In [73]:
```
import plotly.express as px
import plotly.graph_objects as go
import pandas, os, openpyxl, datetime, math # dep xlrd
BASE_DIR='./NetInput_RefineryBlender/'
```

In [2]:
```
file_wk = [BASE_DIR+ff for ff in os.listdir(BASE_DIR) if ff.endswith('_W.xls')
file_mn = [BASE_DIR+ff for ff in os.listdir(BASE_DIR) if ff.endswith('_M.xls')
data_mn = pandas.read_excel(file_mn,sheet_name='Data 1',skiprows=1); data_mn.d
data_wk = pandas.read_excel(file_wk,sheet_name='Data 1',skiprows=1); data_wk.d
# The weekly report period begins at 7:01 a.m. on Friday and ends at 7:00 a.m.
data_mn = data_mn[['Sourcekey','MCRRIUS1']]; data_wk = data_wk[['Sourcekey','W
# MCRRIUS1
#
data_mn['Sourcekey']=pandas.to_datetime(data_mn['Sourcekey'])
# Mon of days in month
data_wk['Sourcekey']=pandas.to_datetime(data_wk['Sourcekey'])
#

# Calculate the number of days in the current month and hence convert monthly
def numDaysInMonth(x):
    first_day_current_month = datetime.datetime(x.year, x.month, 1)
    if x.month == 12: first_day_next_month = datetime.datetime(x.year+1, 1, 1)
    else            : first_day_next_month = datetime.datetime(x.year, x.month
    num_days = (first_day_next_month - first_day_current_month).days
    #print(f"Number of days in {first_day_current_month.strftime('%B')} {x.yea
    return num_days
data_mn['days_in_month']=data_mn['Sourcekey'].apply(lambda x:numDaysInMonth(x)
data_mn['MCRRIUS1 in (thousand_barrels_per_day)']= data_mn['MCRRIUS1']/data_mn

# From weekly data create a monthy production data. Remember the Units for wee
# dataFrame data_wk2mn contains monthly data.

# Function to calculate the proportion of a week in each month
def get_monthly_contribution(row):
    # date range for the week
    week_dates = pandas.date_range(row['Sourcekey'], row['Sourcekey']+pandas.T
    monthly_contributions= week_dates.to_period('M').value_counts(normalize=Tr
    return monthly_contributions

# Apply the function to each row
data_wk2mn = data_wk.apply(get_monthly_contribution, axis=1).stack().reset_ind
data_wk2mn = data_wk2mn.reset_index()
data_wk2mn.columns = ['Sourcekey', 'WCRRIUS2']
data_wk2mn = data_wk2mn.groupby('Sourcekey')['WCRRIUS2'].sum().reset_index()
data_wk2mn['Sourcekey'] = data_wk2mn['Sourcekey'].dt.to_timestamp()
data_wk2mn['Sourcekey'] = data_wk2mn['Sourcekey'] + pandas.Timedelta(days=14)
```

In [3]:
```
# merged dataFrame MCRRIUS1 and WCRRIUS2
combined=data_wk2mn.merge(data_mn, how='inner', on='Sourcekey')
```

2. Q Compare the weekly and monthly time series data. What is the monthly deviation between the monthly and weekly data? What is the range and the average of the monthly deviation between the monthly and the data?

Deviation = Monthly (MCRRIUS1) - Monthly (obtained from weekly WCRRIUS2)

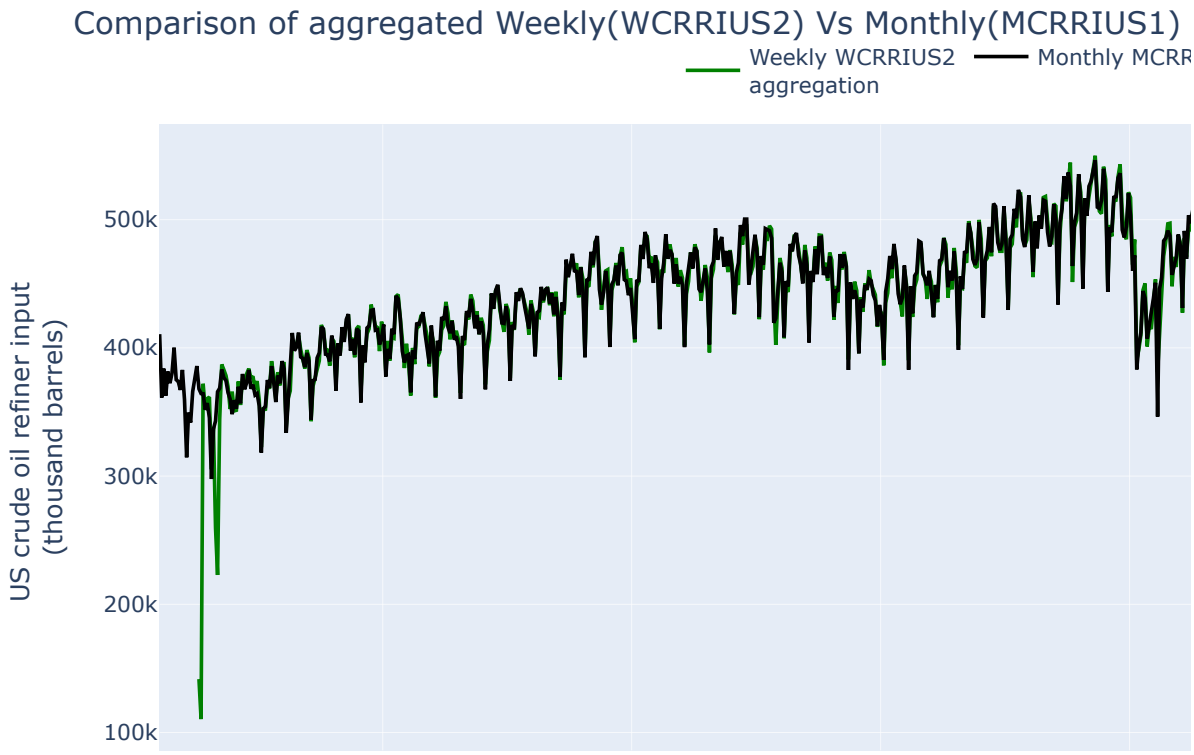
In [4]:
```
combined['monthly_deviation'] = combined['MCRRIUS1'] - combined['WCRRIUS2']

print('Deviation Min'.ljust(20),': ',combined['monthly_deviation'].min(), 'thou
print('Deviation Max'.ljust(20),': ',combined['monthly_deviation'].max(), 'thou
print('Deviation Median'.ljust(20),': ',combined['monthly_deviation'].median()
print('Deviation Mean'.ljust(20),': ',combined['monthly_deviation'].mean(), 'th
```

```
Deviation Min        :  -34781.0 thousand barrels
Deviation Max        :  253913.0 thousand barrels
Deviation Median     :  131.0 thousand barrels
Deviation Mean       :  1428.935166994106 thousand barrels
```

2. Q Plot the monthly and weekly data in a time series

Solution Visual time series plot to see any data discrepency

In [5]:
```
fig = go.Figure()
fig.update_layout(title='Comparison of aggregated Weekly(WCRRIUS2) Vs Monthly(M
## Weekly to monthly data
fig.add_trace(go.Scatter(x=data_wk2mn['Sourcekey'], y=data_wk2mn['WCRRIUS2'],mo
## Monthly data MCRRIUS1
fig.add_trace(go.Scatter(x=data_mn['Sourcekey'],y=data_mn['MCRRIUS1'],mode='lin
fig.update_layout(xaxis_title='Date',yaxis_title='US crude oil refiner input <b
fig.update_layout(legend=dict(    orientation="h",    yanchor="bottom",
    y=1.02,    xanchor="right",    x=1 ))
fig.show()
```



3. Q Use the more timely weekly data to estimate the lagged monthly

data.

In [62]:
```
## import library and Evaluation metrics
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

In [43]:
```
## Prepare Data
# Select a date range when both monthly and weekly data is available
start_date = '1982-10-01';    end_date   = '2025-02-28'

## Prepare weekly data with features and target (from monthly data)
weekly_data = data_wk.copy()
weekly_data = weekly_data[(weekly_data['Sourcekey']>=start_date) & (weekly_dat

# Feature engineering
weekly_data['year']         = weekly_data['Sourcekey'].dt.year
weekly_data['month']        = weekly_data['Sourcekey'].dt.month
weekly_data['WeekStartDay'] = weekly_data['Sourcekey'].dt.day
weekly_data['days_in_month']= weekly_data['Sourcekey'].apply(lambda x:numDaysI
weekly_data.drop(columns=['Sourcekey'],inplace=True)

## add target from monthly data
# Target from Monthly data
data_mn['year'] = data_mn['Sourcekey'].dt.year
data_mn['month'] = data_mn['Sourcekey'].dt.month
target = 'MCRRIUS1' # from MONTHLY data
weekly_data=weekly_data.merge(data_mn[['MCRRIUS1', 'year', 'month']], how='inn


## Split weekly data into training and testing sets (e.g., 80% train, 20% test
train_size = int(len(weekly_data) * 0.8)
train, test = weekly_data.iloc[:train_size].copy(), weekly_data.iloc[train_siz
```

In [83]:
```
## Input features
# WCRRIUS2 (week's production in thousand barrels per day)
# month (month number 1-12)
# WeekStartDay (on which date of month week started)
# days_in_month (number of days in month)

## TARGET for prediction
# MCRRIUS1 (Our target to estimate monthly Production "in thousand barrels")
```

In [86]:
```
print('Input Features: ',str(['WCRRIUS2', 'month', 'WeekStartDay', 'days_in_mo
print('Target (thousand barrels): ',target)
```

```
Input Features:  ['WCRRIUS2', 'month', 'WeekStartDay', 'days_in_month']
Target (thousand barrels):  MCRRIUS1
```

In [51]:
```
## Training
train_cols =['WCRRIUS2', 'month', 'WeekStartDay', 'days_in_month']
model = LinearRegression()
model.fit(X=train[train_cols], y=train[target])
```

Out[51]:

▾ LinearRegression

LinearRegression()

In [57]:
```
train['Predicted_Production'] = model.predict(X=train[train_cols])
test['Predicted_Production'] = model.predict(X=test[train_cols])
```

In [74]:
```
mae_train= mean_absolute_error(train['MCRRIUS1'], train['Predicted_Production'
mae_test = mean_absolute_error(test['MCRRIUS1'], test['Predicted_Production'])
rmse_train= math.sqrt(mean_squared_error(train['MCRRIUS1'], train['Predicted_P
rmse_test = math.sqrt(mean_squared_error(test['MCRRIUS1'], test['Predicted_Pro
r2_train= r2_score(train['MCRRIUS1'], train['Predicted_Production'])
r2_test = r2_score(test['MCRRIUS1'], test['Predicted_Production'])
```

In [82]:
```
print('(train) Root Mean Squared Error.: '.ljust(35), "{:.2f}".format(rmse_tra
print('(train) Mean Average Error.: '.ljust(35), "{:.2f}".format(mae_train), '
print('(train) R-squared: '.ljust(35), "{:.2f}".format(r2_train))
print('-----------------')
print('(test) Root Mean Squared Error.: '.ljust(35), "{:.2f}".format(rmse_test
print('(test) Mean Average Error.: '.ljust(35), "{:.2f}".format(mae_test), 'th
print('(test) R-squared: '.ljust(35), "{:.2f}".format(r2_test))
```

```
(train) Root Mean Squared Error.:   8080.74 thousand barrels
(train) Mean Average Error.:        5896.98 thousand barrels
(train) R-squared:                  0.96
-----------------
(test) Root Mean Squared Error.:    12593.48 thousand barrels
(test) Mean Average Error.:         8514.60 thousand barrels
(test) R-squared:                   0.88
```

Model's R-Square on test data is 0.88, this implies model can explain 88% of the variance in data.

In [ ]: