

# Clustering Canadian Crude: Unsupervised Learning for Oil Grade Classification

---

## Project Summary

This project applies unsupervised machine learning to predict and group the grades of Canadian crude oils based on their chemical and physical characteristics. The dataset includes four target categories—Light & Medium Sweet Crude Oil, Light & Medium Sour Crude Oil, Heavy Sour Crude Oil, and Sweet Synthetic Crude Oil—sourced from CrudeMonitor.ca. Having worked as a Petroleum Engineer for over 25 years and now nearing completion of a Master of Science in Data Science, this project bridges my engineering experience with modern data-driven methods to address a real-world problem commonly faced in crude oil quality analysis and classification.

The central hypothesis is that the Light & Medium Sweet and Light & Medium Sour crude oils share nearly identical physical properties, differing mainly in sulfur content and related compounds. Therefore, they may not form truly distinct groups in an unsupervised context. If clustering algorithms naturally group these two together, it would indicate that the optimal structure of the data reflects three clusters rather than four—thus rejecting the null hypothesis that four separate groups best represent the dataset.

The analysis proceeds through several structured stages: data summary to understand the dataset's structure, exploratory data analysis (EDA) to identify patterns and correlations, and data cleaning to handle missing or inconsistent values. Three clustering algorithms—K-Means, Agglomerative Clustering, and Gaussian Mixture Models (GMM)—are then applied and evaluated using multiple performance metrics and visualized through PCA and t-SNE. The unsupervised results are later compared with supervised learning models—Multinomial Logistic Regression, Support Vector Classifier, and Gradient Boosting—to assess alignment with true class labels. The project concludes with final discussion and takeaways, highlighting key insights and implications for applying machine learning to crude quality analysis in the energy industry.

---

## Data Summary

This section provides an initial overview of the dataset to establish a clear understanding of its structure and contents before any analysis begins. It outlines the data source, the number of records and features, and the types of variables included. Basic statistics are used to summarize the range of values, while data types are confirmed to distinguish between numerical and categorical columns.

### Import Python Packages

```
In [56]: import pandas as pd
import numpy as np
from IPython.display import display, Markdown, Image
from datetime import datetime
from dateutil import relativedelta
import requests

from sklearn.impute import KNNImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.cluster import MiniBatchKMeans, AgglomerativeClustering
from sklearn.metrics.cluster import contingency_matrix
from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.linear_model import LogisticRegressionCV
from sklearn.svm import SVC
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.metrics import (
    silhouette_score,
    davies_bouldin_score,
    calinski_harabasz_score,
    adjusted_rand_score,
    normalized_mutual_info_score,
    fowlkes_mallows_score,
    silhouette_samples,
    classification_report,
    confusion_matrix
)
from scipy.optimize import linear_sum_assignment

import altair as alt
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from IPython.display import Image

pd.set_option('future.no_silent_downcasting', True)
alt.data_transformers.disable_max_rows()
```

```
Out[56]: DataTransformerRegistry.enable('default')
```

### Data Source

The data for this project was obtained from CrudeMonitor.ca, a publicly available resource that provides detailed information on the quality characteristics of various crude oil grades produced in Western Canada.

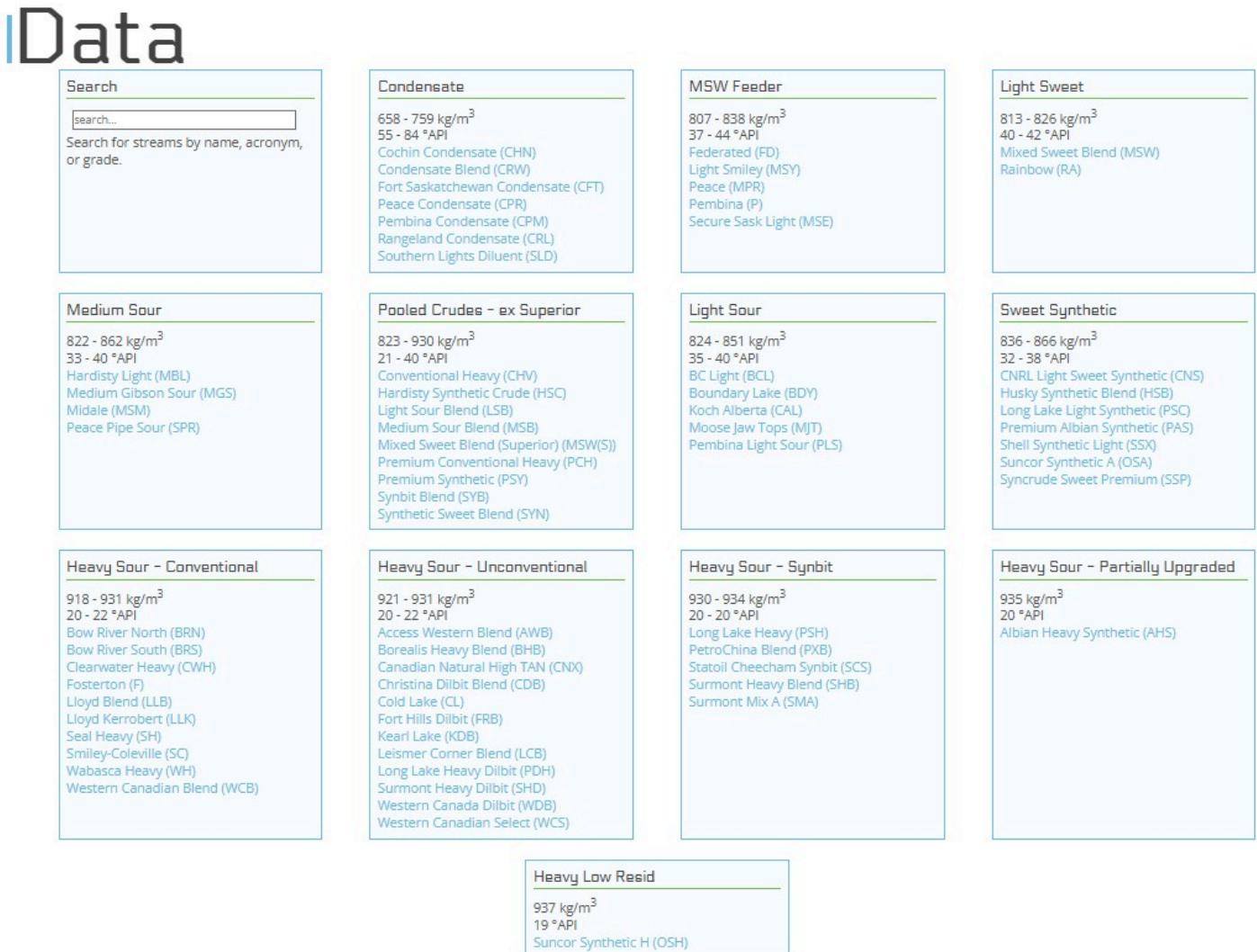
CrudeMonitor.ca groups grades of crude oils into 12 general groupings. I've chosen to use six of these groups for this analysis, in particular the Heavy Sour - Conventional, Medium Sour, Light Sour, Light Sweet, MSW Feeder, and Sweet Synthetic groups.

#### Reference:

Crude Quality Inc. (n.d.). CrudeMonitor.ca. Retrieved October 16, 2025, from <https://www.crudemonitor.ca/>

```
In [57]: # Display an image of the crude oil groups as defined by CrudeMonitor.ca
Image(url='CrudeMonitor-Screenshot-2025-10-18.jpg', width=1200)
```

Out[57]:



## Target Groups

For this project, I combined the six CrudeMonitor.ca categories—Heavy Sour – Conventional, Medium Sour, Light Sour, Light Sweet, MSW Feeder, and Sweet Synthetic—into **four broader target groups: Heavy Sour, Sweet Synthetic, Light & Medium Sour, and Light & Medium Sweet** crude oils. Based on my industry experience, these four groups represent grades with comparable quality characteristics. While I could have merged the Light & Medium Sweet and Sour categories into a single group, I intentionally kept them separate to test the hypothesis that they may, in fact, belong together rather than being treated as distinct classes.

The four target groups for this project are defined as:

- **Heavy Sour Crude Oils**
  - "Heavy Sour - Conventional" group as defined by CrudeMonitor.ca
  - Density range: 918 - 931 kg/m<sup>3</sup>
  - Quality range: 20 - 22 °API
  - Grades:
    - Bow River North (BRN)
    - Bow River South (BRS)
    - Fosterton (F)
    - Lloyd Blend (LLB)
    - Lloyd Kerrobert (LLK)
    - Seal Heavy (SH)
    - Smiley-Coleville (SC)
    - Wabasca Heavy (WH)
    - Western Canadian Blend (WCB)

- Western Canadian Select (WCS)

- **Sweet Synthetic Crude Oils**

- "Sweet Synthetic" group as defined by CrudeMonitor.ca
- Density range: 836 - 866 kg/m<sup>3</sup>
- Quality range: 32 - 38 °API
- Grades:
  - CNRL Light Sweet Synthetic (CNS)
  - Husky Synthetic Blend (HSB)
  - Long Lake Light Synthetic (PSC)
  - Premium Albian Synthetic (PAS)
  - Shell Synthetic Light (SSX)
  - Suncor Synthetic A (OSA)
  - Syncrude Sweet Premium (SSP)

- **Light & Medium Sour Crude Oils**

- A combination of the "Light Sour" and "Medium Sour" groups as defined by CrudeMonitor.ca
- Density range: 822 - 862 kg/m<sup>3</sup>
- Quality range: 33 - 41 °API
- Grades:
  - Hardisty Light (MBL)
  - Medium Gibson Sour (MGS)
  - Midale (MSM)
  - Peace Pipe Sour (SPR)
  - BC Light (BCL)
  - Boundary Lake (BDY)
  - Koch Alberta (CAL)
  - Moose Jaw Tops (MJT)
  - Pembina Light Sour (PLS)

- **Light & Medium Sweet Crude Oils**

- A combination of the "MSW Feeder" and "Light Sweet" groups as defined by CrudeMonitor.ca
- Density range: 807 - 838 kg/m<sup>3</sup>
- Quality range: 37 - 44 °API
- Grades:
  - Federated (FD)
  - Light Smiley (MSY)
  - Peace (MPR)
  - Pembina (P)
  - Secure Sask Light (MSE)
  - Mixed Sweet Blend (MSW)
  - Rainbow (RA)

## Create Dataset

- The dataset was created by extracting data from the CrudeMonitor.ca API using a custom-built function. This function references a predefined dictionary of crude oil groups, loops through each crude grade to retrieve its data, and then combines all groups into a single, consolidated dataset for analysis.

```
In [58]: # Define dictionary of crude oil types & associated metadata
crude_oils_dict = {
    'Heavy Sour':{
        'target_label':0,
        'min_date':'2001-01-02',
        'grades':{
            'Bow River North':'BRN',
            'Bow River South':'BRS',
            'Fosterton':'F',
            'Lloyd Blend':'LLB',
            'Lloyd Kerrobert':'LLK',
            'Seal Heavy':'SH',
            'Smiley-Coleville':'SC',
            'Wabasca Heavy':'WH',
            'Western Canadian Blend':'WCB',
            'Western Canadian Select':'WCS'
        }
    },
    'Sweet Synthetic':{
        'target_label':1,
        'min_date':'2004-01-05',
        'grades':{
            'CNRL Light Sweet Synthetic':'CNS',
            'Husky Synthetic Blend':'HSB',
            'Long Lake Light Synthetic':'PSC',
            'Premium Albian Synthetic':'PAS',
            'Shell Synthetic Light':'SSX',
            'Suncor Synthetic A':'OSA',
            'Syncrude Sweet Premium':'SSP'
        }
    },
    'Light & Medium Sour Crude Oils':{
        'target_label':2,
        'min_date':'1999-11-21',
    }
}
```

```

'grades':{
    'Hardisty Light':'MBL',
    'Medium Gibson Sour':'MGS',
    'Midale':'MSM',
    'Peace Pipe Sour':'SPR',
    'BC Light':'BCL',
    'Boundary Lake':'BDY',
    'Koch Alberta':'CAL',
    'Moose Jaw Tops':'MJT',
    'Pembina Light Sour':'PLS'
}
},
'Light & Medium Sweet Crude Oils':{
    'target_label':3,
    'min_date':1999-11-12,
    'grades':{
        'Federated':'FD',
        'Light Smiley':'MSY',
        'Peace':'MPR',
        'Pembina':'P',
        'Secure Sash Light':'MSE',
        'Mixed Sweet Blend':'MSW',
        'Rainbow':'RA'
    }
},
}

```

In [59]: # Function to create a dataframe from historical crude oil properties by crude\_grade

```

def get_data(crude_grade, crude_oils_dict):
    # Create URL parts
    start_date = crude_oils_dict[crude_grade]['min_date'] # Start date is the min_date defined in the crude_oils_dict for each crude_grade
    end_date = datetime.now().date().strftime('%Y-%m-%d') # End date is the current date
    base_url = 'https://www.crudemonitor.ca/api/1.1/json.php?'
    crude_properties_url = '&crudeProperties%5B0%5D=crudes-BA&crudeProperties%5B1%5D=crudes-LE&crudeProperties%5B2%5D=crudes-HTSD'
    crudes_url = ''.join([f'&crudes%5B{i}%5D={x.replace(' ', '+)}' for i, x in enumerate([k for k in crude_oils_dict[crude_grade][['grades']].keys()])])
    date_url = f'&date%5Bstart%5D={start_date}&date%5Bend%5D={end_date}'

    # Merge url parts into full url
    url = base_url + crudes_url + crude_properties_url + date_url

    # Request data from API
    response = requests.get(url)
    response.raise_for_status()

    # Create dataframe
    response_df = pd.DataFrame.from_dict(response.json())

    # Replace values
    response_df = response_df.replace({np.nan, 'ND': np.nan}).infer_objects(copy=False)

    # Correct Location names
    response_df['Location'] = response_df['Location'].str.replace(r'(?i)milk.+river', 'Milk River', regex=True)

    # Change numeric columns datatypes
    num_cols = response_df.columns[4:]
    response_df[num_cols] = response_df[num_cols].apply(pd.to_numeric, errors='coerce')

    # Downcast numeric columns where possible
    for col in response_df.select_dtypes(include=['float64']).columns:
        response_df[col] = pd.to_numeric(response_df[col], downcast='float')
    for col in response_df.select_dtypes(include=['int64']).columns:
        response_df[col] = pd.to_numeric(response_df[col], downcast='integer')

    # Convert Sample Date (yyyy-mm-dd) column to datetime
    response_df['Sample Date (yyyy-mm-dd)'] = pd.to_datetime(response_df['Sample Date (yyyy-mm-dd)'])

    # Add target labels to df
    response_df = response_df.assign(target_label=crude_oils_dict[crude_grade]['target_label'])

    return response_df

```

In [60]: # Loop over crude\_grade types and concatenate datasets into one

```

df = pd.concat([
    get_data(crude_grade, crude_oils_dict) for crude_grade in crude_oils_dict.keys(),
    ignore_index=True,
    copy=False
])

# Display the dataset
df

```

Out[60]:

	Crude	Batch	Sample Date (yyyy-mm-dd)	Location	Density (kg/m³) [ASTM D5002]	Gravity (°API) [ASTM D5002]	Sulphur (wt%) [ASTM D4294]	Micro Carbon Residue (wt%) [ASTM D4530]	Sediment (ppmw) [ASTM D4807]	Total Acid Number (mgKOH/g) [ASTM D664]	...	91 Mass% Recovered (°C) [ASTM D7169]	92 Mass% Recovered (°C) [ASTM D7169]	93 Mass% Recovered (°C) [ASTM D7169]	94 Mass% Recovered (°C) [ASTM D7169]	95 Mass% Recovered (°C) [ASTM D7169]	96 Mass% Recovered (°C) [ASTM D7169]
0	Bow River North	BR-030	2001-01-02	Hardisty	927.099976	21.000000	2.95	8.9	170.0	1.24	...	664.900024	676.000000	689.400024	704.700012	733.200012	NaN
1	Bow River North	BR-073	2001-01-07	Hardisty	927.299988	21.000000	2.96	9.4	150.0	1.16	...	719.099976	NaN	NaN	NaN	NaN	NaN
2	Lloyd Blend	LLB-464	2001-01-03	Hardisty	920.799988	22.000000	3.17	9.4	230.0	0.75	...	NaN	NaN	NaN	NaN	NaN	NaN
3	Lloyd Blend	LLB-468	2001-01-18	Hardisty	922.500000	21.799999	3.11	9.6	300.0	0.91	...	NaN	NaN	NaN	NaN	NaN	NaN
4	Lloyd Kerrobert	LLK-062	2001-01-04	Kerrobert	926.299988	21.100000	2.80	9.6	230.0	1.56	...	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5298	Secure Sask Light	MSE-1434	2024-08-04	Kerrobert	832.900024	38.200001	0.38	2.5	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
5299	Secure Sask Light	MSE-1056	2024-09-12	Kerrobert	844.799988	35.799999	0.35	2.5	NaN	NaN	...	651.200012	668.900024	688.700012	706.799988	NaN	NaN
5300	Secure Sask Light	MSE-1183	2024-10-03	Kerrobert	829.700012	38.900002	0.33	2.5	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
5301	Secure Sask Light	MSE-1267	2024-11-01	Kerrobert	834.700012	37.900002	0.30	2.4	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
5302	Secure Sask Light	MSE-1211	2024-12-05	Kerrobert	835.000000	37.799999	0.38	2.6	NaN	NaN	...	666.400024	685.500000	703.099976	NaN	NaN	NaN

5303 rows × 133 columns



## Dataset Info

- `df.info()` provides a concise summary of a DataFrame, including:
  - Number of rows
  - Column names
  - Data types
- It also shows the count of non-null entries for each column, which makes it easy to identify missing values.
- In addition, it displays the memory usage of the DataFrame, helping to assess the size and efficiency of the dataset in memory.

```
In [61]: # Inspect column data types and size of the dataframe
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5303 entries, 0 to 5302
Data columns (total 133 columns):
 #   Column                      Dtype  
 --- 
 0   Crude                       object  
 1   Batch                        object  
 2   Sample Date (yyyy-mm-dd)    datetime64[ns]
 3   Location                     object  
 4   Density (kg/m³) [ASTM D5002] float32
 5   Gravity (°API) [ASTM D5002] float32
 6   Sulphur (wt%) [ASTM D4294]  float32
 7   Micro Carbon Residue (wt%) [ASTM D4530] float32
 8   Sediment (ppmw) [ASTM D4807] float32
 9   Total Acid Number (mgKOH/g) [ASTM D664] float32
 10  Salt (ptb) [ASTM D3230]     float32
 11  Nickel (mg/kg) [ASTM D5708A] float32
 12  Vanadium (mg/kg) [ASTM D5708A] float32
 13  C1 Methane (vol%) [GC/FID] float32
 14  C2 Ethane (vol%) [GC/FID]  float32
 15  C3 Propane (vol%) [GC/FID] float32
 16  iC4 iso-Butane (vol%) [GC/FID] float32
 17  nC4 n-Butane (vol%) [GC/FID] float32
 18  iC5 iso-Pentane (vol%) [GC/FID] float32
 19  nC5 n-Pentane (vol%) [GC/FID] float32
 20  C6 Hexanes (vol%) [GC/FID]  float32
 21  C7 Heptanes (vol%) [GC/FID] float32
 22  C8 Octanes (vol%) [GC/FID]  float32
 23  C9 Nonanes (vol%) [GC/FID]  float32
 24  C10 Decanes (vol%) [GC/FID] float32
 25  Nitrogen - C4- (vol%) [GPA 2177M] float32
 26  Carbon Dioxide - C4- (vol%) [GPA 2177M] float32
 27  C1 Methane - C4- (vol%) [GPA 2177M] float32
 28  C2 Ethane - C4- (vol%) [GPA 2177M] float32
 29  C3 Propane - C4- (vol%) [GPA 2177M] float32
 30  iC4 iso-Butane - C4- (vol%) [GPA 2177M] float32
 31  nC4 n-Butane - C4- (vol%) [GPA 2177M] float32
 32  IBP (°C) [ASTM D7169]      float32
 33  1 Mass% Recovered (°C) [ASTM D7169] float32
 34  2 Mass% Recovered (°C) [ASTM D7169] float32
 35  3 Mass% Recovered (°C) [ASTM D7169] float32
 36  4 Mass% Recovered (°C) [ASTM D7169] float32
 37  5 Mass% Recovered (°C) [ASTM D7169] float32
 38  6 Mass% Recovered (°C) [ASTM D7169] float32
 39  7 Mass% Recovered (°C) [ASTM D7169] float32
 40  8 Mass% Recovered (°C) [ASTM D7169] float32
 41  9 Mass% Recovered (°C) [ASTM D7169] float32
 42  10 Mass% Recovered (°C) [ASTM D7169] float32
 43  11 Mass% Recovered (°C) [ASTM D7169] float32
 44  12 Mass% Recovered (°C) [ASTM D7169] float32
 45  13 Mass% Recovered (°C) [ASTM D7169] float32
 46  14 Mass% Recovered (°C) [ASTM D7169] float32
 47  15 Mass% Recovered (°C) [ASTM D7169] float32
 48  16 Mass% Recovered (°C) [ASTM D7169] float32
 49  17 Mass% Recovered (°C) [ASTM D7169] float32
 50  18 Mass% Recovered (°C) [ASTM D7169] float32
 51  19 Mass% Recovered (°C) [ASTM D7169] float32
 52  20 Mass% Recovered (°C) [ASTM D7169] float32
 53  21 Mass% Recovered (°C) [ASTM D7169] float32
 54  22 Mass% Recovered (°C) [ASTM D7169] float32
 55  23 Mass% Recovered (°C) [ASTM D7169] float32
 56  24 Mass% Recovered (°C) [ASTM D7169] float32
 57  25 Mass% Recovered (°C) [ASTM D7169] float32
 58  26 Mass% Recovered (°C) [ASTM D7169] float32
 59  27 Mass% Recovered (°C) [ASTM D7169] float32
 60  28 Mass% Recovered (°C) [ASTM D7169] float32
 61  29 Mass% Recovered (°C) [ASTM D7169] float32
 62  30 Mass% Recovered (°C) [ASTM D7169] float32
 63  31 Mass% Recovered (°C) [ASTM D7169] float32
 64  32 Mass% Recovered (°C) [ASTM D7169] float32
 65  33 Mass% Recovered (°C) [ASTM D7169] float32
 66  34 Mass% Recovered (°C) [ASTM D7169] float32
 67  35 Mass% Recovered (°C) [ASTM D7169] float32
 68  36 Mass% Recovered (°C) [ASTM D7169] float32
 69  37 Mass% Recovered (°C) [ASTM D7169] float32
 70  38 Mass% Recovered (°C) [ASTM D7169] float32
 71  39 Mass% Recovered (°C) [ASTM D7169] float32
 72  40 Mass% Recovered (°C) [ASTM D7169] float32
 73  41 Mass% Recovered (°C) [ASTM D7169] float32
 74  42 Mass% Recovered (°C) [ASTM D7169] float32
 75  43 Mass% Recovered (°C) [ASTM D7169] float32
 76  44 Mass% Recovered (°C) [ASTM D7169] float32
 77  45 Mass% Recovered (°C) [ASTM D7169] float32
 78  46 Mass% Recovered (°C) [ASTM D7169] float32
 79  47 Mass% Recovered (°C) [ASTM D7169] float32
 80  48 Mass% Recovered (°C) [ASTM D7169] float32
 81  49 Mass% Recovered (°C) [ASTM D7169] float32
 82  50 Mass% Recovered (°C) [ASTM D7169] float32
 83  51 Mass% Recovered (°C) [ASTM D7169] float32
 84  52 Mass% Recovered (°C) [ASTM D7169] float32
 85  53 Mass% Recovered (°C) [ASTM D7169] float32
 86  54 Mass% Recovered (°C) [ASTM D7169] float32
 87  55 Mass% Recovered (°C) [ASTM D7169] float32
 88  56 Mass% Recovered (°C) [ASTM D7169] float32
```

```

89  57 Mass% Recovered (°C) [ASTM D7169] float32
90  58 Mass% Recovered (°C) [ASTM D7169] float32
91  59 Mass% Recovered (°C) [ASTM D7169] float32
92  60 Mass% Recovered (°C) [ASTM D7169] float32
93  61 Mass% Recovered (°C) [ASTM D7169] float32
94  62 Mass% Recovered (°C) [ASTM D7169] float32
95  63 Mass% Recovered (°C) [ASTM D7169] float32
96  64 Mass% Recovered (°C) [ASTM D7169] float32
97  65 Mass% Recovered (°C) [ASTM D7169] float32
98  66 Mass% Recovered (°C) [ASTM D7169] float32
99  67 Mass% Recovered (°C) [ASTM D7169] float32
100 68 Mass% Recovered (°C) [ASTM D7169] float32
101 69 Mass% Recovered (°C) [ASTM D7169] float32
102 70 Mass% Recovered (°C) [ASTM D7169] float32
103 71 Mass% Recovered (°C) [ASTM D7169] float32
104 72 Mass% Recovered (°C) [ASTM D7169] float32
105 73 Mass% Recovered (°C) [ASTM D7169] float32
106 74 Mass% Recovered (°C) [ASTM D7169] float32
107 75 Mass% Recovered (°C) [ASTM D7169] float32
108 76 Mass% Recovered (°C) [ASTM D7169] float32
109 77 Mass% Recovered (°C) [ASTM D7169] float32
110 78 Mass% Recovered (°C) [ASTM D7169] float32
111 79 Mass% Recovered (°C) [ASTM D7169] float32
112 80 Mass% Recovered (°C) [ASTM D7169] float32
113 81 Mass% Recovered (°C) [ASTM D7169] float32
114 82 Mass% Recovered (°C) [ASTM D7169] float32
115 83 Mass% Recovered (°C) [ASTM D7169] float32
116 84 Mass% Recovered (°C) [ASTM D7169] float32
117 85 Mass% Recovered (°C) [ASTM D7169] float32
118 86 Mass% Recovered (°C) [ASTM D7169] float32
119 87 Mass% Recovered (°C) [ASTM D7169] float32
120 88 Mass% Recovered (°C) [ASTM D7169] float32
121 89 Mass% Recovered (°C) [ASTM D7169] float32
122 90 Mass% Recovered (°C) [ASTM D7169] float32
123 91 Mass% Recovered (°C) [ASTM D7169] float32
124 92 Mass% Recovered (°C) [ASTM D7169] float32
125 93 Mass% Recovered (°C) [ASTM D7169] float32
126 94 Mass% Recovered (°C) [ASTM D7169] float32
127 95 Mass% Recovered (°C) [ASTM D7169] float32
128 96 Mass% Recovered (°C) [ASTM D7169] float32
129 97 Mass% Recovered (°C) [ASTM D7169] float32
130 98 Mass% Recovered (°C) [ASTM D7169] float32
131 99 Mass% Recovered (°C) [ASTM D7169] float32
132 target_label int64
dtypes: datetime64[ns](1), float32(128), int64(1), object(3)
memory usage: 2.8+ MB

```

The dataset contains 133 columns and 5,303 rows\* of data (\*Note: new rows of data are added incrementally over time so this row count may be larger than the current count).

- 3 columns are categorical (`Crude`, `Batch`, and `Location`)
- 1 column is datetime (`Sample Date (yyyy-mm-dd)`)
- 1 column is integer (`target_label`)
- 128 columns are numeric

## Feature Descriptions

- These are the key variables that will be used in the modeling process. Together, these variables represent a comprehensive characterization of crude oil samples, covering density, composition, impurities, metal content, acidity, and detailed distillation properties that collectively influence crude quality and processing behavior.

Feature	Description
<b>Crude</b>	Name of the crude oil grade.
<b>Batch</b>	Shipment or lot identifier for the sample.
<b>Sample Date (yyyy-mm-dd)</b>	Date the crude sample was collected.
<b>Location</b>	Site or facility where the sample was taken.
<b>Density (kg/m³) [ASTM D5002]</b>	Mass per unit volume; indicates heaviness of crude.
<b>Gravity (°API) [ASTM D5002]</b>	Measure of crude lightness; inverse of density.
<b>Sulphur (wt%) [ASTM D4294]</b>	Sulphur content; affects refining and emissions.
<b>Micro Carbon Residue (wt%) [ASTM D4530]</b>	Carbon left after pyrolysis; indicator of coke-forming tendency.
<b>Sediment (ppmw) [ASTM D4807]</b>	Solid impurities or particles in crude.
<b>Total Acid Number (mgKOH/g) [ASTM D664]</b>	Measure of acidity; indicates corrosive potential.
<b>Salt (ptb) [ASTM D3230]</b>	Salt concentration; impacts corrosion and desalting needs.
<b>Nickel (mg/kg) [ASTM D5708A]</b>	Metal contaminant affecting catalyst life.
<b>Vanadium (mg/kg) [ASTM D5708A]</b>	Metal impurity impacting refining catalysts.
<b>C1–C10 Components (vol%) [GC/FID]</b>	Light hydrocarbon composition (methane to decane) by gas chromatography; indicates volatility and gas content.
<b>iC4–nC5 (vol%) [GC/FID]</b>	Branched and normal butane/pentane fractions; measure of light-end structure.
<b>C6–C10 (vol%) [GC/FID]</b>	Heavier paraffins; affect vapor pressure and yield profiles.

Feature	Description
Nitrogen – C4- (vol%) [GPA 2177M]	Nitrogen content in light gas fraction.
Carbon Dioxide – C4- (vol%) [GPA 2177M]	CO <sub>2</sub> concentration in gas fraction.
C1-nC4 – C4- (vol%) [GPA 2177M]	Light hydrocarbon breakdown (methane through butane) in gas stream.
IBP (°C) [ASTM D7169]	Initial boiling point; start of vaporization during distillation.
1-99 Mass% Recovered (°C) [ASTM D7169]	Distillation temperatures where 1–99% of sample mass is vaporized; describes boiling range and fractionation behavior.
target_label	Integer encoded variable to describe the crude grade groups (corresponds to 'target_label' in the <code>crude_oils_dict</code> )

## Data Summary - Conclusions/Discussions/Next Steps:

- The Data Summary confirmed that the combined dataset contains over 5,000 samples and 133 features covering physical, chemical, and compositional properties across four crude oil groups. It showed that most variables are numeric and suitable for quantitative analysis, while a few categorical and datetime columns provide sample context. Minor inconsistencies in text formatting and missing values were identified for later cleaning.
- With the dataset now organized and understood, the next step will be Exploratory Data Analysis (EDA) to examine feature distributions, detect missing values, assess correlations, and identify any outliers or patterns that may influence modeling.

## EDA

For the Exploratory Data Analysis (EDA) we begin by organizing the dataset into logical column groups—categorical variables, crude quality features, and distillation temperature features—to simplify analysis. We then examine the numeric column distributions through histograms to visualize how the crude grade groups differ across key quality and distillation variables. After exploring these distributions, the analysis evaluates missing data patterns to identify incomplete or inconsistent features and concludes with a correlation assessment to detect relationships and redundancy among numeric features.

### Column Groups

- These column groups will be used throughout the analysis and to aid in modeling

```
In [62]: # Categorical columns
categorical_cols = ['Crude', 'Batch', 'Sample Date (yyyy-mm-dd)', 'Location', 'target_label']

# Columns that describe the quality of the crude oil
crude_quality_cols = df.loc[:, 'Density (kg/m³) [ASTM D5002]':'IBP (°C) [ASTM D7169]'].columns.to_list()

# Columns that describe distillation temperatures for the crude oil
dist_temp_cols = df.loc[:, '1 Mass% Recovered (°C) [ASTM D7169]':'99 Mass% Recovered (°C) [ASTM D7169]'].columns.to_list()
```

### Crude Quality Histograms

- These histograms give us a sense of the distribution of the `crude_quality_cols`.
- Note: To reduce memory usage for large sets of plots the dataset is temporarily reduced by randomly selecting a subset of data points. Histograms may look different depending on the size of the subset, but generally show the distributions by crude grades.

```
In [63]: # Function to plot a set of histograms for a subset of columns
def plot_column_histograms(input_df, subset, chart_title, col_count, row_sample_size):
    # Melt the numeric columns into one column
    df_melt = input_df[subset + ['target_label']].melt(id_vars='target_label', var_name='feature', value_name='value').sample(n=row_sample_size)

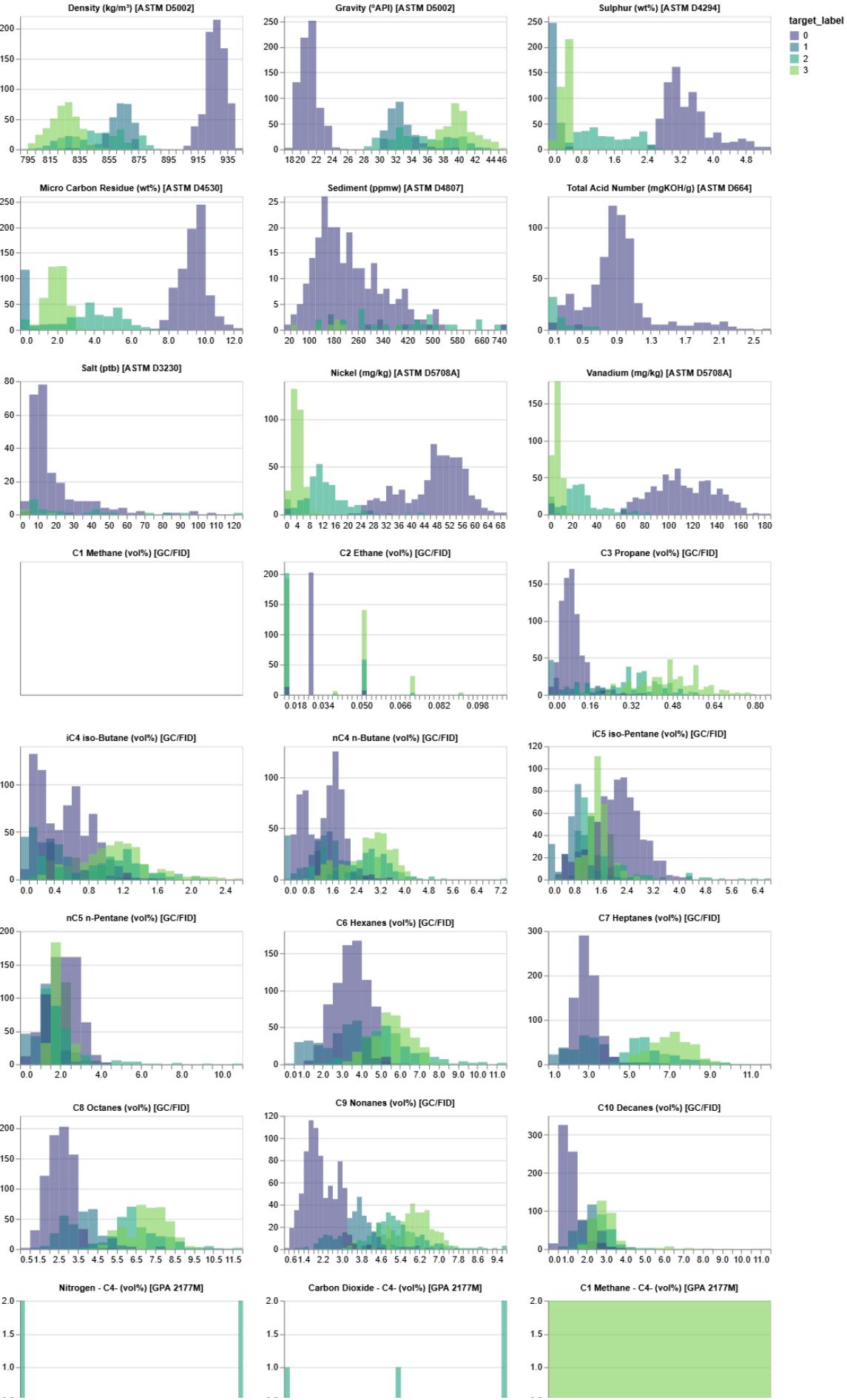
    # Create a base Altair histogram chart
    chart = alt.Chart(df_melt).mark_bar(opacity=0.6, binSpacing=0).encode(
        x=alt.X('value:Q').axis(title='').bin(maxbins=50),
        y=alt.Y('count():Q').axis(title='').stack(None),
        color=alt.Color('target_label:N').scale(scheme='viridis')
    ).properties(
        width=750 / col_count,
        height=150
    )

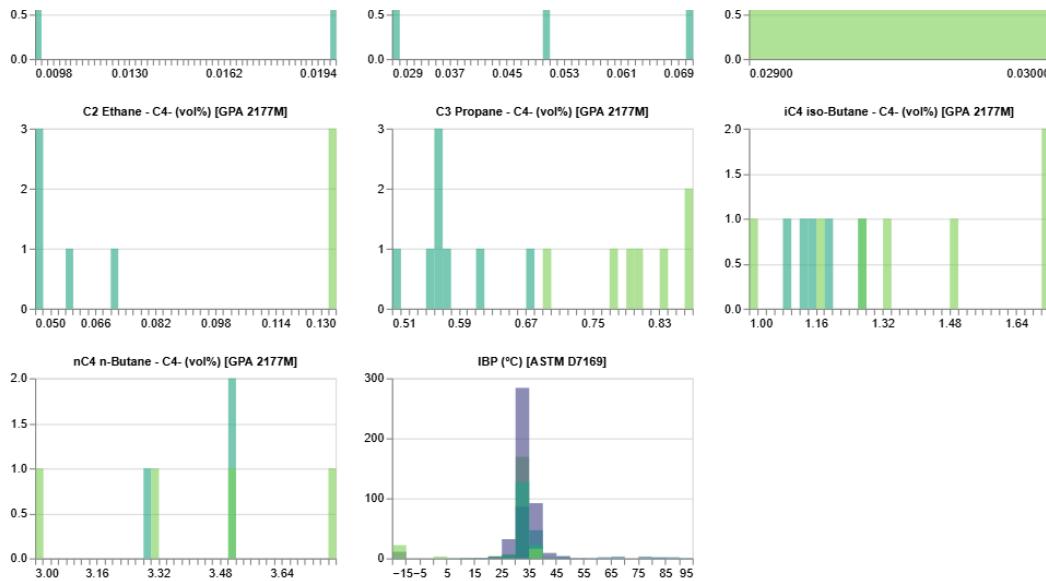
    # Display a histogram for each numeric_columns
    chart = alt.ConcatChart(
        title=alt.Title(f'{chart_title}', fontSize=20),
        concat=[chart.transform_filter(alt.datum.feature == value).properties(title=value) for value in subset],
        columns=col_count,
        .configure_title(
            fontSize=10
        ).resolve_axis(
            x='independent',
            y='independent'
        ).resolve_scale(
            x='independent',
            y='independent'
        )
    )

    return chart

plot_column_histograms(df, crude_quality_cols, 'Crude Quality Columns Data Distributions', 3, 50000)
```

## Out[63]: Crude Quality Columns Data Distributions





The histograms show a clear definition between the Heavy Sour and Sweet Synthetic crude oil groups, when compared to the light-medium grades. Its interesting to note that the histograms for the Light & Medium Sweet and Light & Medium Sour groups often overlap possibly hinting at some relationship between the two groups. In general, there exists some uneven data coverage across the crude grade groups, with several columns containing sparse or missing distributions for certain grades. Features such as Sediment, Total Acid Number, Salt, and the gas composition variables (e.g., C1 Methane, Nitrogen-C4-, CO<sub>2</sub>-C4-, etc.) lack consistent representation and should be considered for removal during the Data Cleaning phase to prevent bias in model training.

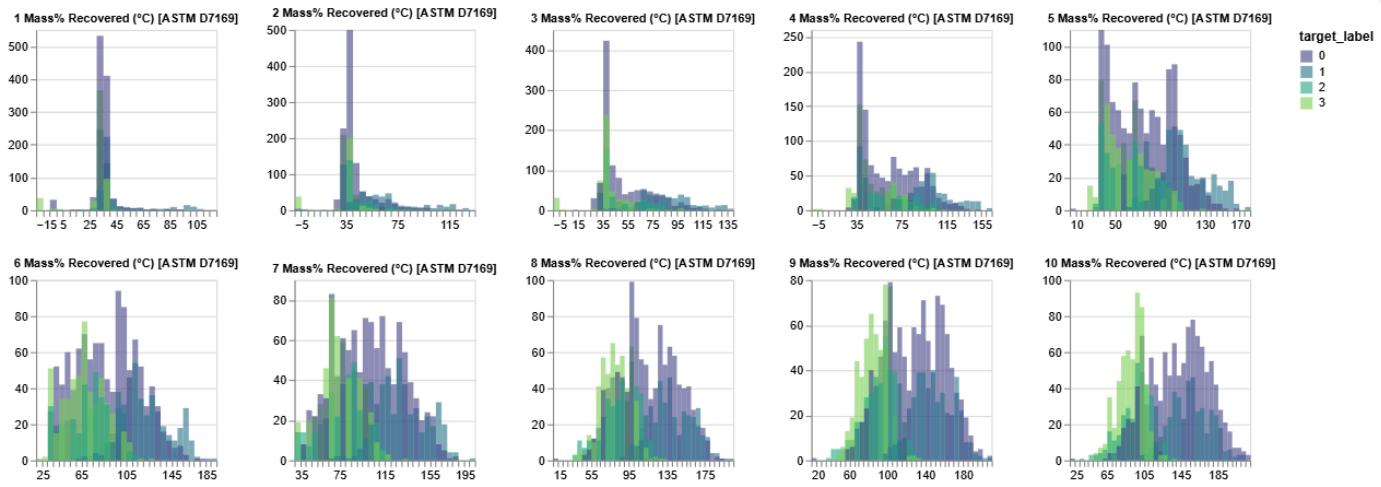
## Distillation Temperature Histograms

- These histograms show how the distributions of crude oil grades vary across the dist\_temp\_cols as the mass percent recovered increases.
- Note: To manage memory when generating multiple plots, a random subset of the dataset is used. The shapes of the histograms may vary slightly with different sample sizes but consistently illustrate the distribution patterns by crude grade.

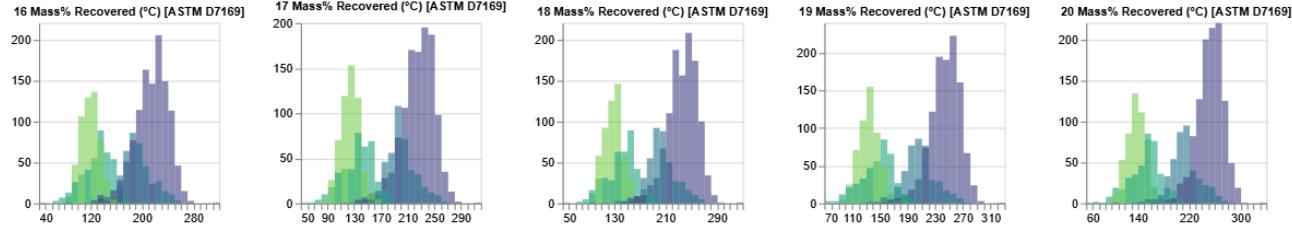
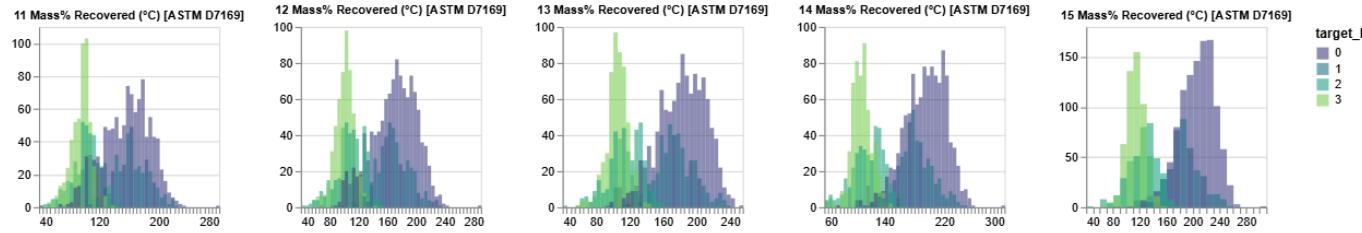
```
In [64]: def chunk(seq, size):
    for i in range(0, len(seq), size):
        yield seq[i:i+size]

for subset in chunk(dist_temp_cols, 10):
    display(plot_column_histograms(df, subset, 'Mass % Recovered Columns Data Distributions', 5, 45000))
```

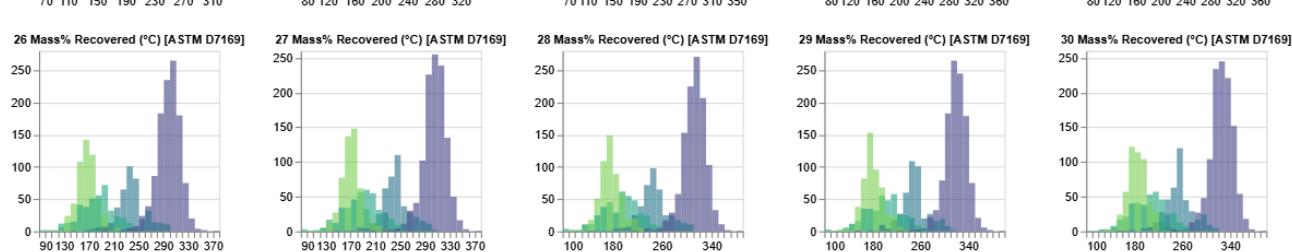
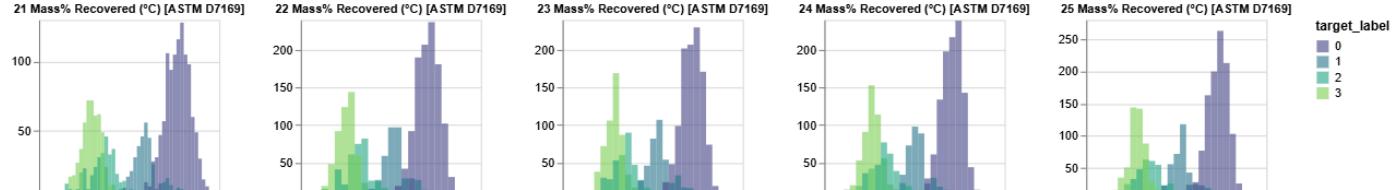
## Mass % Recovered Columns Data Distributions



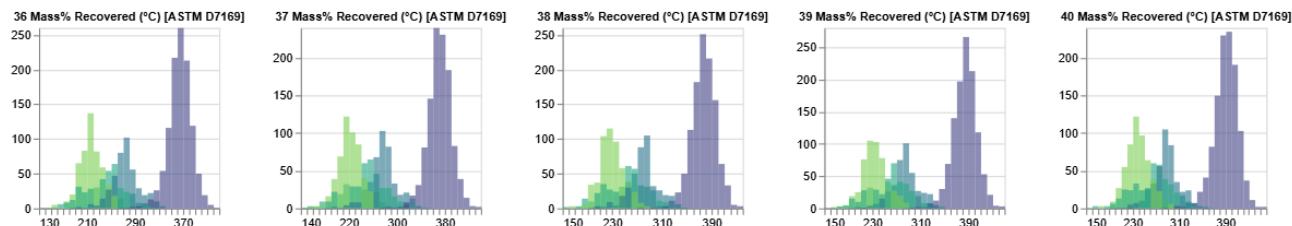
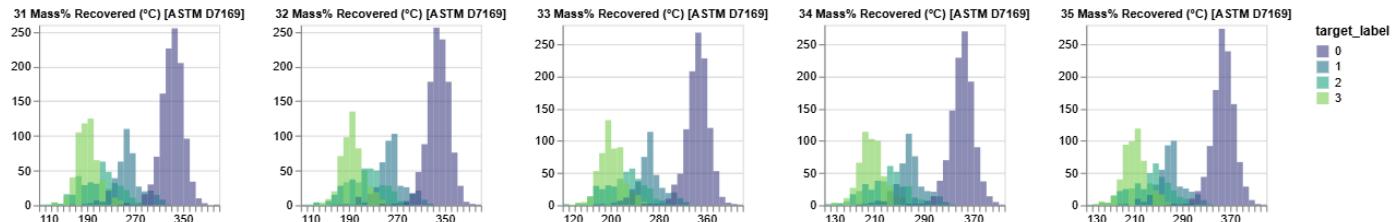
## Mass % Recovered Columns Data Distributions



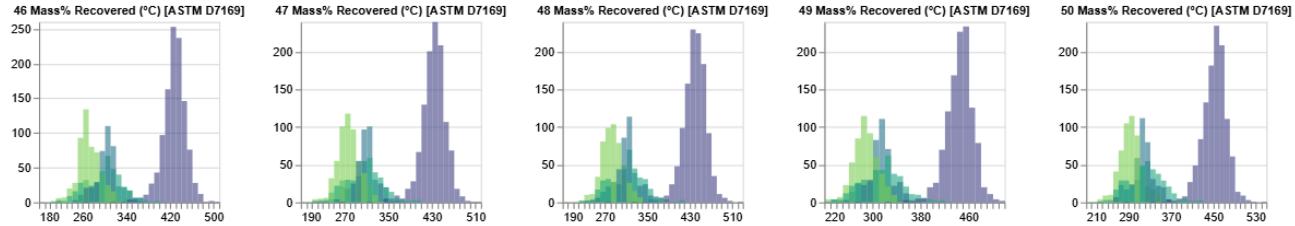
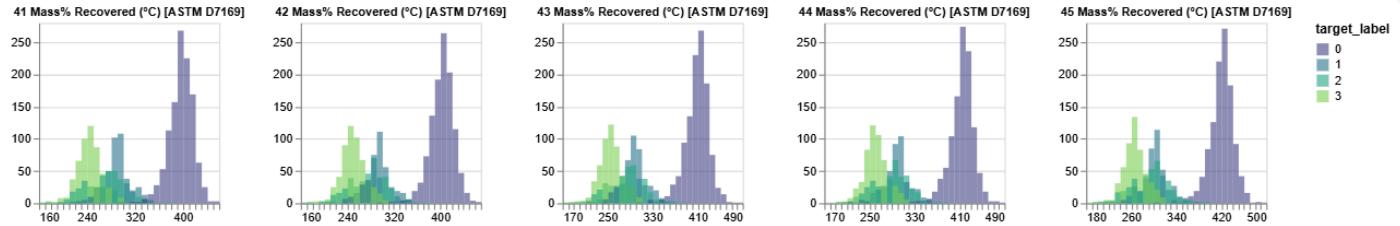
## Mass % Recovered Columns Data Distributions



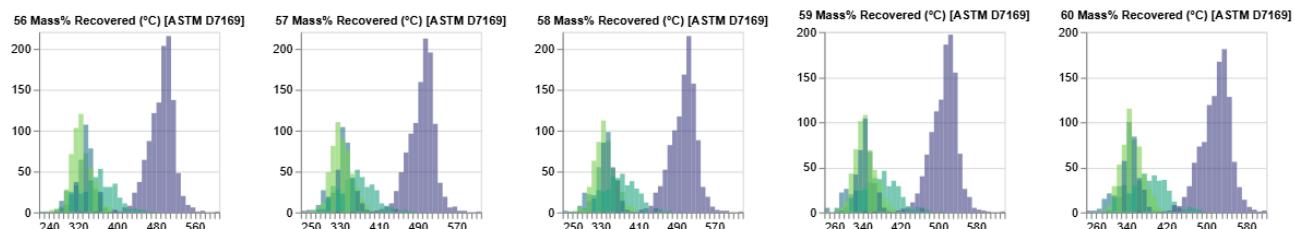
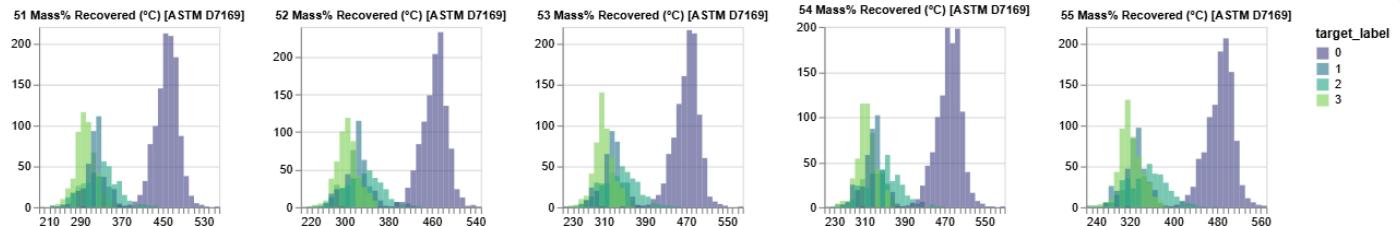
## Mass % Recovered Columns Data Distributions



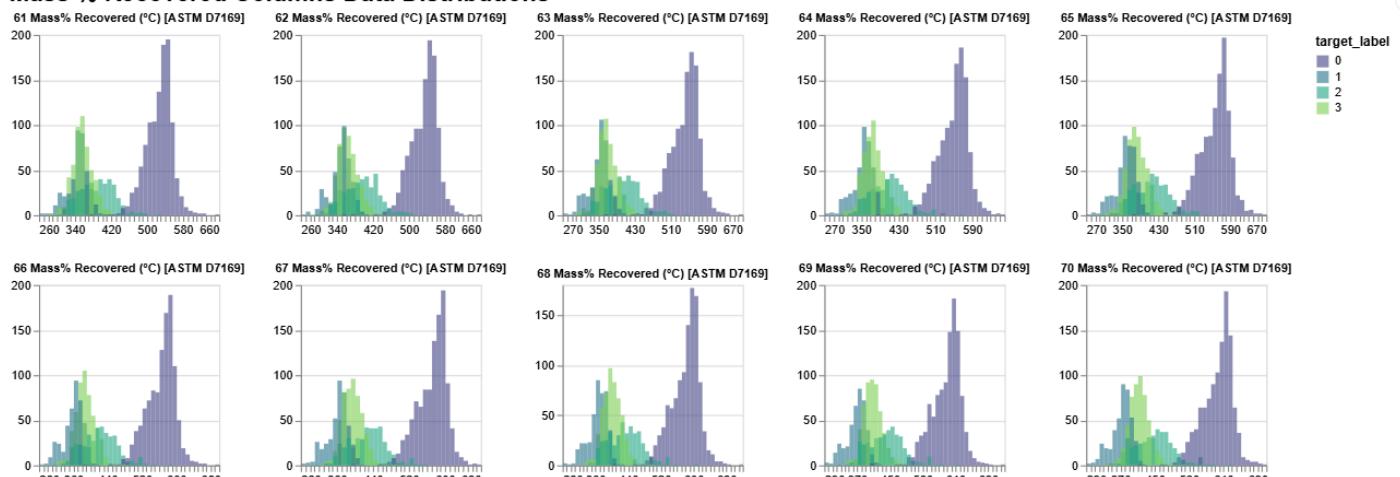
## Mass % Recovered Columns Data Distributions



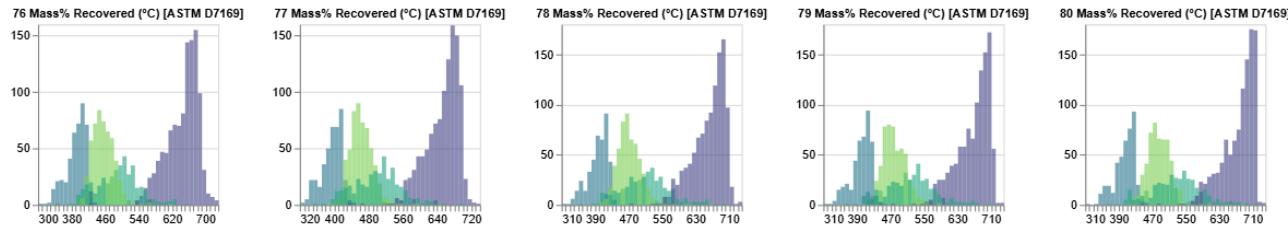
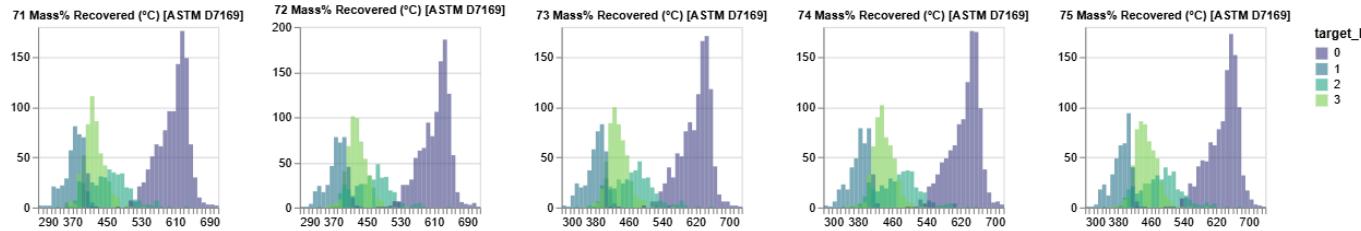
## Mass % Recovered Columns Data Distributions



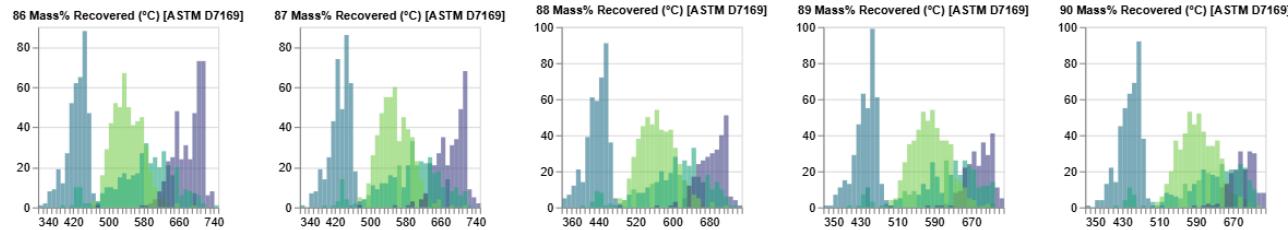
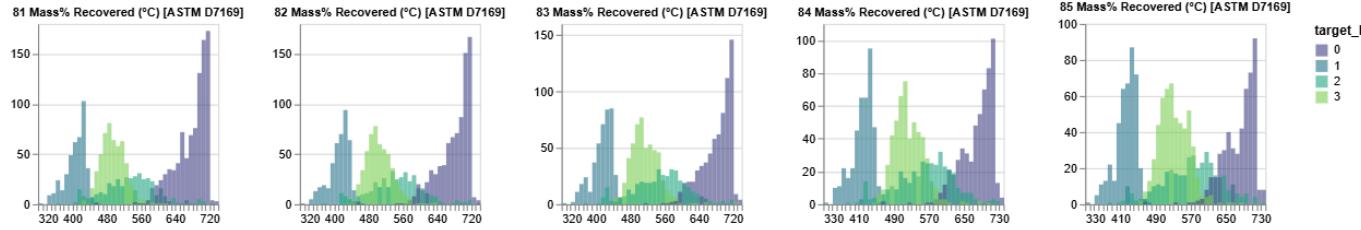
## Mass % Recovered Columns Data Distributions



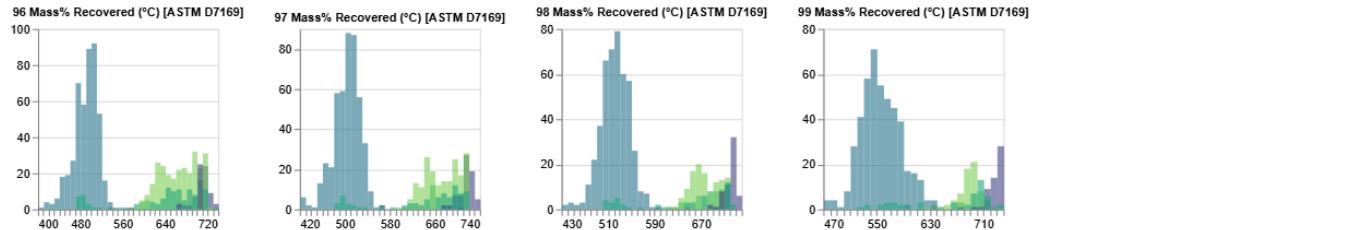
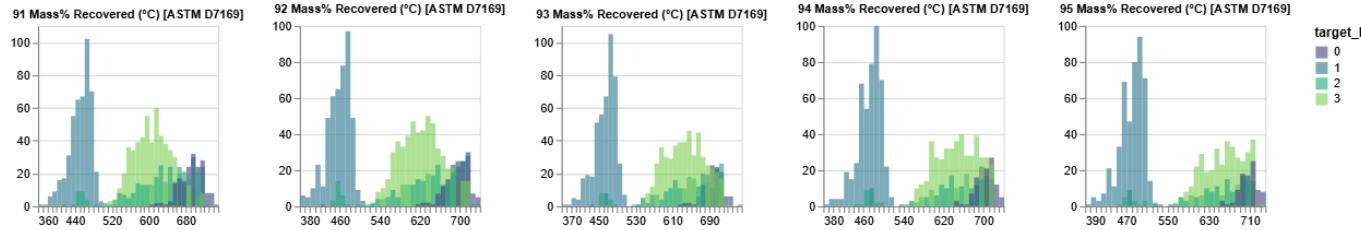
## Mass % Recovered Columns Data Distributions



## Mass % Recovered Columns Data Distributions



## Mass % Recovered Columns Data Distributions



The Mass % Recovered histograms show overlapping and mixed distributions at low recovery levels but become more distinct beyond roughly 5 % recovered. As recovery increases, the Heavy Sour and Synthetic crude grade groups separate clearly. As in the crude quality histograms we find that the light-medium grades are often overlapped indicating some relationship.

## Check for Missing Values

- This step identifies missing data across all features and crude groups to assess dataset completeness and guide later cleaning and imputation decisions.

```
In [65]: # Calculate the total number of values in the dataset
total_value_count = df.notna().sum().sum()
print(f'Total count of values in the dataset = {total_value_count:,d}')

# Calculate the number of missing values in the dataset
```

```

missing_value_count = df.isna().sum().sum()
print(f'Count of missing values in the dataset = {missing_value_count:,d}')

# Calculate the percent missing values in the dataset
percent_missing_values = missing_value_count / total_value_count * 100
print(f'Percent of values that are missing = {percent_missing_values:.1f}%')

```

Total count of values in the dataset = 380,970  
Count of missing values in the dataset = 324,329  
Percent of values that are missing = 85.1%

In [66]: # Create heatmap to visualize where data is missing in the dataset  
fig, (ax1, ax2, ax3, ax4) = plt.subplots(1,4, figsize=(30,30), sharey=True)  
plt.suptitle('Rows with Missing Values by Crude Grade', y=0.92, fontsize = 30)

```

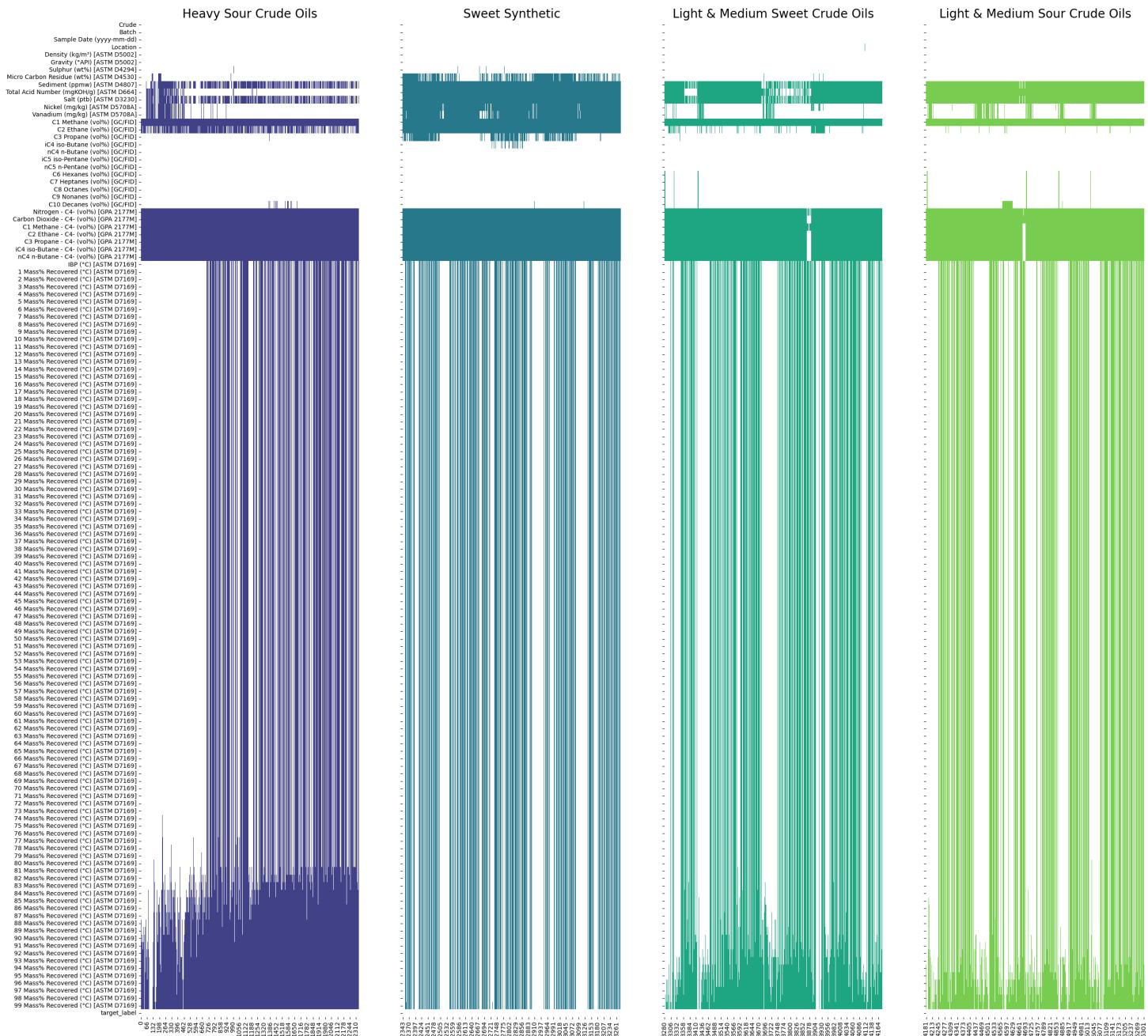
sns.heatmap(df[df['target_label']==0].T.isna(), cmap=sns.color_palette('blend:#ffffff,#414487'), cbar=False, ax=ax1)
sns.heatmap(df[df['target_label']==1].T.isna(), cmap=sns.color_palette('blend:#ffffff,#2a788e'), cbar=False, ax=ax2)
sns.heatmap(df[df['target_label']==2].T.isna(), cmap=sns.color_palette('blend:#ffffff,#22a884'), cbar=False, ax=ax3)
sns.heatmap(df[df['target_label']==3].T.isna(), cmap=sns.color_palette('blend:#ffffff,#7ad151'), cbar=False, ax=ax4)

ax1.set_title('Heavy Sour Crude Oils', fontsize = 20)
ax2.set_title('Sweet Synthetic', fontsize = 20)
ax3.set_title('Light & Medium Sweet Crude Oils', fontsize = 20)
ax4.set_title('Light & Medium Sour Crude Oils', fontsize = 20)

plt.show()

```

Rows with Missing Values by Crude Grade



The missing values heatmap shows significant data gaps across all crude groups. As we found in the crude quality columns histograms the Sediment, Total Acid Number, Salt, and gas composition columns (C1 Methane, Nitrogen–C4–, CO<sub>2</sub>–C4–, etc.) all have a large amount of missing values. These features are largely incomplete across samples and will likely need to be dropped or excluded in later cleaning steps to maintain dataset consistency.

- The `Location` column has one missing value:

```
In [67]: # Find where the missing Location value is  
df[df['Location'].isna()].index
```

```
Out[67]: Index([4108], dtype='int64')
```

```
In [68]: # Extract row where Location is missing  
df.iloc[5230,:]
```

```
Out[68]: Crude           Rainbow  
Batch            RA-739  
Sample Date (yyyy-mm-dd) 2025-09-02 00:00:00  
Location          Edmonton  
Density (kg/m³) [ASTM D5002] 831.400024  
...  
96 Mass% Recovered (°C) [ASTM D7169]      NaN  
97 Mass% Recovered (°C) [ASTM D7169]      NaN  
98 Mass% Recovered (°C) [ASTM D7169]      NaN  
99 Mass% Recovered (°C) [ASTM D7169]      NaN  
target_label          3  
Name: 5230, Length: 133, dtype: object
```

```
In [69]: # Check what the `Location` usually is for when `Crude`=='Pembina Light Sour'  
df[df['Crude']=='Pembina Light Sour'].head(30)
```

Out[69]:

		Crude	Sample Date (yyyy-mm-dd)	Location	Density (kg/m³) [ASTM D5002]	Gravity (°API) [ASTM D5002]	Sulphur (wt%) [ASTM D4294]	Micro Carbon Residue (wt%) [ASTM D4530]	Sediment (ppmw) [ASTM D4807]	Total Acid Number (mgKOH/g) ... [ASTM D664]	91 Mass% Recovered (°C) [ASTM D7169]	92 Mass% Recovered (°C) [ASTM D7169]	93 Mass% Recovered (°C) [ASTM D7169]	94 Mass% Recovered (°C) [ASTM D7169]	95 Mass% Recovered (°C) [ASTM D7169]	96 Mass% Recovered (°C) [ASTM D7169]
4095	Pembina Light Sour	PLS-570	2010-05-04	Edmonton	830.700012	38.599998	0.92	2.2	NaN	NaN ... 600.500000	618.900024	639.400024	663.000000	694.599976	NaN	NaN
4096	Pembina Light Sour	PLS-026	2007-12-02	Edmonton	822.299988	40.400002	0.77	1.4	NaN	NaN ... 539.299988	552.799988	567.299988	582.400024	599.299988	619.000000	NaN
4097	Pembina Light Sour	PLS-042	2008-01-02	Edmonton	823.500000	40.200001	0.70	0.3	NaN	NaN ... 564.400024	579.200012	596.099976	615.599976	638.400024	669.200012	NaN
4098	Pembina Light Sour	PLS-317	2008-04-14	Edmonton	822.400024	40.400002	0.75	1.5	NaN	NaN ... 626.099976	651.400024	686.799988	NaN	NaN	NaN	NaN
4099	Pembina Light Sour	PLS-332	2008-05-14	Edmonton	823.700012	40.099998	0.74	1.3	NaN	NaN ... 618.900024	638.299988	661.000000	691.500000	NaN	NaN	NaN
4100	Pembina Light Sour	PLS-342	2008-06-04	Edmonton	821.900024	40.500000	0.69	1.6	NaN	NaN ... 576.099976	592.200012	610.200012	631.099976	655.500000	689.900024	NaN
4101	Pembina Light Sour	PLS-359	2008-07-08	Edmonton	828.099976	39.200001	0.79	1.5	NaN	NaN ... 576.000000	593.200012	612.599976	635.599976	664.599976	704.900024	NaN
4102	Pembina Light Sour	PLS-376	2008-08-12	Edmonton	822.000000	40.500000	0.74	1.5	NaN	NaN ... 643.400024	677.299988	719.500000	NaN	NaN	NaN	NaN
4103	Pembina Light Sour	PLS-501	2009-03-04	Edmonton	823.099976	40.299999	0.74	1.5	NaN	NaN ... 549.900024	565.000000	580.599976	597.799988	617.799988	641.099976	NaN
4104	Pembina Light Sour	PLS-352	2009-05-04	Edmonton	824.299988	40.000000	0.78	1.7	NaN	NaN ... 587.200012	604.799988	625.200012	649.400024	684.200012	NaN	NaN
4105	Pembina Light Sour	PLS-810	2009-09-02	Edmonton	826.900024	39.500000	0.83	1.6	NaN	NaN ... 549.700012	564.000000	578.500000	594.599976	612.599976	633.200012	NaN
4106	Pembina Light Sour	PLS-440	2009-11-02	Edmonton	822.599976	40.400002	0.77	1.6	NaN	NaN ... 554.400024	568.299988	583.099976	599.500000	618.700012	640.900024	NaN
4107	Pembina Light Sour	PLS-534	2010-03-01	Edmonton	831.000000	38.599998	0.85	2.1	NaN	NaN ... 604.200012	622.299988	642.200012	665.900024	695.500000	NaN	NaN
4108	Pembina Light Sour	PLS-685	2010-09-02	None	825.200012	39.799999	0.77	0.9	NaN	NaN ... 579.599976	596.500000	615.900024	638.500000	661.400024	702.000000	NaN
4109	Pembina Light Sour	PLS-713	2010-10-01	Edmonton	827.599976	39.299999	0.90	2.0	NaN	NaN ... 578.700012	595.500000	614.500000	636.200012	662.299988	696.000000	NaN
4110	Pembina Light Sour	PLS-744	2010-11-02	Edmonton	829.500000	38.900002	0.88	2.1	NaN	NaN ... 570.799988	586.500000	604.000000	624.299988	647.799988	678.299988	NaN
4111	Pembina Light Sour	PLS-594	2011-03-03	Edmonton	830.500000	38.700001	0.87	2.3	NaN	NaN ... 628.799988	649.700012	675.000000	703.099976	NaN	NaN	NaN
4112	Pembina Light Sour	PLS-645	2011-05-09	Edmonton	828.500000	39.099998	0.84	2.3	NaN	NaN ... 597.599976	614.900024	634.099976	655.400024	683.000000	710.599976	NaN
4113	Pembina Light Sour	PLS-283	2011-09-04	Edmonton	831.200012	38.599998	0.83	2.3	NaN	NaN ... 630.099976	649.900024	672.200012	696.099976	NaN	NaN	NaN
4114	Pembina Light Sour	PLS-357	2012-01-12	Edmonton	824.299988	40.000000	0.66	1.7	NaN	NaN ... NaN	NaN	NaN	NaN	NaN	NaN	NaN
4115	Pembina Light Sour	PLS-370	2012-02-07	Edmonton	824.400024	40.000000	0.67	2.0	NaN	NaN ... 551.599976	564.799988	577.799988	592.500000	608.400024	626.099976	NaN
4116	Pembina Light Sour	PLS-385	2012-03-06	Edmonton	836.500000	37.500000	0.68	1.8	NaN	NaN ... NaN	NaN	NaN	NaN	NaN	NaN	NaN
4117	Pembina Light Sour	PLS-430	2012-06-05	Edmonton	819.799988	41.000000	0.40	1.3	NaN	NaN ... NaN	NaN	NaN	NaN	NaN	NaN	NaN

			Sample Date (yyyy-mm-dd)	Location	Density (kg/m³) [ASTM D5002]	Gravity ("API) [ASTM D5002]	Sulphur (wt%) [ASTM D4294]	Micro Carbon Residue (wt%) [ASTM D4530]	Sediment (ppmw) [ASTM D4807]	Total Acid Number (mgKOH/g) [ASTM D664]	91 Mass% Recovered ...	92 Mass% Recovered (°C) [ASTM D7169]	93 Mass% Recovered (°C) [ASTM D7169]	94 Mass% Recovered (°C) [ASTM D7169]	95 Mass% Recovered (°C) [ASTM D7169]	96 Mass% Recovered (°C) [ASTM D7169]
4118	Pembina Light Sour	PLS-460	2012-08-07	Edmonton	834.700012	37.900002	0.76	2.2	NaN	NaN ...	579.299988	593.799988	609.500000	627.000000	646.200012	668.700012
4119	Pembina Light Sour	PLS-176	2012-09-11	Edmonton	823.900024	40.099998	0.59	1.6	NaN	NaN ...	NaN	NaN	NaN	NaN	NaN	NaN
4120	Pembina Light Sour	PLS-483	2012-10-02	Edmonton	828.700012	39.099998	0.59	2.1	NaN	NaN ...	570.900024	584.599976	599.200012	615.599976	633.599976	653.799988
4121	Pembina Light Sour	PLS-505	2012-11-05	Edmonton	828.799988	39.099998	0.77	2.2	NaN	NaN ...	NaN	NaN	NaN	NaN	NaN	NaN
4122	Pembina Light Sour	PLS-516	2012-12-03	Edmonton	823.500000	40.200001	0.80	2.1	NaN	NaN ...	NaN	NaN	NaN	NaN	NaN	NaN
4123	Pembina Light Sour	PLS-231	2013-01-01	Edmonton	827.799988	39.299999	0.78	2.2	NaN	NaN ...	NaN	NaN	NaN	NaN	NaN	NaN
4124	Pembina Light Sour	PLS-241	2013-02-02	Edmonton	846.000000	35.599998	1.09	2.7	NaN	NaN ...	NaN	NaN	NaN	NaN	NaN	NaN

30 rows x 133 columns

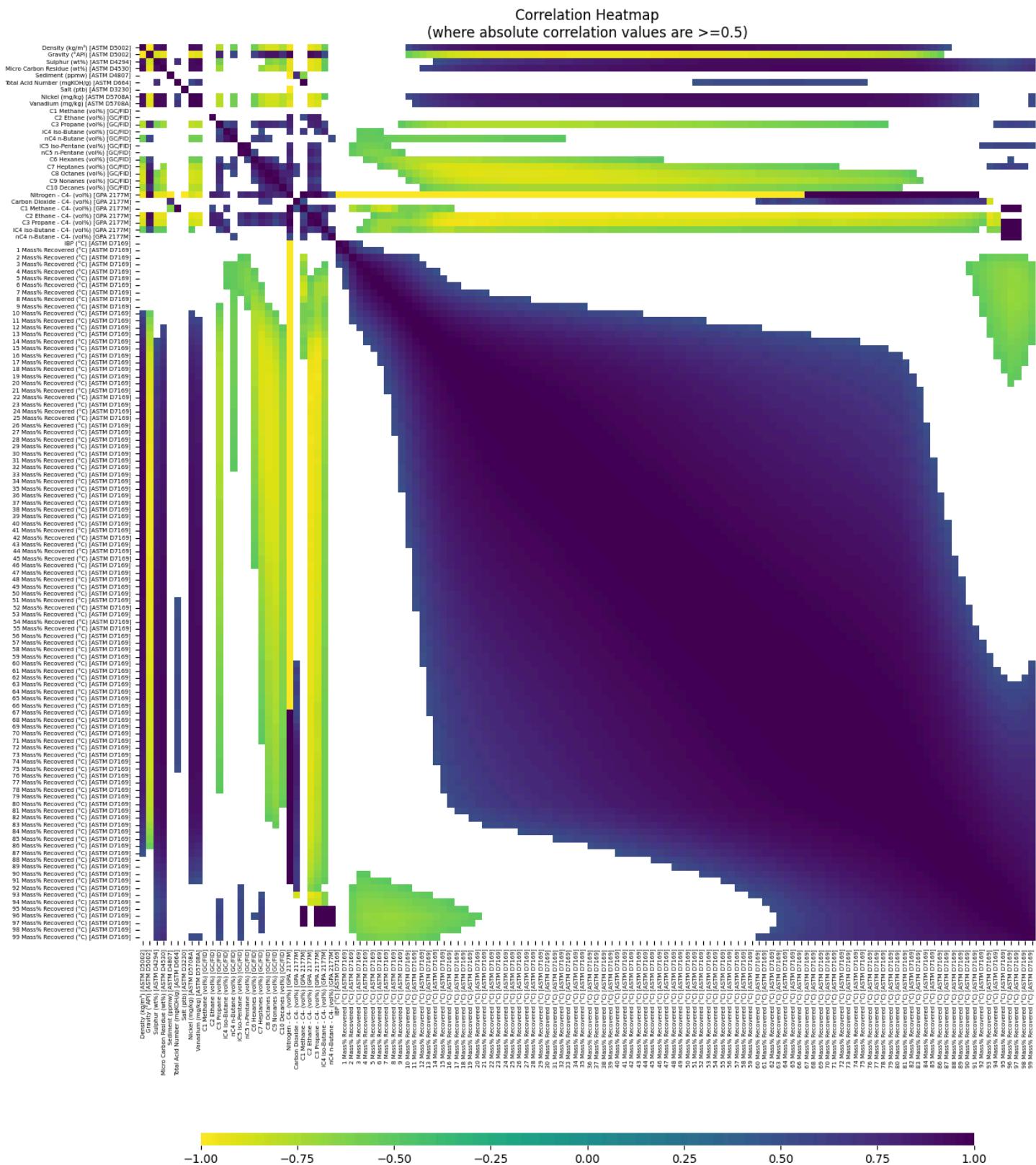
```
In [70]: # Location looks like it is always 'Edmonton', so I will impute this value
df.loc[df['Crude']=='Pembina Light Sour', 'Location'] = 'Edmonton'
```

## Column Correlations

- We need to check if there exists strong correlations between the numeric features. If there are then we should consider dropping one or more of the correlated features from the dataset.
- To aid in this evaluation we will visualize the data using a heatmap, where values of [-1,1] indicate a strong correlation, and values close to 0 indicate a weak correlation.
  - Note: Only the correlation values between [-1.0, -0.5] and [0.5, 1.0] are colored in the heatmap in order to focus on the highly correlated variables.

```
In [71]: # Correlation Heatmap
fig, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(
    df[crude_quality_cols + dist_temp_cols].corr(),
    cmap='viridis_r',
    mask=abs(df[crude_quality_cols + dist_temp_cols].corr())<=0.5, # Plot only absolute correlation values that are >=0.5
    square=True,
    vmin=-1, vmax=1,
    xticklabels=True, yticklabels=True,
    cbar_kws={'location':'bottom', 'aspect':60, 'shrink':0.6}
)

plt.xticks(fontsize=5)
plt.yticks(fontsize=5)
plt.title('Correlation Heatmap\nwhere absolute correlation values are >=0.5')
plt.show()
```



The correlation heatmap shows strong relationships among many of the Mass % Recovered columns, indicating high multicollinearity across distillation temperature features. Other notable correlations appear between related chemical and physical properties (e.g., Density, API Gravity, Sulphur), suggesting that several features convey overlapping information.

## EDA - Conclusions/Discussions/Next Steps:

- The EDA revealed that while the dataset captures a wide range of crude oil properties, several features—such as Sediment, Total Acid Number, Salt, and gas composition variables—contain extensive missing data and inconsistent distributions across crude groups. Strong correlations were observed among the Mass % Recovered columns and between key physical properties like Density, API Gravity, and Sulphur, indicating potential redundancy.

- With these insights, the next step in the analysis will focus on Data Cleaning to prepare the dataset for modeling.

## Data Cleaning

This section focuses on adding datetime-based features, removing low-quality or highly correlated columns, imputing missing values, and developing a preprocessing pipeline to ready the dataset for modeling.

### Add Datetime Columns

- Add month number and year to the dataset to ensure effects of time are captured in the models. Month number is the number of months that have elapsed since the minimum date in the `Sample Date (yyyy-mm-dd)` column.

```
In [72]: # Add Months and Years columns
df['Months'] = (df['Sample Date (yyyy-mm-dd)'].dt.to_period('M') - df['Sample Date (yyyy-mm-dd)'].min()).to_period('M').apply(lambda x: x.n)
df['Year'] = df['Sample Date (yyyy-mm-dd)'].dt.year

# Update the crude_quality_cols list to add Months and Year columns
categorical_cols = categorical_cols + ['Months', 'Year']
```

### Drop Columns

- The missing values analysis showed that several columns contained significant gaps across all four crude oil grades. Because the limited number of available observations would make mean imputation unreliable and potentially introduce bias, these columns were excluded from the dataset to preserve data integrity and modeling accuracy.
- I will also be dropping the `Crude`, `Batch`, and `Sample Date (yyyy-mm-dd)` columns because they will not be used in this analysis.

```
In [73]: # Define the set of crude_quality_cols to drop
crude_quality_cols_to_drop = [
    'Sediment (ppmw) [ASTM D4807]',
    'Total Acid Number (mgKOH/g) [ASTM D664]',
    'Salt (ptb) [ASTM D3230]',
    'C1 Methane (vol%) [GC/FID]',
    'Nitrogen - C4- (vol%) [GPA 2177M]',
    'Carbon Dioxide - C4- (vol%) [GPA 2177M]',
    'C1 Methane - C4- (vol%) [GPA 2177M]',
    'C2 Ethane - C4- (vol%) [GPA 2177M]',
    'C3 Propane - C4- (vol%) [GPA 2177M]',
    'iC4 iso-Butane - C4- (vol%) [GPA 2177M]',
    'nC4 n-Butane - C4- (vol%) [GPA 2177M]']

# Drop columns from dataset
df = df.drop(crude_quality_cols_to_drop, axis=1)

# Update the crude_quality_cols list to remove the crude_quality_cols_to_drop
crude_quality_cols = [col for col in crude_quality_cols if col not in crude_quality_cols_to_drop]
```

```
In [74]: # Define categorical_cols to drop
categorical_cols_to_drop = [
    'Crude',
    'Batch',
    'Sample Date (yyyy-mm-dd)'
]

# Drop categorical columns
df = df.drop(categorical_cols_to_drop, axis=1)

# Update categorical_cols to remove the dropped columns
categorical_cols = [col for col in categorical_cols if col not in categorical_cols_to_drop]

categorical_cols
```

```
Out[74]: ['Location', 'target_label', 'Months', 'Year']
```

### Impute Values

- From my experience the following components should ideally not be present in Sweet Synthetic Crude Oil grades. There are not many occurrences of these components for the Sweet Synthetic group so I'm comfortable setting these values to zero.
  - `Nickel (mg/kg) [ASTM D5708A]`
  - `Vanadium (mg/kg) [ASTM D5708A]`
  - `C2 Ethane (vol%) [GC/FID]`
- After these adjustments, the remaining missing values will be imputed using a KNNImputer, which estimates missing entries based on the mean values of the nearest samples in feature space.
- There is also one missing value in the 'Location' column that will need to be imputed.

```
In [75]: # Fill Sweet Synthetic columns with zeros
df.loc[df['target_label']==1, ['Nickel (mg/kg) [ASTM D5708A]', 'Vanadium (mg/kg) [ASTM D5708A]', 'C2 Ethane (vol%) [GC/FID]']] = 0
```

```
In [76]: # Create a KNNImputer instance which uses the 10 nearest neighbors to the missing value to calculate an imputed mean value
imputer = KNNImputer(n_neighbors=10, weights='distance')
```

```
# Impute missing values for each crude type
for i in range(4):
    df.loc[df['target_label']==i, crude_quality_cols + dist_temp_cols] = imputer.fit_transform(df.loc[df['target_label']==i, crude_quality_cols + dist_temp_cols])

# Check that no missing values exist
df[crude_quality_cols + dist_temp_cols].isna().sum().sum()

Out[76]: np.int64(0)
```

## Pre-Processing

- This step standardizes numeric features and encodes categorical variables using a preprocessing pipeline to ensure consistent scaling across all inputs. The transformed feature matrix X is then prepared for modeling, with the target variable y\_true defined as the encoded crude grade labels.

```
In [77]: # Define the feature matrix
X = df[['Months', 'Year', 'Location'] + crude_quality_cols + dist_temp_cols]

# Define the target label values
y_true = df['target_label']

# Create a preprocessor
preprocessor = ColumnTransformer(
    [('num1', Pipeline([['scaler', StandardScaler()]]), ['Months', 'Year']),
     ('num2', Pipeline([['scaler', StandardScaler()]]), crude_quality_cols),
     ('num3', Pipeline([['scaler', StandardScaler()]]), dist_temp_cols),
     ('cat', OneHotEncoder(drop='first', handle_unknown='ignore'), ['Location']),
     remainder='drop'
    )).fit(X)

# Preprocess the feature matrix
X_transform = preprocessor.transform(X)
```

## Data Cleaning - Conclusions/Discussions/Next Steps:

- The Data Cleaning process successfully refined the dataset by removing unreliable or highly incomplete columns, such as Sediment, Total Acid Number, Salt, and gas composition variables. Datetime-based features were added to capture temporal effects, and missing values were handled through targeted zero-filling for specific synthetic crude components and KNN imputation for the remaining numeric data. All missing values were resolved, and a preprocessing pipeline was created to standardize numeric features and encode categorical ones.
- With a clean and fully prepared dataset, the next step is to begin cluster modeling to explore patterns and groupings among the crude oil samples.

# Clustering Models

The Cluster Modeling section applies three unsupervised learning algorithms—K-Means, Agglomerative Clustering, and Gaussian Mixture Models (GMM)—to identify natural groupings within the crude oil dataset. Each model is tested across cluster sizes ranging from 2 to 6 to evaluate how well the data separates into meaningful clusters. Dimensionality reduction techniques, PCA and t-SNE, are used to visualize the cluster formations and assess their distinctness. Model performance is then compared using quantitative metrics and confusion matrices against the known crude grade labels, followed by a final comparison to supervised learning models to benchmark the clustering accuracy and interpretability.

## Clustering Helper Functions

- The `results_plot` function is a modified version based on an example given by sklearn ([https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_kmeans\\_silhouette\\_analysis.html](https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html))
  - Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (n.d.). K-means clustering and silhouette analysis example. scikit-learn documentation. Retrieved October 16, 2025, from [https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_kmeans\\_silhouette\\_analysis.html](https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html)

```
In [78]: # Function to calculate model performance metrics
def evaluate(X, labels, y_true=None):
    k = len(set(labels)) - (1 if -1 in labels else 0)
    metrics_dict = {
        'n_clusters': k,
        'Silhouette Score': silhouette_score(X, labels),
        'DBI': davies_bouldin_score(X, labels),
        'CHI': calinski_harabasz_score(X, labels),
        'ARI': adjusted_rand_score(y_true, labels) if y_true is not None else np.nan,
        'NMI': normalized_mutual_info_score(y_true, labels) if y_true is not None else np.nan,
        'FMI': fowlkes_mallows_score(y_true, labels) if y_true is not None else np.nan,
    }
    return metrics_dict

# Function to create a silhouette score plot, a PCA scatter plot, and a t-SNE scatter plot
def results_plot(X, labels, model_name, centers=None, random_state=10):
    # Prepare reductions
    pca = PCA(n_components=2, random_state=random_state)
    tsne = TSNE(n_components=2, random_state=random_state, perplexity=30, learning_rate='auto', init='pca')

    X_pca = pca.fit_transform(X)
    X_tsne = tsne.fit_transform(X)

    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16, 5), gridspec_kw={'width_ratios': [1, 1, 1]})

    # Silhouette score plot
    k = len(set(labels)) - (1 if -1 in labels else 0)
    sil_avg = silhouette_score(X, labels)
```

```

sil_vals = silhouette_samples(X, labels)
ax1.set_xlim([-0.6, 1.0])
ax1.set_ylim([0, X.shape[0] + (k + 1) * 10])
y_lower = 10
for i, cl in enumerate(sorted([c for c in set(labels) if c != -1])):
    vals = np.sort(sil_vals[labels == cl])
    y_upper = y_lower + vals.shape[0]
    color = cm.viridis_r(float(i) / max(k, 1))
    ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, vals, facecolor=color, edgecolor=color, alpha=0.7)
    ax1.text(-0.05, y_lower + 0.5 * vals.shape[0], str(cl))
    y_lower = y_upper + 10
ax1.axvline(x=sil_avg, color='red', linestyle='--')
ax1.set_title(f'Silhouette Score = {sil_avg:.3f}', fontdict={'fontsize':10})
ax1.set_xlabel('coef')
ax1.set_yticks([])

# common color map
denom = (labels.max() + 1) if labels.max() >= 0 else 1
colors = cm.viridis_r(labels.astype(float) / denom)

# PCA scatterplot
ax2.scatter(X_pca[:, 0], X_pca[:, 1], c=colors, s=15, lw=0, alpha=0.7, edgecolor='k')
if centers is not None:
    centers_pca = pca.transform(centers)
    ax2.scatter(centers_pca[:, 0], centers_pca[:, 1], marker='o', c='white', s=200, edgecolor='k')
    for i, cpt in enumerate(centers_pca):
        ax2.scatter(cpt[0], cpt[1], marker=f'${i}$', s=50, edgecolor='k')
ax2.set_title(f'PCA projection of clusters into 2 dimensions', fontdict={'fontsize':10})
ax2.set_xlabel('PC1')
ax2.set_ylabel('PC2')

# t-SNE scatterplot
ax3.scatter(X_tsne[:, 0], X_tsne[:, 1], c=colors, s=15, lw=0, alpha=0.7, edgecolor='k')
ax3.set_title(f't-SNE projection of clusters into 2 dimensions', fontdict={'fontsize':10})
ax3.set_xlabel('tSNE-1')
ax3.set_ylabel('tSNE-2')

plt.suptitle(f'{model_name}', horizontalalignment='left', x=0)
plt.tight_layout()
plt.show()

```

## K-means Clustering

- K-Means partitions data into a predefined number of clusters by minimizing the variance within each cluster and assigning points to the nearest cluster centroid through iterative optimization.

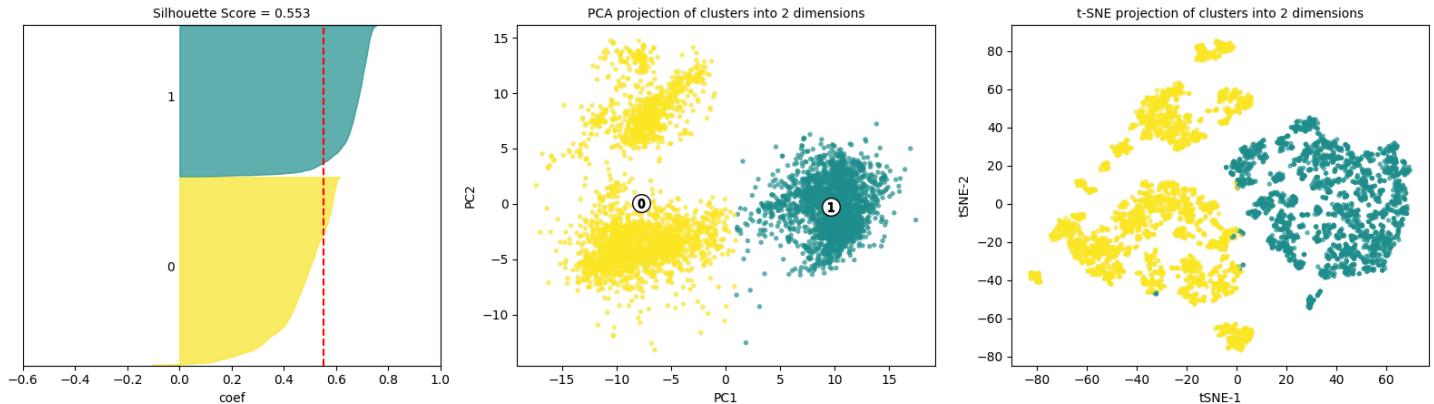
```
In [79]: # # Create empty dicts to store evaluation metrics for each model
results = []
confusion_data = {}
```

```
# Loop over various cluster sizes, fit model, and return evaluation plots
for k in [2, 3, 4, 5, 6]:
    # Fit the model
    model_name = f'K-means Clustering for {k} Clusters'
    kmeans_model = MiniBatchKMeans(n_clusters=k, batch_size=256*14, random_state=10)
    labels = kmeans_model.fit_predict(X_transform)

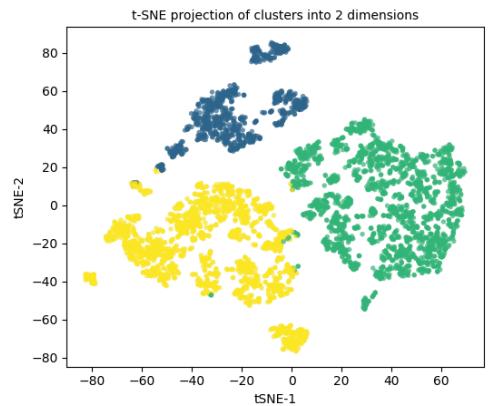
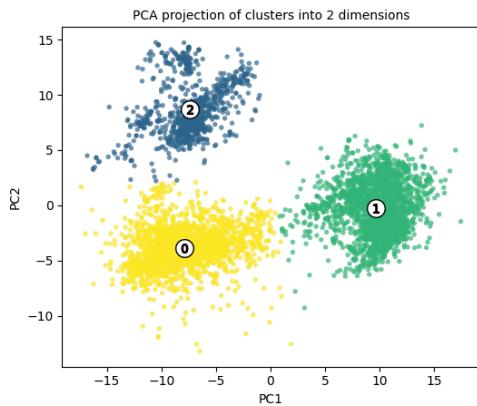
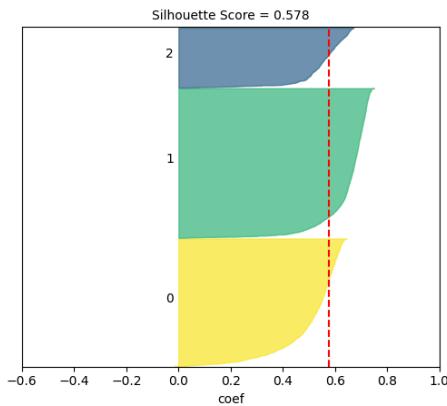
    # Plot the results
    centers = kmeans_model.cluster_centers_
    results_plot(X_transform, labels, model_name, centers=centers)

    # Add the model performance metrics to the results list
    clustering_metrics = evaluate(X_transform, labels, y_true)
    clustering_metrics.update({Model: model_name})
    results.append(clustering_metrics)
```

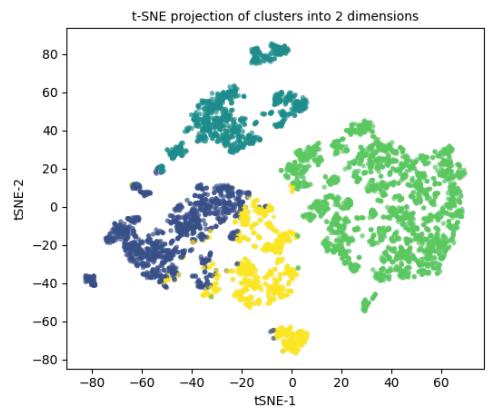
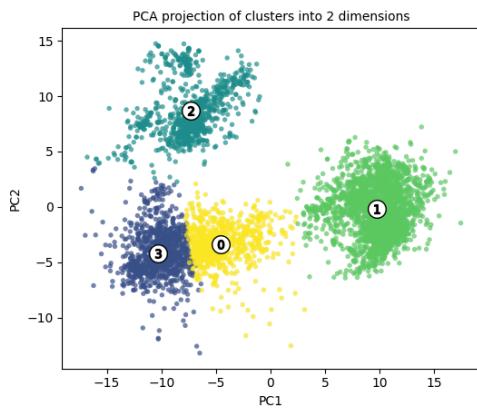
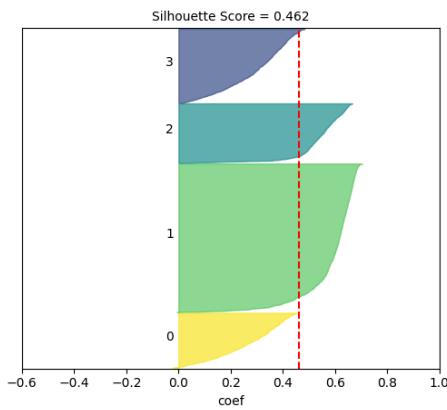
### Kmeans Clustering for 2 Clusters



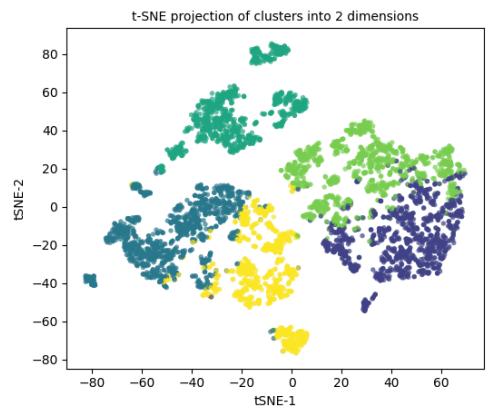
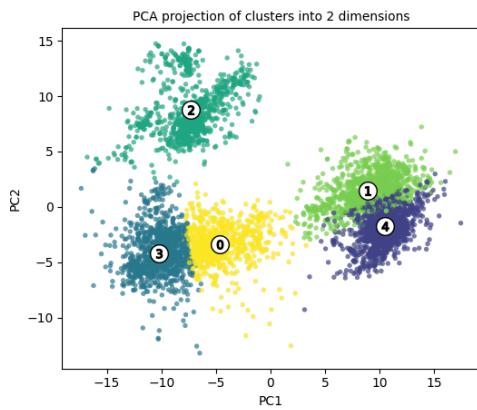
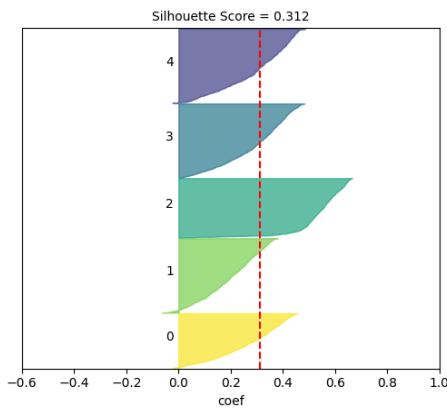
#### K-means Clustering for 3 Clusters



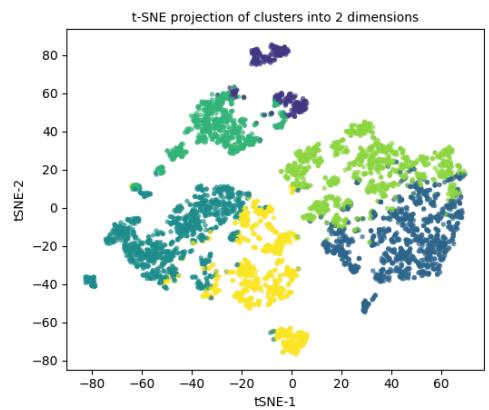
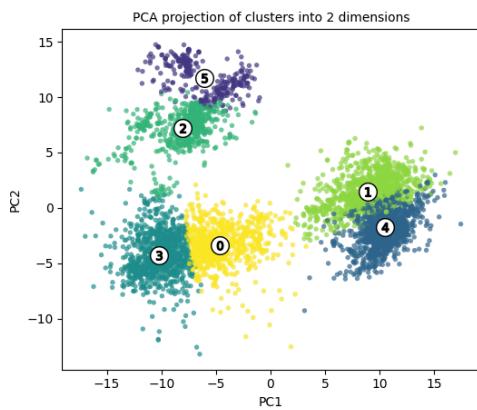
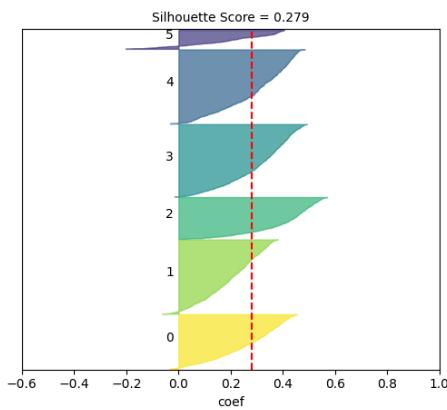
#### K-means Clustering for 4 Clusters



#### K-means Clustering for 5 Clusters



#### K-means Clustering for 6 Clusters



Across cluster sizes 2–6, K-Means shows that the dataset begins with two broad, well-separated groups (silhouette = 0.553) and reaches its strongest structure with three clusters (silhouette = 0.578). This number of clusters aligns with my suspicion that the light-medium crude oil groups may overlap. As cluster count increases beyond three, the

silhouette scores decline and boundaries blur, suggesting over-segmentation without meaningful separation. Overall, 3–4 clusters appear most representative of the underlying structure, consistent with the known four target classes and the suspected similarity between the light-medium crude categories.

## Agglomerative Clustering

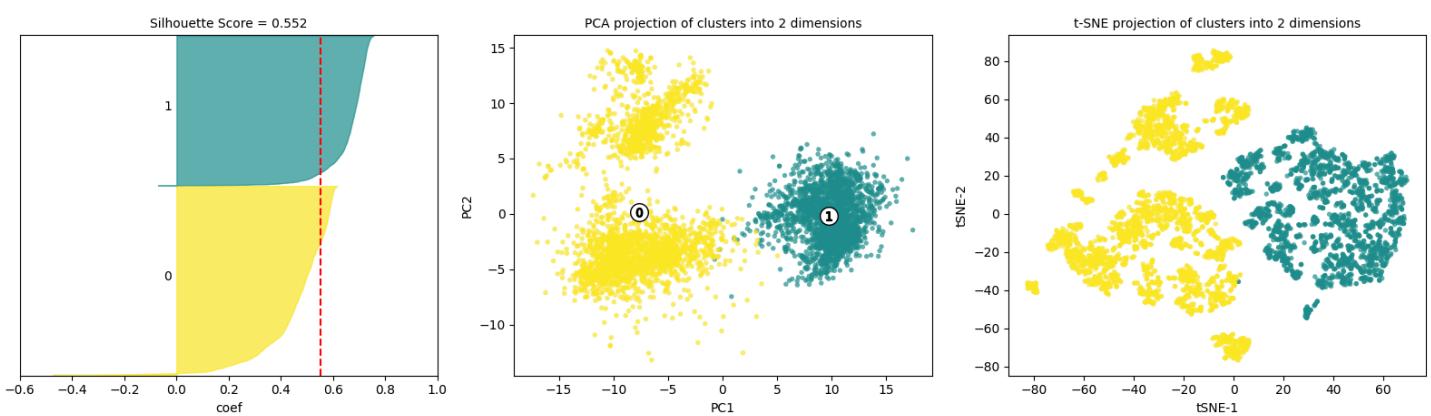
- Agglomerative Clustering is a hierarchical approach that starts with each data point as its own cluster and successively merges the most similar pairs based on linkage criteria until the desired number of clusters is reached.

```
In [80]: for k in [2, 3, 4, 5, 6]:
    # Fit the model
    model_name = f'Agglomerative Clustering for {k} Clusters'
    agglo_model = AgglomerativeClustering(n_clusters=k, linkage='ward')
    labels = agglo_model.fit_predict(X_transform)

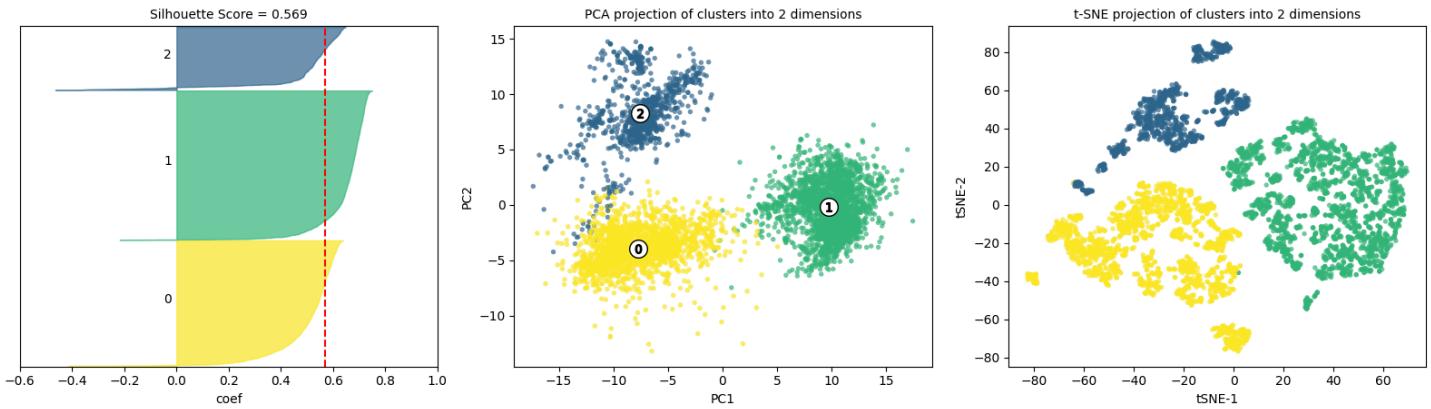
    # Plot the results
    cl_ids = sorted(set(labels))
    centers = np.vstack([X_transform[labels == c].mean(axis=0) for c in cl_ids if np.any(labels == c)])
    results_plot(X_transform, labels, model_name, centers=centers)

    # Add the model performance metrics to the results list
    clustering_metrics = evaluate(X_transform, labels, y_true)
    clustering_metrics.update({'Model': model_name})
    results.append(clustering_metrics)
```

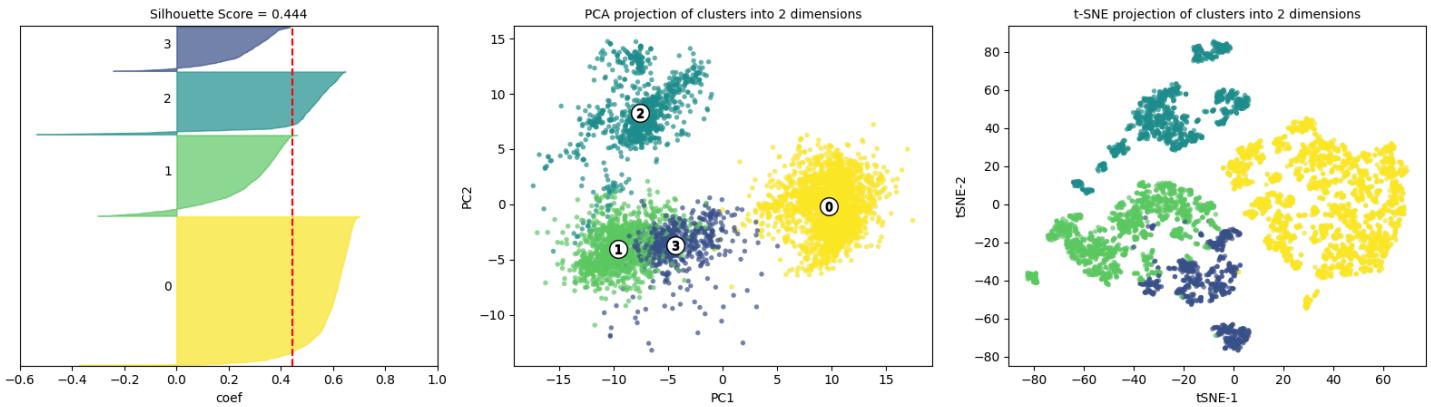
Agglomerative Clustering for 2 Clusters



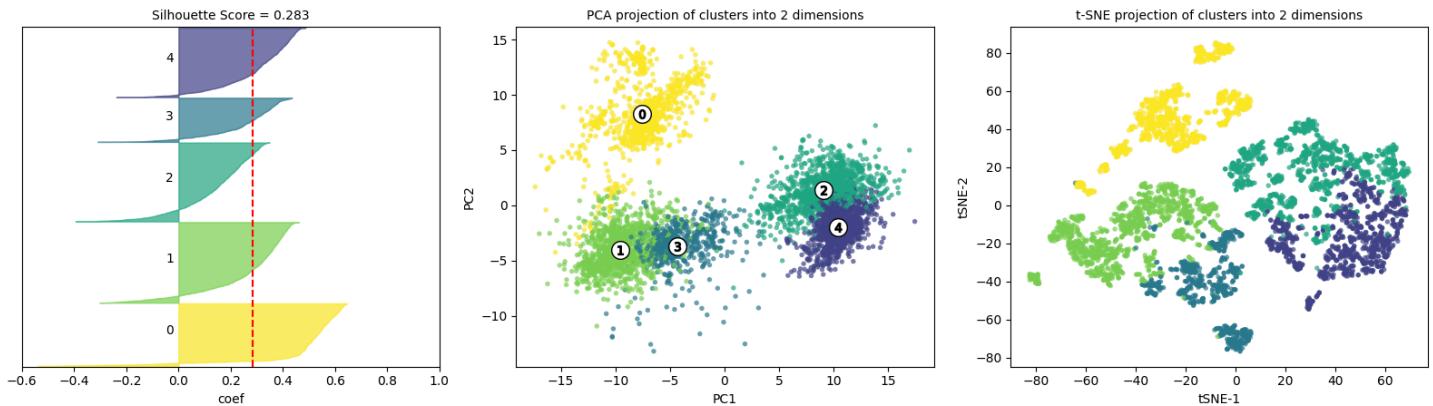
Agglomerative Clustering for 3 Clusters



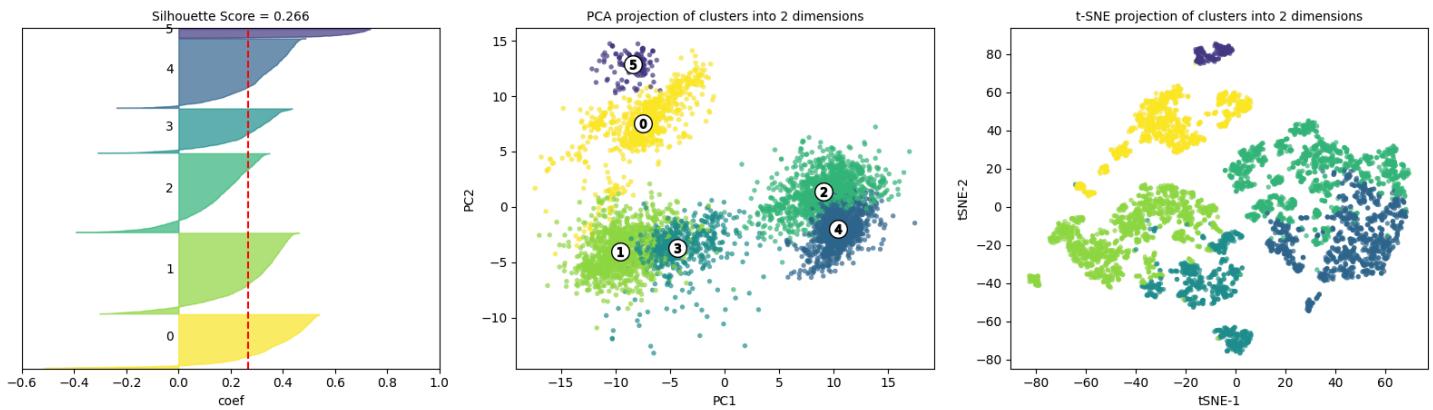
Agglomerative Clustering for 4 Clusters



### Agglomerative Clustering for 5 Clusters



### Agglomerative Clustering for 6 Clusters



Across cluster sizes 2–6, Agglomerative Clustering produces patterns similar to K-Means, with clear separation emerging at 3 clusters (silhouette = 0.569). The 2-cluster result shows two large, well-defined groups, while the 3-cluster structure best captures distinctions among heavy, synthetic, and combined light/medium sour/sweet crudes—supporting the hypothesis that light and medium sour/sweet grades are closely related. As clusters increase beyond 3, the structure fragments and silhouette scores decline, suggesting that 3–4 clusters most accurately represent the underlying relationships among the crude oil samples.

## Gaussian Mixture Model Clustering

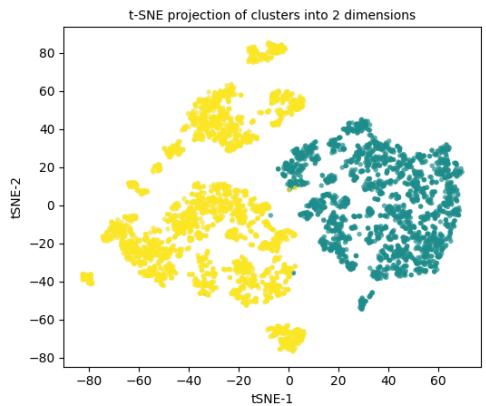
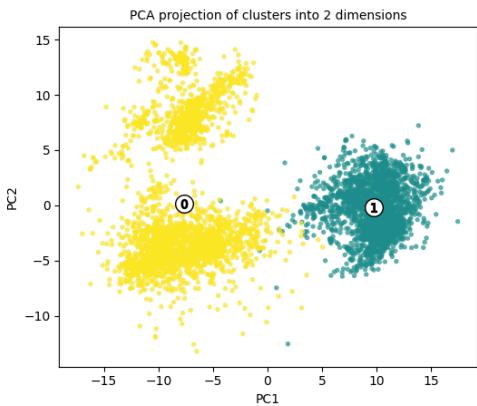
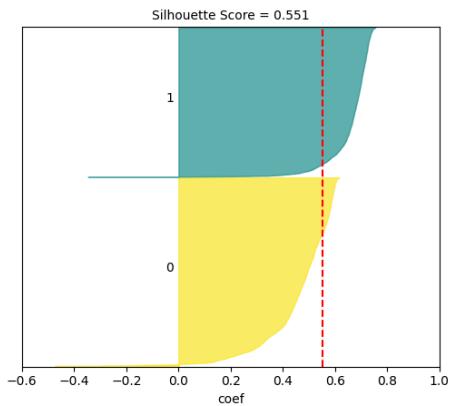
- Gaussian Mixture Models (GMM) assume that the data is generated from a mixture of Gaussian distributions, using probabilistic assignment of points to clusters to capture overlapping or non-spherical group structures more flexibly than K-Means.

```
In [81]: for k in [2, 3, 4, 5, 6]:
    # Fit the model
    model_name = f'Gaussian Mixture Model Clustering for {k} Clusters'
    gmm_model = GaussianMixture(n_components=k, covariance_type='full', random_state=10)
    gmm_model.fit(X_transform)
    labels = gmm_model.predict(X_transform)

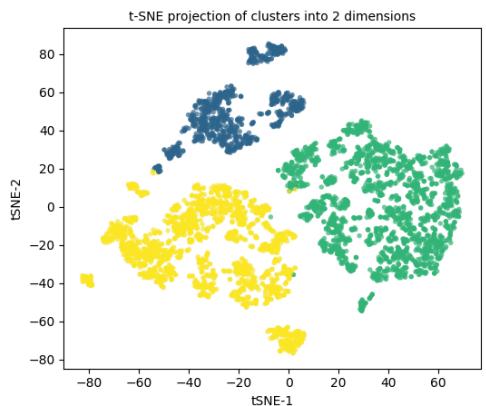
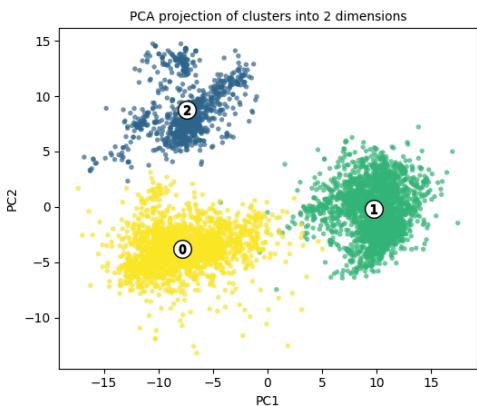
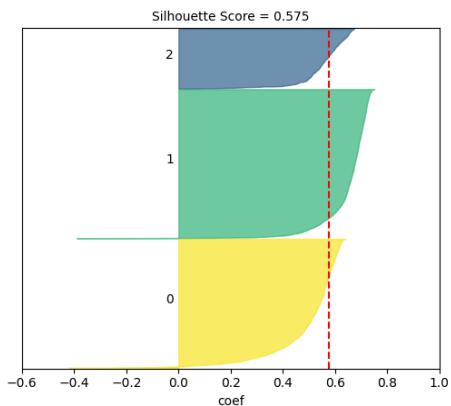
    # Plot the results
    centers = gmm_model.means_
    results_plot(X_transform, labels, model_name, centers=centers)

    # Add the model performance metrics to the results list
    clustering_metrics = evaluate(X_transform, labels, y_true)
    clustering_metrics.update({'Model': model_name})
    results.append(clustering_metrics)
```

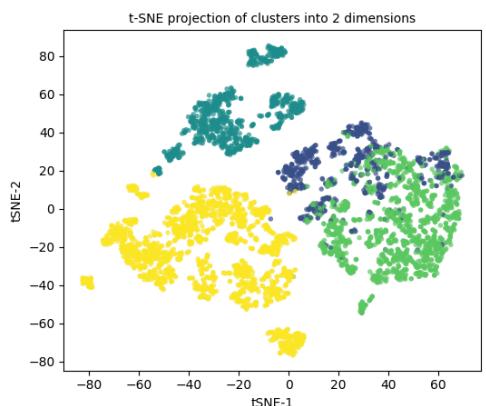
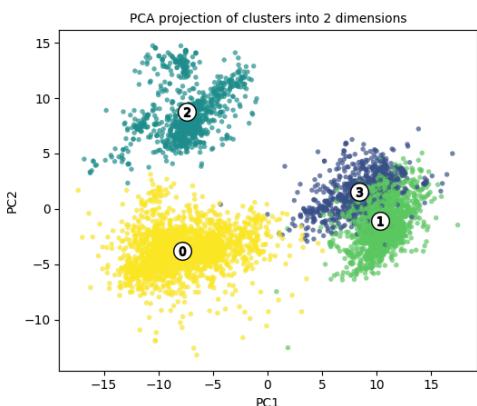
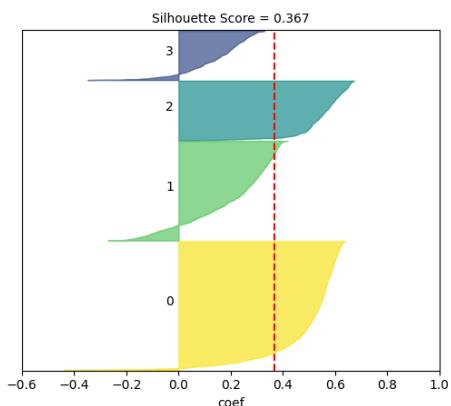
#### Gaussian Mixture Model Clustering for 2 Clusters



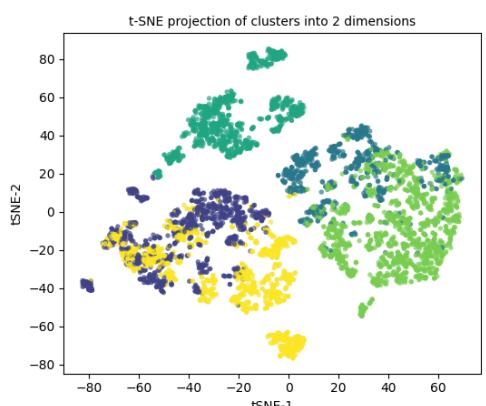
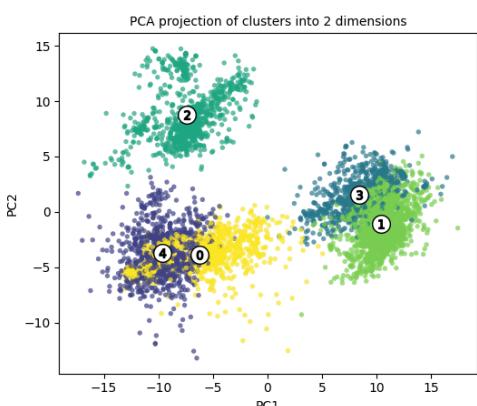
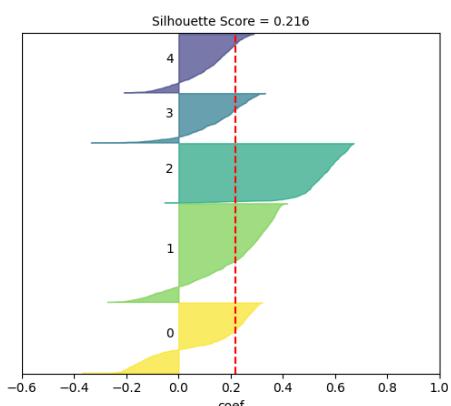
#### Gaussian Mixture Model Clustering for 3 Clusters



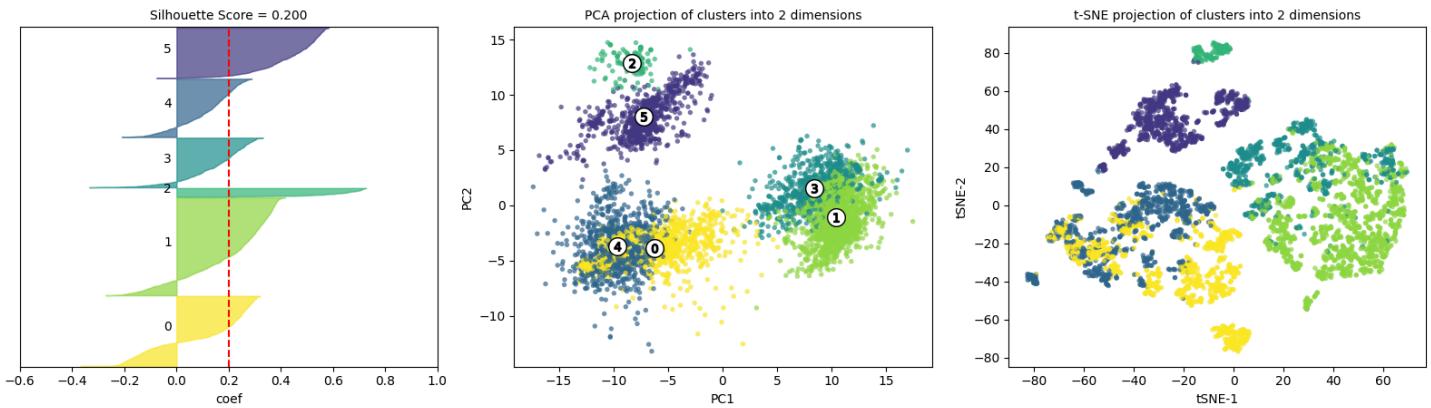
#### Gaussian Mixture Model Clustering for 4 Clusters



#### Gaussian Mixture Model Clustering for 5 Clusters



## Gaussian Mixture Model Clustering for 6 Clusters



The Gaussian Mixture Model results closely mirror K-Means and Agglomerative outcomes, with the most coherent separation appearing at 3 clusters (silhouette = 0.575). At 2 clusters, the model forms two dominant groups, while at 3 clusters it captures distinct heavy, synthetic, and combined light/medium crude categories—supporting the expected overlap between the sour and sweet light-medium grades. Beyond 3 clusters, performance declines as the algorithm overfits and splits natural groups into smaller, less meaningful segments. Overall, the GMM effectively models the probabilistic overlap among crudes, with 3–4 clusters best reflecting the underlying structure of the dataset.

## Performance Evaluation

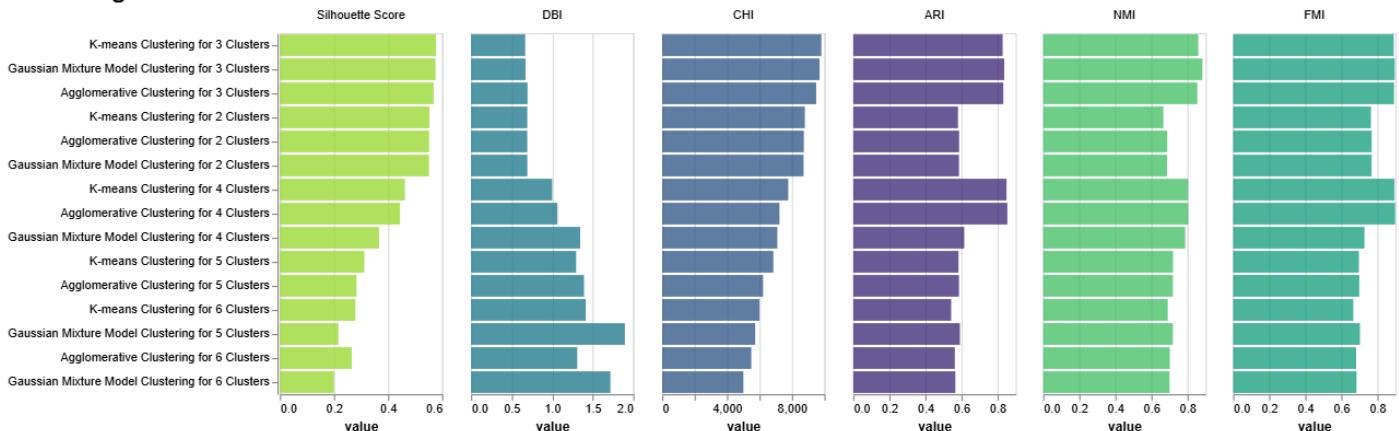
Clustering performance is assessed using **internal** and **external** validation metrics.

- **Internal Metrics** - evaluate the quality of the clusters based solely on the structure of the data, without using any true labels:
  - **Silhouette Score**: Measures how similar each sample is to its own cluster compared to other clusters (higher = better).
  - **Davies–Bouldin Index (DBI)**: Average similarity ratio of each cluster to the most similar one (lower = better).
  - **Calinski–Harabasz Index (CHI)**: Ratio of between-cluster dispersion to within-cluster dispersion (higher = better).
- **External Metrics** - compare the clustering results to known ground truth labels to measure how well the clusters align with the actual classes:
  - **Adjusted Rand Index (ARI)**: Quantifies agreement between predicted clusters and true labels, adjusted for chance.
  - **Normalized Mutual Information (NMI)**: Measures the amount of shared information between cluster assignments and true labels.
  - **Fowlkes–Mallows Index (FMI)**: Geometric mean of precision and recall between predicted clusters and true labels.

```
In [84]: # Create a dataframe from the model results list
results_df = pd.DataFrame(results)[['Model', 'n_clusters', 'Silhouette Score', 'DBI', 'CHI', 'ARI', 'NMI', 'FMI']]

# Plot performance of models
alt.Chart(results_df, title=alt.Title('Clustering Models Performance Metrics', font_size=20)).transform_fold(
    ['Silhouette Score', 'DBI', 'CHI', 'ARI', 'NMI', 'FMI']
).mark_bar(opacity=0.8).encode(
    alt.X('value:Q'),
    alt.Y('Model:N').title(None).sort('-x').axis(labelLimit=500),
    alt.Facet('key:N').sort(['Silhouette Score', 'DBI', 'CHI', 'ARI', 'NMI', 'FMI']).title(None),
    alt.Color('key:N').legend(None).scale(scheme='viridis')
).resolve_axis(
    x='independent',
    y='shared'
).resolve_scale(
    x='independent',
    y='shared'
).properties(width=135)
```

Out[84]: Clustering Models Performance Metrics



Across all clustering models, the **best overall performance occurs at 3 clusters**, where the **Gaussian Mixture Model** and **Agglomerative Clustering** both achieve the highest Silhouette, ARI, NMI, and FMI scores while maintaining low DBI values. K-Means performs comparably but shows slightly less separation between clusters. As the number of

clusters increases beyond three, all models show reduced cohesion and weaker external validation scores. Based on the combined internal and external metrics, **the Gaussian Mixture Model with 3 clusters** provides the best balance of cluster compactness, separation, and alignment with the true crude oil groupings.

## Classification Confusion Matrices

- A set of confusion matrices are used to compare the clustering model performance when compared to actual crude grade labels for 4 clusters

```
In [ ]: def fit_and_align(model, X, y_true):
    # Fit and predict
    if isinstance(model, GaussianMixture):
        model.fit(X)
        z = model.predict(X)
    else:
        z = model.fit_predict(X)

    # Create contingency matrix
    C = contingency_matrix(y_int, z)
    r_ind, c_ind = linear_sum_assignment(C.max() - C)
    mapping = {c: r for r, c in zip(r_ind, c_ind)}
    z_map = np.array([mapping.get(k, k) for k in z])

    # Confusion table
    cm = confusion_matrix(y_int, z_map, labels=np.arange(len(class_names)))
    df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)
    return df_cm

# Encode target labels
le = LabelEncoder()
y_int = le.fit_transform(np.asarray(y_true))
class_names = le.classes_

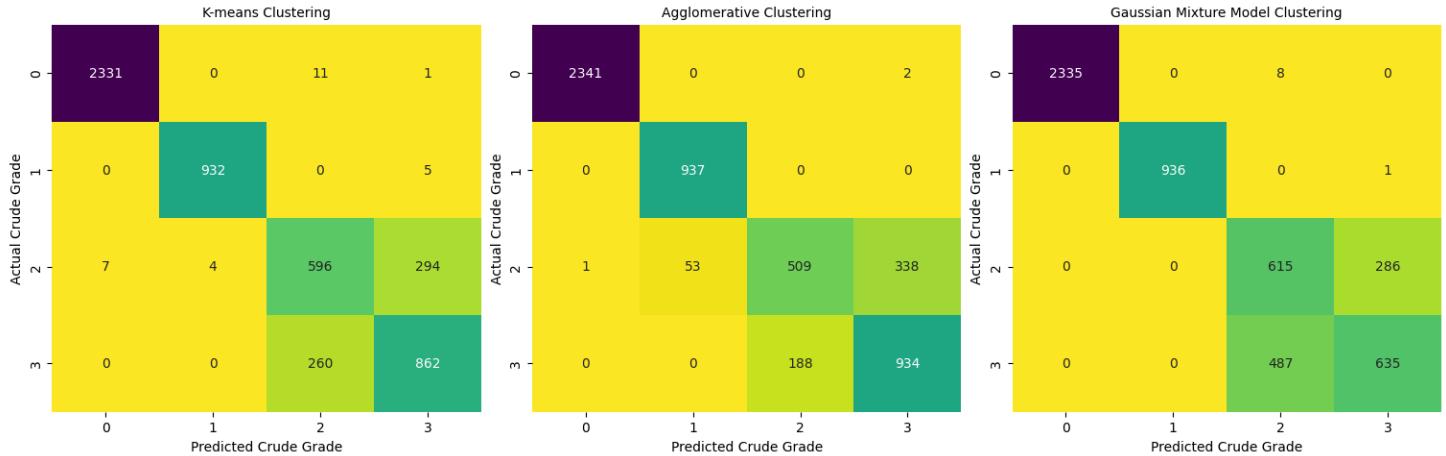
# Run model for k=4
models_4 = {
    'K-means Clustering': MiniBatchKMeans(n_clusters=4, batch_size=256*14, random_state=10),
    'Agglomerative Clustering': AgglomerativeClustering(n_clusters=4, linkage='ward'),
    'Gaussian Mixture Model Clustering': GaussianMixture(n_components=4, n_init=10, covariance_type='full', random_state=10),
}

# Build heatmaps
fig, axes = plt.subplots(1, 3, figsize=(15, 5), constrained_layout=True)

for ax, (name, mdl) in zip(axes, models_4.items()):
    cm_df = fit_and_align(mdl, X_transform, y_int)
    sns.heatmap(cm_df, annot=True, fmt='d', cmap='viridis_r', cbar=False, ax=ax)
    ax.set_title(f'{name}', fontdict={'fontsize':10})
    ax.set_xlabel('Predicted Crude Grade')
    ax.set_ylabel('Actual Crude Grade')

plt.suptitle('Confusion Matrices of Actual vs Predicted Target Labels (for k=4 clusters)', horizontalalignment='left', x=0)
plt.show()
```

Confusion Matrices of Actual vs Predicted Target Labels (for k=4 clusters)



All three models—K-Means, Agglomerative, and Gaussian Mixture—accurately identify the Heavy Sour and Sweet Synthetic crude groups but consistently confuse the Light & Medium Sweet and Light & Medium Sour categories. This overlap appears in each confusion matrix as significant cross-prediction between the 2nd and 3rd targets, indicating that these two groups share similar chemical and distillation characteristics. The consistent misclassification across models reinforces the earlier observation that the light and medium sour/sweet grades are closely related and may represent a single broader class rather than two distinct ones.

## Compare to Supervised Classification Models

```
In [37]: # Define supervised model types
supervised_models = [
    'Multinomial Logistic Classifier': LogisticRegressionCV(solver='lbfgs', penalty='l2', cv=5, max_iter=5000, n_jobs=-1, random_state=10),
    'Support Vector Classifier (with RBF kernel)': SVC(kernel='rbf', gamma='scale', C=1.0, random_state=10),
    'Gradient Boosting Classifier': HistGradientBoostingClassifier(random_state=10)
```

```

}

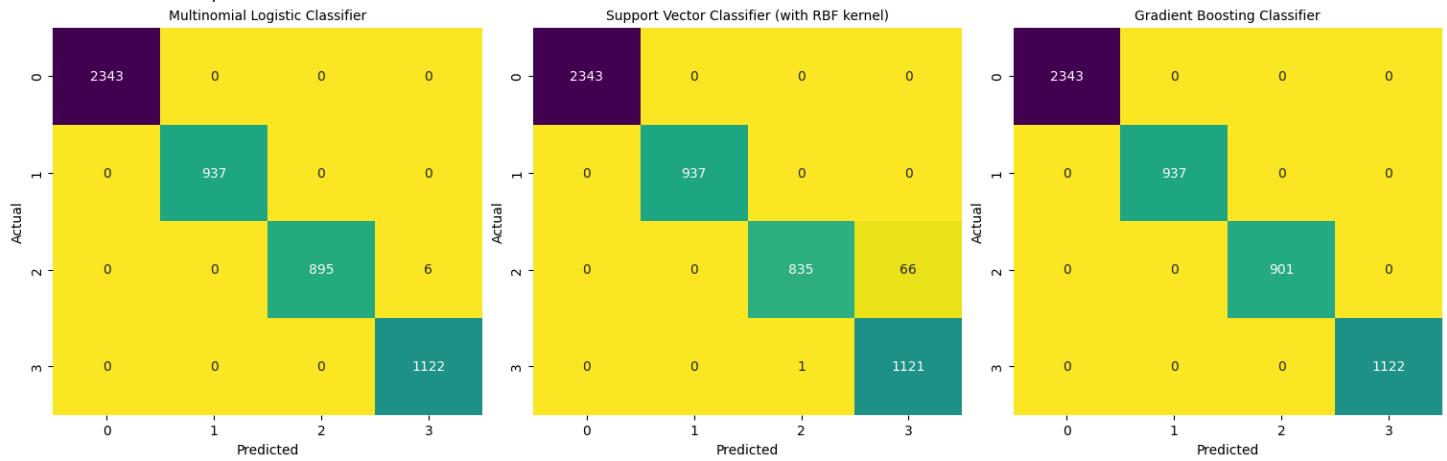
# Fit and predict models
le = LabelEncoder()
y_int = le.fit_transform(np.asarray(y_true))
class_names = le.classes_
confusion_dict = {}
for model_name, supervised_model in supervised_models.items():
    supervised_model.fit(X_transform, y_true)
    y_pred = supervised_model.predict(X_transform)
    confusion_mat = confusion_matrix(y_true, y_pred, labels=np.arange(len(class_names)))
    confusion_dict[model_name] = pd.DataFrame(confusion_mat, index=class_names, columns=class_names)

# Plot confusion matrices
fig, axes = plt.subplots(1, 3, figsize=(15, 5), constrained_layout=True)
for ax, (model_name, confusion_df) in zip(axes, confusion_dict.items()):
    sns.heatmap(confusion_df, annot=True, fmt='d', cmap='viridis_r', cbar=False, ax=ax)
    ax.set_title(model_name, fontsize=10)
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')

plt.suptitle('Confusion Matrices for Supervised Classification Models', x=0, ha='left')
plt.show()

```

Confusion Matrices for Supervised Classification Models



All three supervised models—**Multinomial Logistic Regression**, **Support Vector Classifier (RBF kernel)**, and **Gradient Boosting Classifier**—achieve near-perfect classification, cleanly separating all four crude groups with minimal or no misclassification. This contrasts sharply with the unsupervised models, which consistently confused the Light & Medium Sweet and Light & Medium Sour categories. The key difference is that supervised models are trained with explicit label guidance, allowing them to learn subtle, high-dimensional boundaries between similar classes that unsupervised models cannot infer on their own. However, this also highlights an advantage of unsupervised learning: **its ability to reveal natural similarities**—in this case, the close relationship between the two light/medium crude groups—which may indicate that the labeled classes are less distinct in practice than defined.

## Modeling - Conclusions/Discussions/Next Steps:

- The modeling results confirmed the initial hypothesis that the dataset naturally forms three to four meaningful clusters, supporting the idea that the Light & Medium Sour crude groups are closely related and may belong to a single broader category.
- Among the clustering methods, **the Gaussian Mixture Model with three clusters** achieved the strongest overall performance across both internal and external validation metrics. In contrast, the supervised models accurately classified all four target classes, demonstrating that labeled data can separate even highly similar crude types.
- The next step in the analysis will be to present the final discussion and conclusions, summarizing key insights, comparing supervised and unsupervised performance, and outlining the broader implications of these findings for crude quality classification.

## Discussion & Conclusion

### Learning and Takeaways

This project demonstrated how **unsupervised learning techniques can uncover meaningful structure** in high-dimensional industrial datasets, even without labeled outcomes. By integrating clustering, dimensionality reduction, and supervised benchmarking, the analysis showed that Canadian crude oil grades can be grouped effectively by chemical and physical properties. The results confirmed that the four labeled crude categories largely align with the natural data patterns, though the **Light & Medium Sweet** and **Light & Medium Sour** groups exhibit strong overlap, suggesting their compositional boundaries are more fluid than categorical. The project also reinforced the importance of robust preprocessing—handling missing data, scaling, and feature selection—before model training, as these steps significantly influence model interpretability and performance.

### Why Something Didn't Work

Unsupervised models **struggled to distinguish** between the **Light & Medium Sweet** and **Light & Medium Sour** groups, which repeatedly appeared blended in the clustering outputs. This limitation stems from the inherent similarity of their feature distributions, where overlapping density, API gravity, and distillation temperature characteristics reduced cluster separability. The models, lacking label information, were unable to infer the subtle chemical differences that supervised methods later captured. Additionally, incomplete or uneven data coverage across certain features—such as Sediment, Total Acid Number, and gas composition columns—likely weakened the clustering accuracy by introducing noise and bias into the feature space.

## Suggestions for Improvement

Future work could enhance model precision by incorporating **domain-informed feature engineering**, such as grouping variables into process-relevant ratios (e.g., sulfur-to-density or distillation slope metrics). Expanding the dataset with additional time periods or refinery assay data could also improve model generalization and balance across crude grades. A valuable next step would be to **apply unsupervised learning across all crude oil groups defined by CrudeMonitor.ca**, to explore whether broader clustering patterns emerge beyond the four-group framework used here. Additionally, since **synthetic crudes share many properties with light oils**, it would be interesting to investigate this relationship further and test whether unsupervised learning can naturally distinguish them as separate clusters. Applying **semi-supervised learning** or **latent variable models** may also bridge the gap between unsupervised discovery and supervised precision, while interactive dashboards could help visualize these relationships and translate technical results into actionable refinery insights.

---