e-01,  -1.22258053e+00,  -1.98866590e+00,
         -2.73714067e+00,  -2.21205560e+00,  -2.70098742e+00,
         -1.02509477e+00,  -9.22228138e-01,  -1.78561882e+00,
         -2.72772321e+00,  -1.35200057e+00,   5.36735515e-01]])
```

In [183]:

```
# SPLit the data into training and test data
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3)
```

Regression analysis

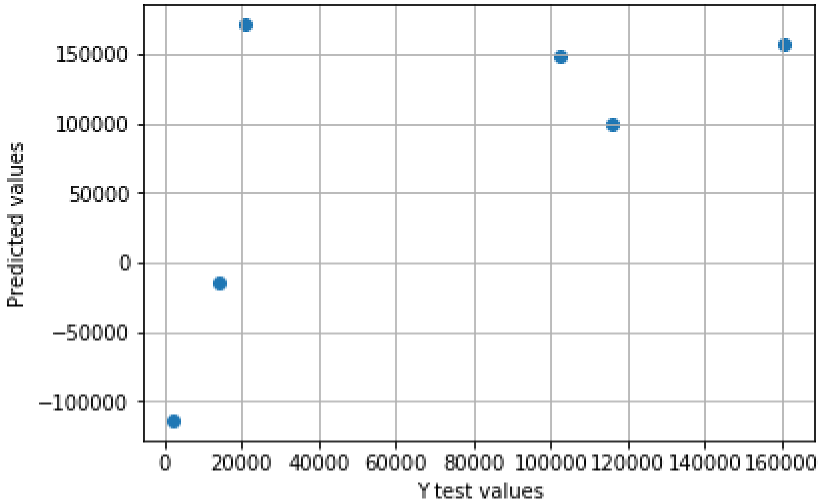
In [184]:
```
lm = LinearRegression() # linear regression

DTR = DecisionTreeRegressor(max_depth=1) # Decision tree regressor
MLPR = MLPRegressor(max_iter = 200, solver = 'lbfgs', verbose=True, tol = 0.000001) # Multilayer perceptron
```

In [185]:
```
SVR = SVR(C = 0.0001, epsilon = 0.2,kernel = 'linear') # Support vector regression
```

In [186]:
```
def Regression_analysis(Regressor,X_train,y_train,X_test,y_test):
    Regressor.fit(X_train,y_train)
    Predict = Regressor.predict(X_test)
    plt.scatter(y_test,Predict)
    plt.xlabel('Y test values')
    plt.ylabel('Predicted values')
    plt.grid()
    plt.show()
    print('MAE:', metrics.mean_absolute_error(y_test, Predict))
    print('MSE:', metrics.mean_squared_error(y_test, Predict))
    print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, Predict)))
```

Linear regression

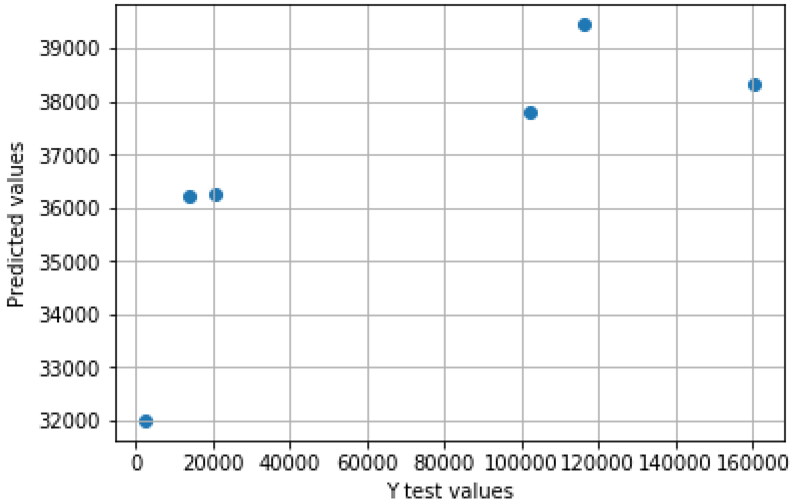
In [187]:
```
Regression_analysis(lm,X_train,y_train,X_test,y_test)
```




```
MAE: 60307.8769227
MSE: 6557027948.81
RMSE: 80975.4774534
```

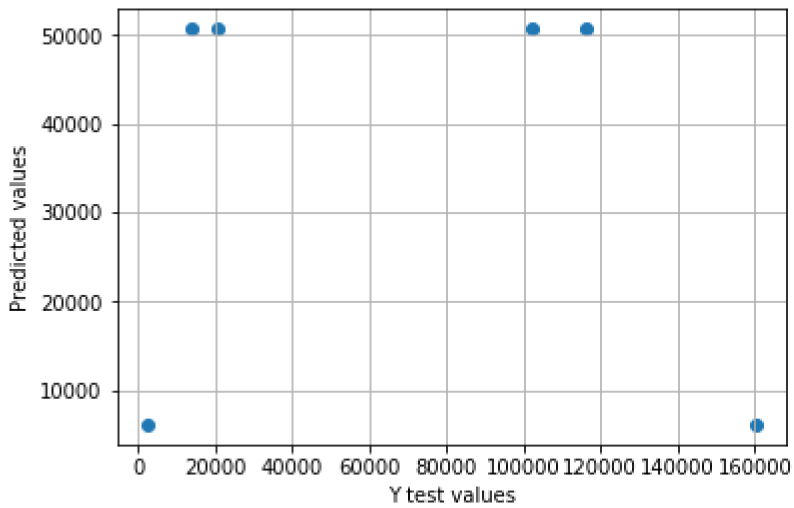
Support Vector Regression

In [188]:
```
Regression_analysis(SVR,X_train,y_train,X_test,y_test)
```



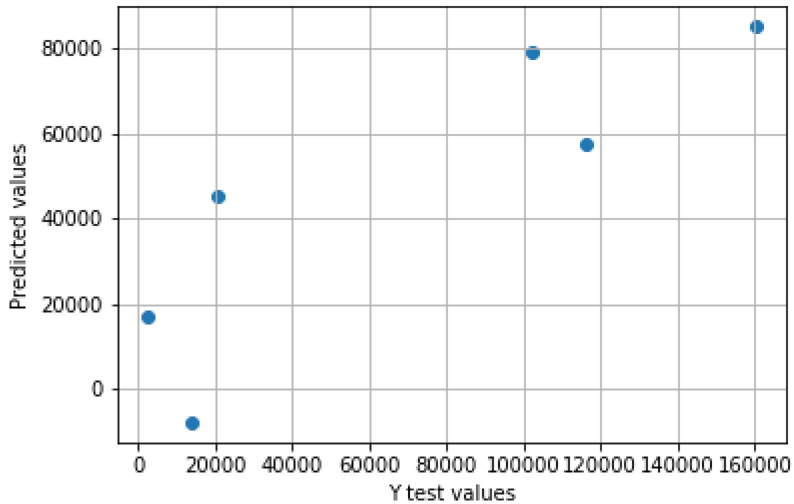

```
MAE: 55086.9794394
MSE: 4429221587.65
RMSE: 66552.3973096
```

In [189]:
```
Regression_analysis(DTR,X_train,y_train,X_test,y_test)
```




```
MAE: 56939.0
MSE: 5503521880.7
RMSE: 74185.7255859
```

In [190]:
```
Regression_analysis(MLPR,X_train,y_train,X_test,y_test)
```




```
MAE: 36307.7778453
MSE: 1822337212.23
RMSE: 42688.8417767
```

In [ ]:
```
```

In [ ]:
```
```

Import test model datasheet

In [125]:
```
test_data = pd.read_excel('IntroEngDataScienceFinalProjectTestModelOutput.xlsx')
print(test_data.head())
print(test_data.isnull().sum())
```

```
   WELL_ID          JOB_DESC_STAGING     PROPPANT_MESH_SIZE  \
0        1  Day 3 Stage 11: Wolfcamp  Sand, White, 100 mesh
1        1   Day 2 Stage 9: Wolfcamp  Sand, White, 100 mesh
2        1   Day 1 Stage 4: Wolfcamp                  40/70
3        1   Day 2 Stage 7: Wolfcamp                  40/70
4        1   Day 2 Stage 9: Wolfcamp                  40/70

  PROPPANT_MESH_DESCRIPTION  PROPPANT_MASS_USED PROPPANT_MASS_UOM  \
0     Sand, White, 100 mesh                  85     CWT = 132 lbs
1     Sand, White, 100 mesh                 109               CWT
2        Sand, White, 40/70                  70               CWT
3        Sand, White, 40/70                 102               CWT
4        Sand, White, 40/70                 110               CWT

   VOLUME_PUMPED_GALLONS  AVERAGE_STP AVERAGE_STP_UOM  FRACTURE_GRADIENT  \
0                 127232         4171             PSI               0.76
1                 126245         4606             PSI               0.76
2                 355844         5517             PSI               0.76
3                 356168         5004             PSI               0.76
4                 355834         4606             PSI               0.76

       ...      MIN_STP  MIN_STP_UOM MAX_STP  MAX_STP_UOM  UPPER_PERF  \
0      ...         4077          PSI    7735          PSI        6260
1      ...         3252          PSI    7683          PSI        6260
2      ...         5130          PSI    7792          PSI        6260
3      ...         3707          PSI    7898          PSI        6260
4      ...         3252          PSI    7683          PSI        6260

   LOWER_PERF  TRUE_VERTICAL_DEPTH  WELL_HORZ_LENGTH NET_PROD_DAYS  \
0       14060                 5810              8345           670
1       14060                 5810              8345           670
2       14060                 5810              8345           670
3       14060                 5810              8345           670
4       14060                 5810              8345           670

   LIQ_CUM_BBLS
0           NaN
1           NaN
2           NaN
3           NaN
4           NaN

[5 rows x 27 columns]
```

```
WELL_ID                         0
JOB_DESC_STAGING                0
PROPPANT_MESH_SIZE              0
PROPPANT_MESH_DESCRIPTION       0
PROPPANT_MASS_USED              0
PROPPANT_MASS_UOM               0
VOLUME_PUMPED_GALLONS           0
AVERAGE_STP                     0
AVERAGE_STP_UOM                 0
FRACTURE_GRADIENT               0
FRACTURE_GRADIENT_UOM           0
MD_MIDDLE_PERFORATION           0
MD_MIDDLE_PERFORATION_UOM       0
TVD_DEPTH                       0
TOP_DEPTH                       0
WELL_LATITUDE                   0
WELL_LONGITUDE                  0
MIN_STP                         0
MIN_STP_UOM                     0
MAX_STP                         0
MAX_STP_UOM                     0
UPPER_PERF                      0
LOWER_PERF                      0
TRUE_VERTICAL_DEPTH             0
WELL_HORZ_LENGTH                0
NET_PROD_DAYS                   0
LIQ_CUM_BBLS                  338
dtype: int64
```

In [126]:

```
test_data=test_data.drop(labels=['JOB_DESC_STAGING','PROPPANT_MESH_DESCRIPTION','PROPPANT_MASS_UOM',
          'AVERAGE_STP_UOM','FRACTURE_GRADIENT_UOM','MD_MIDDLE_PERFORATION_UOM','MIN_STP_UOM',
          'MAX_STP_UOM'], axis = 1)
```

In [127]:

```
# Count the number of rows
Count = rowcount(test_data)
```

In [128]:
```
print(Count)
```

```
   No of rows  Well ID
0          66        1
1           3        5
2          50        9
3          58       13
4          67       17
5          34       20
6          60       27
```

In [129]:

```
# group the data by the well ID
test_group =test_data.groupby('WELL_ID', as_index = False)
test_group=test_group.sum()
test_group['Count'] = Count['No of rows']
print(test_group.head())
```

```
   WELL_ID  PROPPANT_MASS_USED  VOLUME_PUMPED_GALLONS  AVERAGE_STP  \
0        1                7219               14805348       297180   
1        5                 618                 257683        17916   
2        9                8868                8099427       278880   
3       13               12287                6002579       251602   
4       17               37775                7274959       351314   

   FRACTURE_GRADIENT  MD_MIDDLE_PERFORATION  TVD_DEPTH  TOP_DEPTH  \
0              50.16               672484.0     377718     934230   
1               2.25                41838.0      19749      43671   
2              40.00               535220.0     332650     725100   
3              44.08               596267.0     362500     800168   
4              50.92               726750.0     437644     999506   

   WELL_LATITUDE  WELL_LONGITUDE  MIN_STP  MAX_STP  UPPER_PERF  LOWER_PERF  \
0     2046.01650     -6664.53810   230044   466374      413160      927960   
1       93.39516      -303.47820    17103    18519       20808       43671   
2     1569.85600     -5058.59300   127168   329624      348750      721250   
3     1808.08620     -5859.85136   178700   321976      393240      794542   
4     2085.86410     -6771.89368   254762   461354      452853      989657   

   TRUE_VERTICAL_DEPTH  WELL_HORZ_LENGTH  NET_PROD_DAYS  LIQ_CUM_BBLS  Count
0               383460            550770          44220           NaN     66
1                19749             23922           4473           NaN      3
2               332800            392300          44200           NaN     50
3               364646            435522          42398           NaN     58
4               437644            561862          46900           NaN     67
```

In [130]:
```
old_column_names = ['NET_PROD_DAYS','WELL_HORZ_LENGTH','TRUE_VERTICAL_DEPTH'
                   ,'LOWER_PERF','UPPER_PERF','MAX_STP','MIN_STP','WELL_LONGITUDE','WELL_LATITUDE',
                   'TOP_DEPTH','TVD_DEPTH','MD_MIDDLE_PERFORATION','FRACTURE_GRADIENT','AVERAGE_STP']
# new column names
new_column_names=['Mean Production days','Mean Horizontal length',
 'Mean True Vertical Distance','Mean Lower perforation','Mean Upper perforation','Mean Maximum STP',
'Mean Minimum STP','Longitude','Latitude','Mean TOP Depth','Mean TVD depth','Mean Mid perforation',
 'Mean Fracture Gradient','Mean STP']
```

In [131]:
```
# Compute the mean and generate new columns using the predefined columns
mean_calculator(test_group,old_column_names=old_column_names,new_column_names=new_column_names)
```

In [132]:
```
# Drop some  columns
test_group=test_group.drop(labels= ['AVERAGE_STP','FRACTURE_GRADIENT','MD_MIDDLE_PERFORATION','TVD_DEPTH',
                 'TOP_DEPTH','WELL_LATITUDE','WELL_LONGITUDE','MIN_STP','MAX_STP','UPPER_PERF',
                 'LOWER_PERF','TRUE_VERTICAL_DEPTH','WELL_HORZ_LENGTH','NET_PROD_DAYS'
                 ], axis = 1)
```

In [133]:
```
Xtest=test_group.drop(labels=['Count','WELL_ID','Latitude','Longitude','Mean True Vertical Distance'], axis =
 1)
```

In [134]:
```
# re-index the column based on the selected features
Xtest = Xtest[['PROPPANT_MASS_USED', 'Mean STP', 'Mean Fracture Gradient',
       'Mean Mid perforation', 'Mean TVD depth', 'Mean TOP Depth',
       'Mean Minimum STP', 'Mean Maximum STP', 'Mean Upper perforation',
       'Mean Lower perforation', 'Mean Horizontal length',
       'Mean Production days','LIQ_CUM_BBLS']]
```

In [135]:
```
print(Xtest)
```

```
   PROPPANT_MASS_USED     Mean STP  Mean Fracture Gradient  \
0                7219  4502.727273                    0.76
1                 618  5972.000000                    0.75
2                8868  5577.600000                    0.80
3               12287  4337.965517                    0.76
4               37775  5243.492537                    0.76
5               18503  5990.323529                    0.76
6               45934  5743.000000                    0.65

   Mean Mid perforation  Mean TVD depth  Mean TOP Depth  Mean Minimum STP  \
0          10189.151515          5723.0         14155.0       3485.515152
1          13946.000000          6583.0         14557.0       5701.000000
2          10704.400000          6653.0         14502.0       2543.360000
3          10280.465517          6250.0         13796.0       3081.034483
4          10847.014925          6532.0         14918.0       3802.417910
5           9555.088235          7069.0         11727.0       5008.382353
6          11733.866667          7966.0         14850.0       4855.466667

   Mean Maximum STP  Mean Upper perforation  Mean Lower perforation  \
0       7066.272727                  6260.0                 14060.0
1       6173.000000                  6936.0                 14557.0
2       6592.480000                  6975.0                 14425.0
3       5551.310345                  6780.0                 13699.0
4       6885.880597                  6759.0                 14771.0
5       7532.441176                  7410.0                 11671.0
6       7041.033333                  8693.0                 14775.0

   Mean Horizontal length  Mean Production days  LIQ_CUM_BBLS
0                  8345.0                 670.0           NaN
1                  7974.0                1491.0           NaN
2                  7846.0                 884.0           NaN
3                  7509.0                 731.0           NaN
4                  8386.0                 700.0           NaN
5                  4608.0                 761.0           NaN
6                  6884.0                1188.0           NaN
```

In [136]:
```
# Remove the liq cum column prior to Standardizing
Xtestscale = Xtest[['PROPPANT_MASS_USED', 'Mean STP', 'Mean Fracture Gradient',
       'Mean Mid perforation', 'Mean TVD depth', 'Mean TOP Depth',
       'Mean Minimum STP', 'Mean Maximum STP', 'Mean Upper perforation',
       'Mean Lower perforation', 'Mean Horizontal length',
       'Mean Production days']]
```

In [137]:
```
#Standardize
Scaled.fit(Xtestscale)
Scaled.transform(Xtestscale)
```

Out[137]:
```
array([[-0.73904507, -1.33153288,  0.26637086, -0.63208293, -1.48074612,
         0.08100803, -0.5491714 ,  0.61611551, -1.19404533,  0.06505822,
         0.8021137 , -0.86780205],
       [-1.16235774,  1.01023398,  0.03329636,  2.17012293, -0.15325667,
         0.47403666,  1.53900299, -0.85347351, -0.25124164,  0.55496634,
         0.4985891 ,  2.00669845],
       [-0.63329707,  0.38162852,  1.19866888, -0.24776284, -0.04520521,
         0.42026409, -1.43718662, -0.16335594, -0.19684912,  0.4248499 ,
         0.39386903, -0.11854126],
       [-0.4140415 , -1.59413465,  0.26637086, -0.56397248, -0.66727294,
        -0.26998022, -0.93040902, -1.87626114, -0.46881172, -0.29079053,
         0.11816071, -0.65422771],
       [ 1.22046738, -0.15088104,  0.26637086, -0.14138741, -0.23197988,
         0.82698028, -0.25047912,  0.31933907, -0.49810001,  0.7659127 ,
         0.83565684, -0.76276549],
       [-0.01541834,  1.03943852,  0.26637086, -1.1050261 ,  0.59692923,
        -2.29280666,  0.88618587,  1.3830437 ,  0.40983668, -2.28985224,
        -2.25522161, -0.54919115],
       [ 1.74369234,  0.64524755, -2.2974487 ,  0.52010883,  1.98153159,
         0.76049782,  0.74205732,  0.57459231,  2.19921114,  0.76985562,
        -0.39316778,  0.94582921]])
```

We use the multilayer perceptron to predict the liquid produced column

In [138]:
```
Xtest['LIQ_CUM_BBLS'] = MLPR.predict(Xtestscale)
```

In [139]:
```
print(Xtest)
```

```
   PROPPANT_MASS_USED     Mean STP  Mean Fracture Gradient  \
0                7219  4502.727273                    0.76   
1                 618  5972.000000                    0.75   
2                8868  5577.600000                    0.80   
3               12287  4337.965517                    0.76   
4               37775  5243.492537                    0.76   
5               18503  5990.323529                    0.76   
6               45934  5743.000000                    0.65   

   Mean Mid perforation  Mean TVD depth  Mean TOP Depth  Mean Minimum STP  \
0          10189.151515          5723.0         14155.0       3485.515152   
1          13946.000000          6583.0         14557.0       5701.000000   
2          10704.400000          6653.0         14502.0       2543.360000   
3          10280.465517          6250.0         13796.0       3081.034483   
4          10847.014925          6532.0         14918.0       3802.417910   
5           9555.088235          7069.0         11727.0       5008.382353   
6          11733.866667          7966.0         14850.0       4855.466667   

   Mean Maximum STP  Mean Upper perforation  Mean Lower perforation  \
0       7066.272727                  6260.0                 14060.0   
1       6173.000000                  6936.0                 14557.0   
2       6592.480000                  6975.0                 14425.0   
3       5551.310345                  6780.0                 13699.0   
4       6885.880597                  6759.0                 14771.0   
5       7532.441176                  7410.0                 11671.0   
6       7041.033333                  8693.0                 14775.0   

   Mean Horizontal length  Mean Production days    LIQ_CUM_BBLS  
0                  8345.0                 670.0    75964.973443  
1                  7974.0                1491.0   218326.808322  
2                  7846.0                 884.0    29174.453617  
3                  7509.0                 731.0    42287.294083  
4                  8386.0                 700.0    63675.484749  
5                  4608.0                 761.0    35751.015851  
6                  6884.0                1188.0    25599.437312  
```

In [140]:
```
Xtest['Well ID'] = test_group['WELL_ID']
```

In [141]:
```
print(Xtest)
```

```
   PROPPANT_MASS_USED     Mean STP  Mean Fracture Gradient  \
0                7219  4502.727273                    0.76   
1                 618  5972.000000                    0.75   
2                8868  5577.600000                    0.80   
3               12287  4337.965517                    0.76   
4               37775  5243.492537                    0.76   
5               18503  5990.323529                    0.76   
6               45934  5743.000000                    0.65   

   Mean Mid perforation  Mean TVD depth  Mean TOP Depth  Mean Minimum STP  \
0          10189.151515          5723.0         14155.0       3485.515152   
1          13946.000000          6583.0         14557.0       5701.000000   
2          10704.400000          6653.0         14502.0       2543.360000   
3          10280.465517          6250.0         13796.0       3081.034483   
4          10847.014925          6532.0         14918.0       3802.417910   
5           9555.088235          7069.0         11727.0       5008.382353   
6          11733.866667          7966.0         14850.0       4855.466667   

   Mean Maximum STP  Mean Upper perforation  Mean Lower perforation  \
0       7066.272727                  6260.0                 14060.0   
1       6173.000000                  6936.0                 14557.0   
2       6592.480000                  6975.0                 14425.0   
3       5551.310345                  6780.0                 13699.0   
4       6885.880597                  6759.0                 14771.0   
5       7532.441176                  7410.0                 11671.0   
6       7041.033333                  8693.0                 14775.0   

   Mean Horizontal length  Mean Production days   LIQ_CUM_BBLS  Well ID  
0                  8345.0                 670.0   75964.973443        1  
1                  7974.0                1491.0  218326.808322        5  
2                  7846.0                 884.0   29174.453617        9  
3                  7509.0                 731.0   42287.294083       13  
4                  8386.0                 700.0   63675.484749       17  
5                  4608.0                 761.0   35751.015851       20  
6                  6884.0                1188.0   25599.437312       27  
```

In [142]:
```
Xtest
```

Out[142]:

	PROPPANT_MASS_USED	Mean STP	Mean Fracture Gradient	Mean Mid perforation	Mean TVD depth	Mean TOP Depth	Mean Minimum STP	Mean Maximum STP	Mean Upper perforation	p
0	7219	4502.727273	0.76	10189.151515	5723.0	14155.0	3485.515152	7066.272727	6260.0	1
1	618	5972.000000	0.75	13946.000000	6583.0	14557.0	5701.000000	6173.000000	6936.0	1
2	8868	5577.600000	0.80	10704.400000	6653.0	14502.0	2543.360000	6592.480000	6975.0	1
3	12287	4337.965517	0.76	10280.465517	6250.0	13796.0	3081.034483	5551.310345	6780.0	1
4	37775	5243.492537	0.76	10847.014925	6532.0	14918.0	3802.417910	6885.880597	6759.0	1
5	18503	5990.323529	0.76	9555.088235	7069.0	11727.0	5008.382353	7532.441176	7410.0	1
6	45934	5743.000000	0.65	11733.866667	7966.0	14850.0	4855.466667	7041.033333	8693.0	1

Export the result to a csv file

In [143]:
```
# export to a csv
Xtest.to_csv('FinalResult.csv')
```

In [ ]:

In [ ]:

In [ ]: