Source code (part-1) of the paper:

A data − driven approach for identification of coal related lithofacies using singl
A case study from Sohagpur coal field, India

Rupam Roy[1], *Dip Kumar Singha*[1*], *Sayan Ghosh*[2], *Laraib Abbas*[1],
Tarit Narjary[1], *Debjeet Mandal*[2]

This notebook contains part 1 of the workflow for developing machine-learning–based single and ensemble classifiers to predict coal, carbonaceous shale, and non-coal facies from high-resolution well-log data. This notebook primarily presents the data variability analysis as discussed in the paper, which includes the following elements:

1. **Library Imports**:

Loading all required Python libraries and auxiliary functions.

2. **Data Loading**:

Importing training and blind-testing datasets as pandas DataFrames.

3. **Data Description**:

Dataset overview, variability analyses, and outlier detection.

The workflow relies extensively on built-in objects and utilities from scikit-learn, and several plotting routines are adapted from the works of Brendon Hall and Ryan A. Mardani. The raw source files (Excel) used for training and testing are confidential and therefore omitted; their directory paths are replaced with underscores. For illustration, the DataFrame corresponding to one representative training well is provided to demonstrate the structure and naming of input and output columns. To execute the notebook successfully, users must replace the placeholder underscores with appropriate file paths pointing to datasets that share identical variable names for all required input and output features.

Importing all the required libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib as mpl
import matplotlib.colors as colors
from mpl_toolkits.axes_grid1 import make_axes_locatable
import matplotlib.ticker as ticker
from matplotlib.ticker import MultipleLocator, FormatStrFormatter
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize
from itertools import cycle
from imblearn.under_sampling import RandomUnderSampler
import warnings
```

Undersampling: Randomly remove samples from the majority class to balance the class distribution. However, this may lead to loss of important information.

Class Weighting: Most classifiers allow for class weights to be assigned during training. By giving higher weights to minority classes, you can help the classifier focus more on learning them.

Importing training data as a pandas dataframe

In [2]:
```
df_111A = pd.read_excel(r'_.xlsx')
```

In [3]:
```
df_113 = pd.read_excel(r'_.xlsx')
```

In [4]:
```
df_115 = pd.read_excel(r'_.xlsx')
```

In [5]:
```
df_115.describe()
```

Out[5]:

	DEPTH	NGAM	CALP	LSD	HRD	S
count	42590.000000	42590.000000	42590.000000	42590.000000	42590.000000	42590.000
mean	267.549579	94.493711	79.717743	2.703229	2.574046	69.310
std	132.739508	43.077360	3.723658	0.344778	0.414833	46.519
min	45.100000	21.970000	77.244000	1.348000	1.079000	0.200
25%	151.572500	63.293000	78.904000	2.472000	2.228000	33.636
50%	258.045000	87.007000	79.736000	2.779000	2.728000	64.127
75%	387.517500	115.428000	79.839000	2.958000	2.888000	95.764
max	493.990000	372.263000	171.309000	3.768000	3.412000	437.109

Plotting the well log responses of each of the training wells

In [6]:
```
df_111A = df_111A.sort_values(by='DEPTH')
df_113 = df_113.sort_values(by='DEPTH')
df_115 = df_115.sort_values(by='DEPTH')
```

mapping = { 'CARBSHALE': 0, 'HIGH CARBSHALE': 0, 'LOW CARBSHALE': 0, 'COAL': 1, 'INTERCALATION OF SANDSTONE': 2, 'SANDSTONE': 2, 'SANDY SHALE': 3, 'INTERCALATION OF SHALE': 3, 'SHALE': 4, 'SHALY COAL': 5 }

In [7]:
```
facies_colors = ['#2E86C1', '#000000', '#FFD700', '#c1b32e', '#800000', '#9400D3
facies_labels = ['CS','COAL','STN','SNYSH','SH','SHY_COAL']
```

FUNCTION 1 : CALP, NGAM, SPR, LONG_SHORT AVG, LSD_HRD AVG

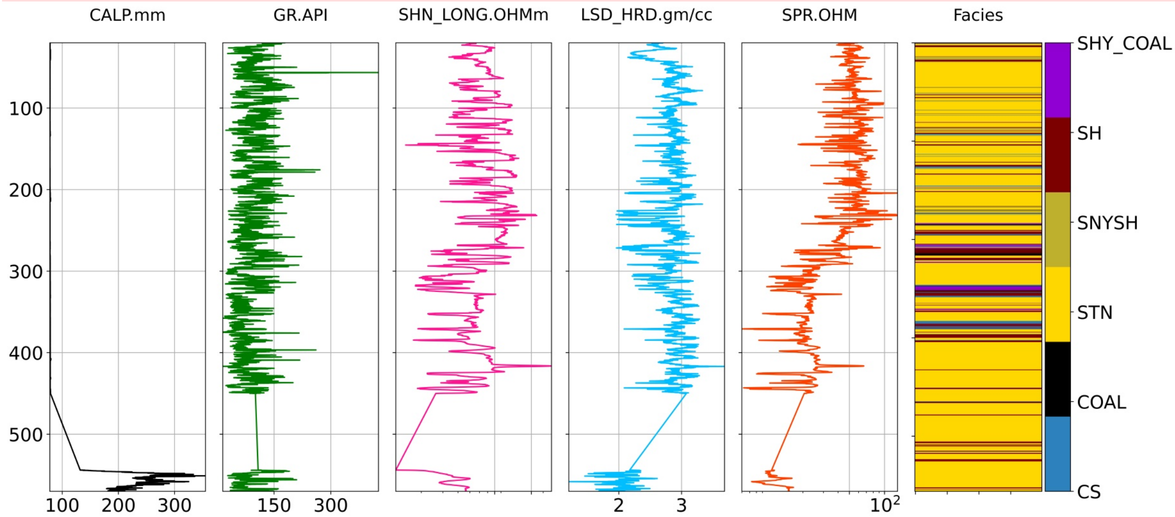
In [8]:
```
def make_facies_log_plot(logs, facies_colors):
    import matplotlib.pyplot as plt
    from matplotlib import colors
    from mpl_toolkits.axes_grid1 import make_axes_locatable
    import numpy as np

    logs = logs.sort_values(by='DEPTH')
    cmap_facies = colors.ListedColormap(facies_colors[0:len(facies_colors)], 'in

    ztop = logs.DEPTH.min()
    zbot = logs.DEPTH.max()

    cluster = np.repeat(np.expand_dims(logs['Encoded_Formation'].values, 1), 100

    f, ax = plt.subplots(nrows=1, ncols=6, figsize=(18, 8), dpi=900)  # Set high

    ax[0].plot(logs.CALP, logs.DEPTH, '-', color='black')
    ax[1].plot(logs.NGAM, logs.DEPTH, '-g')
    ax[2].semilogx(logs.SHN_LONG_AVG, logs.DEPTH, color='deeppink')
    ax[3].plot(logs.LSD_HRD_AVG, logs.DEPTH, color='deepskyblue')
    ax[4].semilogx(logs.SPR, logs.DEPTH, color='orangered')

    im = ax[5].imshow(cluster, interpolation='none', aspect='auto',
                      cmap=cmap_facies, vmin=0, vmax=5)

    divider = make_axes_locatable(ax[5])
    cax = divider.append_axes("right", size="20%", pad=0.05)
    cbar = plt.colorbar(im, cax=cax)
    cbar.set_ticklabels(facies_labels)
    cbar.ax.tick_params(labelsize=20)  # Set colorbar tick fontsize

    for i in range(len(ax)-1):
        ax[i].set_ylim(ztop, zbot)
        ax[i].invert_yaxis()
        ax[i].grid(True)
        ax[i].tick_params(labelsize=20)  # Set tick label font size
        ax[i].xaxis.set_major_locator(plt.MaxNLocator(nbins=3))

    # Axis labels with larger font
    ax[0].set_xlabel("CALP.mm\n", fontsize=18)
    ax[0].set_xlim(logs.CALP.min(), logs.CALP.max())
    ax[0].xaxis.set_label_position('top')

    ax[1].set_xlabel("GR.API\n", fontsize=18)
    ax[1].set_xlim(logs.NGAM.min(), logs.NGAM.max())
    ax[1].xaxis.set_label_position('top')

    ax[2].set_xlabel("SHN_LONG.OHMm\n", fontsize=18)
    ax[2].set_xlim(logs.SHN_LONG_AVG.min(), logs.SHN_LONG_AVG.max())
    ax[2].xaxis.set_label_position('top')

    ax[3].set_xlabel("LSD_HRD.gm/cc\n", fontsize=18)
    ax[3].set_xlim(logs.LSD_HRD_AVG.min(), logs.LSD_HRD_AVG.max())
    ax[3].xaxis.set_label_position('top')
```

```
    ax[4].set_xlabel("SPR.OHM\n", fontsize=18)
    ax[4].set_xlim(logs.SPR.min(), logs.SPR.max())
    ax[4].xaxis.set_label_position('top')

    ax[5].set_xlabel('Facies\n', fontsize=18)
    ax[5].xaxis.set_label_position('top')

    ax[1].set_yticklabels([]); ax[2].set_yticklabels([]); ax[3].set_yticklabels(
    ax[4].set_yticklabels([]); ax[5].set_yticklabels([]); ax[5].set_xticklabels(

    plt.tight_layout()
    plt.savefig("facies_log_plot_high_res.png", dpi=900, bbox_inches='tight')  #
```

Well-1

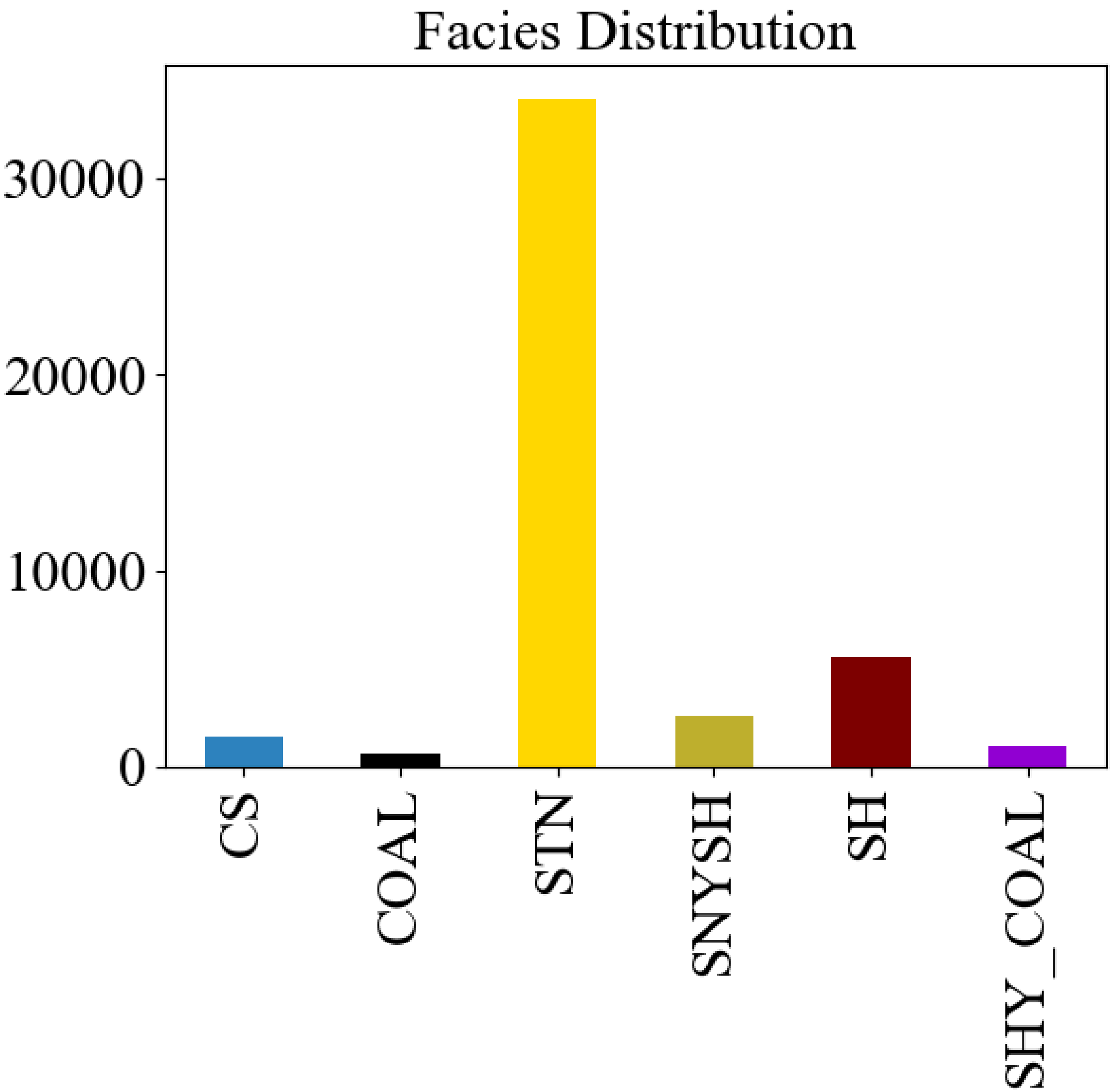
In [9]:
```
make_facies_log_plot(df_111A, facies_colors)
```

```
C:\Users\reser\AppData\Local\Temp\ipykernel_93368\930242987.py:29: UserWarning: s
et_ticklabels() should only be used with a fixed number of ticks, i.e. after set_
ticks() or using a FixedLocator.
  cbar.set_ticklabels(facies_labels)
```



In [10]:
```
#count the number of unique entries for each facies, sort them by
#facies number (instead of by number of entries)
facies_counts = df_111A['Encoded_Formation'].value_counts().sort_index()
#use facies labels to index each count
facies_counts.index = facies_labels

# Increasing fontsize and set font style
plt.rcParams.update({'font.size': 22, 'font.family': 'Times New Roman'})

# Ploting the bar chart
ax = facies_counts.plot(kind='bar', color=facies_colors,
                        title='Facies Distribution')

# Set title fontsize and font style
ax.title.set_fontsize(22)
ax.title.set_fontname('Times New Roman')


# Set tick labels fontsize and font style
ax.tick_params(axis='both', which='major', labelsize=22)
```

```
# Set tick labels font style
for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontname('Times New Roman')

# Show the plot
plt.show()

facies_counts
```

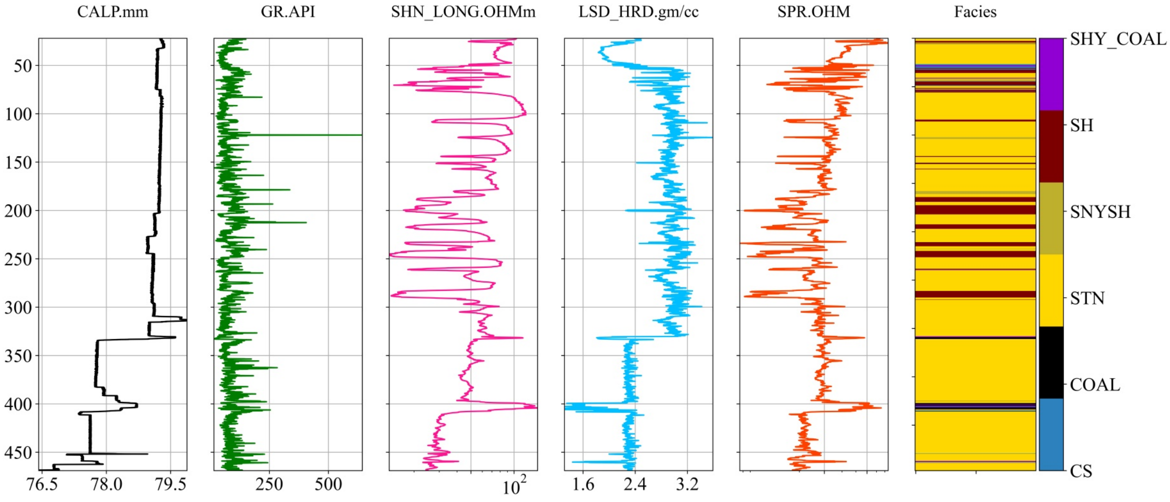


Out[10]:
```
CS           1583
COAL          660
STN         34069
SNYSH        2638
SH           5596
SHY_COAL     1056
Name: count, dtype: int64
```

2. Well-2

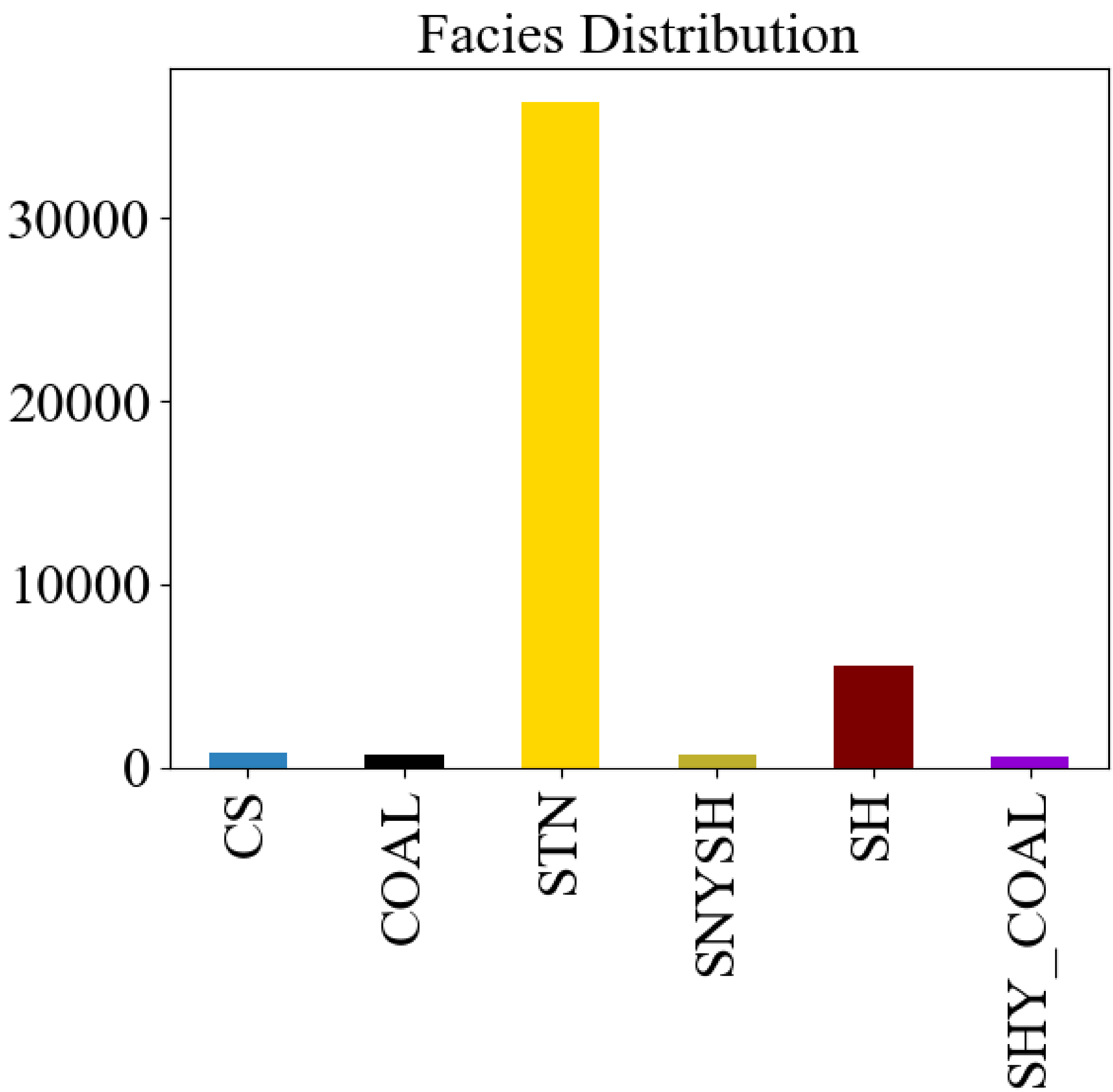
In [11]:
```
make_facies_log_plot(df_113, facies_colors)
```

```
C:\Users\reser\AppData\Local\Temp\ipykernel_93368\930242987.py:29: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
  cbar.set_ticklabels(facies_labels)
```



In [12]:

```
#count the number of unique entries for each facies, sort them by
#facies number (instead of by number of entries)
facies_counts = df_113['Encoded_Formation'].value_counts().sort_index()
#use facies labels to index each count
facies_counts.index = facies_labels

# Increasing fontsize and set font style
plt.rcParams.update({'font.size': 22, 'font.family': 'Times New Roman'})

# Ploting the bar chart
ax = facies_counts.plot(kind='bar', color=facies_colors,
                        title='Facies Distribution')

# Set title fontsize and font style
ax.title.set_fontsize(22)
ax.title.set_fontname('Times New Roman')

# Set tick labels fontsize and font style
ax.tick_params(axis='both', which='major', labelsize=22)

# Set tick labels font style
for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontname('Times New Roman')

# Show the plot
plt.show()

facies_counts
```

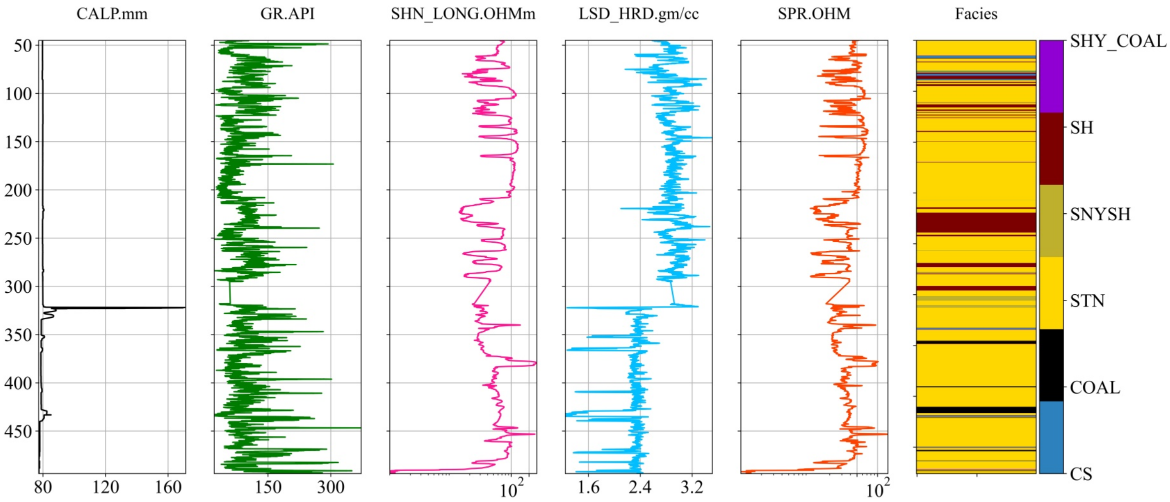


Out[12]:
```
CS           794
COAL         746
STN        36307
SNYSH        732
SH          5539
SHY_COAL     583
Name: count, dtype: int64
```

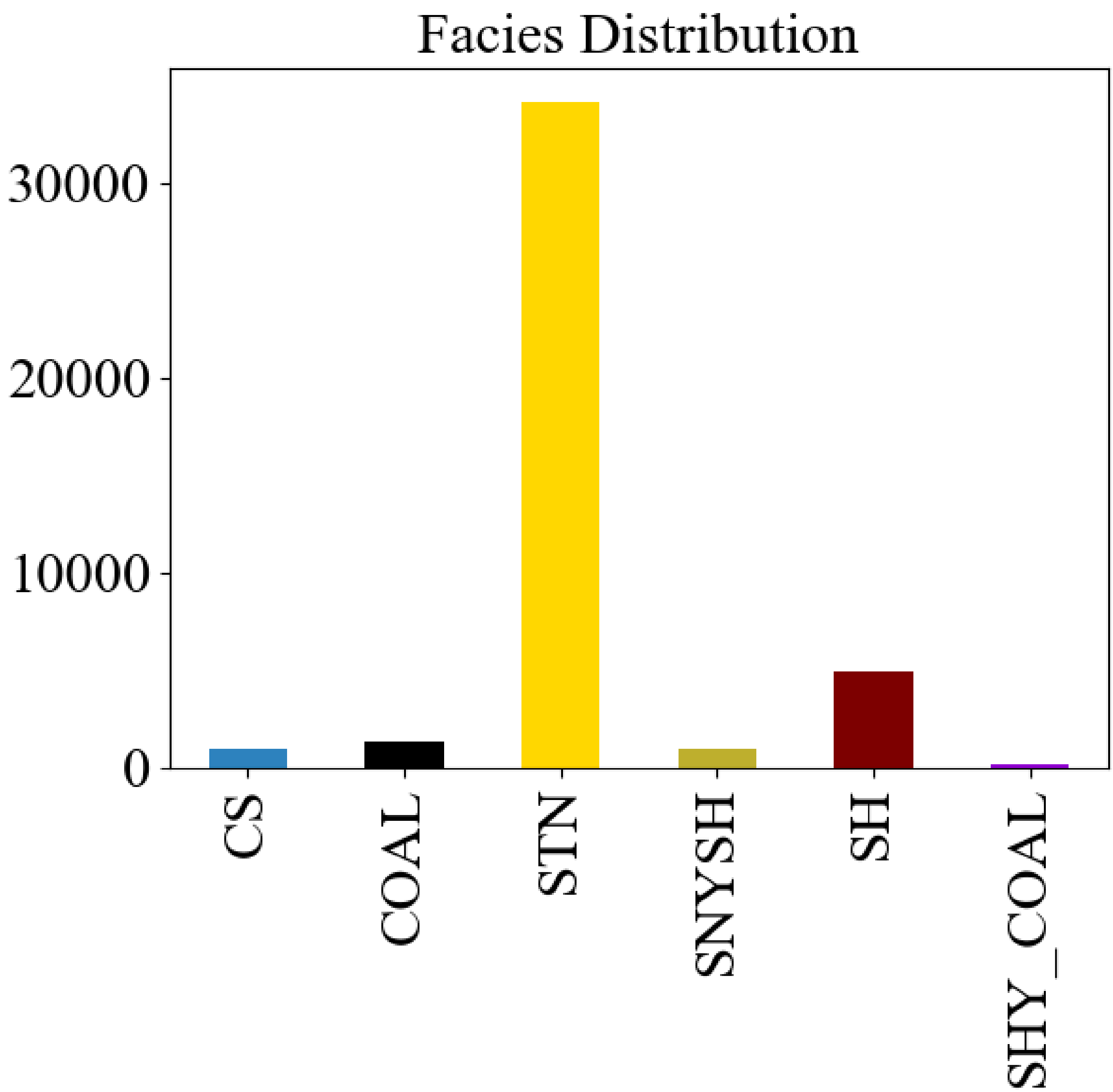
3. Well-3

In [13]:
```
make_facies_log_plot(df_115, facies_colors)
#plt.savefig("SHRIMPLIN_X1", dpi=400)
```

```
C:\Users\reser\AppData\Local\Temp\ipykernel_93368\930242987.py:29: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
  cbar.set_ticklabels(facies_labels)
```



In [14]:
```
#count the number of unique entries for each facies, sort them by
#facies number (instead of by number of entries)
facies_counts = df_115['Encoded_Formation'].value_counts().sort_index()
#use facies labels to index each count
facies_counts.index = facies_labels

# Increasing fontsize and set font style
plt.rcParams.update({'font.size': 22, 'font.family': 'Times New Roman'})

# Ploting the bar chart
ax = facies_counts.plot(kind='bar', color=facies_colors,
                        title='Facies Distribution')

# Set title fontsize and font style
ax.title.set_fontsize(22)
ax.title.set_fontname('Times New Roman')

# Set tick labels fontsize and font style
ax.tick_params(axis='both', which='major', labelsize=22)

# Set tick labels font style
for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontname('Times New Roman')

# Show the plot
plt.show()

facies_counts
```



Out[14]:
```
CS            950
COAL         1306
STN         34177
SNYSH        1003
SH           4936
SHY_COAL      218
Name: count, dtype: int64
```

The overall training data

In [15]:
```
Training_df = pd.read_excel(r'D:\6. PHD_copy\Coal work\Training wells\Main files
```

In [16]:
```
Training_df
```

Out[16]:

	DEPTH	NGAM	CALP	SPR	SHN_LONG_AVG	LSD_HRD_AVG	Encoded_For
0	20.00	178.024	78.223	58.400	50.2635	2.6490	
1	20.01	176.280	78.201	58.000	50.1455	2.6570	
2	20.02	172.095	78.219	57.600	50.0270	2.6425	
3	20.03	166.864	78.219	56.527	49.9090	2.6385	
4	20.04	164.598	78.214	55.455	49.7905	2.6455	
...	...	...	...	...	...	...	
132888	569.96	130.772	209.926	15.200	46.2730	1.9905	
132889	569.97	132.515	209.317	15.200	46.2180	1.9900	
132890	569.98	130.938	208.741	15.200	46.1635	1.9855	
132891	569.99	128.613	208.309	15.200	46.1090	1.9835	
132892	570.00	125.126	208.184	15.200	46.0910	1.9790	

132893 rows × 7 columns

Importing Blind Test well data (Well-4)

In [17]:
```
Blind_Testing_df = pd.read_excel(r'D:\6. PHD_copy\Coal work\Training wells\Main
```

In [18]:
```
Blind_Testing_df
```

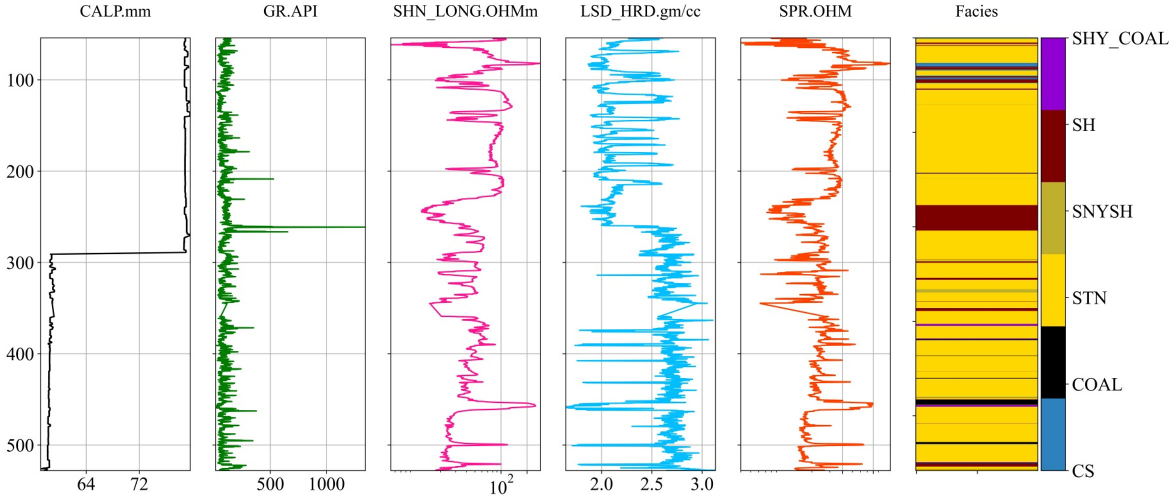
Out[18]:

	DEPTH	NGAM	CALP	SPR	SHN_LONG_AVG	LSD_HRD_AVG	Encoded_Form
0	54.00	107.924	78.913	9.582	107.6545	2.5965	
1	54.01	104.797	78.904	9.618	107.6545	2.6000	
2	54.02	108.281	78.913	9.655	107.6545	2.6105	
3	54.03	112.458	78.913	9.691	107.6545	2.6065	
4	54.04	108.982	78.926	9.727	107.0730	2.6085	
...	...	...	...	...	...	...	
45798	527.96	80.207	57.758	15.655	18.3455	2.7605	
45799	527.97	78.114	57.758	15.564	18.2635	2.7355	
45800	527.98	80.207	57.758	15.527	18.1635	2.7140	
45801	527.99	82.299	57.736	15.491	18.1640	2.6800	
45802	528.00	87.530	57.736	15.455	18.1455	2.6490	

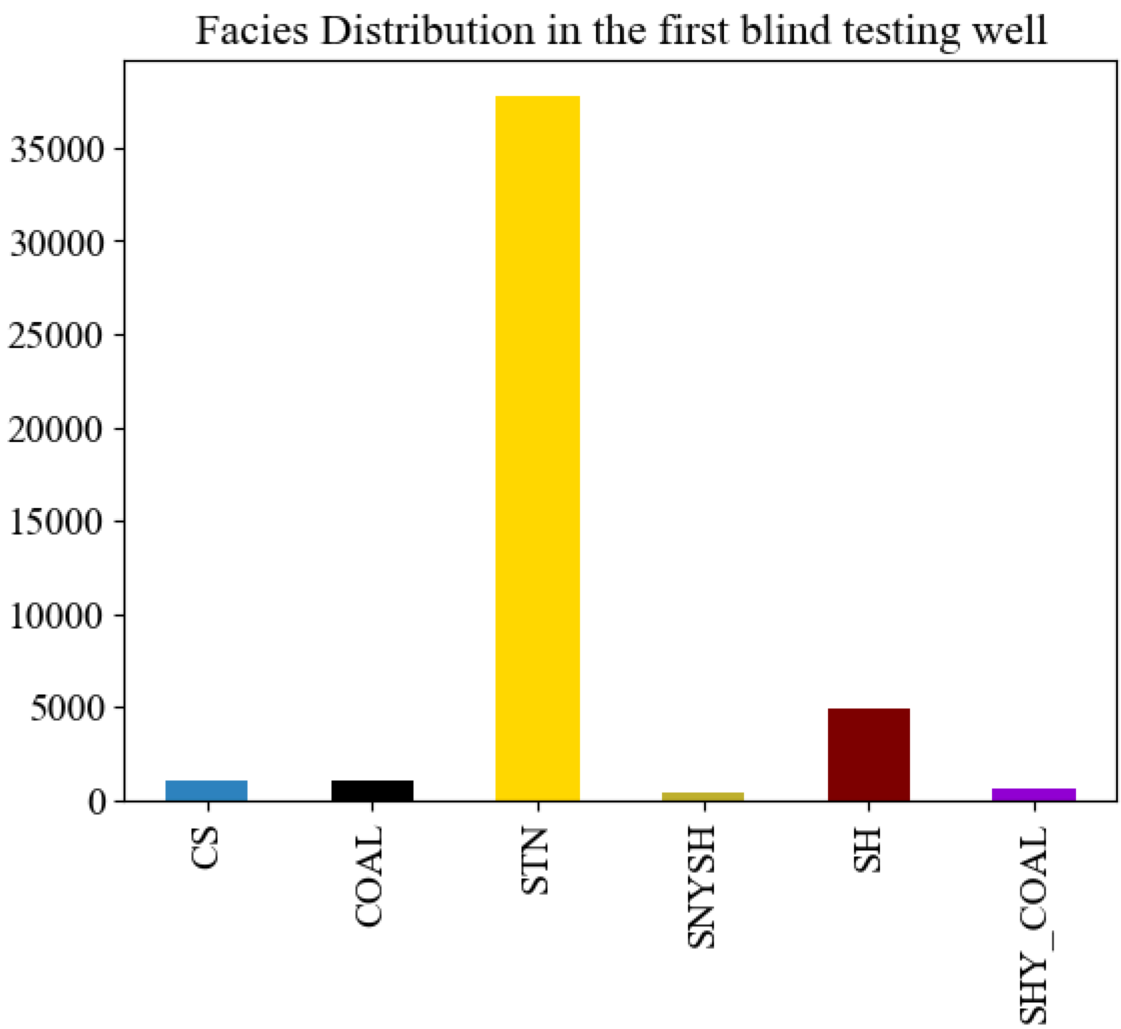
45803 rows × 7 columns

In [19]:
```
make_facies_log_plot(Blind_Testing_df, facies_colors)
```

```
C:\Users\reser\AppData\Local\Temp\ipykernel_93368\930242987.py:29: UserWarning: s
et_ticklabels() should only be used with a fixed number of ticks, i.e. after set_
ticks() or using a FixedLocator.
  cbar.set_ticklabels(facies_labels)
```



In [20]:
```
#count the number of unique entries for each facies, sort them by
#facies number (instead of by number of entries)
facies_counts = Blind_Testing_df['Encoded_Formation'].value_counts().sort_index(
#use facies labels to index each count
facies_counts.index = facies_labels

# Increasing fontsize and set font style
plt.rcParams.update({'font.size': 12, 'font.family': 'Times New Roman'})

# Ploting the bar chart
ax = facies_counts.plot(kind='bar', color=facies_colors,
                        title='Facies Distribution in the first blind testing we

# Set title fontsize and font style
ax.title.set_fontsize(16)
ax.title.set_fontname('Times New Roman')

# Set tick labels fontsize and font style
ax.tick_params(axis='both', which='major', labelsize=14)

# Set tick labels font style
for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontname('Times New Roman')

# Show the plot
plt.show()

facies_counts
```



Out[20]:
```
CS           1045
COAL         1064
STN         37811
SNYSH         380
SH           4903
SHY_COAL      600
Name: count, dtype: int64
```

In [21]:
```
Blind_X_test = Blind_Testing_df.iloc[:, 1:-1]
Blind_X_test
```

Out[21]:

	NGAM	CALP	SPR	SHN_LONG_AVG	LSD_HRD_AVG
0	107.924	78.913	9.582	107.6545	2.5965
1	104.797	78.904	9.618	107.6545	2.6000
2	108.281	78.913	9.655	107.6545	2.6105
3	112.458	78.913	9.691	107.6545	2.6065
4	108.982	78.926	9.727	107.0730	2.6085
...	...	...	...	...	...
45798	80.207	57.758	15.655	18.3455	2.7605
45799	78.114	57.758	15.564	18.2635	2.7355
45800	80.207	57.758	15.527	18.1635	2.7140
45801	82.299	57.736	15.491	18.1640	2.6800
45802	87.530	57.736	15.455	18.1455	2.6490

45803 rows × 5 columns

Defining a color map for the facies

'CARBSHALE': 0, 'COAL': 1, 'SANDSTONE': 2, 'SANDY SHALE',: 3, 'SHALE': 4, 'SHALY COAL': 5

In [22]:

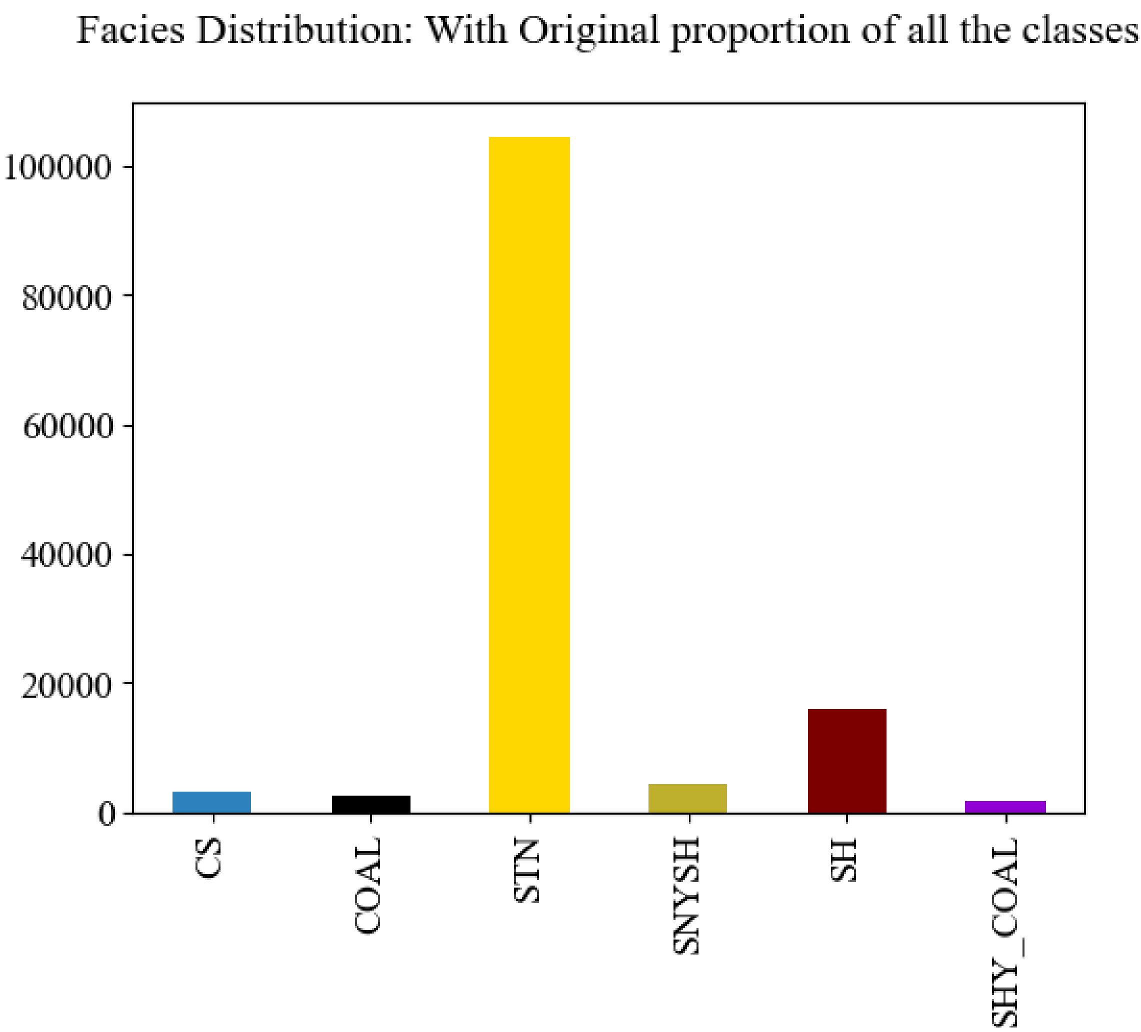
```
facies_colors = ['#2E86C1', '#000000', '#FFD700', '#c1b32e', '#800000', '#9400D3
facies_labels = ['CS','COAL','STN','SNYSH','SH','SHY_COAL']

# 1. cmap_facies
cmap_facies = colors.ListedColormap(facies_colors[0:len(facies_colors)], 'indexe
```

In addition to individual wells, we can look at how the various facies are represented by the entire training set. Let's plot a histogram of the number of training examples for each facies class.

In [23]:

```
#count the number of unique entries for each facies, sort them by
#facies number (instead of by number of entries)
facies_counts = Training_df['Encoded_Formation'].value_counts().sort_index()
#use facies labels to index each count
facies_counts.index = facies_labels

# Increasing fontsize and set font style
plt.rcParams.update({'font.size': 12, 'font.family': 'Times New Roman'})

# Ploting the bar chart
ax = facies_counts.plot(kind='bar', color=facies_colors,
                        title='Facies Distribution: With Original proportion of

# Set title fontsize and font style
ax.title.set_fontsize(16)
ax.title.set_fontname('Times New Roman')
```

```
# Set tick labels fontsize and font style
ax.tick_params(axis='both', which='major', labelsize=14)

# Set tick labels font style
for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontname('Times New Roman')

# Show the plot
plt.show()

facies_counts
```



Out[23]:
```
CS            3327
COAL          2712
STN         104553
SNYSH         4373
SH           16071
SHY_COAL      1857
Name: count, dtype: int64
```

In [24]:
```
# np.bincount(Training_df["Encoded_Formation"] == 1)
```

In [25]:
```
# 132893/(6 * np.bincount(Training_df["Encoded_Formation"] == 1))
```

Outlier detection

In [26]:
```python
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest
from scipy.stats import zscore

# Assuming your DataFrame is named 'Training_df'
numeric_cols = ['DEPTH', 'NGAM', 'CALP', 'SPR', 'SHN_LONG_AVG', 'LSD_HRD_AVG']

# 1️⃣ Z-Score Method
z_scores = np.abs(zscore(Training_df[numeric_cols]))
Training_df['Z_Outlier'] = (z_scores > 3).any(axis=1)  # Flag if any feature is

# 2️⃣ IQR Method
Q1 = Training_df[numeric_cols].quantile(0.25)
Q3 = Training_df[numeric_cols].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

is_iqr_outlier = ((Training_df[numeric_cols] < lower_bound) | (Training_df[numer
Training_df['IQR_Outlier'] = is_iqr_outlier.any(axis=1)  # Any column outside bo

# 3️⃣ Isolation Forest
iso_forest = IsolationForest(contamination=0.01, random_state=42)
Training_df['IF_Outlier'] = iso_forest.fit_predict(Training_df[numeric_cols])
Training_df['IF_Outlier'] = Training_df['IF_Outlier'] == -1  # Convert to Boolea

# ✅ Summary: How many outliers detected by each
print("Z-score outliers:", Training_df['Z_Outlier'].sum())
print("IQR outliers:", Training_df['IQR_Outlier'].sum())
print("IsolationForest outliers:", Training_df['IF_Outlier'].sum())
```

```
Z-score outliers: 7307
IQR outliers: 9171
IsolationForest outliers: 1328
```

In [27]:
```python
# Show rows where at least one method flags an outlier
outliers = Training_df[Training_df[['Z_Outlier', 'IQR_Outlier', 'IF_Outlier']].a
print(outliers.head())
```

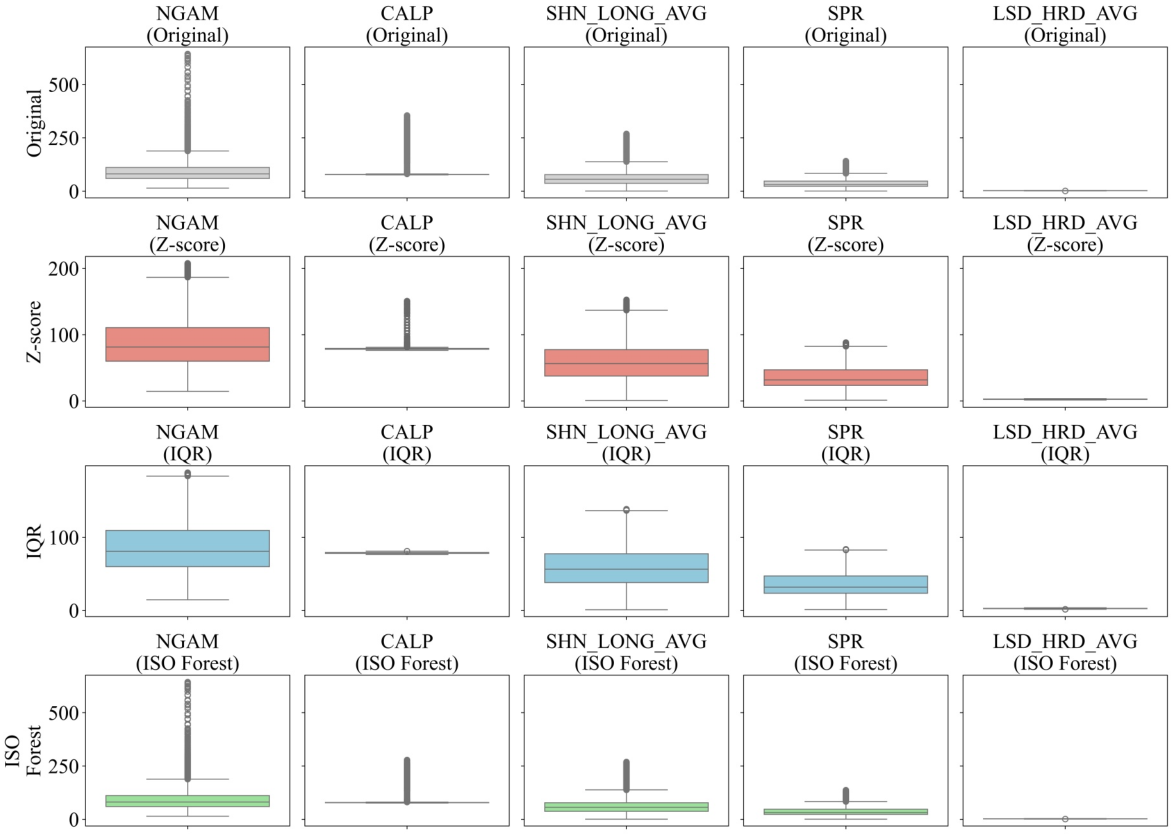
```
      DEPTH     NGAM    CALP     SPR  SHN_LONG_AVG  LSD_HRD_AVG  \
1085  26.42   94.330  79.318  83.891       90.5730       2.3730
1087  26.43   90.145  79.322  84.145       90.6725       2.3650
1089  26.44   94.853  79.318  84.400       90.7730       2.3675
1091  26.45   99.735  79.318  84.418       91.1450       2.3795
1093  26.46  102.525  79.318  84.436       91.3455       2.3780

      Encoded_Formation  Z_Outlier  IQR_Outlier  IF_Outlier
1085                  2      False         True       False
1087                  2      False         True       False
1089                  2      False         True       False
1091                  2      False         True       False
1093                  1      False         True       False
```

In [28]:
```python
# ---------------------------------------------------------------
# ❶  Detect outliers and build per-method “clean” DataFrames
# ---------------------------------------------------------------
import numpy as np
import pandas as pd
from scipy.stats import zscore
```

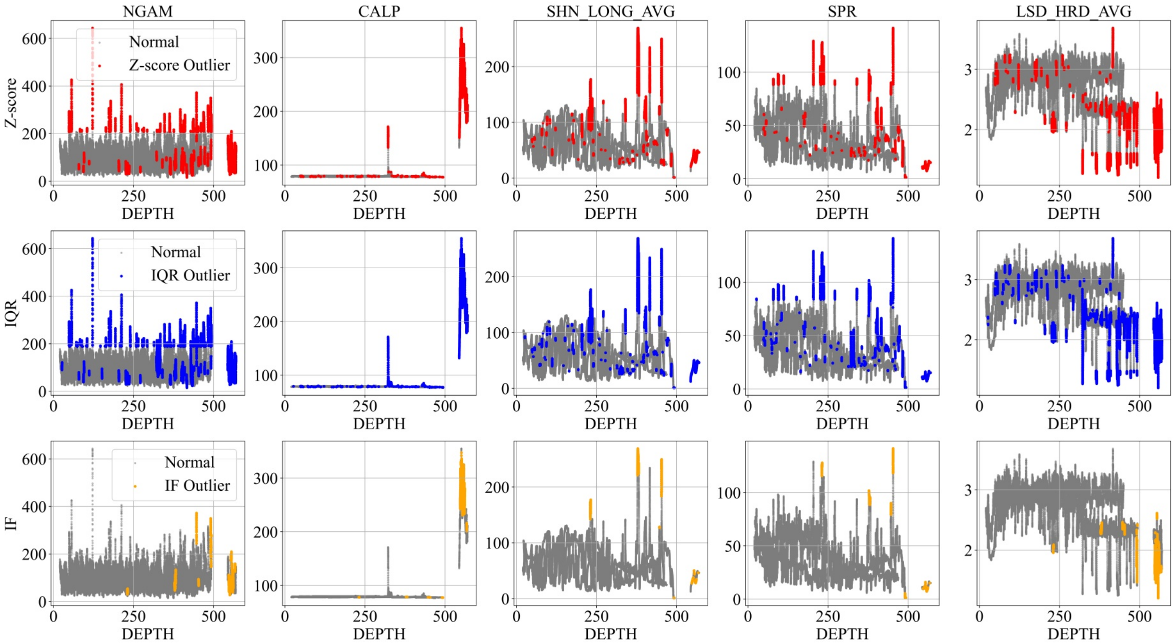
```
from sklearn.ensemble import IsolationForest

# -----------------------
# Configuration
# -----------------------
numeric_cols = ['NGAM', 'CALP', 'SHN_LONG_AVG', 'SPR', 'LSD_HRD_AVG']
z_thresh      = 3          # |z| > z_thresh  ⇒  outlier
iqr_multiplier = 1.5       # Tukey rule (Q1 – 1.5·IQR,  Q3 + 1.5·IQR)
iso_contam     = 0.01      # expected fraction of outliers for IsolationForest

# -----------------------
# Make a working copy of your original Training_df
# -----------------------
df = Training_df.copy()

# ---- Z-score flags ---------------------------------------------
z_scores = np.abs(zscore(df[numeric_cols]))
df['Z_Outlier'] = (z_scores > z_thresh).any(axis=1)

# ---- IQR flags ------------------------------------------------
Q1  = df[numeric_cols].quantile(0.25)
Q3  = df[numeric_cols].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - iqr_multiplier * IQR
upper = Q3 + iqr_multiplier * IQR
df['IQR_Outlier'] = ((df[numeric_cols] < lower) | (df[numeric_cols] > upper)).ar

# ---- Isolation-Forest flags ----------------------------------------
iso = IsolationForest(contamination=iso_contam, random_state=42)
df['IF_Outlier'] = iso.fit_predict(df[numeric_cols]) == -1   # True  = outlier

# ---- Build cleaned frames ------------------------------------------
cleaned_df_z    = df[~df['Z_Outlier']].copy()
cleaned_df_iqr = df[~df['IQR_Outlier']].copy()
cleaned_df_if   = df[~df['IF_Outlier']].copy()

print("Rows kept  |  Z-score:", cleaned_df_z.shape[0],
      "|  IQR:", cleaned_df_iqr.shape[0],
      "|  IsolationForest:", cleaned_df_if.shape[0])

# -------------------------------------------------------------------
# ❷  Visualise: 4-row × 5-column box-plot grid
# -------------------------------------------------------------------
import matplotlib.pyplot as plt
import seaborn as sns

# Plot style parameters
fig, axes = plt.subplots(
    nrows=4, ncols=len(numeric_cols),
    figsize=(18, 14), sharey='row', dpi=900
)

#fig.suptitle("Outlier Cleaning Comparison by Method", fontsize=28)

title_fs  = 22.5   # per-subplot title size
label_fs  = 22.5   # ylabel font size
tick_fs   = 22.5   # tick-label font size

# Row 0 – Original
for j, col in enumerate(numeric_cols):
```

```
    sns.boxplot(y=df[col], ax=axes[0, j], color='lightgray')
    axes[0, j].set_title(f"{col}\n(Original)", fontsize=title_fs)
    axes[0, j].tick_params(axis='both', labelsize=tick_fs)
    axes[0, 0].set_ylabel("Original", fontsize=label_fs)

# Row 1 – Z-score cleaned
for j, col in enumerate(numeric_cols):
    sns.boxplot(y=cleaned_df_z[col], ax=axes[1, j], color='salmon')
    axes[1, j].set_title(f"{col}\n(Z-score)", fontsize=title_fs)
    axes[1, j].tick_params(axis='both', labelsize=tick_fs)
    axes[1, 0].set_ylabel("Z-score", fontsize=label_fs)

# Row 2 – IQR cleaned
for j, col in enumerate(numeric_cols):
    sns.boxplot(y=cleaned_df_iqr[col], ax=axes[2, j], color='skyblue')
    axes[2, j].set_title(f"{col}\n(IQR)", fontsize=title_fs)
    axes[2, j].tick_params(axis='both', labelsize=tick_fs)
    axes[2, 0].set_ylabel("IQR", fontsize=label_fs)

# Row 3 – Isolation-Forest cleaned
for j, col in enumerate(numeric_cols):
    sns.boxplot(y=cleaned_df_if[col], ax=axes[3, j], color='lightgreen')
    axes[3, j].set_title(f"{col}\n(ISO Forest)", fontsize=title_fs)
    axes[3, j].tick_params(axis='both', labelsize=tick_fs)
    axes[3, 0].set_ylabel("ISO\nForest", fontsize=label_fs)

# Remove redundant x-labels and tidy
for ax in axes.flatten():
    ax.set_xlabel("")

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.savefig("outlier_cleaning_comparison_by_method.png", dpi=900, bbox_inches='t
plt.show()
```

```
Rows kept  |  Z-score: 125586 |  IQR: 123722 |  IsolationForest: 131564
```



In [29]:
```
import matplotlib.pyplot as plt

# Features and outlier method mapping
features = ['NGAM', 'CALP', 'SHN_LONG_AVG', 'SPR', 'LSD_HRD_AVG']
methods = {
    'Z-score': ('Z_Outlier', 'red'),
    'IQR': ('IQR_Outlier', 'blue'),
    'IF': ('IF_Outlier', 'orange')
}

# Font configuration
title_fontsize = 25
label_fontsize = 25
tick_fontsize = 25
legend_fontsize = 25

# Create subplot grid: 3 rows (methods) × 5 columns (features)
n_rows, n_cols = len(methods), len(features)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(25, 15), dpi=900)
#fig.suptitle("Comparison of Outlier Detection Methods vs. DEPTH", fontsize=24)

# Iterate over each method-feature pair
for row_idx, (method_name, (flag_col, color)) in enumerate(methods.items()):
    for col_idx, feature in enumerate(features):
        ax = axes[row_idx, col_idx]

        # Scatter plot: normal points
        ax.scatter(Training_df['DEPTH'], Training_df[feature],
                   c='gray', s=5, alpha=0.4, label='Normal')

        # Scatter plot: outliers
        outliers = Training_df[Training_df[flag_col]]
        ax.scatter(outliers['DEPTH'], outliers[feature],
                   c=color, s=10, label=f'{method_name} Outlier')

        ax.set_xlabel("DEPTH", fontsize=label_fontsize)
        # Set titles and labels
        if row_idx == 0:
            ax.set_title(f"{feature}", fontsize=title_fontsize)
        if col_idx == 0:
            ax.set_ylabel(f"{method_name}", fontsize=label_fontsize)

        ax.tick_params(axis='both', labelsize=tick_fontsize)
        ax.grid(True)

        # Show legend and outlier count in first column
        if col_idx == 0:
            # total = len(Training_df)
            # count = len(outliers)
            # percent = (outliers[feature].notna().sum() / total) * 100
            ax.legend(fontsize=legend_fontsize, title_fontsize=legend_fontsize)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.savefig("outlier_methods_colored_comparison.png", dpi=900, bbox_inches='tigh
plt.show()
```


NGAM
CALP
SHN_LONG_AVG
SPR
LSD_HRD_AVG
Normal
Z-score Outlier
IQR Outlier
IF Outlier
Z-score
IQR
IF
DEPTH