Report

February 13, 2024

1 Breakdown of the business problem

The company, a key player in the Oil and Gas industry, has been grappling with a decline in profits from its oil and gas trading operations. The management attributes this downturn to potentially inefficient trading strategies and is keen on leveraging the power of data to bolster decision-making processes. In the Oil and Gas sector, the ability to understand, leverage, and unleash the power of data is important. It equips firms with the competitive edge needed throughout planning, exploration, production, and field development stages. Moreover, it enables them to maximize production in relation to maintenance and forecasting. This, in turn, lowers operating costs and enhances the productivity of assets across their life cycle. One of the key challenges that the firm is facing is ensuring the seamless, automated availability of the right information to the workforce at the right time. Most of the analysis in the company is done mannully which is a slow process and in many occasion does not provide accurate information. With the advanced technology in the field of data analysis this should not be the case.

1.1 Business Question

How can we leverage Big Data to improve our trading strategies, reduce operating costs, and increase the profitability of our oil and gas trading operations?

Is it possible for the comapnay to transition from manual to automated data analysis to increase efficiency, speed, and accuracy of information for better business decisions?

What is the furture of oil production in the us?

2 Findings and recommendations to the business questions

2.1 Findings

2.2 Problem 1

The data uncovers Texas as the biggest producer, with an all out production of 290,863 units, followed by the Federal Offshore Gulf of Mexico with 174,756 units, and North Dakota with 90,312 units. Texas stands out not only for its sheer production volume but also for the considerable variability in its production figures, evidenced by a mean production of 2,403 units and a standard deviation of approximately 1,025 units. In contrast, Alaska's production, totaling 65,389 units, displays less variability, indicating a more stable output. California, with 64,903 units, demonstrates consistent production levels, reflecting a mean production of 536 units and a relatively low standard deviation of approximately 33 units. Oklahoma and New Mexico contribute significantly to the overall production, with 38,915 units and 36,965 units, respectively, showcasing varying degrees of variability in their production figures. Colorado, with 25,666 units, also presents a noteworthy contribution to the national production, albeit with moderate variability.

2.3 Problem 2 findings

The correlation analysis reveals significant relationships between the oil production of various states in the US. Texas, being the leading producer, demonstrates strong positive correlations with several other states, notably North Dakota, Oklahoma, New Mexico, and Colorado, with correlation coefficients ranging from approximately 0.93 to 0.99. These strong positive correlations indicate a tendency for these states to exhibit similar trends in oil production over time. Conversely, Alaska and California exhibit negative correlations with many other states, particularly Texas, Oklahoma, and New Mexico, suggesting an inverse relationship in their production trends.

2.4 Problem 3 findings

The trend analysis of oil production in the US uncovers an a general upward trajectory in overall production between 2008 and 2018. This positive trend lines up with more extensive industry trends driven by variables like technological advancements, increased exploration activities, and developing worldwide interest for oil. Upon closer assessment of individual states, it is obvious that states like Texas, Federal Offshore Gulf of Mexico, and North Dakota shows an upward tredn in the data, demonstrating steady development in oil production inside these states. On the other hand, states like Oklahoma, New Mexico, and The Frozen North show a decreasing trend in oil production.

From the anlysis done, it has proven that, it is possible form the company to shift from the manual way of doing analysis and make use of the growing technology. The analysis done shows we still have a upward trend in oil production in the unites states. Some staes such as texas have seen a significant increase in oil production.

2.5 Recomendations

2.6 General Reccomendations for All Problems

The company should focus on states on states displaying up trends in oil production, like Texas,Federal Offshore Gulf of Mexico, and North Dakota, as these areas offer promising speculation possibilities with potential for development and profitability.The companys should also incorporate advanced predictive modeling techniques to anticipate changes in oil production levels and market elements, enabling proactive decision-making and strategic portfolio management.. Recognize states with descending trends in oil production, like Oklahoma, New Mexico, and Gold country, aand explore opportunities to negotiate favorable trading agreements or divest from underperforming assets to mitigate risks and minimize operational expenses. To improve on decision making the company should develop customized data analytics tools and dashboards to visualize key performance indicators, monitor oil production activities in real-time, and identify actionable insights for informed decision-making. The company should focus on states with strong positive correlations with leading producers like Texas, such as North Dakota, Oklahoma, and New Mexico, as these regions are likely to exhibit similar production trends and offer potential for profitable trading opportunities. In oreder to shift from manual to advanced technology the company should determine the technical feasibility of transitioning from manual to automated data analysis. This involves evaluating the company's current technological infrastructure, data availability, and the capabilities of existing or potential automated data analysis tools.

3 Assumptions made, data quality and availability constraints

The suggestions are based on the premise that both trend and correlation analyses provide a true representation of future production patterns and market behaviors. The analysis hinges on the assumption that the data used for production, correlation, and trend analyses are correct, dependable, and upto date. Any data inaccuracies or inconsistencies could affect the credibility of the conclusions and suggestions. Regulatory constraints or privacy considerations may restrict access to certain types of data, particularly sensitive information related to production rights, contractual agreements, or proprietary trading strategies. Timely access to relevant data is essential for conducting real-time analyses and making informed decisions. Delays in data availability could hinder the ability to respond promptly to market changes or emerging trends. In order to determine the future of oil production the company should look at other factors that might be conritbuting to the increase in supply. for example, demand. How and what is driving the demand of oil in the USA desipte the adoption of other technologues such as electric cars? The company can also go for predictive models e.g time sereis analysis, and type curve that will predict the direction of oil production in the USA.

4 Detailed analysis of the data provided

4.1 Load libraries

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import datetime
from scipy.optimize import curve_fit
import geopandas as gpd
from datetime import datetime
from pathlib import Path
import seaborn as sns

# path data
datapath = Path('Assignment data')
```

4.2 Load Data.

[ ]:
```
filepath = './U.S._crude_oil_production.csv'
# load data
df_oil = pd.read_csv(filepath)

# Loading Excel files using pandas
well_stats_2021 = pd.read_excel(datapath / '2021-OCD-Well-Statistics.xlsx')
flaring_data = pd.read_excel(datapath /␣
↪'c-115_flaring_and_venting_data_from_2014_district_and_year.xlsx')
metadata = pd.read_excel(datapath / 'Metadata.xlsx')
flaring_and_venting_data_operator_year = pd.read_excel(datapath /␣
↪'c-115_flaring_and_venting_data_from_2014_operator_and_year.xlsx')
```

```
produced_water_data_district_year = pd.read_excel(datapath /␣
    ↪'c-115_produced_water_data_from_2014_district_and_year.xlsx')
produced_water_data_operator_year = pd.read_excel(datapath /␣
    ↪'c-115_produced_water_data_from_2014_operator_and_year.xlsx')
#active_water_haulers = pd.read_excel(datapath / 'active_water_hauler.xlsx')
production_data = pd.read_excel(datapath / 'Production Data.xlsx')
operators_quarterly = pd.read_excel(datapath / 'operators_quarterly.xlsx')
#monthly_inspections = pd.read_excel(datapath / 'monthly_inspection.xlsx')
ocd_well_statistics_2020 = pd.read_excel(datapath / 'OCDWellStatistics2020.
    ↪xlsx')
oil_and_gas_sales_volume = pd.read_excel(datapath /␣
    ↪'oil_and_gas_sales_by_volume_by_year.xlsx')
gas_produced_by_stripper_wells = pd.read_excel(datapath /␣
    ↪'Gas-Produced-By-Stripper-Wells.xlsx')
# Loading GeoJSON files using geopandas
oil_gas_wells = gpd.read_file(datapath / 'New_Mexico_Oil_and_Gas_Wells.geojson')
```

```
/home/ricmwas/.local/lib/python3.10/site-packages/geopandas/io/file.py:414:
UserWarning: Could not infer format, so each element will be parsed
individually, falling back to `dateutil`. To ensure parsing is consistent and
as-expected, please specify a format.
  as_dt = pd.to_datetime(df[k])
/home/ricmwas/.local/lib/python3.10/site-packages/geopandas/io/file.py:421:
UserWarning: Could not infer format, so each element will be parsed
individually, falling back to `dateutil`. To ensure parsing is consistent and
as-expected, please specify a format.
  as_dt = pd.to_datetime(df[k], utc=True)
```

4.3 Inspect the data

[ ]:
```
df_oil.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 121 entries, 0 to 120
Data columns (total 36 columns):
 #   Column                                   Non-Null Count  Dtype
---  ------                                   --------------  -----
 0   Month                                    121 non-null    object
 1   U.S. Crude Oil                           121 non-null    int64
 2   Alabama                                  121 non-null    int64
 3   Alaska                                   121 non-null    int64
 4   Arkansas                                 121 non-null    int64
 5   Arizona                                  121 non-null    int64
 6   California                               121 non-null    int64
 7   Colorado                                 121 non-null    int64
 8   Federal Offshore Gulf of Mexico Crude Oil 121 non-null    int64
 9   Federal Offshore Pacific Crude Oil       121 non-null    int64
```

```
 10  Florida                                    121 non-null    int64
 11  Idaho                                      121 non-null    int64
 12  Illinois                                   121 non-null    int64
 13  Indiana                                    121 non-null    int64
 14  Kansas                                     121 non-null    int64
 15  Kentucky                                   121 non-null    int64
 16  Louisiana                                  121 non-null    int64
 17  Michigan                                   121 non-null    int64
 18  Mississippi                                121 non-null    int64
 19  Missouri                                   121 non-null    int64
 20  Nebraska                                   121 non-null    int64
 21  Montana                                    121 non-null    int64
 22  Nevada                                     121 non-null    int64
 23  New Mexico                                 121 non-null    int64
 24  New York                                   121 non-null    int64
 25  North Dakota                               121 non-null    int64
 26  Ohio                                       121 non-null    int64
 27  Oklahoma                                   121 non-null    int64
 28  Pennsylvania                               121 non-null    int64
 29  South Dakota                               121 non-null    int64
 30  Wyoming                                    121 non-null    int64
 31  West Virginia                              121 non-null    int64
 32  Virginia                                   121 non-null    int64
 33  Utah                                       121 non-null    int64
 34  Texas                                      121 non-null    int64
 35  Tennessee                                  121 non-null    int64
dtypes: int64(35), object(1)
memory usage: 34.2+ KB
```

[ ]:
```
df_oil.head()
```

[ ]:
```
        Month  U.S. Crude Oil   Alabama  Alaska  Arkansas  Arizona  \
0  2008-06-01            5138        21     655        17        0
1  2008-07-01            5177        21     640        17        0
2  2008-08-01            5003        21     544        17        0
3  2008-09-01            3974        21     681        16        0
4  2008-10-01            4738        21     716        17        0

   California  Colorado  Federal Offshore Gulf of Mexico Crude Oil  \
0         583        82                                       1326
1         586        81                                       1372
2         588        82                                       1272
3         587        88                                        242
4         586        86                                        803

   Federal Offshore Pacific Crude Oil  …  Ohio  Oklahoma  Pennsylvania  \
0                                  67  …    14       186             8
```

```
1                                   61  ...    14       184              8
2                                   70  ...    14       188              8
3                                   67  ...    14       186              8
4                                   66  ...    14       185              8

   South Dakota  Wyoming  West Virginia  Virginia  Utah  Texas  Tennessee
0             5      144              6         0    60   1097          1
1             5      145              5         0    61   1111          1
2             5      145              6         0    62   1110          1
3             5      144              6         0    63   1055          1
4             5      145              6         0    64   1125          1

[5 rows x 36 columns]
```

[ ]:
```python
# MetaData
metadata.head()
```

[ ]:
```
        API_UWI  Unformatted_API_UWI        API_UWI_12  Unformatted_API_UWI_12  \
0  30-015-43706           3001543706  30-015-43706-00            300154370600
1  30-025-44687           3002544687  30-025-44687-00            300254468700
2  30-025-46812           3002546812  30-025-46812-00            300254681200
3  30-025-44013           3002544013  30-025-44013-00            300254401300
4  30-015-49326           3001549326  30-015-49326-00            300154932600

           API_UWI_14  Unformatted_API_UWI_14           WellID  \
0  30-015-43706-00-00          30015437060000  840300003495031
1  30-025-44687-00-00          30025446870000  840300004752843
2  30-025-46812-00-00          30025468120000  840300005432761
3  30-025-44013-00-00          30025440130000  840300004671472
4  30-015-49326-00-00          30015493260000  840300019360808

      CompletionID     WellPadID WellPadDirection  ...  \
0  840301002145615  30-015-43706                N  ...
1  840301006255780  30-025-44687                N  ...
2  840301006778568  30-025-46812                S  ...
3  840301006220087  30-025-44013                N  ...
4  840301007128722  30-015-44453                S  ...

   EURWH_MBOE_60Per1000FT  EURWH_MBOE_120Per1000FT EURWH_MBOE_180Per1000FT  \
0                    93.0                    125.0                   145.0
1                    44.0                     54.0                    60.0
2                    67.0                     94.0                   111.0
3                    59.0                     78.0                    91.0
4                    97.0                    120.0                   133.0

  EURWH_MBOE_360PEr1000FT EURWH_BCFE_360Per1000FT OilEURWH_MBBLPer1000FT  \
0                   177.8                    1.07                  104.0
```

```
1                   69.8                    0.42                    57.0
2                  143.6                    0.86                   121.0
3                  116.0                    0.70                    95.0
4                  155.3                    0.93                    82.0

   EURWH_MBBL_360Per1000FT GasEURWH_BCFPer1000FT EURWH_BCF_360Per1000FT  \
0                    100.0                   0.5                    0.5
1                     55.0                   0.1                    0.1
2                    118.0                   0.2                    0.2
3                     89.0                   0.2                    0.2
4                     81.0                   0.4                    0.4

  EURPP_MBOEPer1000FT
0              208.38
1               75.69
2              153.90
3              126.91
4              184.44

[5 rows x 505 columns]
```

4.4 Clean the metadata

Remove columns with more than 50% of missing values.

```
[ ]: # Given columns names
columns = [
    'API_UWI', 'Unformatted_API_UWI', 'API_UWI_12', 'Unformatted_API_UWI_12',
    'API_UWI_14', 'Unformatted_API_UWI_14', 'WellID', 'CompletionID',
    'WellPadID', 'WellPadDirection', 'EURWH_MBOE_60Per1000FT',
    'EURWH_MBOE_120Per1000FT', 'EURWH_MBOE_180Per1000FT',␣
  ↪'EURWH_MBOE_360PEr1000FT',
    'EURWH_BCFE_360Per1000FT', 'OilEURWH_MBBLPer1000FT',␣
  ↪'EURWH_MBBL_360Per1000FT',
    'GasEURWH_BCFPer1000FT', 'EURWH_BCF_360Per1000FT', 'EURPP_MBOEPer1000FT'
]

# Assuming df_metadata is your DataFrame
def clean_metadata(df, threshold_percentage=0.5, columns_to_keep=[]):
    threshold = threshold_percentage / 100 * len(df)  # Calculate threshold as␣
  ↪a number of rows
    for column in df.columns:
        if column not in columns_to_keep and df[column].isnull().mean() >␣
  ↪threshold / 100:
            df.drop(column, axis=1, inplace=True)
    return df
```

```
# Specify which columns you want to keep, regardless of their missing values
columns_to_keep = [
    'API_UWI', 'Unformatted_API_UWI', 'API_UWI_12', 'Unformatted_API_UWI_12',
    'API_UWI_14', 'Unformatted_API_UWI_14', 'WellID', 'CompletionID'
]

# Cleaning the DataFrame
metadata_cleaned = clean_metadata(metadata, 0.5, columns_to_keep)
```

[ ]:
```
metadata_cleaned.columns
```

[ ]:
```
Index(['API_UWI', 'Unformatted_API_UWI', 'API_UWI_12',
       'Unformatted_API_UWI_12', 'API_UWI_14', 'Unformatted_API_UWI_14',
       'WellID', 'CompletionID', 'WellPadID', 'WellPadDirection',
       …
       'EURWH_MBOE_60Per1000FT', 'EURWH_MBOE_120Per1000FT',
       'EURWH_MBOE_180Per1000FT', 'EURWH_MBOE_360PEr1000FT',
       'EURWH_BCFE_360Per1000FT', 'OilEURWH_MBBLPer1000FT',
       'EURWH_MBBL_360Per1000FT', 'GasEURWH_BCFPer1000FT',
       'EURWH_BCF_360Per1000FT', 'EURPP_MBOEPer1000FT'],
      dtype='object', length=345)
```

[ ]:
```
# Convert 'object' type columns to 'category' type where appropriate
for col in metadata_cleaned.columns:
    if metadata_cleaned[col].dtype == 'object':
        # Convert to category if the number of unique values is significantly␣
↪less than the total number of values
        if metadata_cleaned[col].nunique() / len(metadata_cleaned[col]) < 0.5: ␣
↪# Adjust the threshold as needed
            metadata_cleaned[col] = metadata_cleaned[col].astype('category')

print(metadata_cleaned.dtypes)
```

```
API_UWI                     object
Unformatted_API_UWI          int64
API_UWI_12                  object
Unformatted_API_UWI_12       int64
API_UWI_14                  object
                            …
OilEURWH_MBBLPer1000FT     float64
EURWH_MBBL_360Per1000FT    float64
GasEURWH_BCFPer1000FT      float64
EURWH_BCF_360Per1000FT     float64
EURPP_MBOEPer1000FT        float64
Length: 345, dtype: object
```

[ ]:
```
# Convert date columns to datetime format
production_data['ProducingMonth'] = pd.
↪to_datetime(production_data['ProducingMonth'], errors='coerce')

# Fill missing values or drop rows with missing dates or critical values
#production_data.dropna(subset=['ProducingMonth', 'OilProduction',␣
↪'GasProduction'], inplace=True)

# Convert columns to float, setting errors to 'coerce' to handle invalid parsing
numeric_columns = ['Prod_BOE', 'Prod_MCFE', 'LiquidsProd_BBL', 'GasProd_MCF',␣
↪'WaterProd_BBL',
                   'RepGasProd_MCF', 'CDProd_BOEPerDAY', 'CDProd_MCFEPerDAY',␣
↪'CDLiquids_BBLPerDAY',
                   'CDGas_MCFPerDAY', 'CDWater_BBLPerDAY',␣
↪'CDRepGas_MCFPerDAY', 'PDProd_BOEPerDAY',
                   'PDProd_MCFEPerDAY', 'PDLiquids_BBLPerDAY',␣
↪'PDGas_MCFPerDAY', 'PDWater_BBLPerDAY',
                   'PDRepGas_MCFPerDAY', 'CumProd_BOE', 'CumProd_MCFE',␣
↪'CumLiquids_BBL', 'CumGas_MCF',
                   'CumWater_BBL', 'CumRepGas_MCF'
]

for col in numeric_columns:
    production_data[col] = pd.to_numeric(production_data[col], errors='coerce')

# After converting, you might want to check how many NaNs were introduced
print(production_data.isnull().sum())

# Example of handling missing data
for name, df in production_data.items():
    production_data[name] = df.dropna()  # or df.fillna(method='ffill') etc.

# Removing duplicates
production_data.drop_duplicates(inplace=True)

# Assuming there's a 'WellID' and 'Region' column for comparative analysis
# Ensure categorical columns are of type 'category'
production_data['WellID'] = production_data['WellID'].astype('category')

# Initial cleanup complete
production_data_cleaned = production_data
```

```
WellID                                             0
CompletionID                                       0
API_UWI                                            0
WellName                                           0
WellboreId                                         0
```

```
ProducingMonth                                  0
TotalProdMonths                                 0
TotalCompletionMonths                           0
ProducingDays                                   0
Prod_BOE                                        0
Prod_MCFE                                       0
LiquidsProd_BBL                                 0
GasProd_MCF                                     0
WaterProd_BBL                                   0
RepGasProd_MCF                                  0
CDProd_BOEPerDAY                                0
CDProd_MCFEPerDAY                               0
CDLiquids_BBLPerDAY                             0
CDGas_MCFPerDAY                                 0
CDWater_BBLPerDAY                               0
CDRepGas_MCFPerDAY                              0
PDProd_BOEPerDAY                                0
PDProd_MCFEPerDAY                               0
PDLiquids_BBLPerDAY                             0
PDGas_MCFPerDAY                                 0
PDWater_BBLPerDAY                               0
PDRepGas_MCFPerDAY                              0
CumProd_BOE                                     0
CumProd_MCFE                                    0
CumLiquids_BBL                                  0
CumGas_MCF                                      0
CumWater_BBL                                    0
CumRepGas_MCF                                   0
ProductionReportedMethod                        0
ProducingOperator                            1383
InjectionGas_MCF                           106835
InjectionSolvent_BBL                       106835
InjectionSteam_BBL                         106835
InjectionWater_BBL                         106835
InjectionOther_BBL                         106835
CalendarDayInjectionWater_BBLPerDAY        106835
CalendarDayInjectionSteam_BBLPerDAY        106835
CalendarDayInjectionSolvent_BBLPerDAY      106835
CalendarDayInjectionGas_MCFPerDAY          106835
CalendarDayInjectionOther_BBLPerDAY        106835
ENVProdID                                       0
dtype: int64
```

[ ]:
```
# Convert 'object' type columns to 'category' type where appropriate
for col in production_data_cleaned.columns:
    if production_data_cleaned[col].dtype == 'object':
```

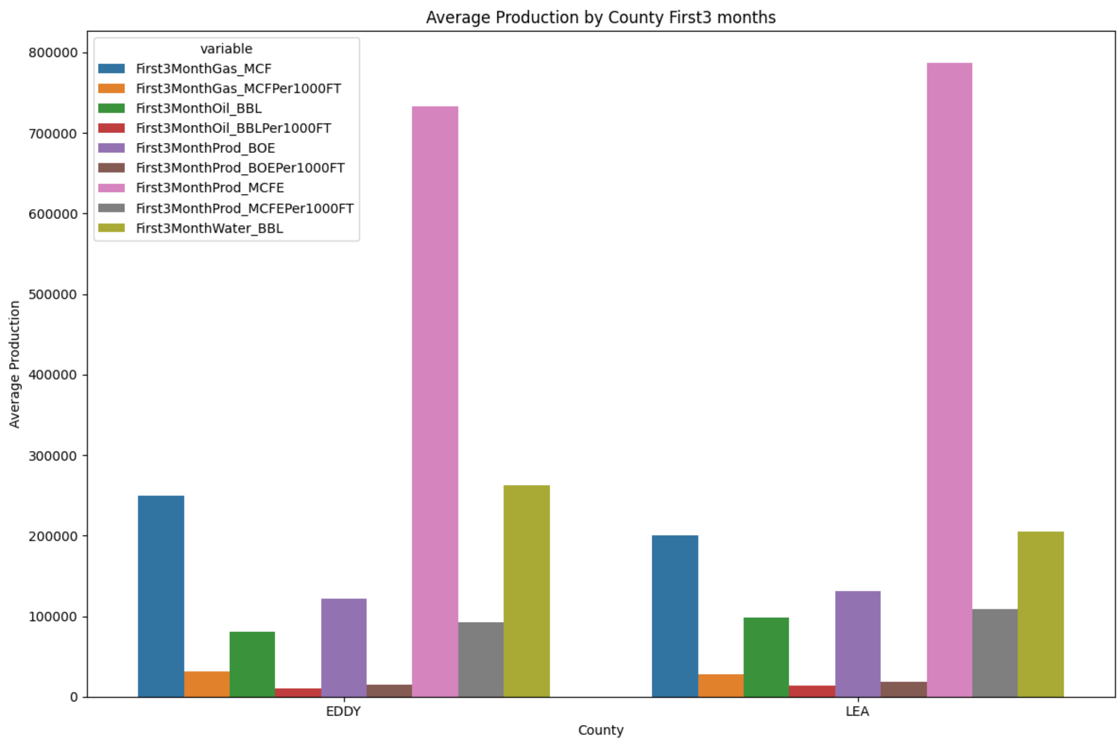
```
        # Convert to category if the number of unique values is significantly␣
↪less than the total number of values
        if production_data_cleaned[col].nunique() /␣
↪len(production_data_cleaned[col]) < 0.5:  # Adjust the threshold as needed
            production_data_cleaned[col] = production_data_cleaned[col].
↪astype('category')

# Display updated data types for verification
print(production_data_cleaned.dtypes)
```

```
WellID                                           category
CompletionID                                        int64
API_UWI                                          category
WellName                                         category
WellboreId                                          int64
ProducingMonth                             datetime64[ns]
TotalProdMonths                                     int64
TotalCompletionMonths                               int64
ProducingDays                                       int64
Prod_BOE                                            int64
Prod_MCFE                                           int64
LiquidsProd_BBL                                     int64
GasProd_MCF                                         int64
WaterProd_BBL                                       int64
RepGasProd_MCF                                      int64
CDProd_BOEPerDAY                                    int64
CDProd_MCFEPerDAY                                   int64
CDLiquids_BBLPerDAY                                 int64
CDGas_MCFPerDAY                                     int64
CDWater_BBLPerDAY                                   int64
CDRepGas_MCFPerDAY                                  int64
PDProd_BOEPerDAY                                    int64
PDProd_MCFEPerDAY                                   int64
PDLiquids_BBLPerDAY                                 int64
PDGas_MCFPerDAY                                     int64
PDWater_BBLPerDAY                                   int64
PDRepGas_MCFPerDAY                                  int64
CumProd_BOE                                         int64
CumProd_MCFE                                        int64
CumLiquids_BBL                                      int64
CumGas_MCF                                          int64
CumWater_BBL                                        int64
CumRepGas_MCF                                       int64
ProductionReportedMethod                         category
ProducingOperator                                category
InjectionGas_MCF                                  float64
InjectionSolvent_BBL                              float64
```

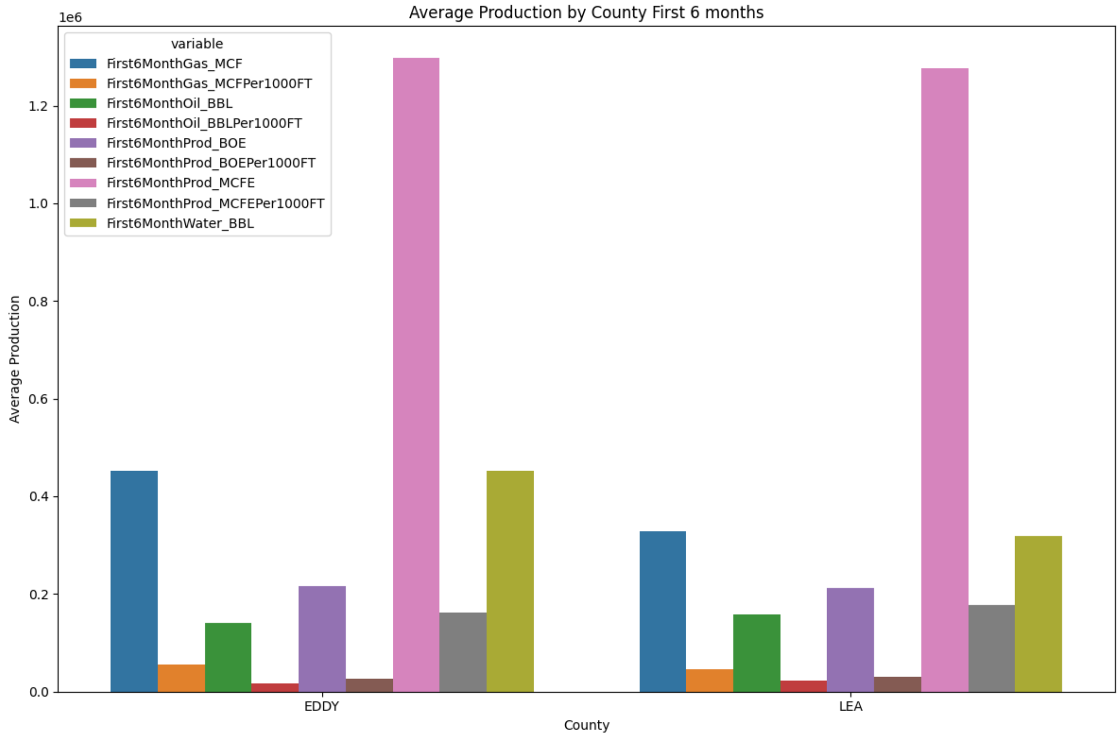
```
InjectionSteam_BBL                                  float64
InjectionWater_BBL                                  float64
InjectionOther_BBL                                  float64
CalendarDayInjectionWater_BBLPerDAY                 float64
CalendarDayInjectionSteam_BBLPerDAY                 float64
CalendarDayInjectionSolvent_BBLPerDAY               float64
CalendarDayInjectionGas_MCFPerDAY                   float64
CalendarDayInjectionOther_BBLPerDAY                 float64
ENVProdID                                             int64
dtype: object
```

4.5 Production my County

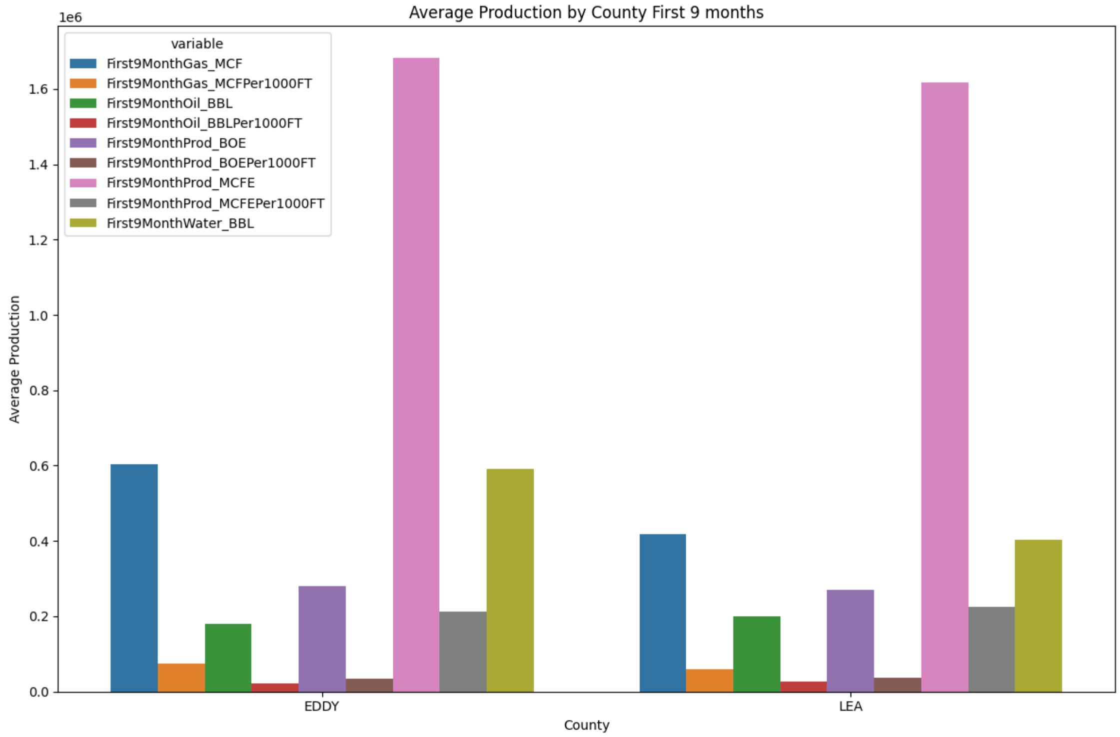
```
[ ]: filter_col = [col for col in metadata if col.startswith('First3M')]
     filter_col.append('County')
     first_df= metadata[filter_col]
     #first_df.drop(['FirstProdDate', 'FirstProdQuarter', 'FirstProdMonth',␣
      ↪'FirstRigDay','FirstProdYear'], axis=1, inplace=True)
     first_df=pd.melt(first_df, ['County'])
     first_df.head()
     first_df=first_df.groupby(['County','variable'])['value'].mean().reset_index()
     #first_df=first_df.to_frame()

     # plot with seaborn barplot
     plt.figure(figsize=(12, 8))
     sns.barplot(data=first_df, x='County', y='value', hue='variable')
     # plt.figure(figsize=(12, 8))
     # first_df.plot(kind='barh', color='')
     plt.xlabel('County')
     plt.ylabel('Average Production')
     plt.title('Average Production by County First3 months')
     plt.tight_layout()
     plt.show()
```




```
[ ]: filter_col = [col for col in metadata if col.startswith('First6M')]
filter_col.append('County')
first_df= metadata[filter_col]
#first_df.drop(['FirstProdDate', 'FirstProdQuarter', 'FirstProdMonth',␣
↪'FirstRigDay','FirstProdYear'], axis=1, inplace=True)
first_df=pd.melt(first_df, ['County'])
first_df.head()
first_df=first_df.groupby(['County','variable'])['value'].mean().reset_index()
#first_df=first_df.to_frame()

# plot with seaborn barplot
plt.figure(figsize=(12, 8))
sns.barplot(data=first_df, x='County', y='value', hue='variable')
# plt.figure(figsize=(12, 8))
# first_df.plot(kind='barh', color='')
plt.xlabel('County')
plt.ylabel('Average Production')
plt.title('Average Production by County First 6 months')
plt.tight_layout()
plt.show()
```




```
[ ]: filter_col = [col for col in metadata if col.startswith('First9M')]
     filter_col.append('County')
     first_df= metadata[filter_col]
     #first_df.drop(['FirstProdDate', 'FirstProdQuarter', 'FirstProdMonth',␣
      ↪'FirstRigDay','FirstProdYear'], axis=1, inplace=True)
     first_df=pd.melt(first_df, ['County'])
     first_df.head()
     first_df=first_df.groupby(['County','variable'])['value'].mean().reset_index()
     #first_df=first_df.to_frame()

     # plot with seaborn barplot
     plt.figure(figsize=(12, 8))
     sns.barplot(data=first_df, x='County', y='value', hue='variable')
     # plt.figure(figsize=(12, 8))
     # first_df.plot(kind='barh', color='')
     plt.xlabel('County')
     plt.ylabel('Average Production')
     plt.title('Average Production by County First 9 months')
     plt.tight_layout()
     plt.show()
```



[ ]:
```
# filter_col = [col for col in metadata if col.startswith('First12M')]
# filter_col.append('County')
# first_df= metadata[filter_col]
# #first_df.drop(['FirstProdDate', 'FirstProdQuarter', 'FirstProdMonth',␣
↪'FirstRigDay','FirstProdYear'], axis=1, inplace=True)
# first_df=pd.melt(first_df, ['County'])
# first_df.head()
# first_df=first_df.groupby(['County','variable'])['value'].mean().reset_index()
# #first_df=first_df.to_frame()
# first_df
# # plot with seaborn barplot
# plt.figure(figsize=(12, 8))
# sns.barplot(data=first_df, x='County', y='value', hue='variable')
# # plt.figure(figsize=(12, 8))
# # first_df.plot(kind='barh', color='')
# plt.xlabel('County')
# plt.ylabel('Average Production')
# plt.title('Average Production by County First 12 months')
# plt.tight_layout()
# plt.show()
```

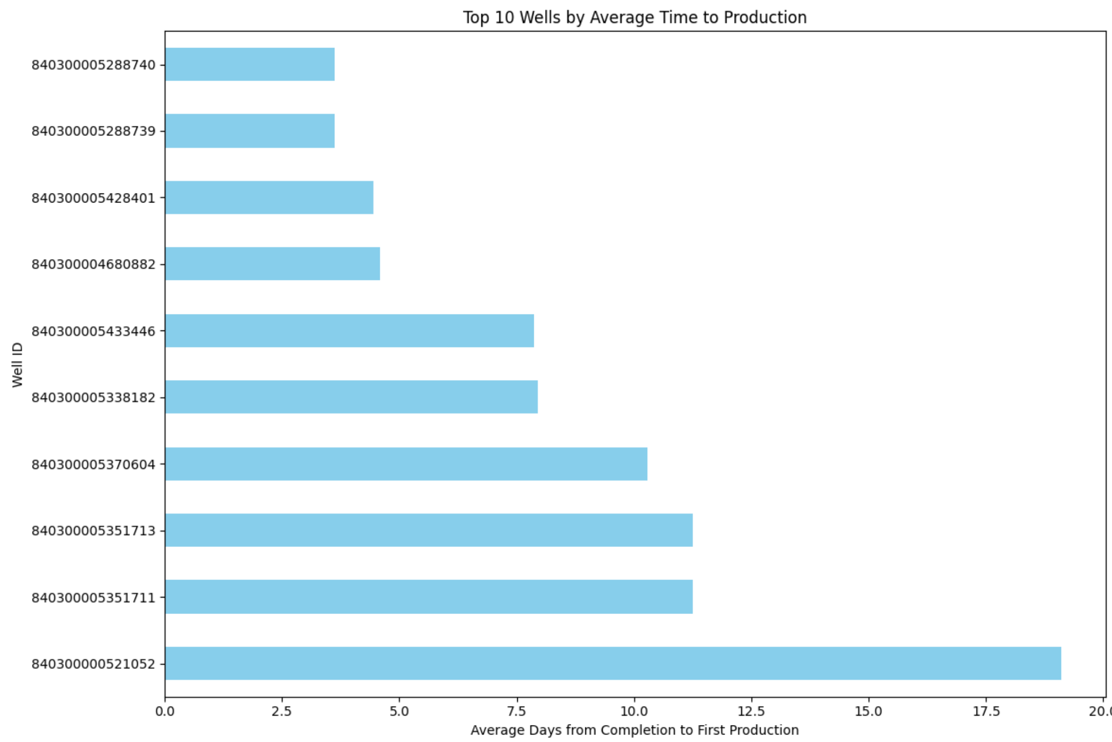
[ ]:
```
# Assuming df_metadata_cleaned is your cleaned DataFrame
num_columns = len(production_data_cleaned.columns)
```

```
print(f"The cleaned DataFrame has {num_columns} columns.")
```

```
The cleaned DataFrame has 46 columns.
```

4.6 Number of days in production

```
[ ]: # Average time to production
import matplotlib.pyplot as plt

# Assuming merged_df is your DataFrame after merging and calculating␣
↪'TimeToProduction'

merged_df = pd.merge(metadata_cleaned, production_data_cleaned, on='WellID')

# Calculate the difference between first production and completion dates to get␣
↪the time to production in days
# Correcting the calculation by removing .dt.days for numeric columns
merged_df['TimeToProduction'] = merged_df['TotalProdMonths'] -␣
↪merged_df['TotalCompletionMonths']

# Calculate the average time to production for each WellID, then select the top␣
↪10
top_wells = merged_df.groupby('WellID')['TimeToProduction'].mean().nlargest(10)

plt.figure(figsize=(12, 8))
top_wells.plot(kind='barh', color='skyblue')
plt.xlabel('Average Days from Completion to First Production')
plt.ylabel('Well ID')
plt.title('Top 10 Wells by Average Time to Production')
plt.tight_layout()
plt.show()
```



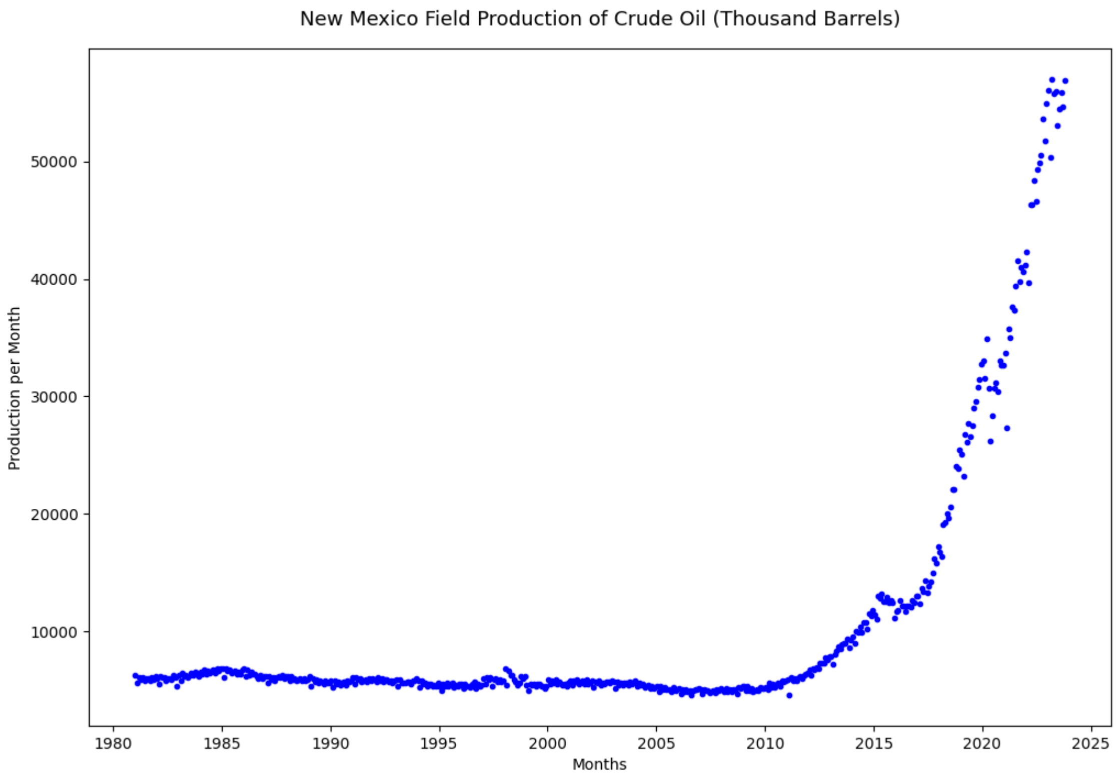
4.7 Oil Production in new mexico

[ ]:
```
# load data

df = pd.read_excel(datapath / 'MCRFPNM1m.xls',sheet_name="Data 1", skiprows= 2 )

# rename the columns
df.columns= ["Date", "Barrels"]

#df=df.loc[(df['Date'] >= '2015-06-15')]

t = df['Date']
q = df['Barrels']

# display the data
df.info()
```

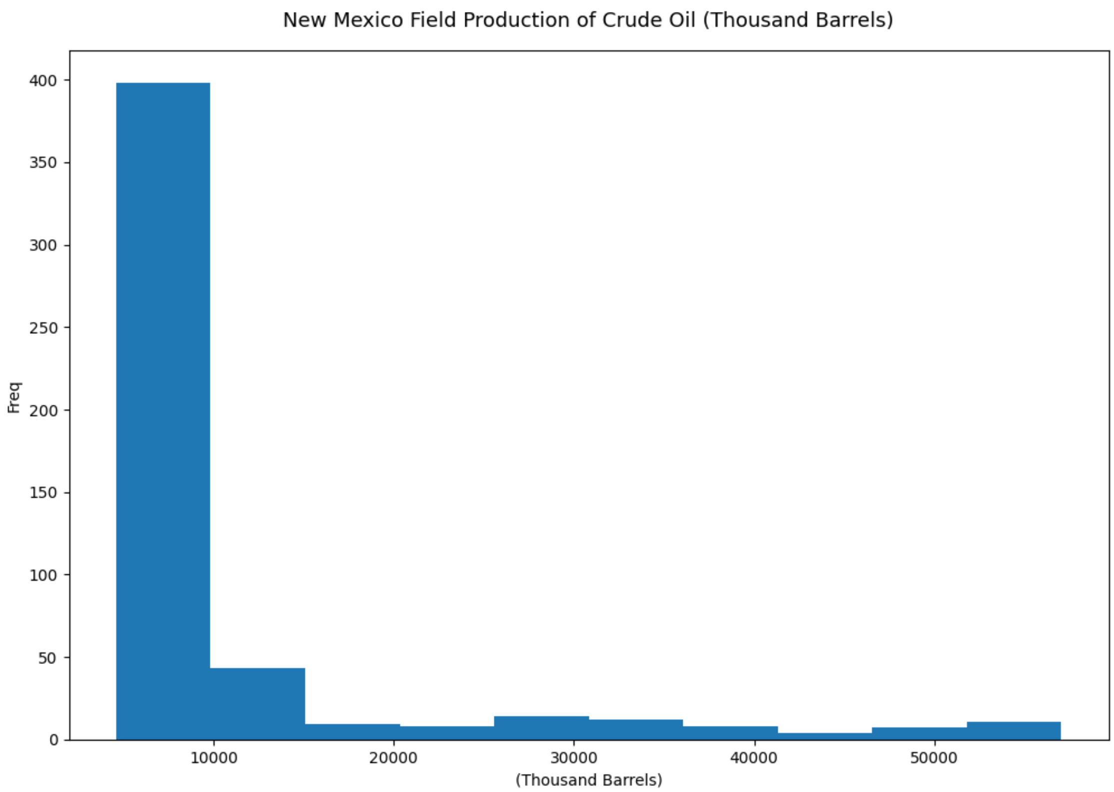
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 514 entries, 0 to 513
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Date     514 non-null    datetime64[ns]
```

```
 1   Barrels  514 non-null    int64
dtypes: datetime64[ns](1), int64(1)
memory usage: 8.2 KB
```

[ ]:
```
plt.figure(figsize=(12, 8))
plt.plot(t,q, '.', color='blue')
plt.title('New Mexico Field Production of Crude Oil (Thousand Barrels)',␣
 ↪size=13, pad=15)
plt.xlabel('Months')
plt.ylabel('Production per Month')
plt.show()
```



[ ]:
```
plt.figure(figsize=(12, 8))
plt.hist(q)
plt.title('New Mexico Field Production of Crude Oil (Thousand Barrels)',␣
 ↪size=13, pad=15)
plt.ylabel('Freq')
plt.xlabel('(Thousand Barrels)')
plt.show()
```



We see a sharp increase in oil production as from 2010. The histogram shows that most of the oil production per day in new mexico is below 10000 barrels.

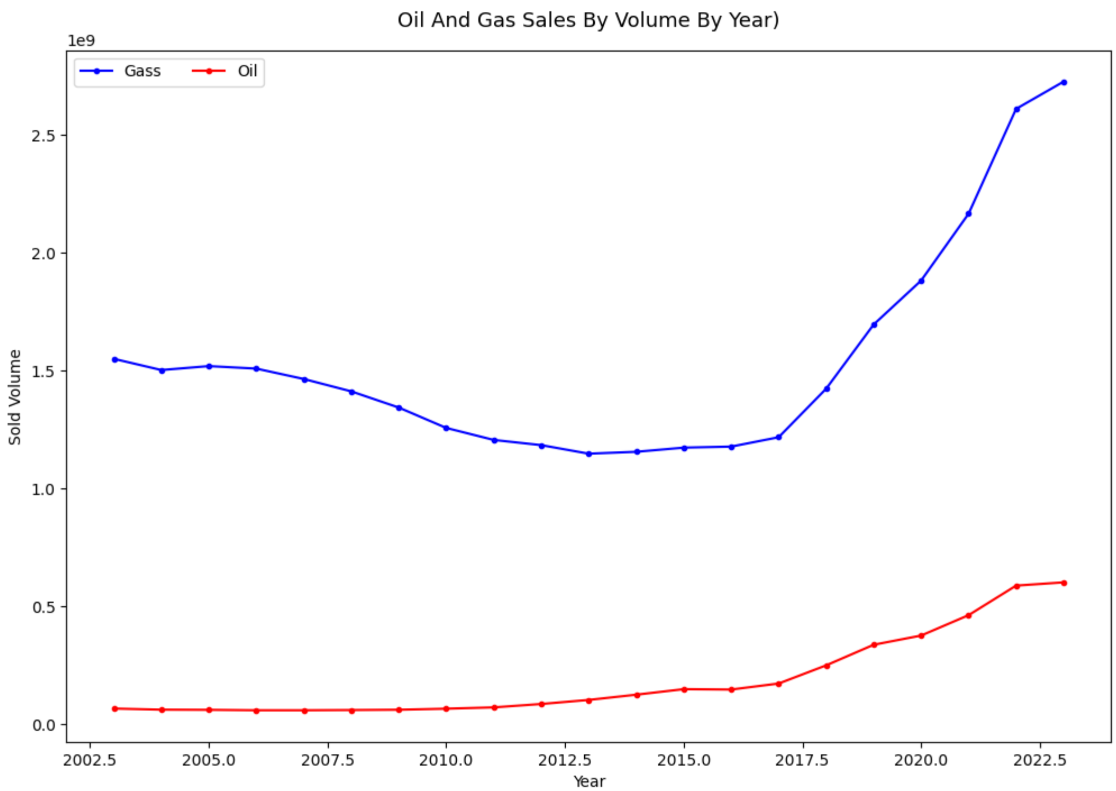
```
[ ]: oil_and_gas_sales_volume = pd.read_excel(datapath /␣
      ↪'oil_and_gas_sales_by_volume_by_year.xlsx', skiprows=8)

     columns_to_keep2= ['Year', 'Gas Sold*', 'Year.1', 'Oil Sold**']
     clean_metadata(oil_and_gas_sales_volume, 0.5, columns_to_keep2)
     oil_and_gas_sales_volume['Gas Sold*'] = oil_and_gas_sales_volume['Gas Sold*'].
      ↪str.replace(',', '').astype(float)
     oil_and_gas_sales_volume['Oil Sold**'] = oil_and_gas_sales_volume['Oil Sold**'].
      ↪str.replace(',', '').astype(float)
     oil_and_gas_sales_volume.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Year        21 non-null     int64
 1   Gas Sold*   21 non-null     float64
 2   Year.1      21 non-null     int64
```

```
 3   Oil Sold**  21 non-null     float64
dtypes: float64(2), int64(2)
memory usage: 800.0 bytes
```

[ ]:

```
plt.figure(figsize=(12, 8))
plt.plot(oil_and_gas_sales_volume['Year'],oil_and_gas_sales_volume['Gas␣
 ↪Sold*'], '.', color='blue', label="Gass", ls='solid')
plt.plot(oil_and_gas_sales_volume['Year.1'],oil_and_gas_sales_volume['Oil␣
 ↪Sold**'], '.', color='red', label="Oil", ls= 'solid')
plt.title('Oil And Gas Sales By Volume By Year)', size=13, pad=15)
plt.xlabel('Year')
plt.ylabel('Sold Volume')
plt.legend(ncol=2)
plt.show()
```



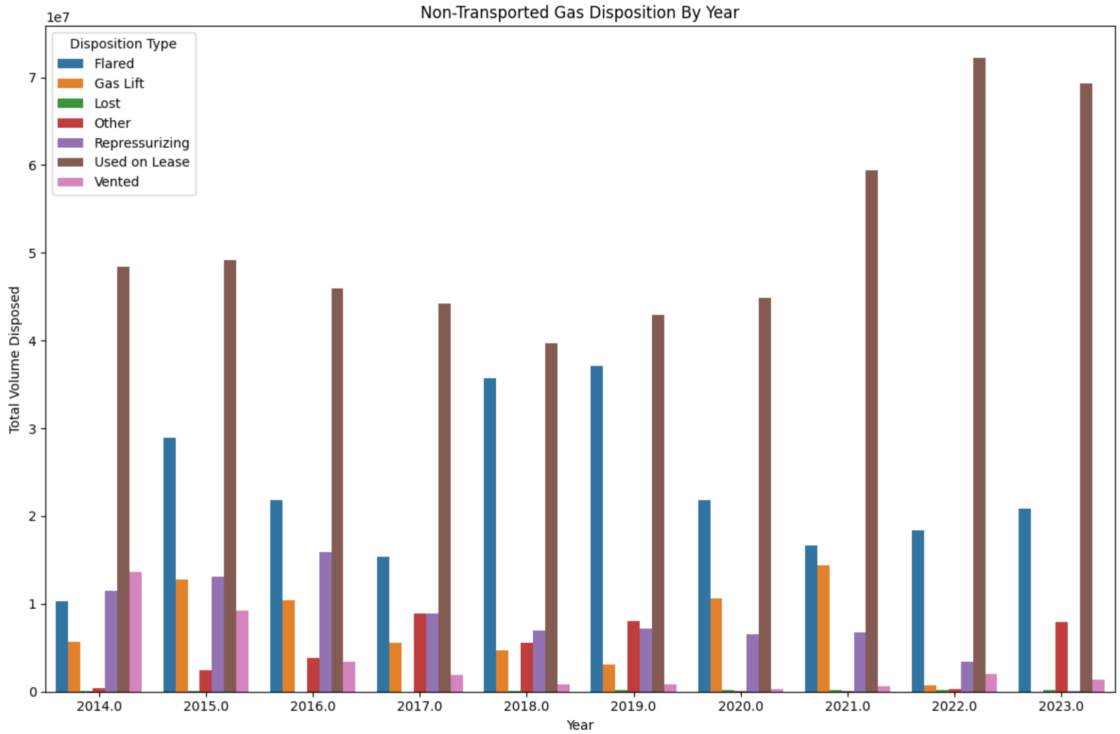
The plot shows a decrease in gas sold between 2002 and 2015 after which there is an upward increase to 2022. on the other hand oil sold has been on upward trend since 2002. This is an indication that oil demand is on constant upward trend.

4.8 Flaring and Venting data

[ ]:
```
flaring_and_venting_data_operator_year = pd.read_excel(datapath /␣
↪'c-115_flaring_and_venting_data_from_2014_operator_and_year.xlsx',
                                                       skiprows=7)
flaring_and_venting_data_operator_year.columns
filter_col_out = [col for col in flaring_and_venting_data_operator_year if col.
↪startswith('Unnamed')]
flaring_and_venting_data_operator_year.drop(filter_col_out, axis=1,␣
↪inplace=True)
flaring_and_venting_data_operator_year.dropna(inplace=True)
flaring_and_venting_data_operator_year.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 28847 entries, 1 to 28848
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   OGRID             28847 non-null  float64
 1   Operator          28847 non-null  object
 2   Year              28847 non-null  float64
 3   Month             28847 non-null  object
 4   District          28847 non-null  float64
 5   Disposition Type  28847 non-null  object
 6   Volume            28847 non-null  float64
dtypes: float64(4), object(3)
memory usage: 1.8+ MB
```

[ ]:
```
flare_group= flaring_and_venting_data_operator_year.
↪groupby(['Year','Disposition Type'])['Volume'].sum().reset_index()
# plot with seaborn barplot
plt.figure(figsize=(12, 8))
sns.barplot(data=flare_group, x='Year', y='Volume', hue='Disposition Type')
# plt.figure(figsize=(12, 8))
# first_df.plot(kind='barh', color='')
plt.xlabel('Year')
plt.ylabel('Total Volume Disposed')
plt.title('Non-Transported Gas Disposition By Year')
plt.tight_layout()
plt.show()
```



The image you sent me is a graph of non-transported gas disposals by year. The graph shows that the number of non-transported gas disposals has been increasing over time, with a particularly sharp increase in recent years. The main findings from the image are:

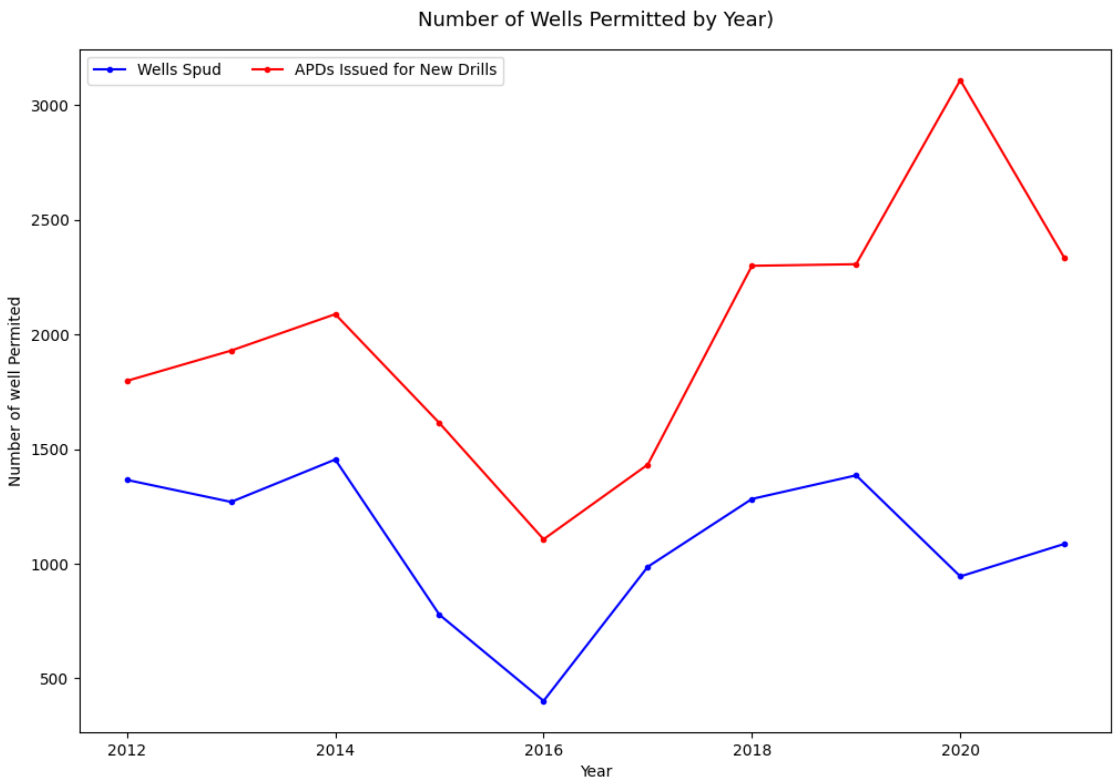
The number of non-transported gas disposals has increased from around 1e6 in 2014 to around 6e7 in 2023. The increase in non-transported gas disposals could have been driven by a number of factors, such as increased oil and gas production, a lack of infrastructure to transport gas to market, and environmental regulations that limit the amount of gas that can be flared or vented.The increase in non-transported gas disposals has a number of environmental and economic consequences, including greenhouse gas emissions, air pollution, and lost revenue.

4.9 OCD Well Statistics

```
[ ]: Oil_stats=  pd.read_excel(datapath / '2021-OCD-Well-Statistics.xlsx',
                                                          skiprows=48,nrows=10 )
     keep= ['Calendar Year', 'Wells Spud**', 'APDs Issued for New Drills']
     Oil_stats= clean_metadata(Oil_stats, 0.5, keep)
```

```
[ ]: plt.figure(figsize=(12, 8))
     plt.plot(Oil_stats['Calendar Year'],Oil_stats['Wells Spud**'], '.',␣
      ↪color='blue', label="Wells Spud", ls= 'solid')
     plt.plot(Oil_stats['Calendar Year'],Oil_stats['APDs Issued for New Drills'], '.
      ↪', color='red', label="APDs Issued for New Drills",
             ls='solid')
```

```
plt.title('Number of Wells Permitted by Year)', size=13, pad=15)
plt.xlabel('Year')
plt.ylabel('Number of well Permited')
plt.legend(ncol=2)
plt.show()
```



We see an increase in number of permissions given for drilling as from 2016.

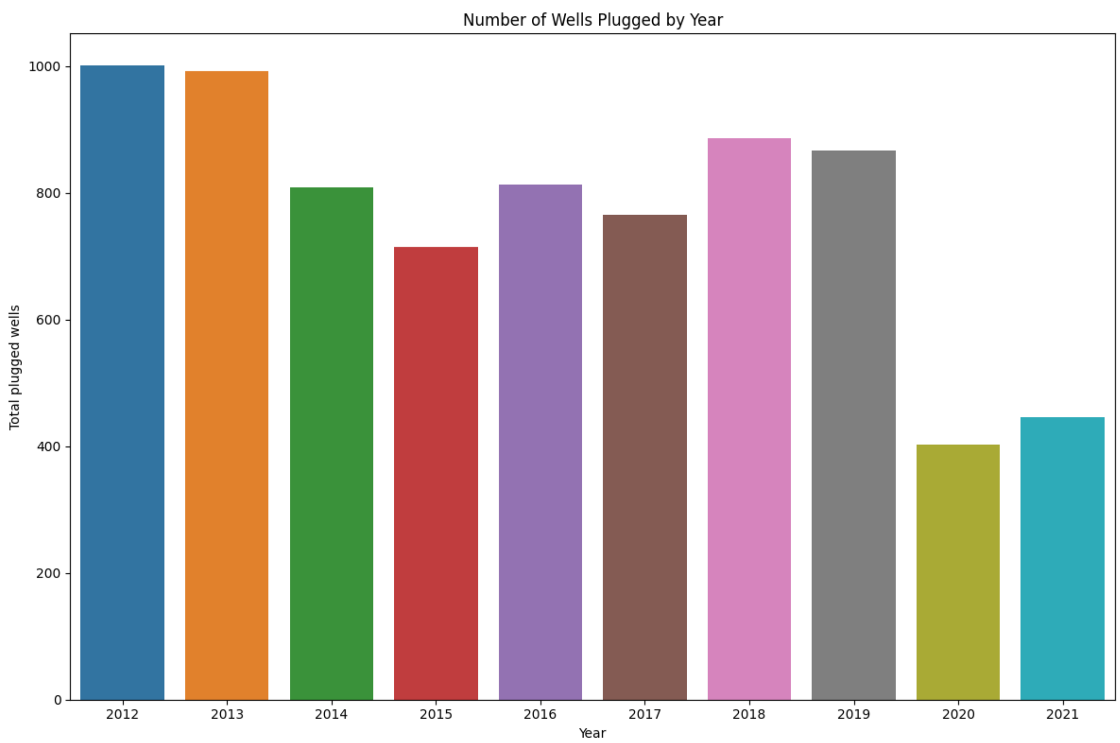
```
[ ]: Well_plugged=  pd.read_excel(datapath / '2021-OCD-Well-Statistics.xlsx',
                                          skiprows=63,nrows=11 )
     Well_plugged
     keep= ['Calendar Year', 'Wells Plugged and Site Released*']
     Well_plugged= clean_metadata(Well_plugged, 0.5, keep)
     Well_plugged
```

[ ]:

	Calendar Year	Wells Plugged and Site Released*
0	2012	1001
1	2013	992
2	2014	809
3	2015	714
4	2016	813
5	2017	765

6	2018	886
7	2019	867
8	2020	403
9	2021	446

[ ]:
```
plt.figure(figsize=(12, 8))
sns.barplot(data=Well_plugged, x='Calendar Year', y='Wells Plugged and Site␣
 ↪Released*')
# plt.figure(figsize=(12, 8))
# first_df.plot(kind='barh', color='')
plt.xlabel('Year')
plt.ylabel('Total plugged wells')
plt.title('Number of Wells Plugged by Year')
plt.tight_layout()
plt.show()
```



We see a decrease in the number of plugged wells in the recent years. This could be as a result of increase in demand for oil and gas in the recent years.

4.10 Production between 2008 and 2018

[ ]:
```
df_sum= df_oil.drop("Month",axis=1, inplace=False)
s= df_sum.sum().sort_values(ascending=False)
s.index[1:10]
Top_ten= df_oil[s.index[1:10]]
#Top_ten["Month"]= pd.to_datetime(df_oil['Month'])

Top_ten.head()
```

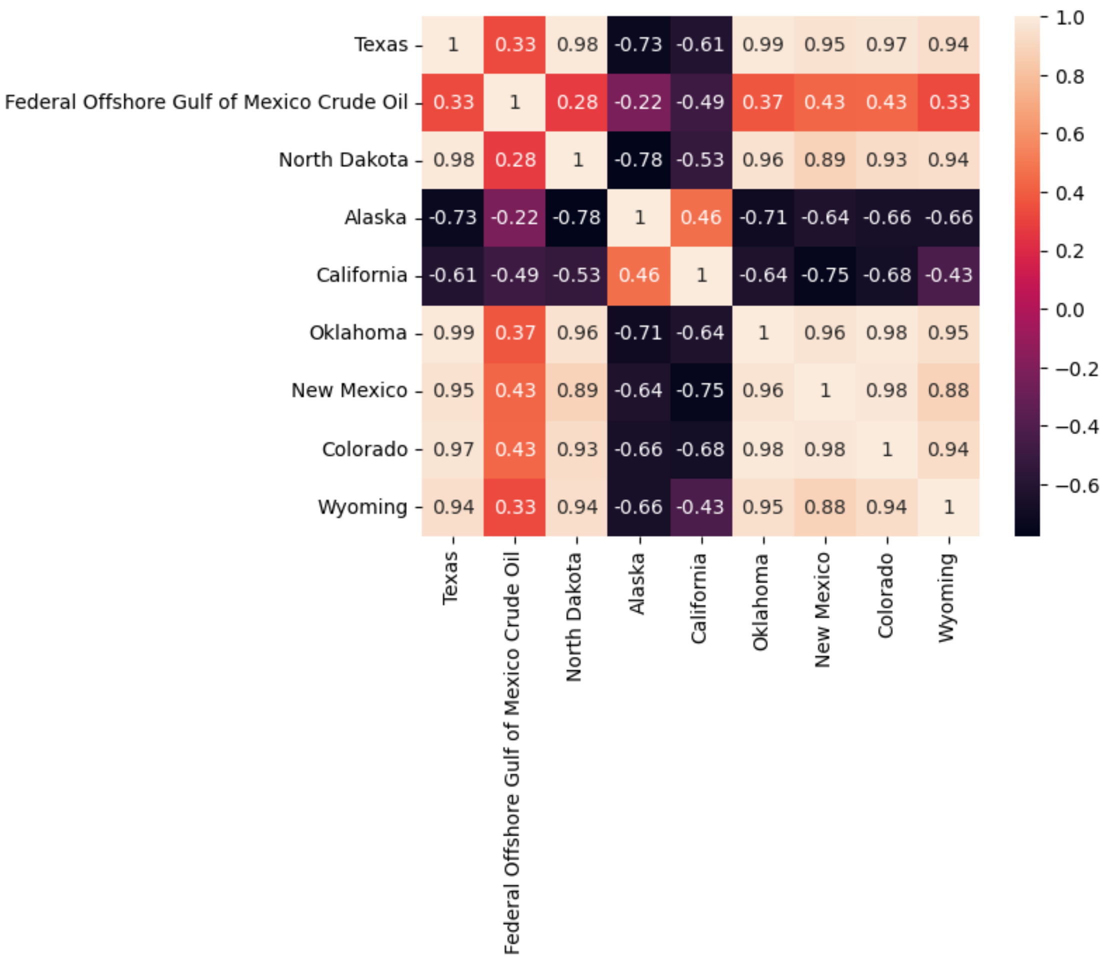
[ ]:
```
   Texas  Federal Offshore Gulf of Mexico Crude Oil  North Dakota  Alaska  \
0   1097                                       1326           165     655
1   1111                                       1372           172     640
2   1110                                       1272           178     544
3   1055                                        242           189     681
4   1125                                        803           203     716

   California  Oklahoma  New Mexico  Colorado  Wyoming
0         583       186         161        82      144
1         586       184         163        81      145
2         588       188         163        82      145
3         587       186         157        88      144
4         586       185         169        86      145
```

[ ]:
```
## Top ten states in term of production
s[1:10]
```

[ ]:
```
Texas                                         290863
Federal Offshore Gulf of Mexico Crude Oil     174756
North Dakota                                   90312
Alaska                                         65389
California                                     64903
Oklahoma                                       38915
New Mexico                                     36965
Colorado                                       25666
Wyoming                                        21832
dtype: int64
```

4.11 Correlation

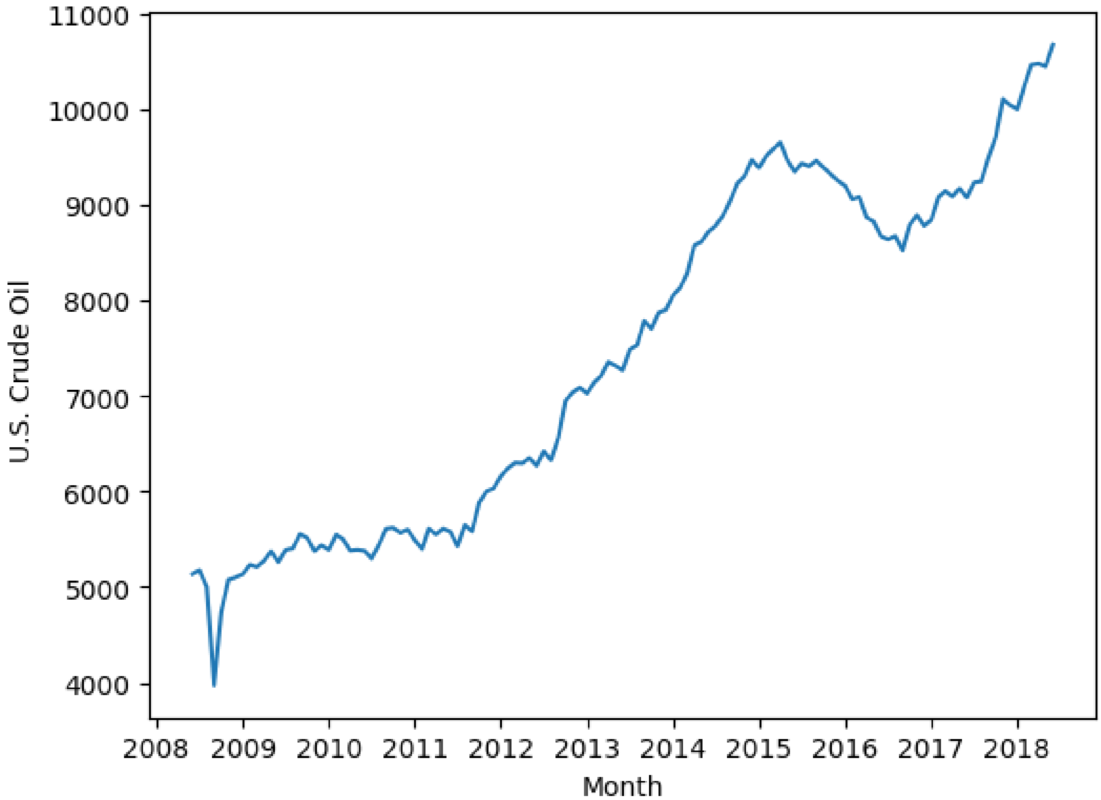
[ ]:
```
import seaborn as sns

ax = sns.heatmap(Top_ten.corr(), annot=True)
```



4.12 Trend Analysis

[ ]:
```
us_df=df_oil[['Month', 'U.S. Crude Oil ']]
us_df["Month"]= pd.to_datetime(us_df['Month'])
sns.lineplot(x='Month', y='U.S. Crude Oil ',
             data=us_df)
plt.show()
```

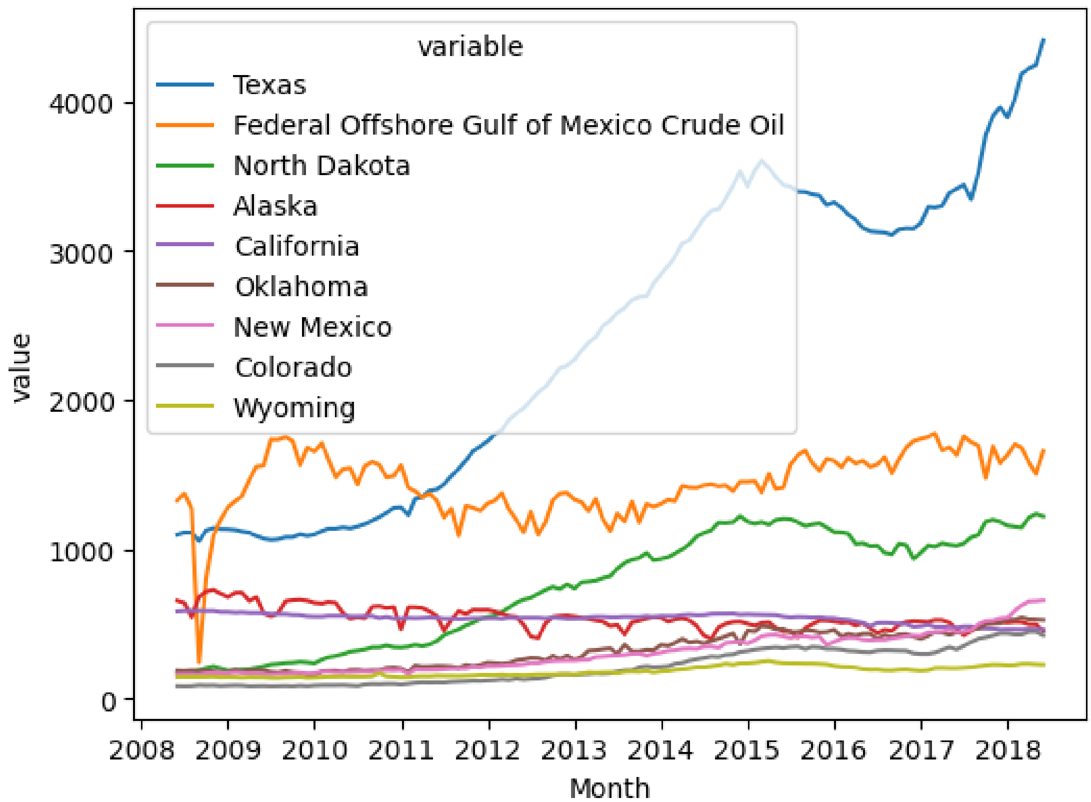
```
/tmp/ipykernel_954730/3959123548.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  us_df["Month"]= pd.to_datetime(us_df['Month'])
```



4.13 Trend analysis by state

[ ]:
```
#Line plot
Top_ten["Month"]= pd.to_datetime(df_oil['Month'])
sns.lineplot(x='Month', y='value', hue='variable',
             data=pd.melt(Top_ten, ['Month']))
plt.show()
```

```
/tmp/ipykernel_954730/3007367102.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  Top_ten["Month"]= pd.to_datetime(df_oil['Month'])
```



4.14 Descriptive statistics

[ ]:
```
#USA Production
us_df['U.S. Crude Oil '].describe()
```

[ ]:
```
count      121.000000
mean      7423.694215
std       1801.581601
min       3974.000000
25%       5555.000000
50%       7355.000000
75%       9085.000000
max      10674.000000
Name: U.S. Crude Oil , dtype: float64
```

[ ]:
```
# State prduction
Top_ten.describe()
```

[ ]:
```
             Texas  Federal Offshore Gulf of Mexico Crude Oil  North Dakota  \
count   121.000000                                 121.000000    121.000000
mean   2403.826446                                1444.264463    746.380165
```

```
min     1055.000000                                   242.000000    165.000000
25%     1243.000000                                  1322.000000    343.000000
50%     2533.000000                                  1452.000000    820.000000
75%     3301.000000                                  1593.000000   1096.000000
max     4410.000000                                  1775.000000   1236.000000
std     1025.413694                                   216.197200    374.123158

           Alaska  California    Oklahoma  New Mexico    Colorado     Wyoming  \
count  121.000000  121.000000  121.000000  121.000000  121.000000  121.000000
mean   540.404959  536.388430  321.611570  305.495868  212.115702  180.429752
min    398.000000  461.000000  152.000000  157.000000   81.000000  137.000000
25%    497.000000  534.000000  201.000000  188.000000   97.000000  147.000000
50%    523.000000  544.000000  320.000000  285.000000  169.000000  174.000000
75%    582.000000  559.000000  433.000000  405.000000  320.000000  209.000000
max    728.000000  588.000000  543.000000  657.000000  447.000000  251.000000
std     73.691879   33.228595  119.141958  128.069846  117.403165   34.097367

                               Month
count                            121
mean   2013-06-01 03:34:12.892561920
min              2008-06-01 00:00:00
25%              2010-12-01 00:00:00
50%              2013-06-01 00:00:00
75%              2015-12-01 00:00:00
max              2018-06-01 00:00:00
std                              NaN
```

5 Details of how the answers have been derived from the data

The answers to the business question on leveraging Big Data to improve trading strategies, reduce operating costs, and increase profitability in oil and gas trading operations were derived through a thorough analysis of us oil production data. Historical production figures, market trends, and correlation analysis served as foundational elements in deriving insights. By leveraging Data analytics, patterns and trends in oil production and market dynamics were identified, guiding the optimization of decision making. Correlation analysis provided valuable insights into the relationships between oil-producing states, enabling the identification of states with promising investment prospects for trading rights. States exhibiting upward trends in oil production, such as Texas, Federal Offshore Gulf of Mexico, and North Dakota, were pinpointed as potential targets for investment, aligning with the goal of maximizing profitability. Furthermore, the analysis delved into cost reduction opportunities by identifying states with declining production trends, such as Oklahoma, New Mexico, and Alaska. Recommendations were derived from data anlysis, trend analysis, and correlation findings, providing a robust foundation for data-driven decision-making. By leveraging the wealth of available data, the company can optimize trading strategies, mitigate risks, and capitalize on emerging opportunities, ultimately enhancing operational efficiency and profitability in the oil and gas trading market.