Lab6_Part_1_Time

In [35]:

```
import pandas as pd
import matplotlib.pyplot as plt
```

Assignment 1: Date Formats and Date Parts

- First, convert the `date` column to datetime64, by any method.
- Then, create a column representing the time difference between the last date in the data and each date.
- Next, create columns for the date parts year, month, and weekday.
- Finally, format the date to Year-Month-Day (This will be a string/object).

In [3]:

```
transactions = pd.read_csv("transactions.csv")
```

In [4]:

```
transactions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83488 entries, 0 to 83487
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   date          83488 non-null  object
 1   store_nbr     83488 non-null  int64 
 2   transactions  83488 non-null  int64 
dtypes: int64(2), object(1)
memory usage: 1.9+ MB
```

In [5]:

```
# conversion with parse dates in read_csv

transactions = pd.read_csv("transactions.csv", parse_dates=["date"])
transactions
```

Out[5]:

	date	store_nbr	transactions
0	2013-01-01	25	770
1	2013-01-02	1	2111
2	2013-01-02	2	2358
3	2013-01-02	3	3487
4	2013-01-02	4	1922
...	...	...	...
83483	2017-08-15	50	2804
83484	2017-08-15	51	1573
83485	2017-08-15	52	2255
83486	2017-08-15	53	932
83487	2017-08-15	54	802

83488 rows × 3 columns

In [6]:
```
transactions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83488 entries, 0 to 83487
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   date          83488 non-null  datetime64[ns]
 1   store_nbr     83488 non-null  int64
 2   transactions  83488 non-null  int64
dtypes: datetime64[ns](1), int64(2)
memory usage: 1.9 MB
```

In [39]:
```
# conversion with to_datetime
```

In [40]:
```
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83488 entries, 0 to 83487
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   date          83488 non-null  datetime64[ns]
 1   store_nbr     83488 non-null  int64
 2   transactions  83488 non-null  int64
dtypes: datetime64[ns](1), int64(2)
memory usage: 1.9 MB
```

In [7]:
```
transactions_2 = pd.read_csv("transactions.csv")
transactions_2['date'] = pd.to_datetime(transactions_2['date'], errors='coerce
```

In [8]:
```
# conversion with to_datetime
transactions_2 = pd.read_csv("transactions.csv")
transactions_2['date'] = pd.to_datetime(
    transactions_2['date'],
    errors='coerce',
    infer_datetime_format=True)
transactions_2
```

Out[8]:

	date	store_nbr	transactions
0	2013-01-01	25	770
1	2013-01-02	1	2111
2	2013-01-02	2	2358
3	2013-01-02	3	3487
4	2013-01-02	4	1922
...	...	...	...
83483	2017-08-15	50	2804
83484	2017-08-15	51	1573
83485	2017-08-15	52	2255
83486	2017-08-15	53	932
83487	2017-08-15	54	802

83488 rows × 3 columns

In [9]:

```
transactions_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83488 entries, 0 to 83487
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   date          83488 non-null  datetime64[ns]
 1   store_nbr     83488 non-null  int64
 2   transactions  83488 non-null  int64
dtypes: datetime64[ns](1), int64(2)
memory usage: 1.9 MB
```

In [41]:

```
# conversion with astype
```

In [42]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83488 entries, 0 to 83487
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   date          83488 non-null  datetime64[ns]
 1   store_nbr     83488 non-null  int64
 2   transactions  83488 non-null  int64
dtypes: datetime64[ns](1), int64(2)
memory usage: 1.9 MB
```

In [10]:

```
transactions_3 = pd.read_csv("transactions.csv")
transactions_3 = transactions_3.astype({"date":"datetime64"})
transactions_3
```

Out[10]:

	date	store_nbr	transactions
0	2013-01-01	25	770
1	2013-01-02	1	2111
2	2013-01-02	2	2358
3	2013-01-02	3	3487
4	2013-01-02	4	1922
...	...	...	...
83483	2017-08-15	50	2804
83484	2017-08-15	51	1573
83485	2017-08-15	52	2255
83486	2017-08-15	53	932
83487	2017-08-15	54	802

83488 rows × 3 columns

In [11]:

```
transactions_3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83488 entries, 0 to 83487
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   date          83488 non-null  datetime64[ns]
 1   store_nbr     83488 non-null  int64
 2   transactions  83488 non-null  int64
dtypes: datetime64[ns](1), int64(2)
memory usage: 1.9 MB
```

In [43]:

```
```

Out[43]:

	date	store_nbr	transactions
0	2013-01-01	25	770
1	2013-01-02	1	2111
2	2013-01-02	2	2358
3	2013-01-02	3	3487
4	2013-01-02	4	1922

In [10]:

```
transactions.head(5)
```

Out[10]:

	date	store_nbr	transactions
0	2013-01-01	25	770
1	2013-01-02	1	2111
2	2013-01-02	2	2358
3	2013-01-02	3	3487
4	2013-01-02	4	1922

In [ ]:

```
# Calcualte the maximum datetime

```

In [12]:

```
max_date = transactions['date'].max()
max_date
```

Out[12]:

```
Timestamp('2017-08-15 00:00:00')
```

In [45]:

```
# Difference between date and max date



# Dateparts


# Format Date

```

Out[45]:

	date	store_nbr	transactions	time_to_last_date	year	month	day_of_week
0	2013-January-01	25	770	1687 days	2013	1	1
1	2013-January-02	1	2111	1686 days	2013	1	2
2	2013-January-02	2	2358	1686 days	2013	1	2
3	2013-January-02	3	3487	1686 days	2013	1	2
4	2013-January-02	4	1922	1686 days	2013	1	2

In [13]:

```
# Difference between date and max date
transactions["time_to_last_date"] = ""
transactions["time_to_last_date"] = max_date - transactions['date']
transactions

# Dateparts
transactions['year'] = transactions['date'].dt.strftime('%Y')
transactions['month'] = transactions['date'].dt.strftime('%m')
transactions['day_of_week'] = transactions['date'].dt.dayofweek

# Format Date
transactions['date'] = transactions['date'].dt.strftime('%Y-%B-%d')
transactions.head()
```

Out[13]:

	date	store_nbr	transactions	time_to_last_date	year	month	day_of_week
0	2013-January-01	25	770	1687 days	2013	01	1
1	2013-January-02	1	2111	1686 days	2013	01	2
2	2013-January-02	2	2358	1686 days	2013	01	2
3	2013-January-02	3	3487	1686 days	2013	01	2
4	2013-January-02	4	1922	1686 days	2013	01	2

Assignment 2: Time Arithmetic

max date in our data was three weeks after 2017-08-15.

- Can you add three weeks to the ‘time_to_last_date’ column?
- Then, calculate ‘weeks_to_last_date’ by dividing the number of days in ‘time_to_last_date’ by 7.

In [14]:

```
# overwrite previous transactions df

transactions = pd.read_csv("transactions.csv", parse_dates=["date"])
```

In [15]:

```
transactions.tail()
```

Out[15]:

	date	store_nbr	transactions
83483	2017-08-15	50	2804
83484	2017-08-15	51	1573
83485	2017-08-15	52	2255
83486	2017-08-15	53	932
83487	2017-08-15	54	802

In [16]:

```
# recreate columns from assignment 1 using assign

transactions = transactions.assign(
    year=transactions["date"].dt.year,
    month=transactions["date"].dt.month,
    day_of_week=transactions["date"].dt.dayofweek,
    time_to_last_date=transactions["date"].max() - transactions["date"],
)

transactions.head()
```

Out[16]:

	date	store_nbr	transactions	year	month	day_of_week	time_to_last_date
0	2013-01-01	25	770	2013	1	1	1687 days
1	2013-01-02	1	2111	2013	1	2	1686 days
2	2013-01-02	2	2358	2013	1	2	1686 days
3	2013-01-02	3	3487	2013	1	2	1686 days
4	2013-01-02	4	1922	2013	1	2	1686 days

In [33]:

```
# Add three weeks to time to last date column
# Then divide the timedelta (converted to integer) into integer weeks
```

Out[33]:

	date	store_nbr	transactions	year	month	day_of_week	time_to_last_date	weeks_to_last_d
0	2013-01-01	25	770	2013	1	1	1708 days	244.000
1	2013-01-02	1	2111	2013	1	2	1707 days	243.857
2	2013-01-02	2	2358	2013	1	2	1707 days	243.857
3	2013-01-02	3	3487	2013	1	2	1707 days	243.857
4	2013-01-02	4	1922	2013	1	2	1707 days	243.857

In [17]:

```
# Add three weeks to time to last date column
transactions['time_to_last_date'] = transactions['time_to_last_date'] + pd.to_

# Then divide the timedelta (converted to integer) into integer weeks
transactions['weeks_to_last_date'] = transactions['time_to_last_date'].dt.days

transactions.head()
```

Out[17]:

	date	store_nbr	transactions	year	month	day_of_week	time_to_last_date	weeks_to_last_d
0	2013-01-01	25	770	2013	1	1	1708 days	244.000
1	2013-01-02	1	2111	2013	1	2	1707 days	243.857
2	2013-01-02	2	2358	2013	1	2	1707 days	243.857
3	2013-01-02	3	3487	2013	1	2	1707 days	243.857
4	2013-01-02	4	1922	2013	1	2	1707 days	243.857

Assignment 3: Missing Time Series Data

Take a look at the mean value for the oil price using forward fill, backfill, and interpolation. Are they very different?

Then, plot the series with forward fill for:

- The year 2014.
- The month of December 2014.
- The days from December 1st to December 15th, 2014.

In [18]:
```
# Read in oil csv with date as index (and converted to datetime64)
oil = pd.read_csv("oil.csv",
                  index_col="date",
                  parse_dates=True)
```

In [19]:
```
# This is a synonym for datetime64

oil.index.dtype
```

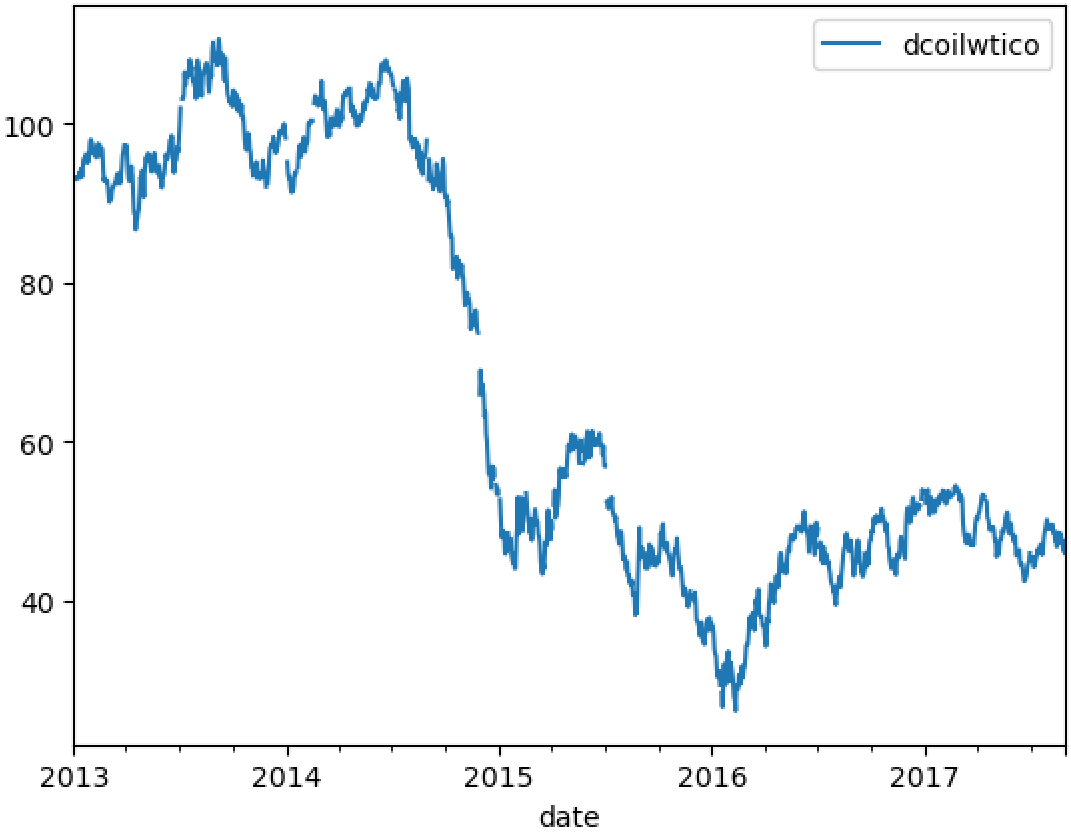
Out[19]:
```
dtype('<M8[ns]')
```

In [20]:
```
# mean of original series

oil.mean()
```

Out[20]:
```
dcoilwtico    67.714366
dtype: float64
```

In [21]:
```
# original plot

oil.plot()
```

Out[21]:
```
<AxesSubplot:xlabel='date'>
```

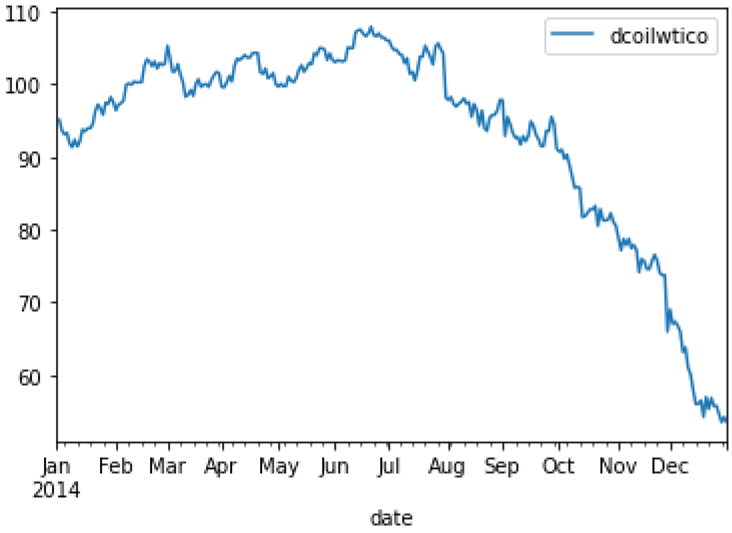


In [22]:
```
# mean of each type of missing value handling for time series

print(oil.ffill().mean(),
      oil.bfill().mean(),
      oil.interpolate().mean()
     )
```

```
dcoilwtico    67.671249
dtype: float64 dcoilwtico    67.673325
dtype: float64 dcoilwtico    67.661824
dtype: float64
```

In [27]:
```
# Filter to 2014 then plot forward filled Series
```

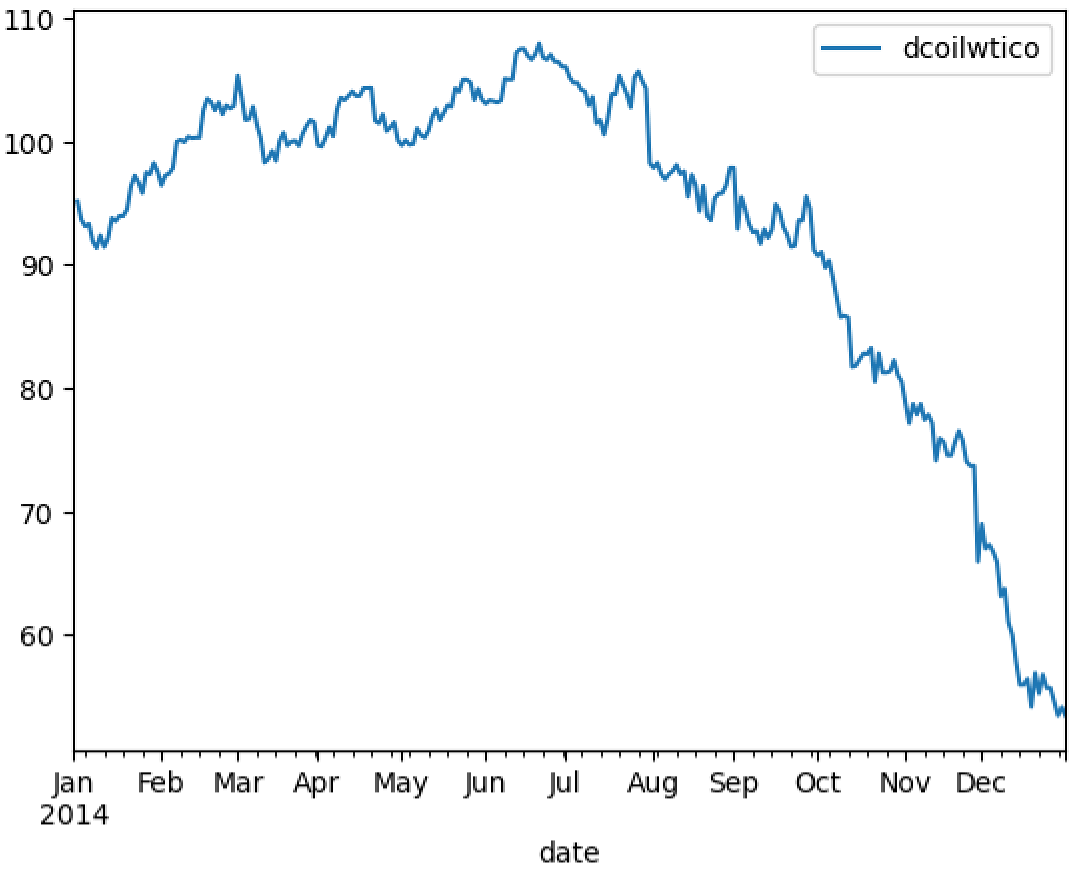
Out[27]:
```
<AxesSubplot:xlabel='date'>
```



In [24]:
```
# Filter to 2014 then plot forward filled Series

oil['2014'].ffill().plot()
```

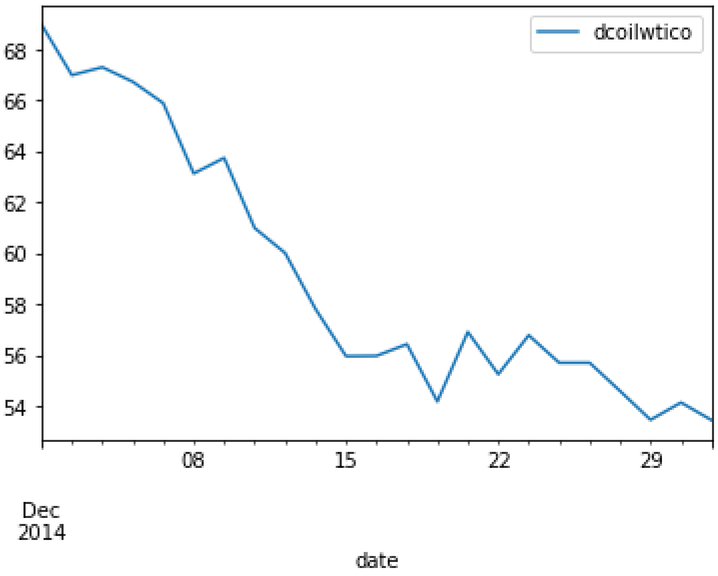
```
C:\Users\bhave\AppData\Local\Temp\ipykernel_29448\2682096312.py:3: FutureWarn
ing: Indexing a DataFrame with a datetimelike index using a single string to
slice the rows, like `frame[string]`, is deprecated and will be removed in a
future version. Use `frame.loc[string]` instead.
  oil['2014'].ffill().plot()
```

Out[24]:
```
<AxesSubplot:xlabel='date'>
```



In [28]:
```
# Filter to December 2014 then plot forward filled Series
```

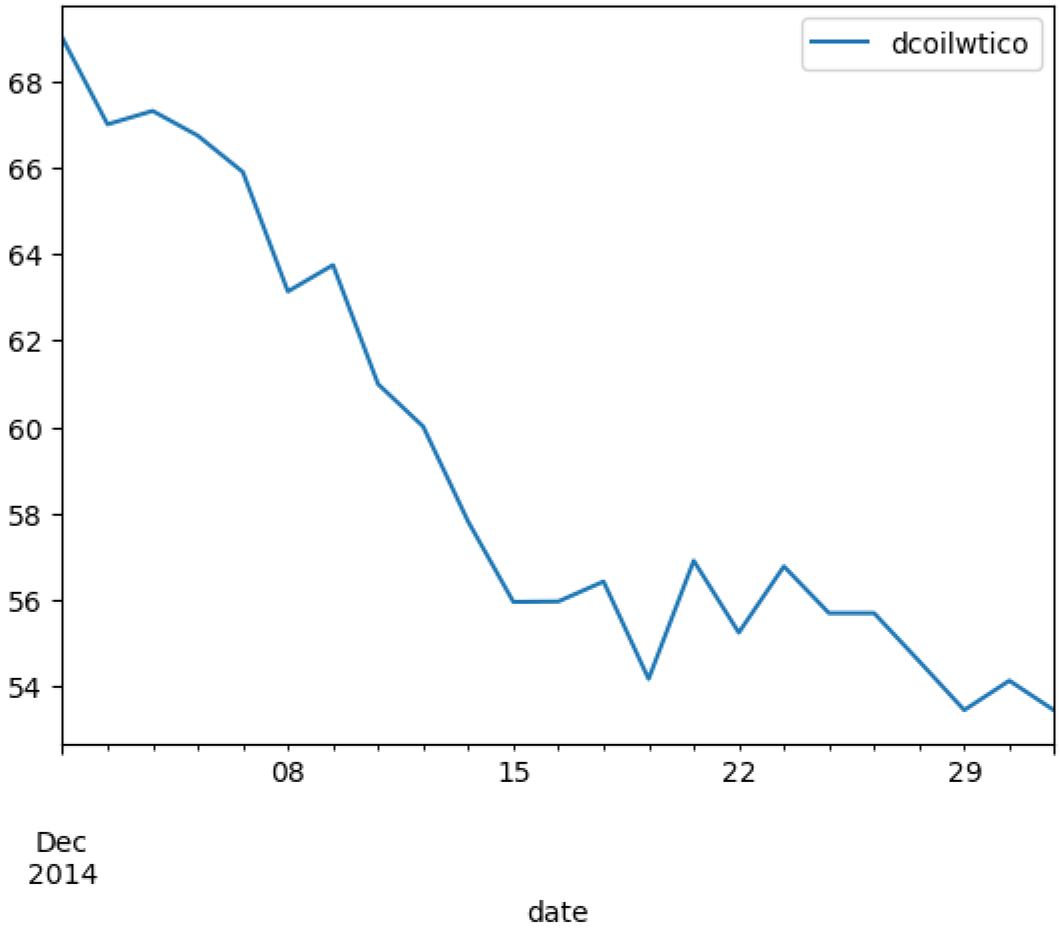
Out[28]:
```
<AxesSubplot:xlabel='date'>
```



In [36]:
```
# Filter to December 2014 then plot forward filled Series
oil['2014-12'].ffill().plot()
```

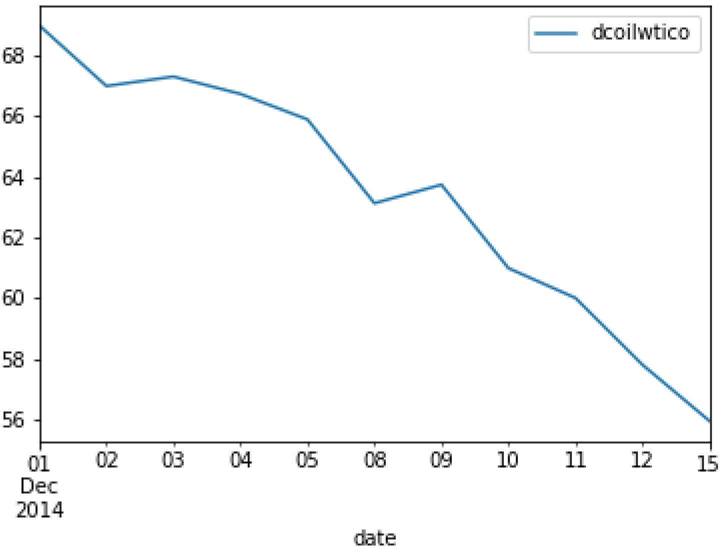
```
C:\Users\bhave\AppData\Local\Temp\ipykernel_29448\548675971.py:2: FutureWarni
ng: Indexing a DataFrame with a datetimelike index using a single string to s
lice the rows, like `frame[string]`, is deprecated and will be removed in a f
uture version. Use `frame.loc[string]` instead.
  oil['2014-12'].ffill().plot()
```

Out[36]:
```
<AxesSubplot:xlabel='date'>
```



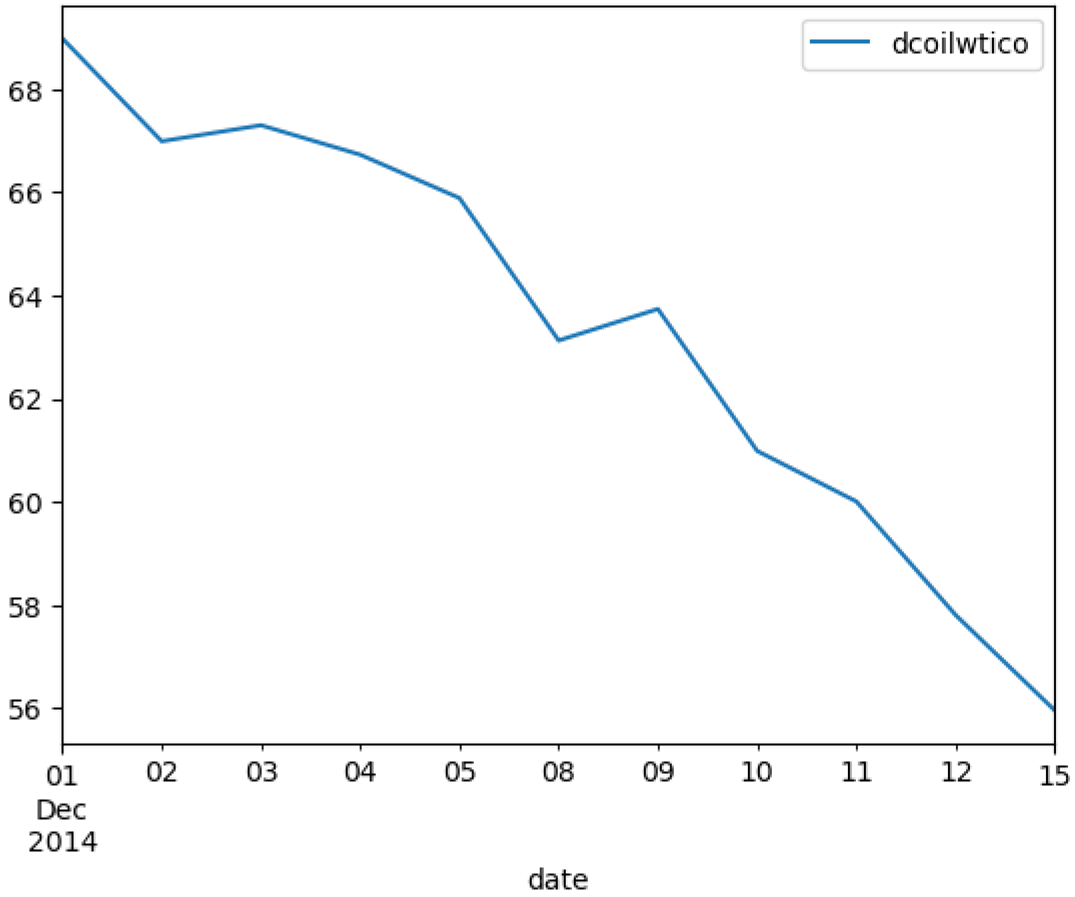
In [29]:
```
# Filter to first two weeks of December 2014 then plot forward filled Series
```

Out[29]:
```
<AxesSubplot:xlabel='date'>
```



In [26]:
```
# Filter to first two weeks of December 2014 then plot forward filled Series
oil['2014-12-01' : '2014-12-15'].ffill().plot()
```

Out[26]:
```
<AxesSubplot:xlabel='date'>
```



Assignment 4: Shift and Diff

looking into a few different year over year trends related to changes made at store 47.

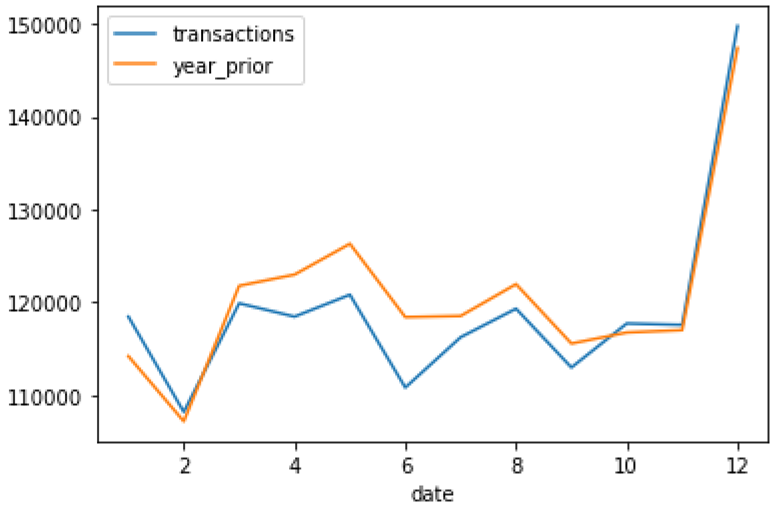
Can you plot the sum of monthly of transactions in year 2015 vs the sum of monthly transactions in the year prior for store 47?

Make sure to group your DataFrame by year AND month!

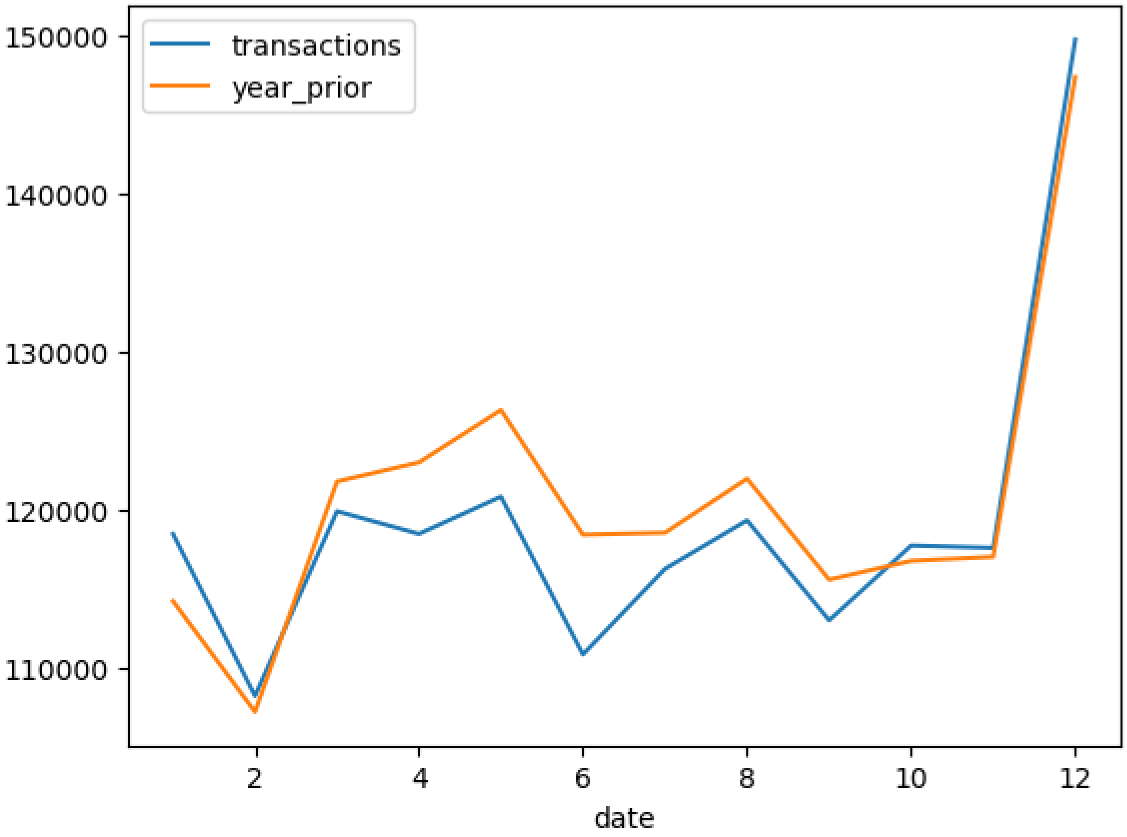
Thanks

In [37]:

```
# filter df to store 47, 'drop' store_nbr column via loc



# Calculate sum of sales by year and month


# Calculate a 'year_prior' column by shiftly monthly sales series forward by 1



# Filter to 2015 and plot
```



In [57]:
```
# filter df to store 47, 'drop' store_nbr column via loc

transactions_47 = transactions.loc[transactions['store_nbr'] == 47].drop(colum



# Calculate sum of sales by year and month
sales_by_year_and_month = df.groupby(['year', 'month'])['transactions'].sum().


# Calculate a 'year_prior' column by shiftly monthly sales series forward by 1
sales_by_year_and_month['year_prior'] = sales_by_year_and_month['transactions'



# Filter to 2015 and plot
filtered_data = sales_by_year_and_month[sales_by_year_and_month['year'] == 201
filtered_data.plot(x='month', y=['transactions', 'year_prior'])
plt.xlabel('date')
plt.legend(['transactions', 'year_prior'])
plt.show()
```



Assignment 5: Resampling Time Series

Plot the monthly and yearly average oil prices.

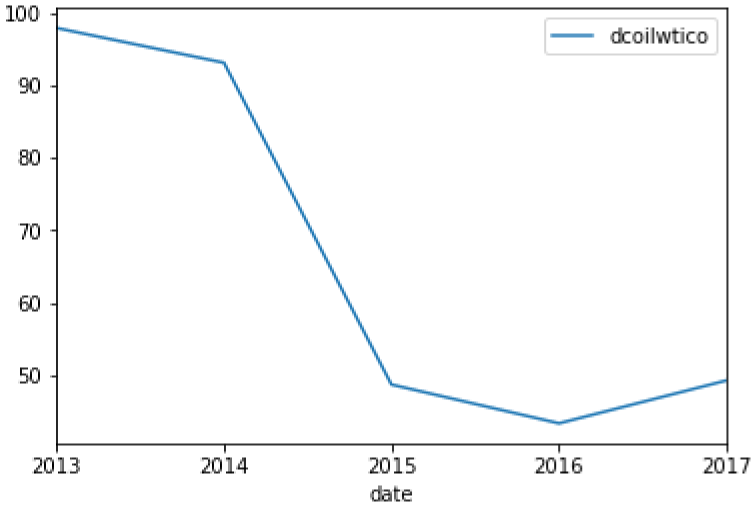
In [39]:
```
oil.head()
```

Out[39]:

	dcoilwtico
date	
2013-01-01	NaN
2013-01-02	93.14
2013-01-03	92.97
2013-01-04	93.12
2013-01-07	93.20

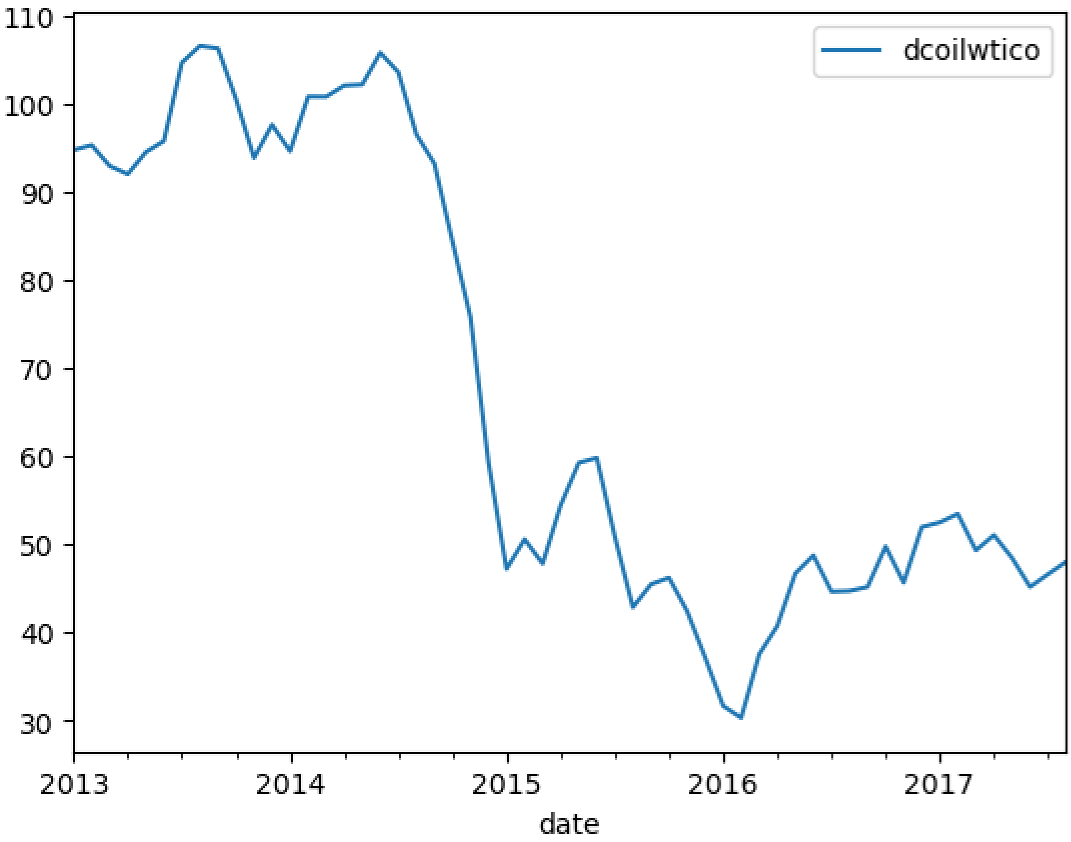
In [24]:
```
# Monthly average oil price
```

Out[24]:
```
<AxesSubplot:xlabel='date'>
```

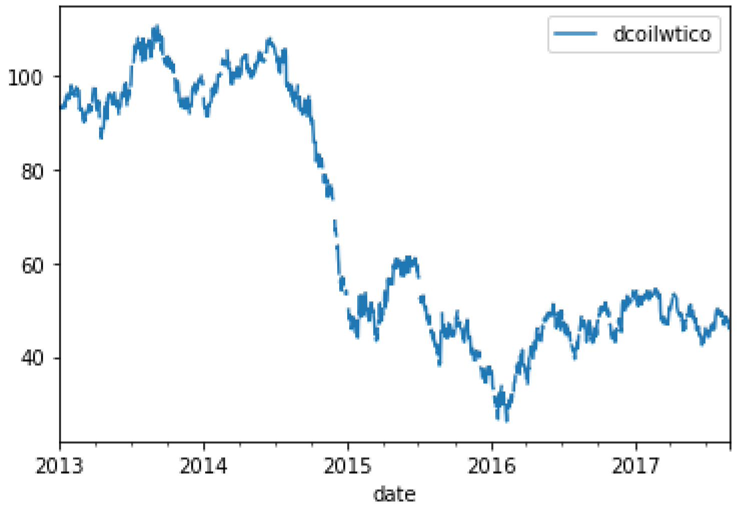


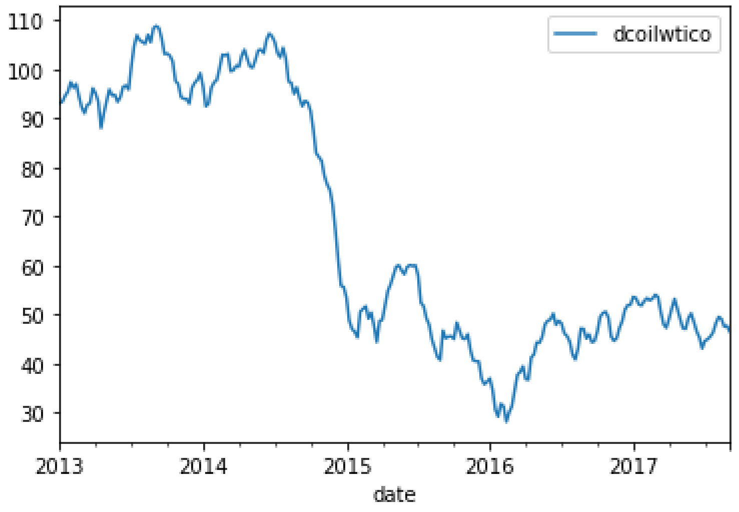
In [49]:
```
# Monthly average oil price

oil.resample('M').mean().plot()
```

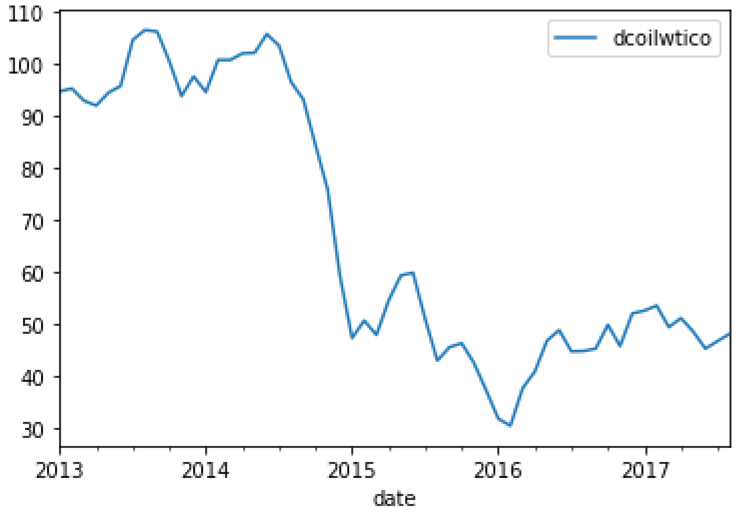
Out[49]:
```
<AxesSubplot:xlabel='date'>
```

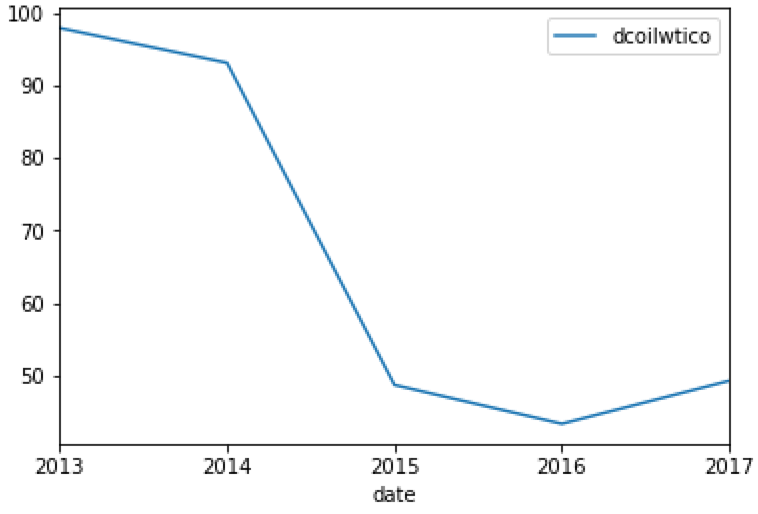


In [25]:
```
# A loop to create various time period averages and plot them
```



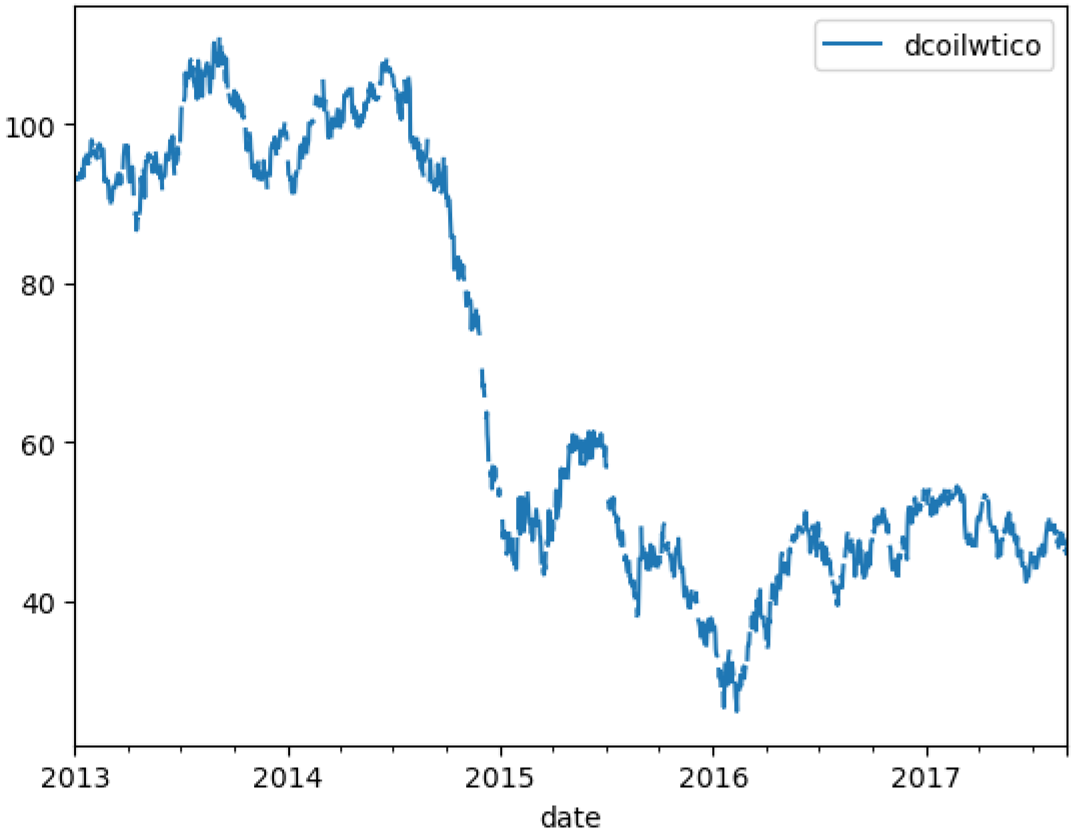


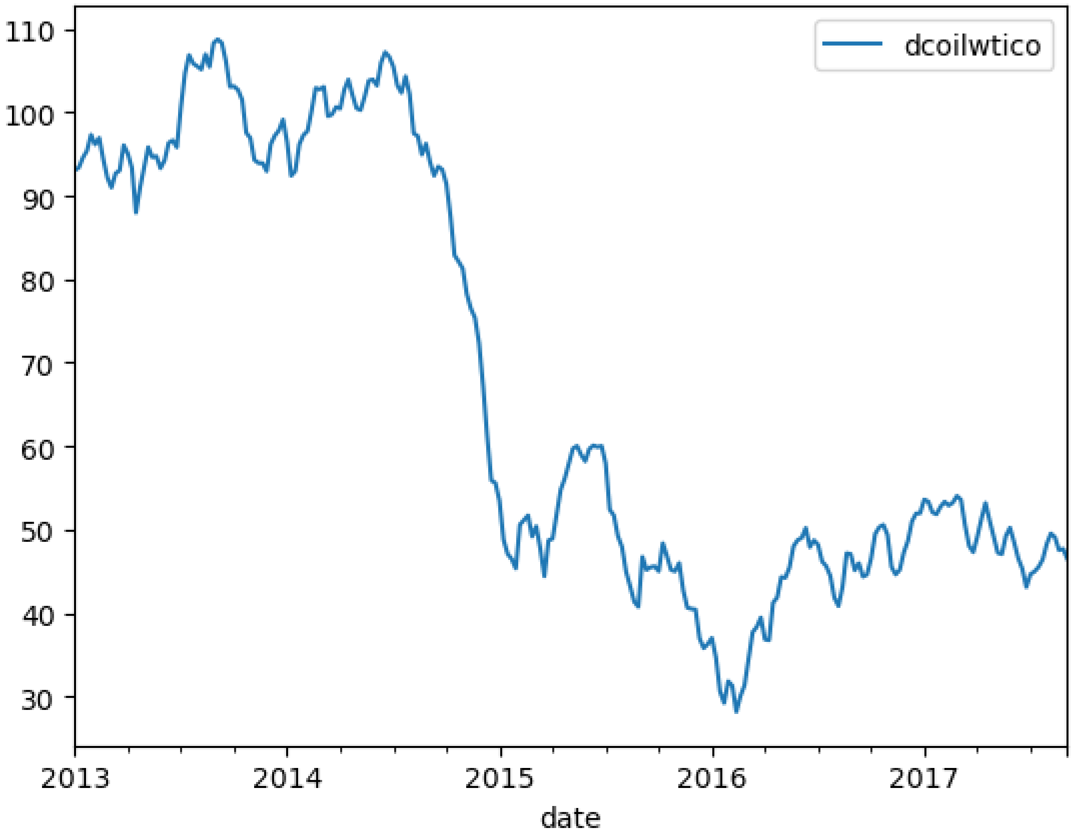


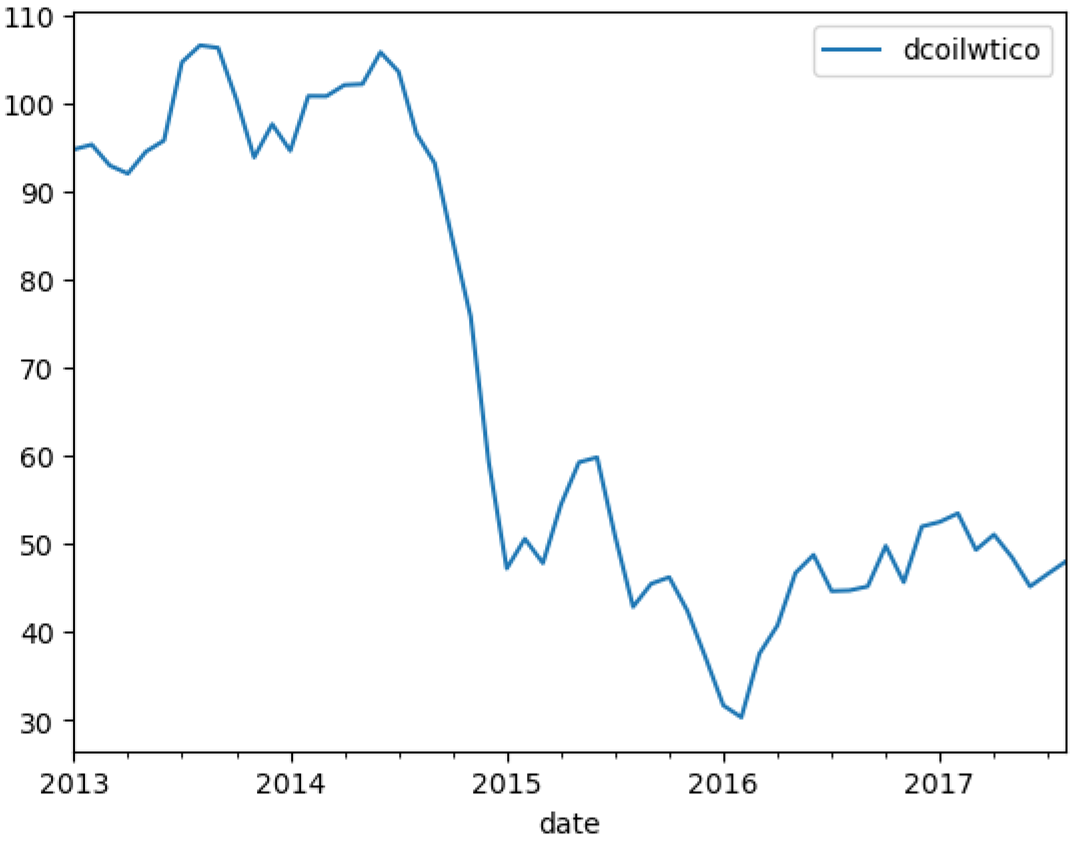


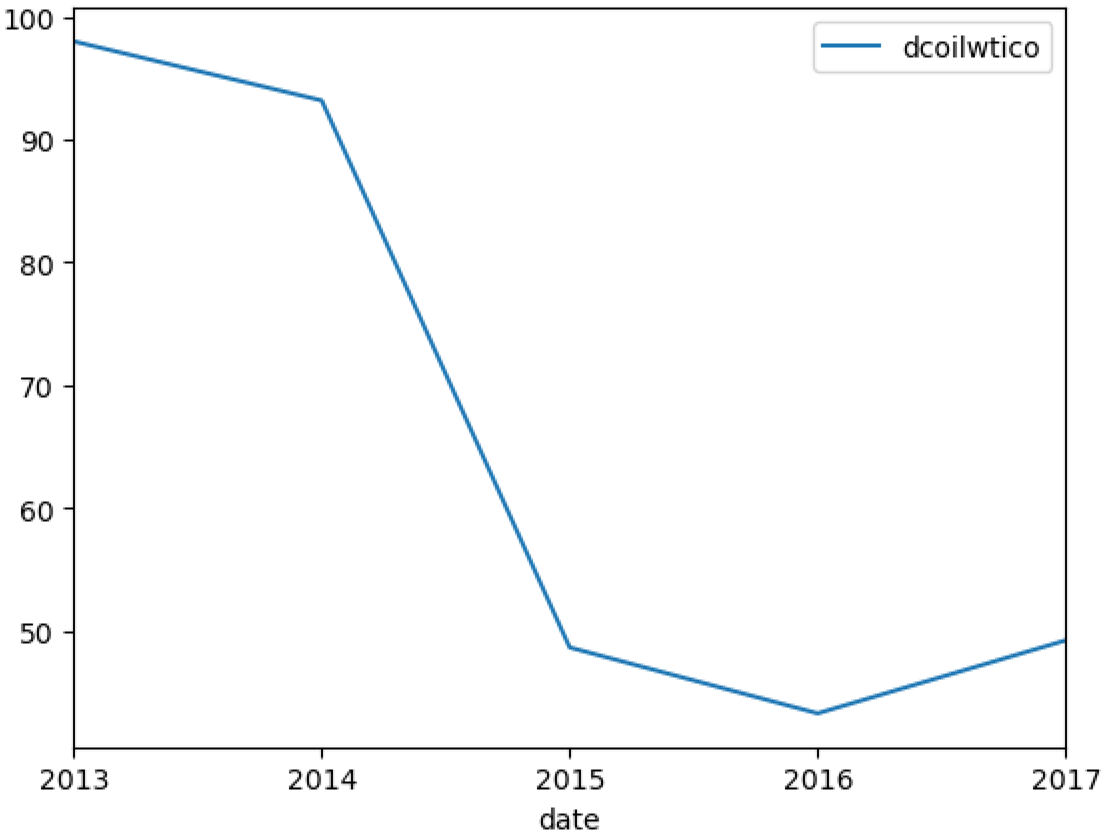
In [47]:

```
# A loop to create various time period averages and plot them

for _ in range(1):
    oil.resample('D').mean().plot()
    oil.resample('W').mean().plot()
    oil.resample('M').mean().plot()
    oil.resample('Y').mean().plot()
```









Assignment 6: Rolling Averages

Plot the 90-day moving average for transactions for store 47.

This will help remove some of the noise from our series.

In [69]:

```
# recreate transactions47 with date as index

transactions = pd.read_csv("transactions.csv")
transactions['date'] = pd.to_datetime(transactions['date'])
transactions47 = transactions[transactions['store_nbr'] == 47]

# Set the 'date' column as the index
transactions47 = transactions47.set_index('date')
transactions47.drop('store_nbr', axis=1,inplace=True)
transactions47.head()
```

Out[69]:

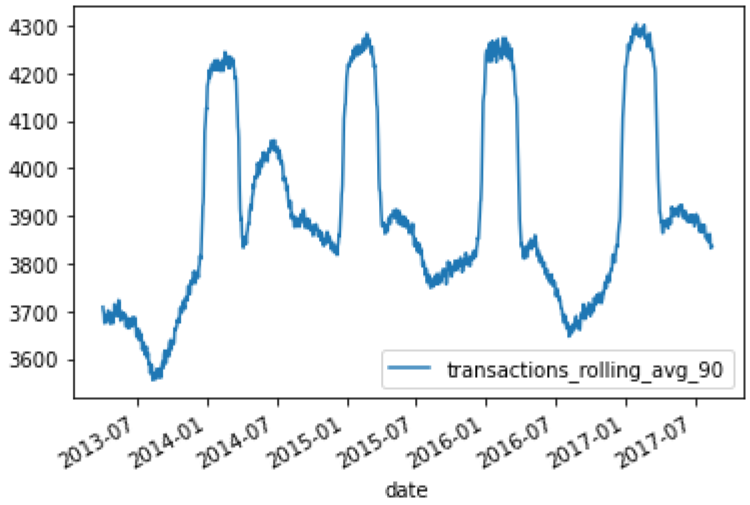
	transactions
date	
2013-01-02	4161
2013-01-03	3660
2013-01-04	3915
2013-01-05	4764
2013-01-06	4935

In [27]:

```
# Create 90 day rolling average column, drop original transactions column and
```

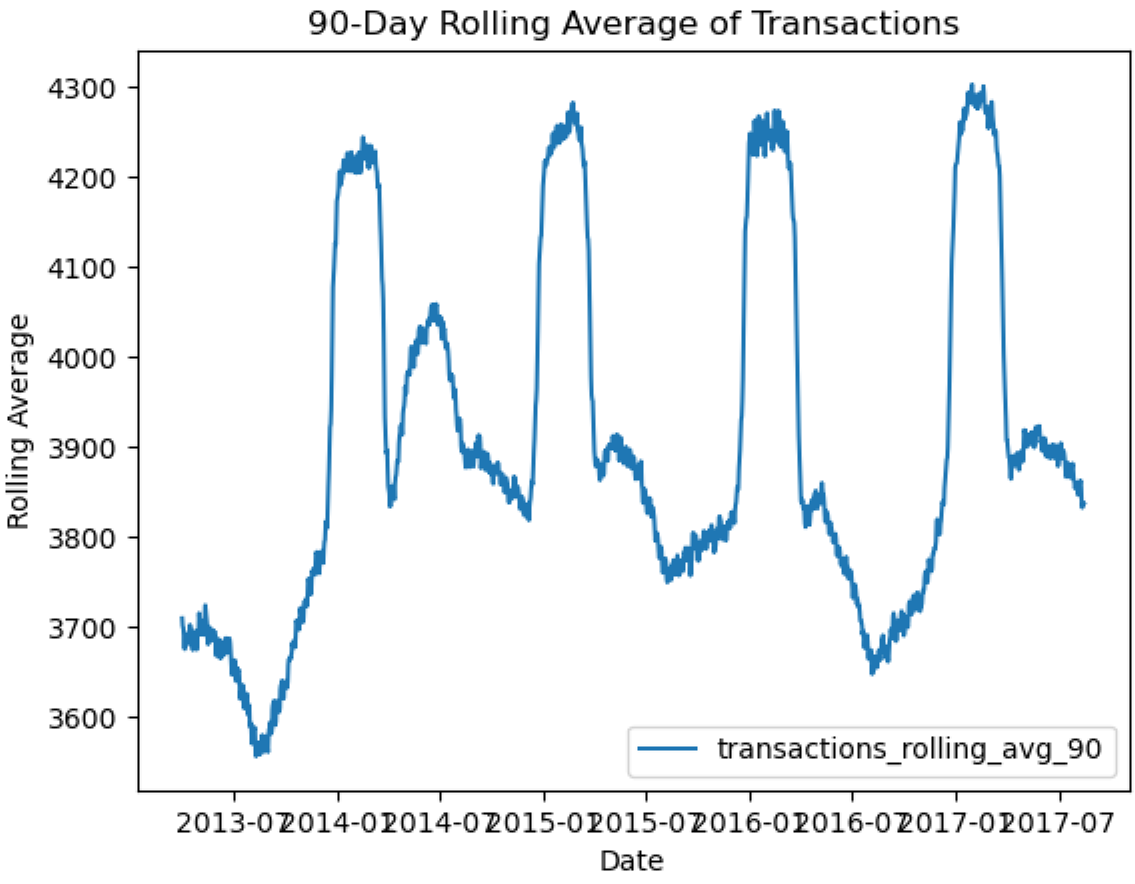
Out[27]:

```
<AxesSubplot:xlabel='date'>
```



In [75]:

```
# Create 90 day rolling average column, drop original transactions column and 

transactions47['rolling_avg_90'] = transactions47['transactions'].rolling(wind


transactions47_roll = transactions47.drop('transactions', axis=1,inplace = Fal

plt.plot(transactions47_roll['rolling_avg_90'], label = 'transactions_rolling_
plt.title('90-Day Rolling Average of Transactions')
plt.xlabel('Date')
plt.ylabel('Rolling Average')
plt.legend()
plt.show()
```

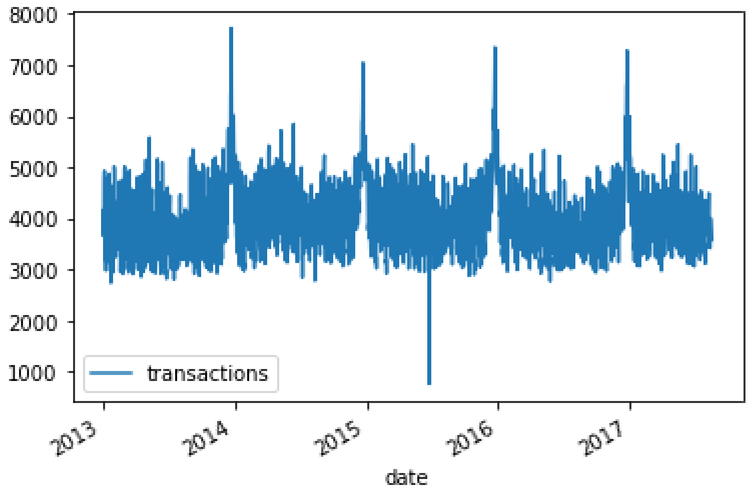


In [28]:

```
# original daily series for comparison
```

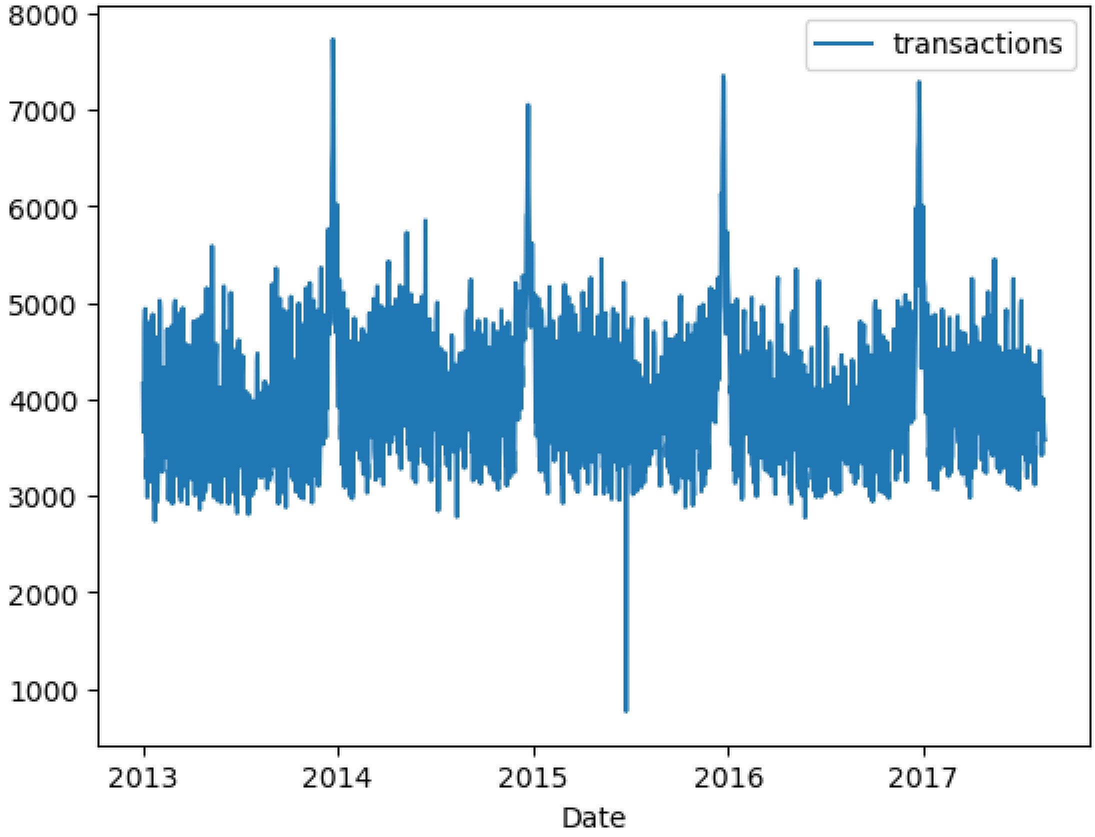
Out[28]:

```
<AxesSubplot:xlabel='date'>
```



In [74]:
```
plt.plot(transactions47.index, transactions47['transactions'], label='transact
plt.xlabel('Date')
plt.legend()
```

Out[74]:
```
<matplotlib.legend.Legend at 0x1ce8123d9a0>
```



In [ ]: