Hydraulic Health Monitoring

Objective:

The project aims to predict the stability of a hydraulic system in a manufacturing plant, thereby reducing downtime and maintenance costs. By leveraging machine learning, the goal is to proactively identify potential equipment failures before they impact production, ensuring continuous operation and optimizing maintenance practices.

Problem Statement:

Manufacturing plants heavily rely on hydraulic systems, and unexpected failures can lead to unplanned downtime and increased costs, negatively affecting overall production and profitability. The challenge is to develop a predictive maintenance solution using machine learning to anticipate and address potential failures, maximizing uptime and minimizing maintenance expenses.

Business Requirements:

● Detect and prevent sensor failures proactively. ● Develop a data-driven approach to establish relationships between sensor data and prediction accuracy. ● Apply algorithms to analyze data and detect anomalies in hydraulic system performance. ● Provide predictive maintenance recommendations for component servicing or replacement. ● Store historical data for trend analysis, performance assessment, and compliance reporting.

Data Entities:

Sensor Data:

This is a primary data entity representing the raw sensor measurements from the hydraulic test rig. Each sensor, such as PS1, PS2, EPS1, FS1, TS1, VS1, CE, and CP, generates data points at specific time intervals (e.g., 100 Hz, 10 Hz, 1 Hz). Sensor data is essential for understanding the operational state of the hydraulic system.

Hydraulic Components:

The condition of four key hydraulic components (cooler, valve, pump, and accumulator) is another set of data entities. These components are crucial to the hydraulic system and are assessed based on the sensor data. The condition of these components is expressed as percentages or pressure values.

Target Condition Values:

The annotated condition values associated with each hydraulic component, such as cooler condition, valve condition, internal pump leakage, and hydraulic accumulator pressure, represent distinct data entities. These values are derived from sensor data and are used for classification and regression tasks.

Cycles:

Cycles represent a temporal data entity within the dataset. Each cycle corresponds to a period of constant load, lasting 60 seconds. The data is organized into cycles, with each cycle containing sensor measurements and target condition values.

In [ ]:

```
```

In [28]:

```
#Importing all the necessary libraries
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from scipy.stats import spearmanr
# sklearn libraries
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import OneHotEncoder, QuantileTransformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.feature_selection import chi2
from sklearn.preprocessing import StandardScaler
from sklearn import model_selection
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
# imbalanced-learn library
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE
#joblib for model persistence
from joblib import dump, load
```

In [29]:

```
#Reading the data
hydraulic_systemdf = pd.read_csv('manufacturing_data.csv')
profile_data = pd.read_csv('profile.txt', sep="\t", header=None)
profile_data.columns = ['cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_accumulator', 'stable_flag']
#Data pre processing
hydraulic_systemdf = hydraulic_systemdf.drop(columns=["Unnamed: 0", "Time"])
hydraulic_systemdf = hydraulic_systemdf.rename(columns={
    "Cooling efficiency": "CE",
    "Cooling power": "CP",
    "Motor power W": "EPS1",
    "Volume flow l/min 1": "FS1",
    "Volume flow l/min 2": "FS2",
    "Pressure bar 1": "PS1",
    "Pressure bar 2": "PS2",
    "Pressure bar 3": "PS3",
    "Pressure bar 4": "PS4",
    "Pressure bar 5": "PS5",
    "Pressure bar 6": "PS6",
    "Efficiency factor": "SE",
    "Temperature 1": "TS1",
    "Temperature  2": "TS2",
    "Temperature  3": "TS3",
    "Temperature  4": "TS4",
    " Vibration mm/s": "VS1"
```

```
})
hydraulic_systemdf = pd.concat([hydraulic_systemdf,profile_data], axis=1)
```

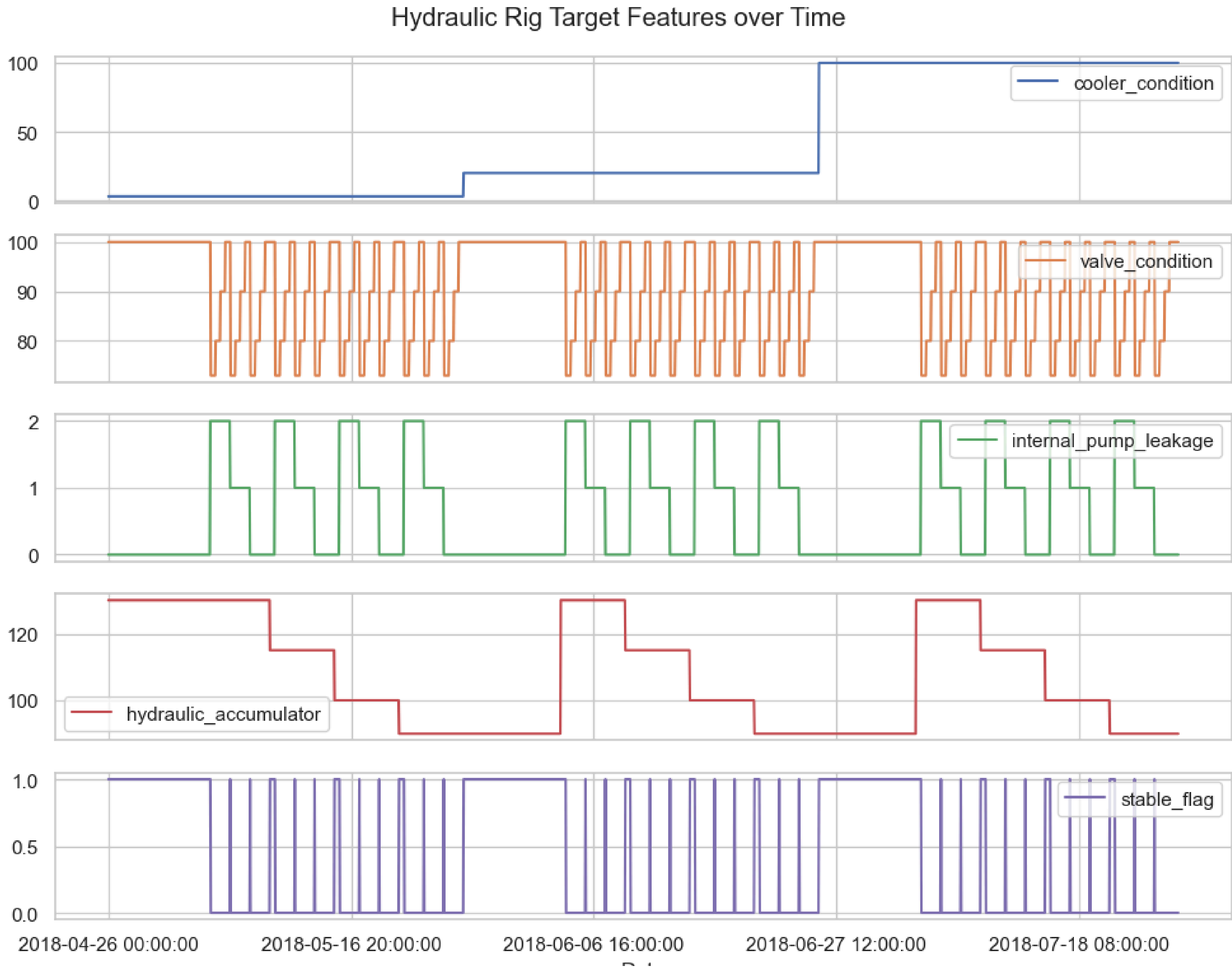
In [30]:
```
hydraulic_systemdf.head(5)
```

Out[30]:

	CE	CP	EPS1	FS1	FS2	PS1	PS2	PS3	PS4	PS5	...	TS2	TS3	TS4	VS1	Date	cooler_condition	valve_condition	ir
0	47.202	2.184	2411.6	8.990	10.179	151.47	125.50	2.305	0.0	9.936	...	40.961	38.320	30.363	0.604	2018-04-26 00:00:00	3	100	
1	29.208	1.414	2409.6	8.919	10.408	151.11	125.06	2.281	0.0	9.700	...	41.258	38.680	33.648	0.590	2018-04-26 01:00:00	3	100	
2	23.554	1.159	2397.8	9.179	10.392	150.81	125.13	2.227	0.0	9.606	...	42.129	39.234	35.113	0.578	2018-04-26 02:00:00	3	100	
3	21.540	1.101	2383.8	9.034	10.329	150.48	124.93	2.320	0.0	9.528	...	43.039	40.086	36.133	0.565	2018-04-26 03:00:00	3	100	
4	20.460	1.086	2372.0	8.729	10.276	150.41	124.72	2.250	0.0	9.408	...	44.031	40.934	36.992	0.570	2018-04-26 04:00:00	3	100	

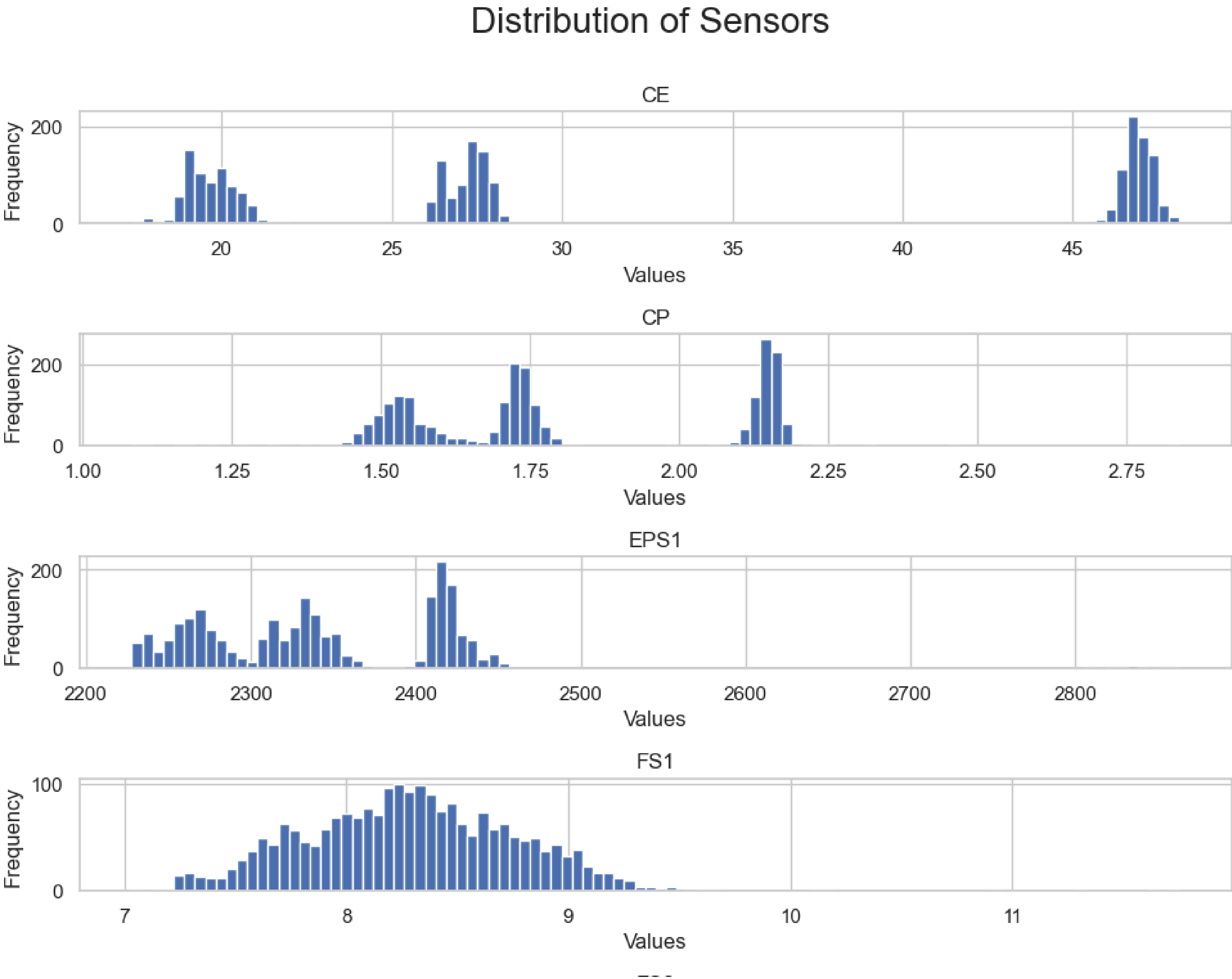
5 rows × 23 columns

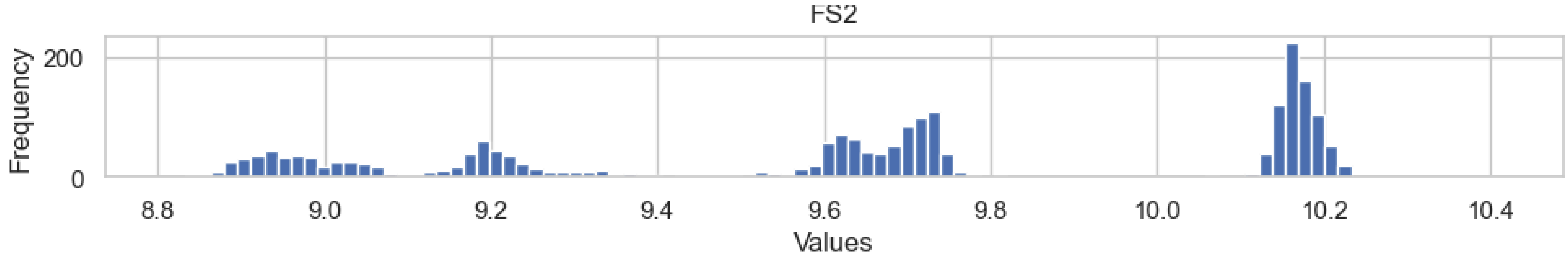
In [31]:
```
#Explortory Data Analysis
hydraulic_systemdf.plot(x='Date',
              title = "Hydraulic Rig Target Features over Time",
              y=['cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_accumulator', 'stable_flag'],
              figsize=(10,8),
              subplots=True)
plt.tight_layout()
plt.savefig('target_features.png', format='png')
plt.show()
```

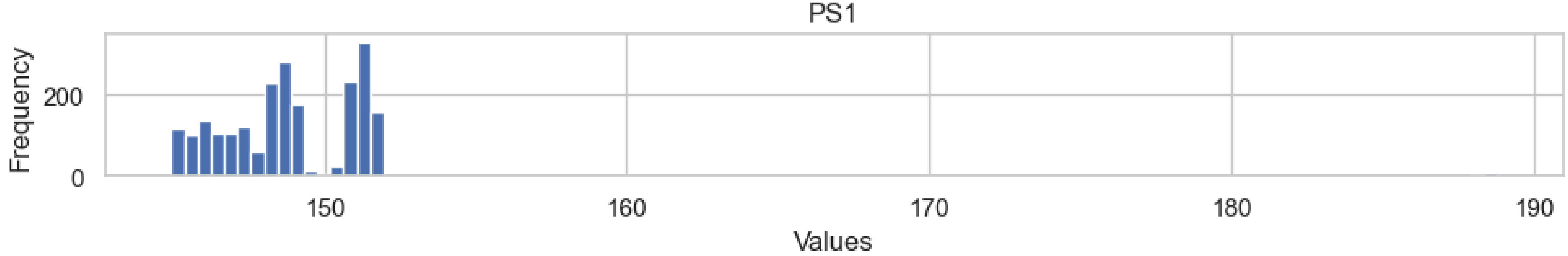

Hydraulic Rig Target Features over Time
100
50
0
cooler_condition
100
90
80
valve_condition
2
1
0
internal_pump_leakage
120
100
hydraulic_accumulator
1.0
0.5
0.0
stable_flag
2018-04-26 00:00:00
2018-05-16 20:00:00
2018-06-06 16:00:00
2018-06-27 12:00:00
2018-07-18 08:00:00

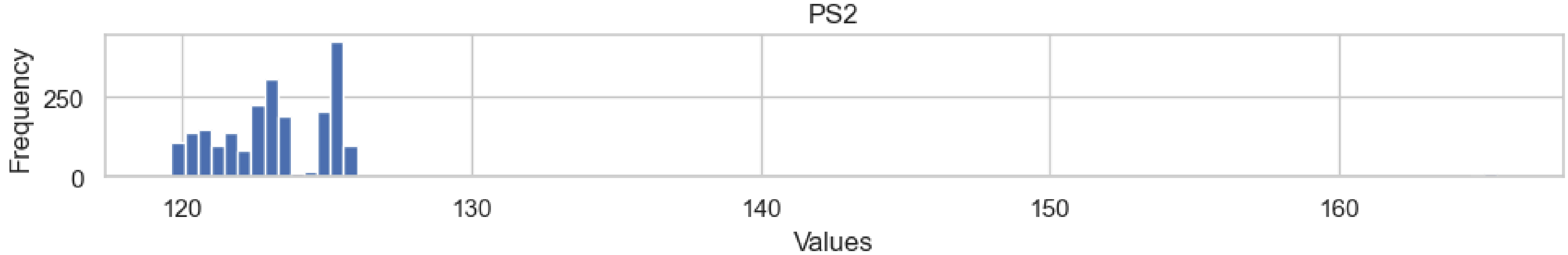
Date

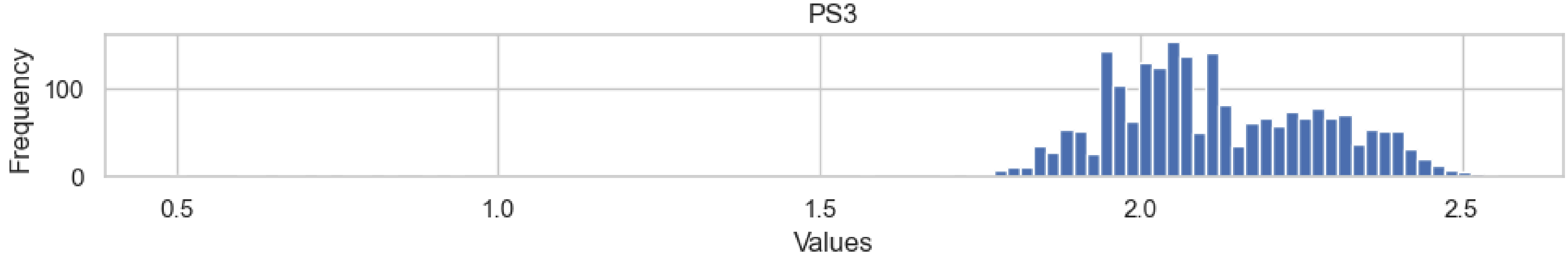
In [32]:
```
#Sensor Data Histogram
all_sensors = ['CE', 'CP', 'EPS1', 'FS1', 'FS2', 'PS1', 'PS2', 'PS3', 'PS4', 'PS5',
                  'PS6', 'SE', 'TS1', 'TS2', 'TS3', 'TS4', 'VS1']

#Creating subplots for each sensor
fig, axes = plt.subplots(nrows=len(all_sensors), figsize=(10, 30))

#Setting the title for the whole figure
fig.suptitle('Distribution of Sensors', fontsize=20, x=.53, y=1)

#Iterating over each sensor column and plot histogram
for i, column in enumerate(all_sensors):
    ax = axes[i]
    ax.hist(hydraulic_systemdf[column], bins=100)
    ax.set_title(column)
    ax.set_xlabel('Values')
    ax.set_ylabel('Frequency')

#Adjust spacing between subplots
plt.tight_layout()
plt.savefig('sensordistribution.png', format='png')
#Show the plot
plt.show()
```

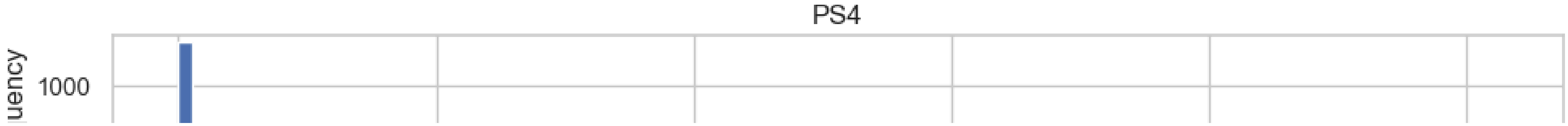

Distribution of Sensors
CE
Frequency
Values
CP
Frequency
Values
EPS1
Frequency
Values
FS1
Frequency
Values


FS2
Frequency
200
0
8.8
9.0
9.2
9.4
9.6
9.8
10.0
10.2
10.4
Values

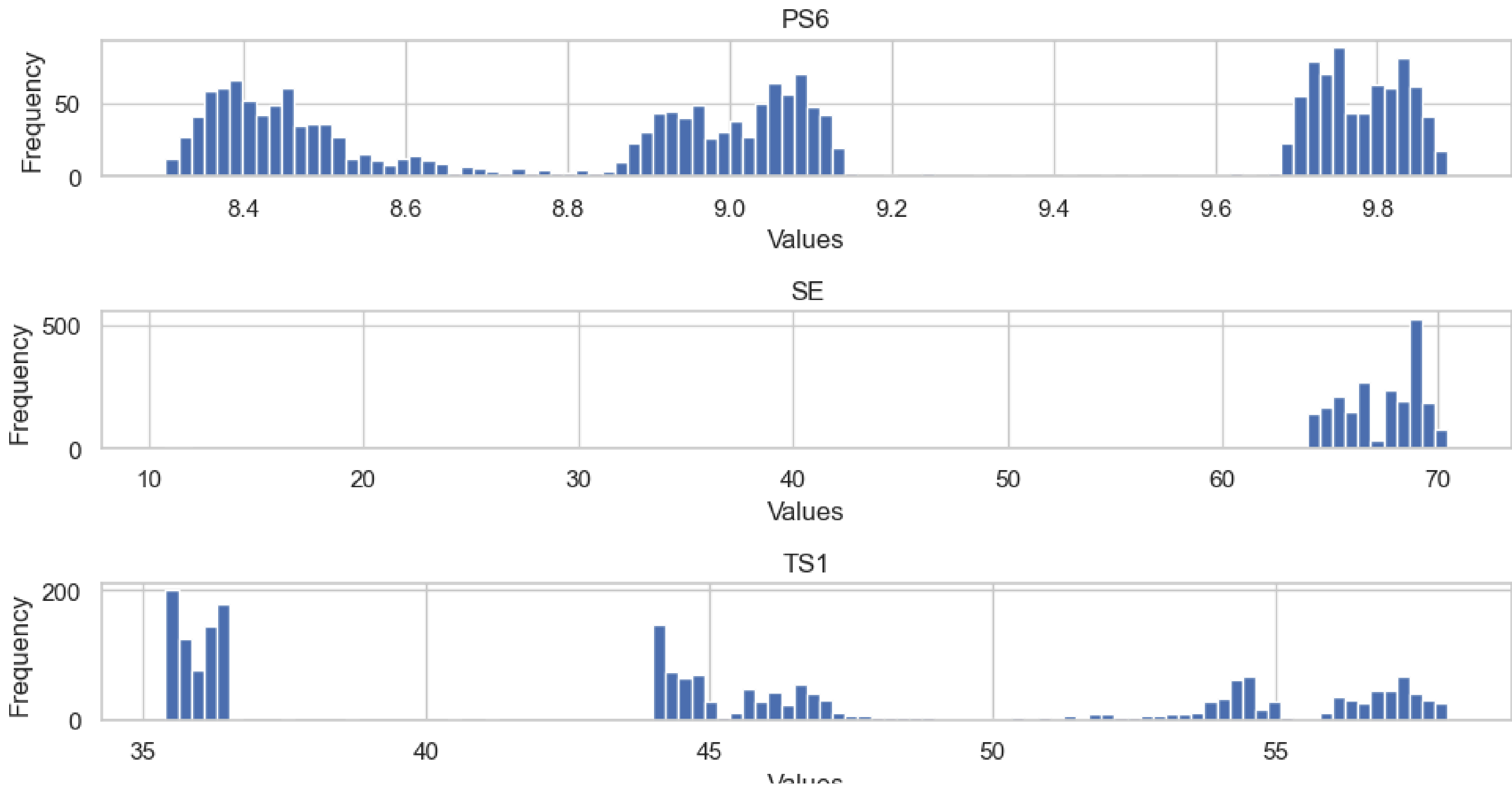

PS1
Frequency
200
0
150
160
170
180
190
Values

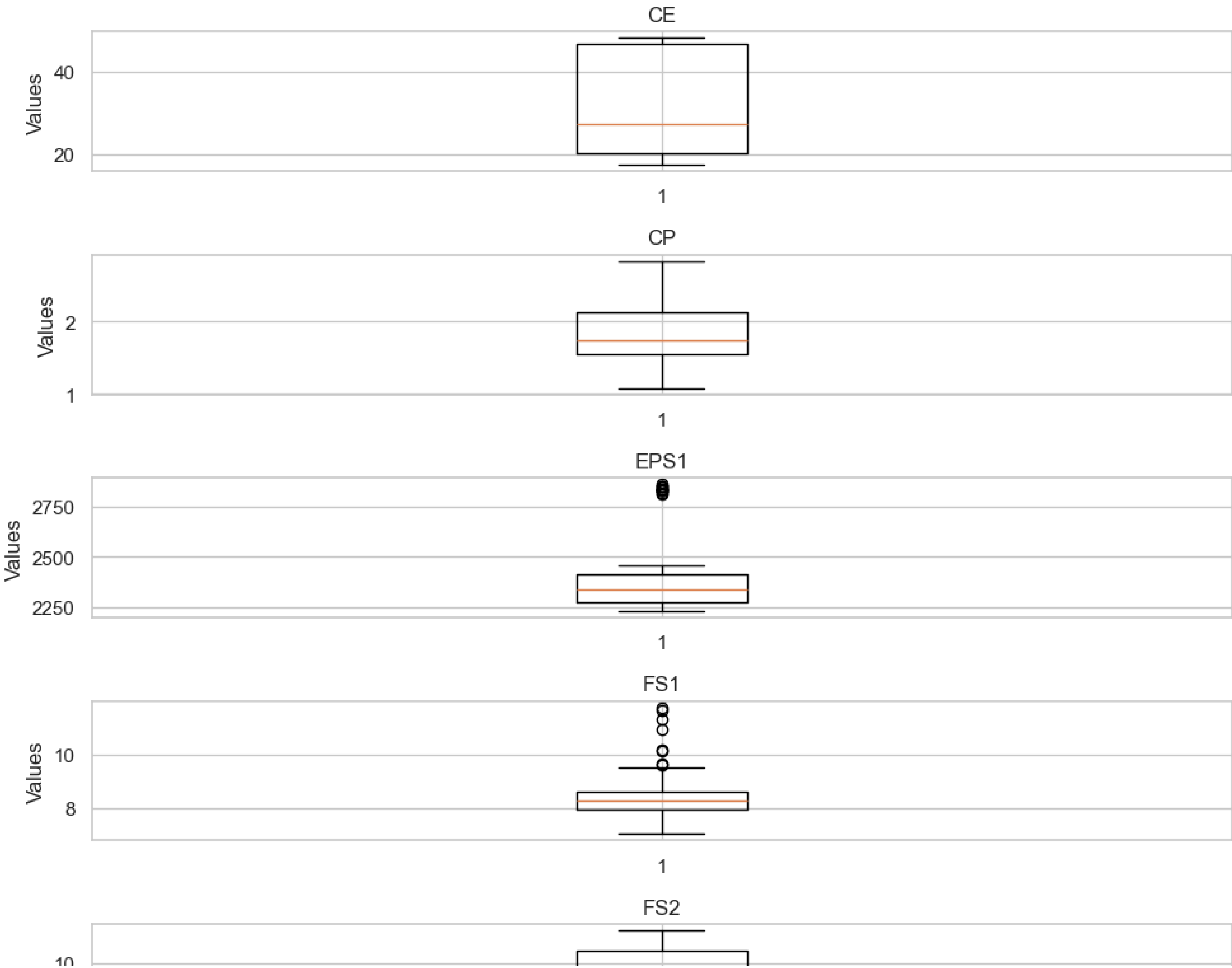

PS2
Frequency
250
0
120
130
140
150
160
Values

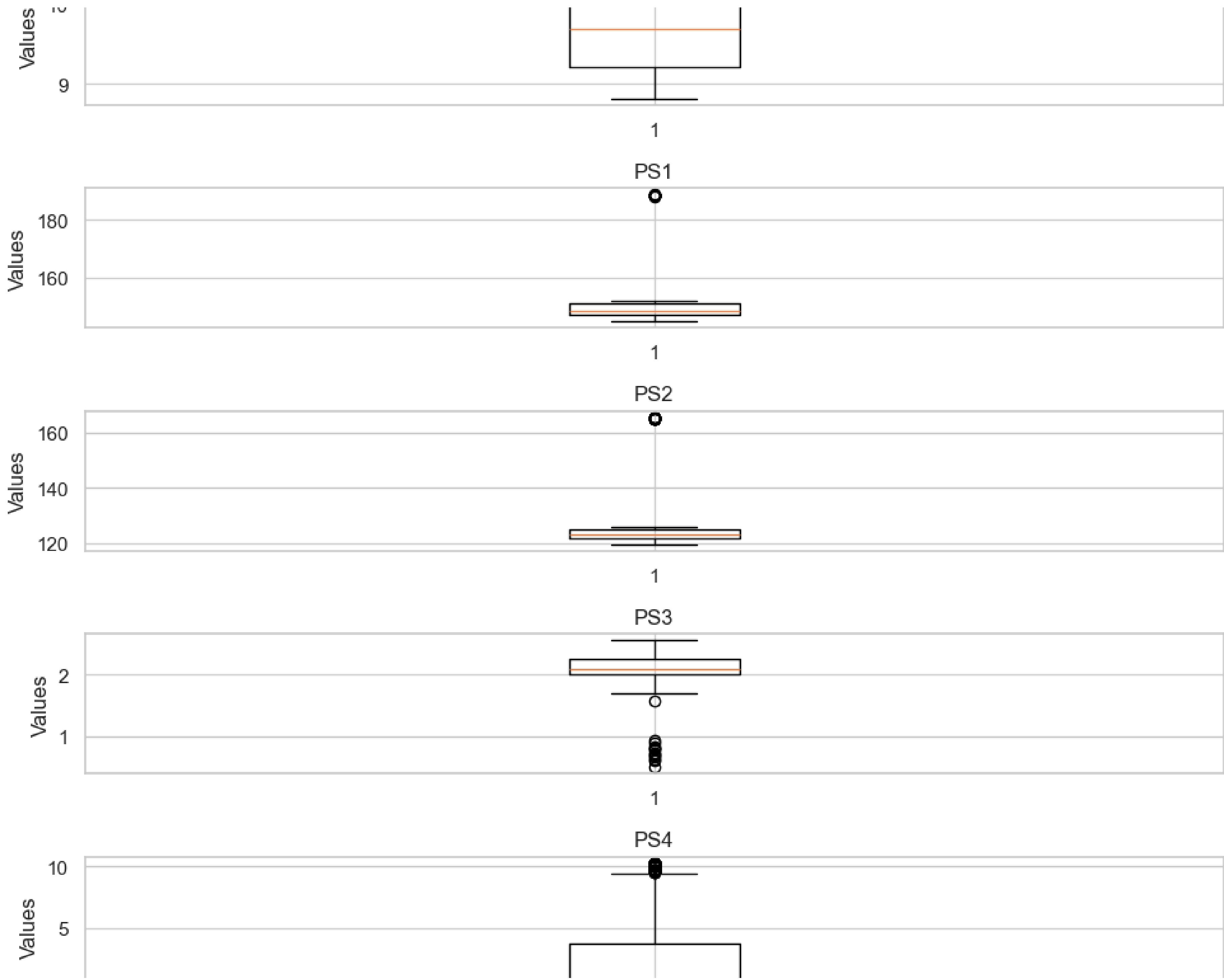

PS3
Frequency
100
0
0.5
1.0
1.5
2.0
2.5
Values


PS4
1000

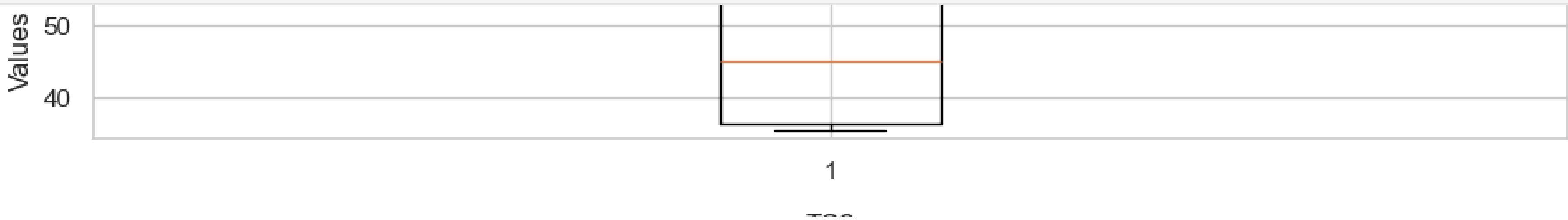
In [33]:
```
# Create subplots for each sensor
fig, axes = plt.subplots(nrows=len(all_sensors), figsize=(10, 30))
for i, column in enumerate(all_sensors):
    ax = axes[i]
    ax.boxplot(hydraulic_systemdf[column])
    ax.set_title(column)
    ax.set_ylabel('Values')

# Adjust spacing between subplots
plt.tight_layout()
plt.savefig('sensorboxplot.png', format='png')
# Show the plot
plt.show()
```



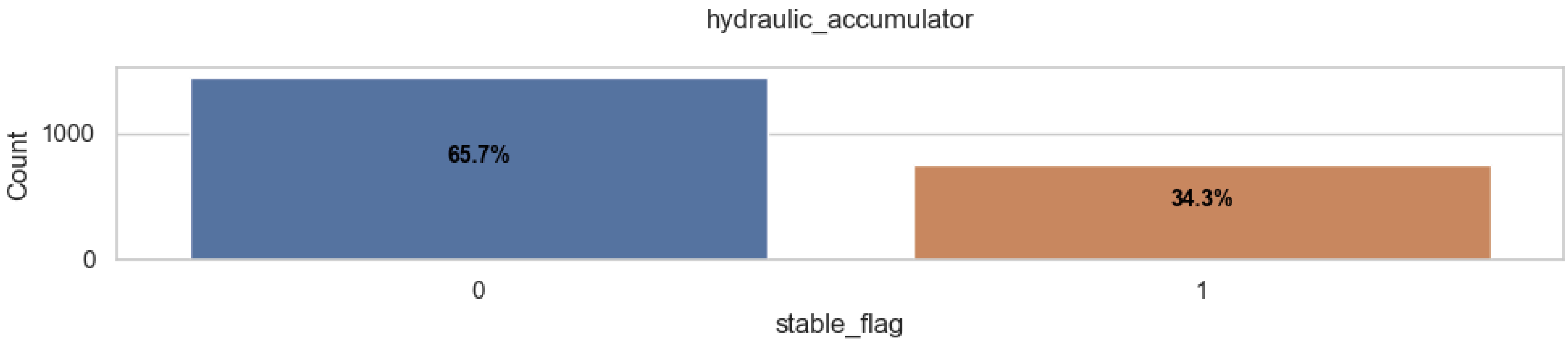

CE
Values
40
20
1
CP
Values
2
1
1
EPS1
Values
2750
2500
2250
1
FS1
Values
10
8
1
FS2
10


Values
9
1
PS1
180
160
Values
1
PS2
160
140
120
Values
1
PS3
2
1
Values
1
PS4
10
5
Values

In [34]:
```python
# Subset the columns
profile_columns = ['cooler_condition','valve_condition', 'internal_pump_leakage', 'hydraulic_accumulator', 'stable_flag']

# Create subplots
fig, axes = plt.subplots(len(profile_columns), 1, figsize=(10, 10))

# Set the title for the whole figure
fig.suptitle('Class Distribution(Profile)', fontsize=20)

# Iterate over columns and plot count plots
for i, column in enumerate(profile_columns):
    ax = axes[i]
    sns.countplot(x=column, data=hydraulic_systemdf, ax=ax)
    ax.set_xlabel(column)
    ax.set_ylabel('Count')

    # Calculate total count of records for the current column
    total = len(hydraulic_systemdf[column])

    # Iterate over all bars and add percentage text inside each bar
    for p in ax.patches:
        height = p.get_height()
        # If height is 0, we want to avoid division by zero error
        if height == 0:
            continue
        percentage = f'{100 * height/total:.1f}%'
        ax.text(p.get_x()+p.get_width()/2., height/2, percentage, ha='center', va='bottom', fontsize=10, fontweight='bold', color

# Adjust spacing between subplots
plt.tight_layout()

plt.savefig('class_distributionsprofile.png', format='png')

# Show the plot
plt.show()
```

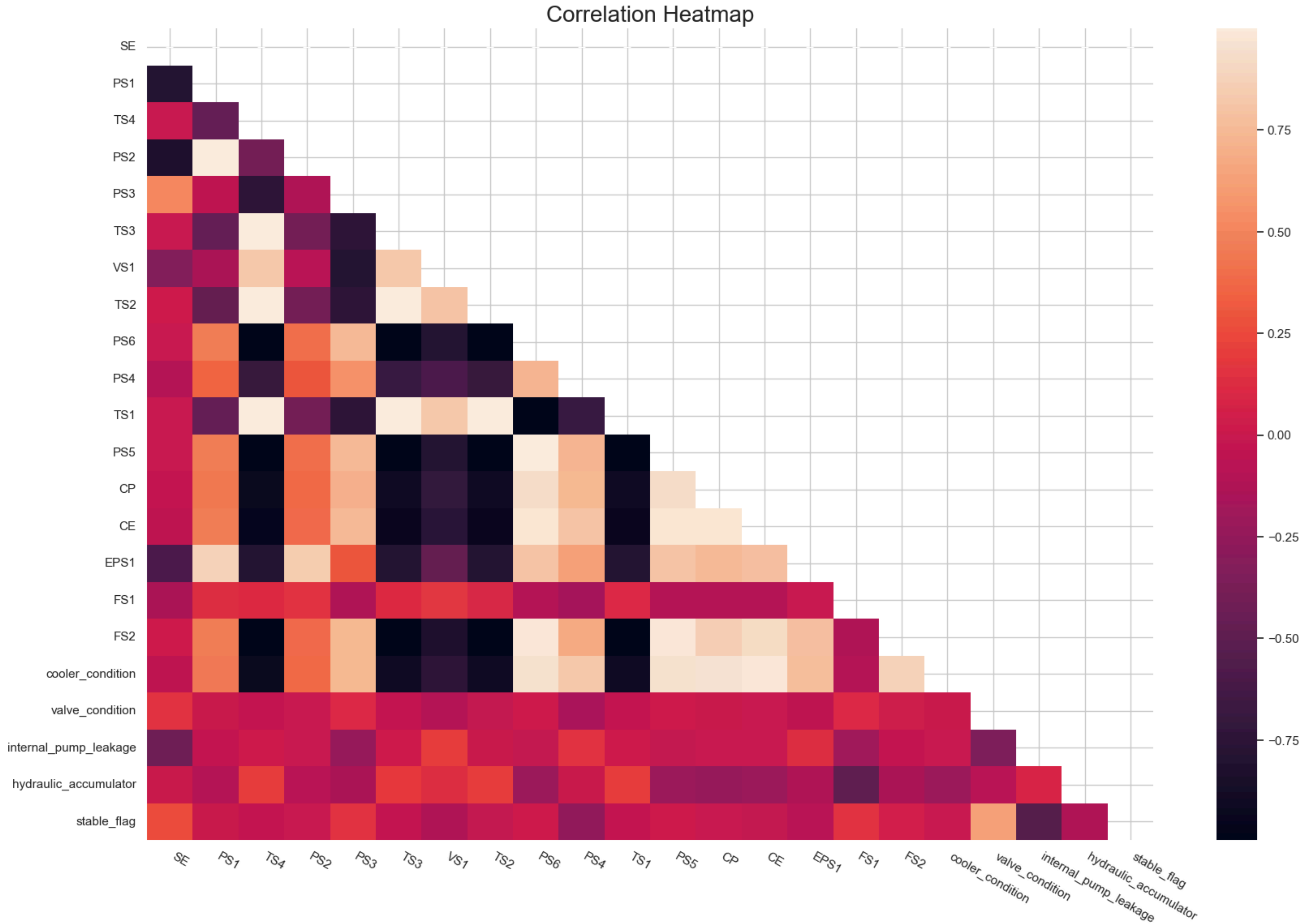



Class Distribution(Profile)
Count
500
0
33.2%
33.2%
33.6%
3
20
100
cooler_condition
Count
1000
500
0
16.3%
16.3%
16.3%
51.0%
73
80
90
100
valve_condition
Count
1000
500
0
55.4%
22.3%
22.3%
0
1
2
internal_pump_leakage
Count
500
0
36.6%
18.1%
18.1%
27.2%
90
100
115
130



In [35]:
```
#Co-relation Heatmap
# Select the columns for correlation heatmap
all_columns = ['SE', 'PS1', 'TS4', 'PS2', 'PS3', 'TS3', 'VS1', 'TS2', 'PS6', 'PS4', 'TS1',
           'PS5', 'CP', 'CE', 'EPS1', 'FS1', 'FS2', 'cooler_condition', 'valve_condition',
           'internal_pump_leakage', 'hydraulic_accumulator', 'stable_flag']

# Create a new figure with a size of 20x13
fig, ax = plt.subplots(figsize=(20, 13))

# Extract the selected columns and compute the correlation matrix
correlation_matrix = hydraulic_systemdf[all_columns].corr()

# Create a mask to hide the upper triangle
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# Plot a heatmap of the correlation matrix with the mask applied
sns.heatmap(correlation_matrix, annot=True, ax=ax, mask=mask)

# Set the title and rotate x-axis labels
ax.set_title('Correlation Heatmap', fontsize=20)
ax.set_xticklabels(ax.get_xticklabels(), rotation=-30, ha='left')

plt.savefig('correlation_heatmap.png', format='png')
# Show the plot
plt.show()
```

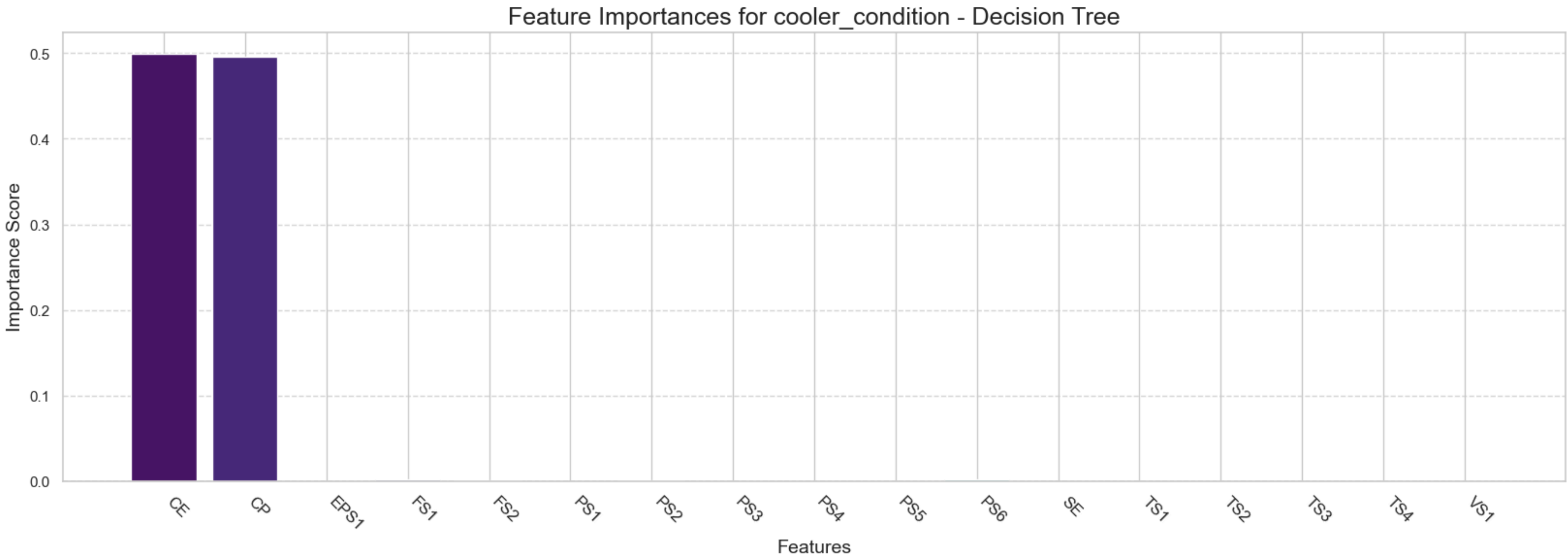
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\matrix.py:260: FutureWarning: Format strings passed to MaskedConstant are ignore
d, but in future may error or produce different behavior
  annotation = ("{:" + self.fmt + "}").format(val)
```

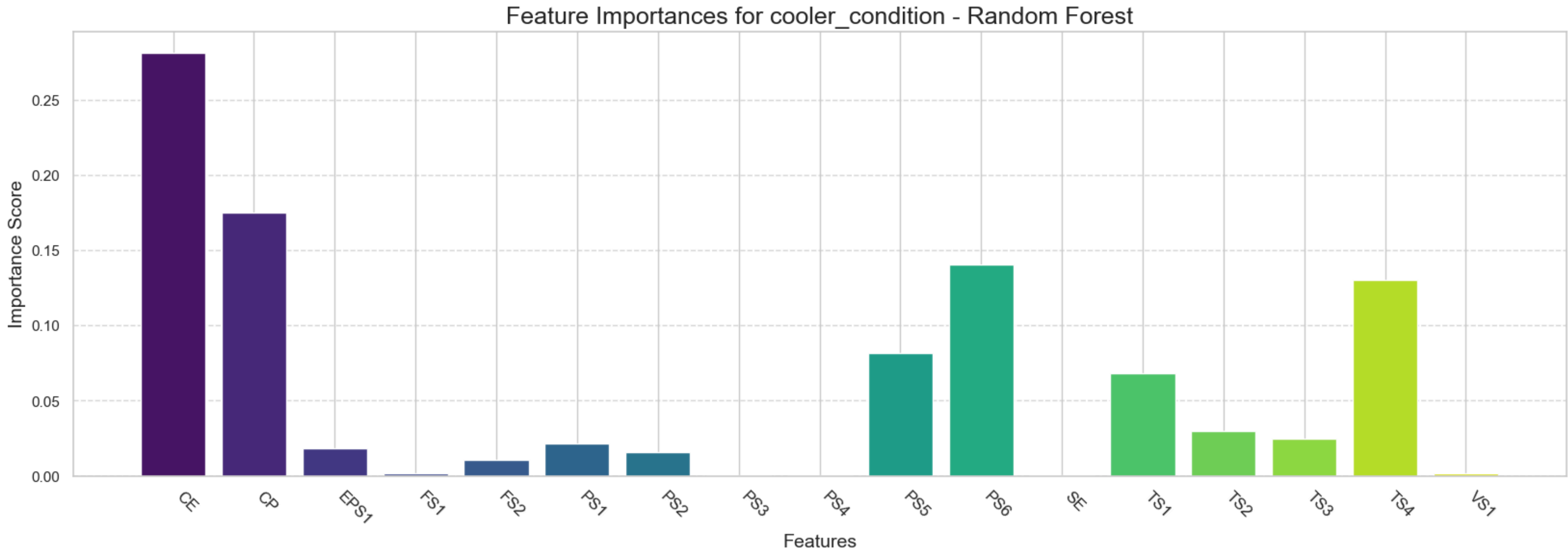


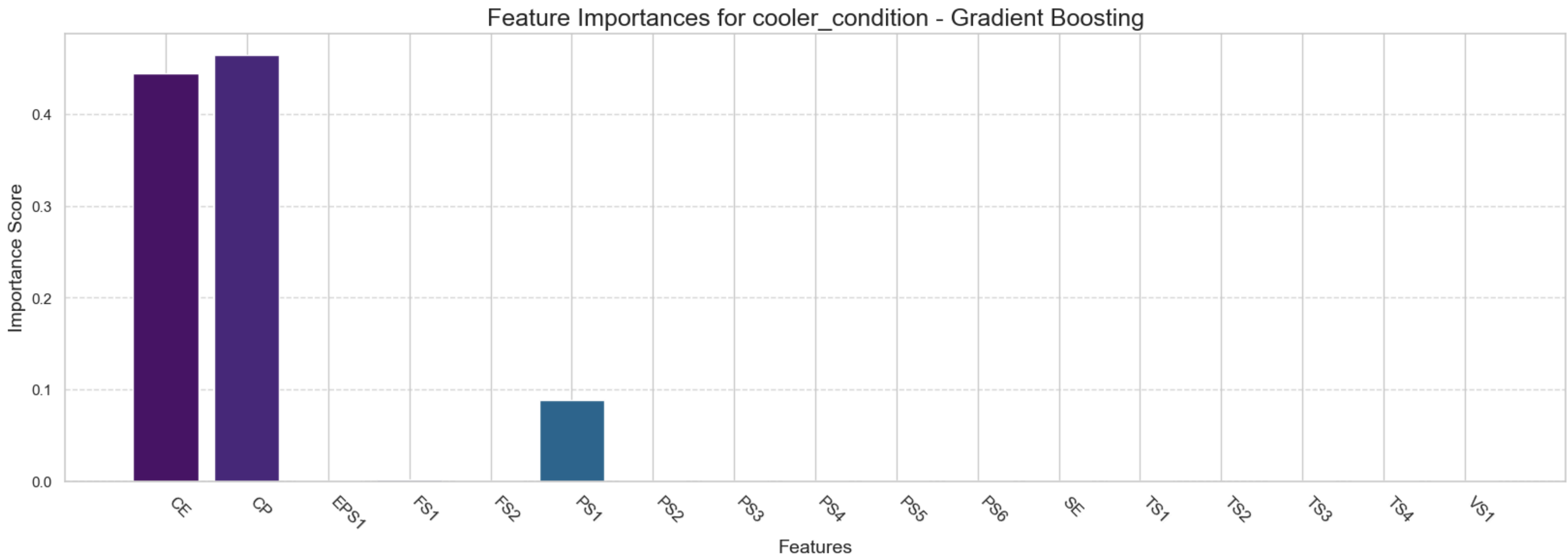
In [36]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

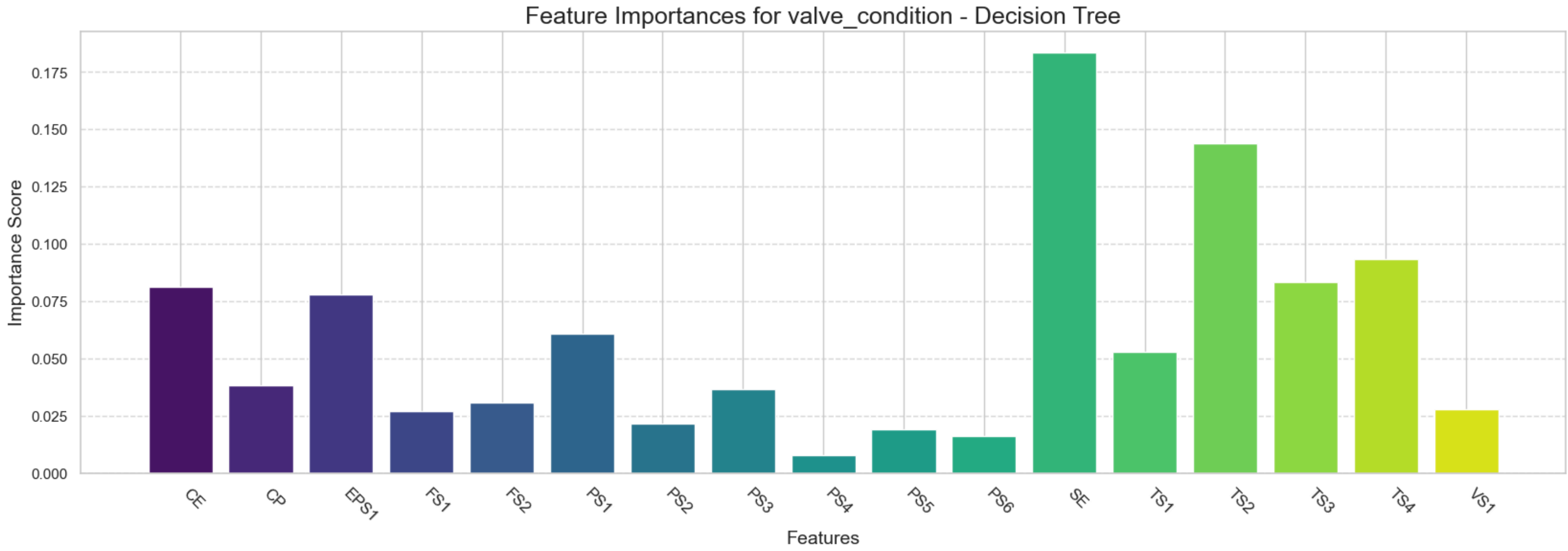
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

sns.set(style="whitegrid")
target_variables = ['cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_accumulator', 'stable_flag']
for target in target_variables:

    y = hydraulic_systemdf[target]

    if target == "stable_flag":
        X = hydraulic_systemdf.drop(columns=['Date', 'stable_flag'])
    else:
        X = hydraulic_systemdf.drop(columns=['Date','cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_ac
        # Binary classification for stable_flag
        #X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

        classifiers = {
            'Decision Tree': DecisionTreeClassifier(),
            'Random Forest': RandomForestClassifier(),
            'Gradient Boosting': GradientBoostingClassifier(),
            'KNN': KNeighborsClassifier(),
            'SVM': SVC()
        }

        for name, model in classifiers.items():
            model.fit(X, y)

            # For tree-based models, plot feature importances
            if name in ['Decision Tree', 'Random Forest', 'Gradient Boosting']:
                importances = model.feature_importances_

                # Plot feature importances
                fig, ax = plt.subplots(figsize=(20, 6))
                color = sns.color_palette("viridis", len(importances))
                ax.bar(X.columns, importances, color=color)
                ax.set_xlabel('Features', fontsize=14)
                ax.set_ylabel('Importance Score', fontsize=14)
                ax.set_title(f"Feature Importances for {target} - {name}", fontsize=18)
                plt.xticks(rotation=-45, ha='left', fontsize=12)

                # Add grid lines
                ax.grid(axis='y', linestyle='--', alpha=0.7)
```

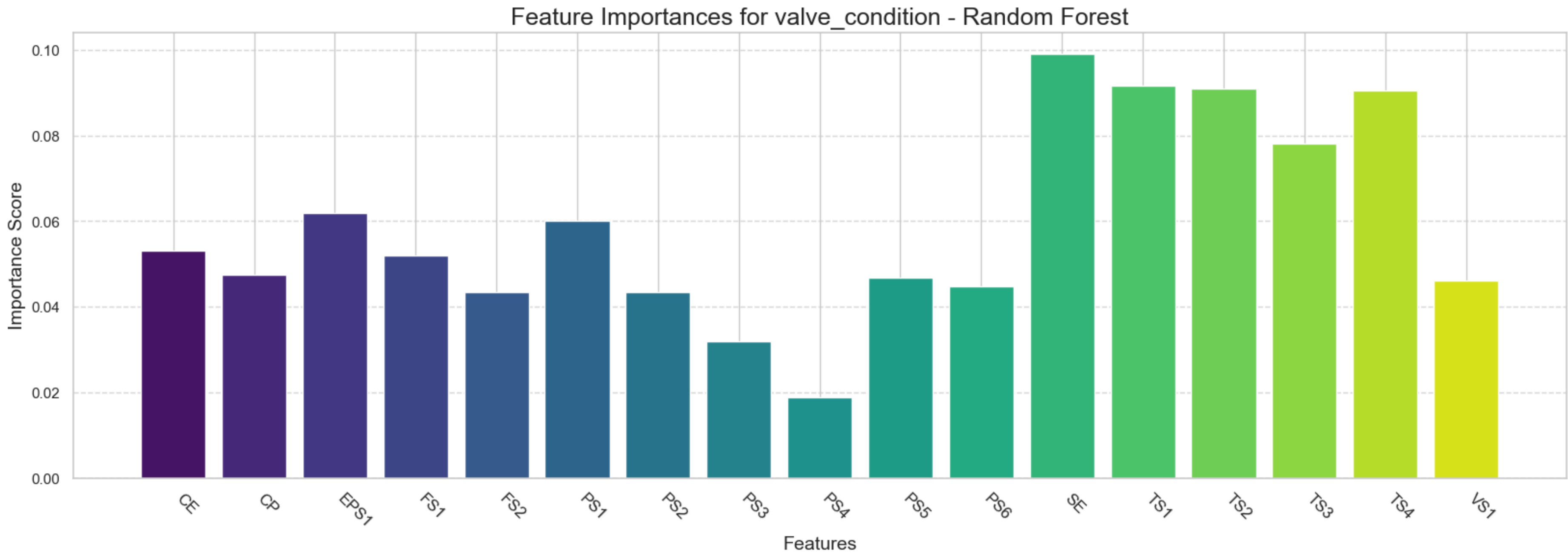
```
    # Save the figure as a png file
    plt.savefig("featureimportances.png", bbox_inches='tight')

    # Show the plot
    plt.show()
```

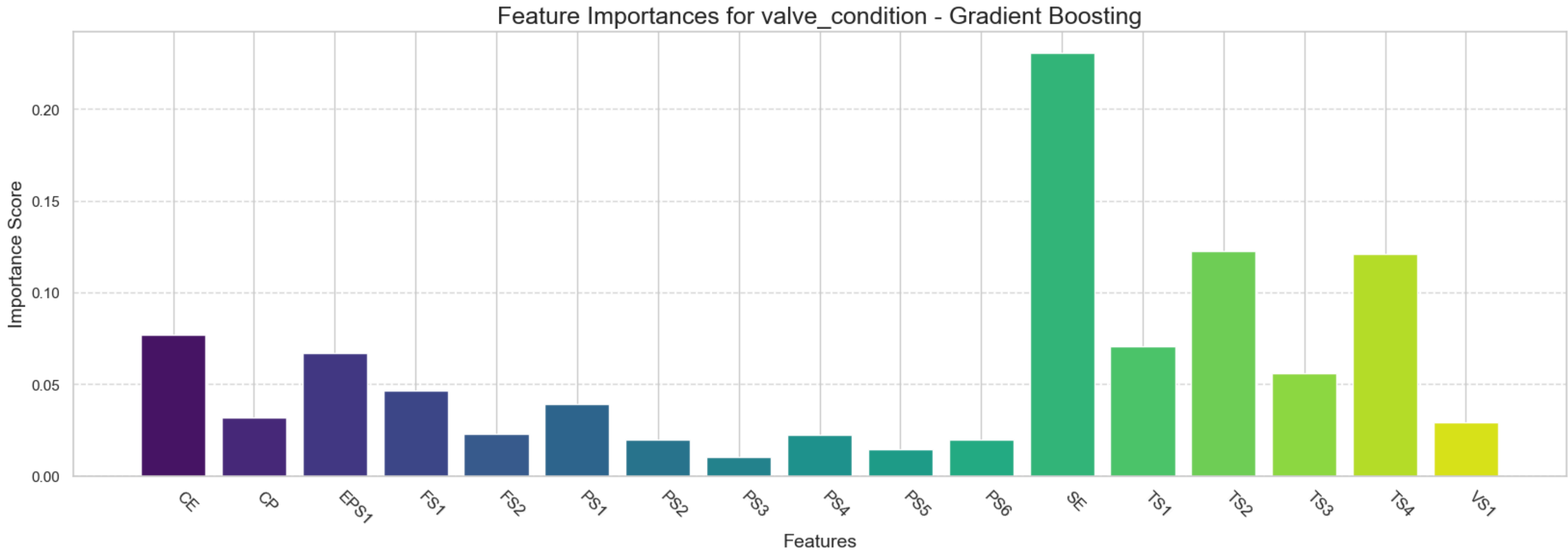


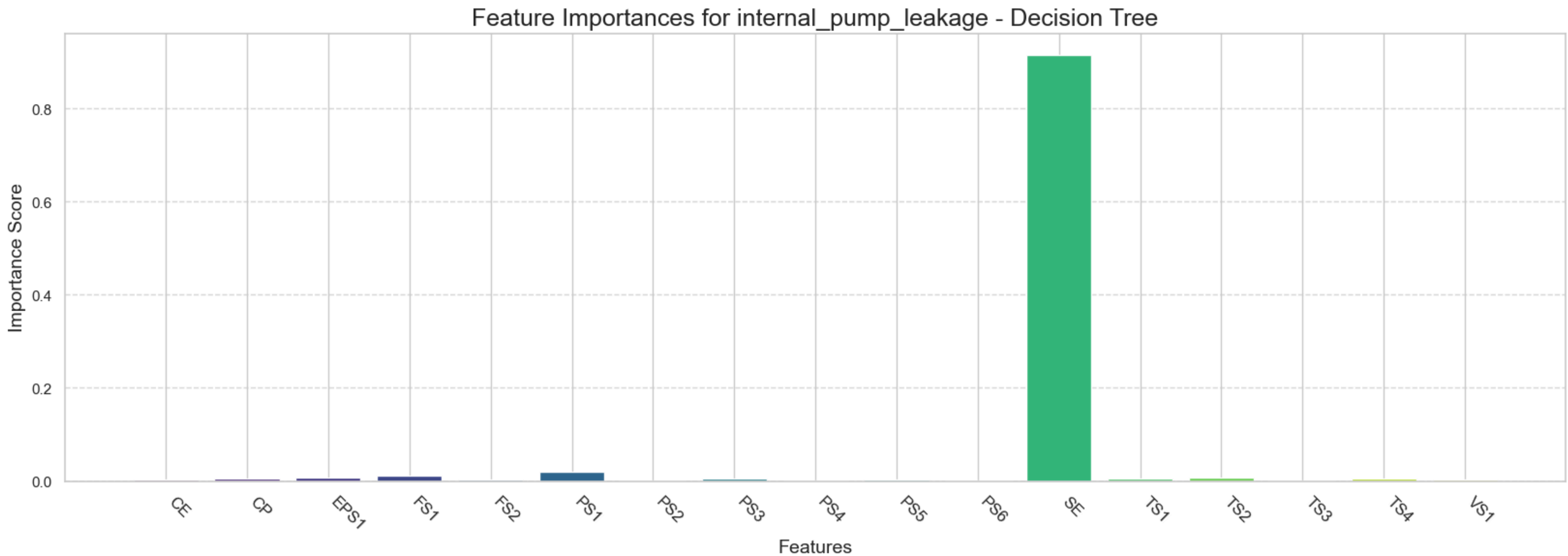

Feature Importances for cooler_condition - Random Forest
Importance Score
0.25
0.20
0.15
0.10
0.05
0.00
CE
CP
EPS1
FS1
FS2
PS1
PS2
PS3
PS4
PS5
PS6
SE
TS1
TS2
TS3
TS4
VS1
Features

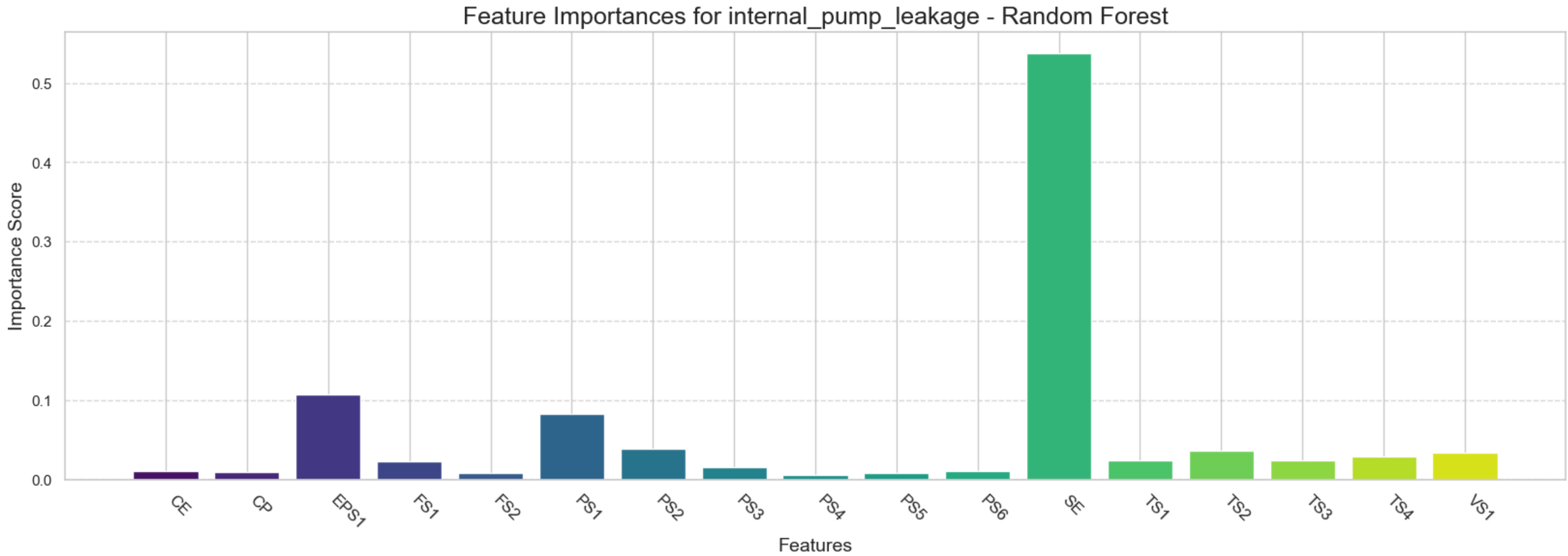

Feature Importances for cooler_condition - Gradient Boosting
Importance Score
0.4
0.3
0.2
0.1
0.0
CE
CP
EPS1
FS1
FS2
PS1
PS2
PS3
PS4
PS5
PS6
SE
TS1
TS2
TS3
TS4
VS1
Features

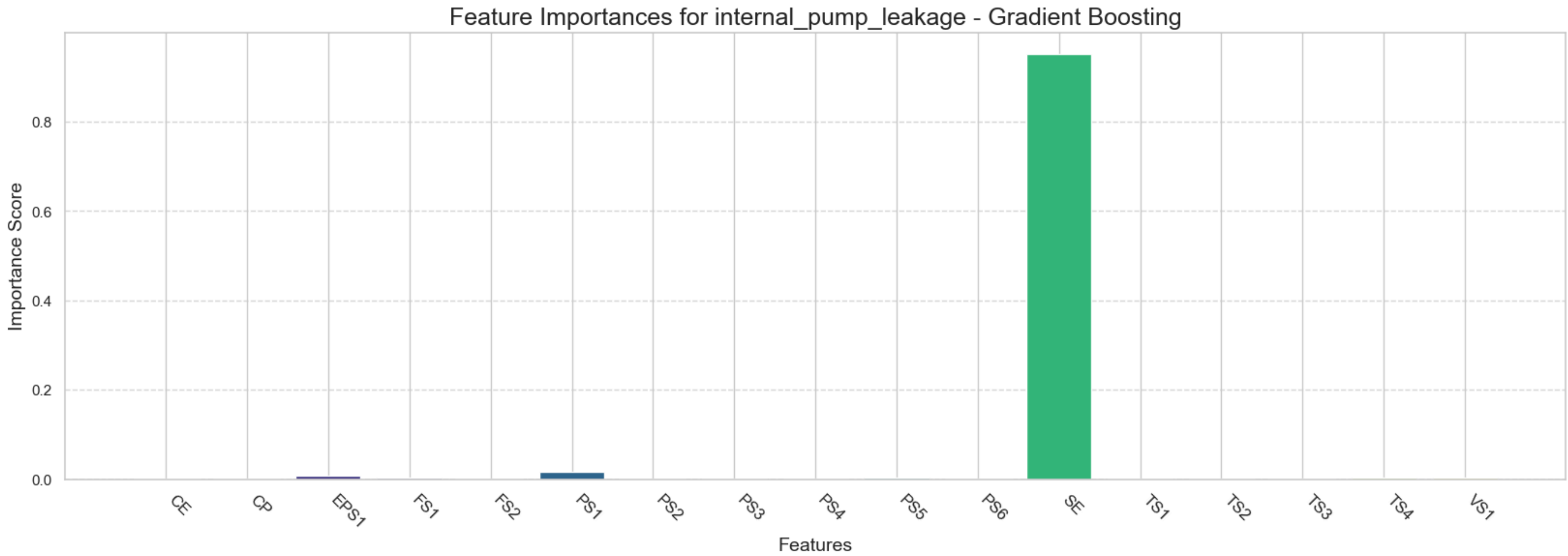

Feature Importances for valve_condition - Decision Tree
Importance Score
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
CE
CP
EPS1
FS1
FS2
PS1
PS2
PS3
PS4
PS5
PS6
SE
TS1
TS2
TS3
TS4
VS1
Features

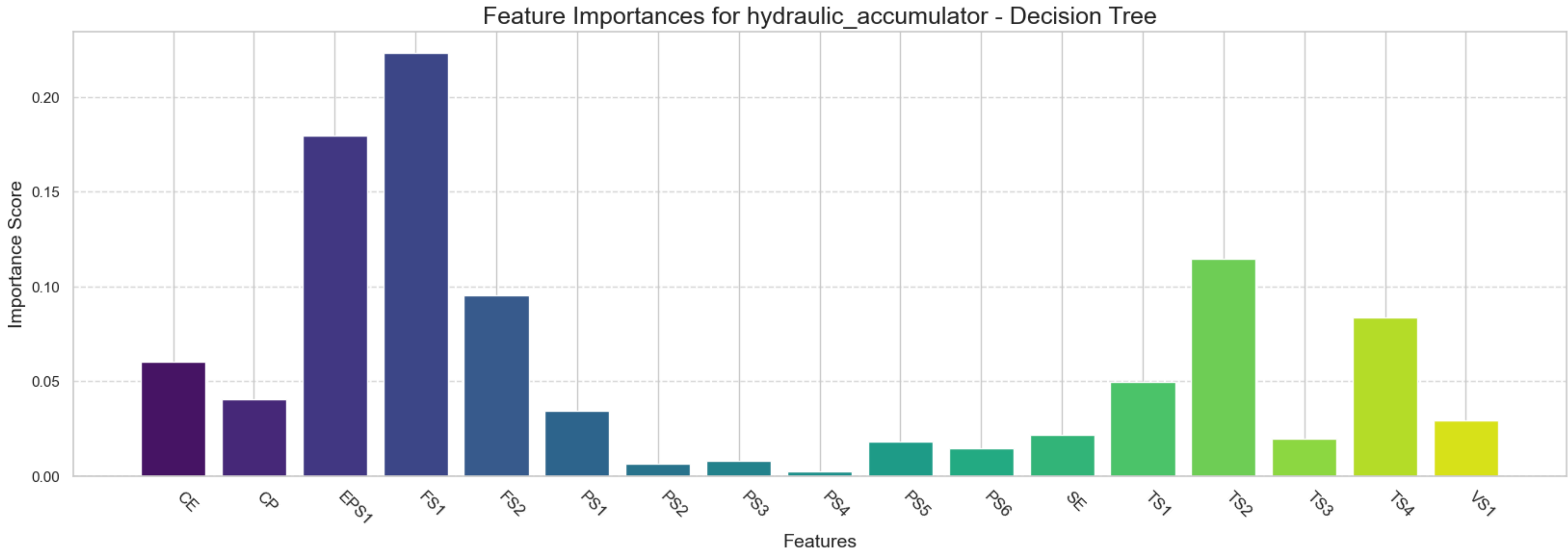

Feature Importances for valve_condition - Random Forest
Importance Score
0.10
0.08
0.06
0.04
0.02
0.00
CE
CP
EPS1
FS1
FS2
PS1
PS2
PS3
PS4
PS5
PS6
SE
TS1
TS2
TS3
TS4
VS1
Features

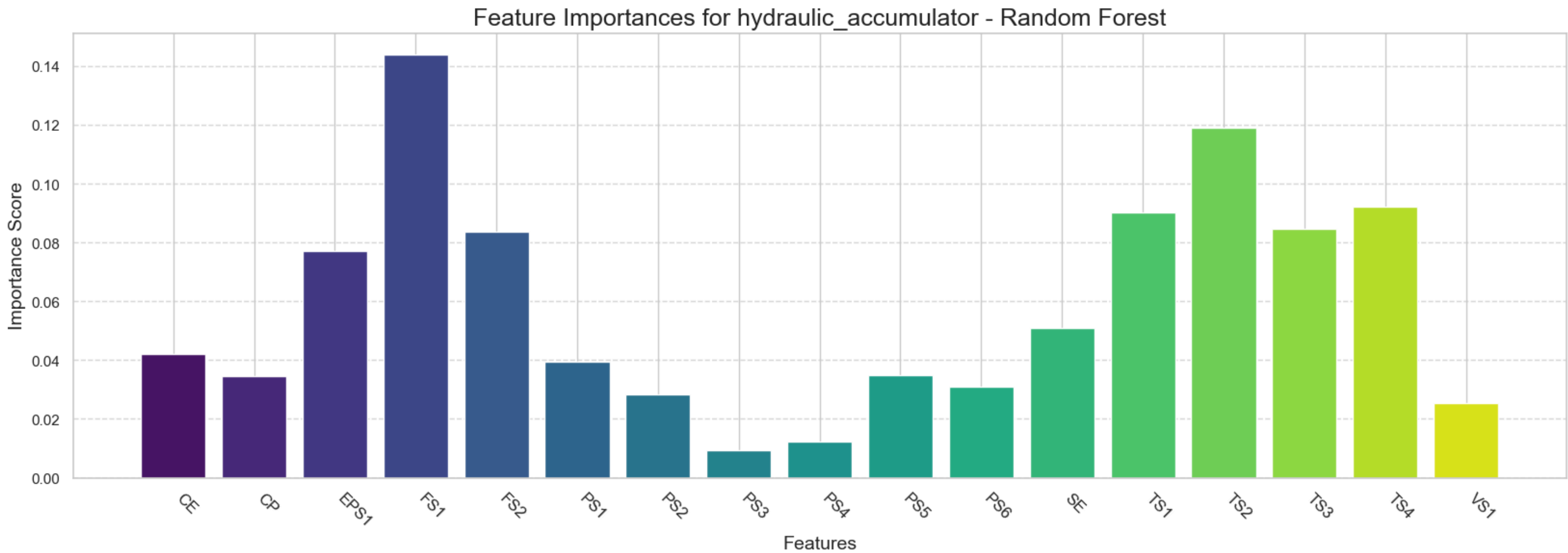

Feature Importances for valve_condition - Gradient Boosting
Importance Score
0.00
0.05
0.10
0.15
0.20
CE
CP
EPS1
FS1
FS2
PS1
PS2
PS3
PS4
PS5
PS6
SE
TS1
TS2
TS3
TS4
VS1
Features

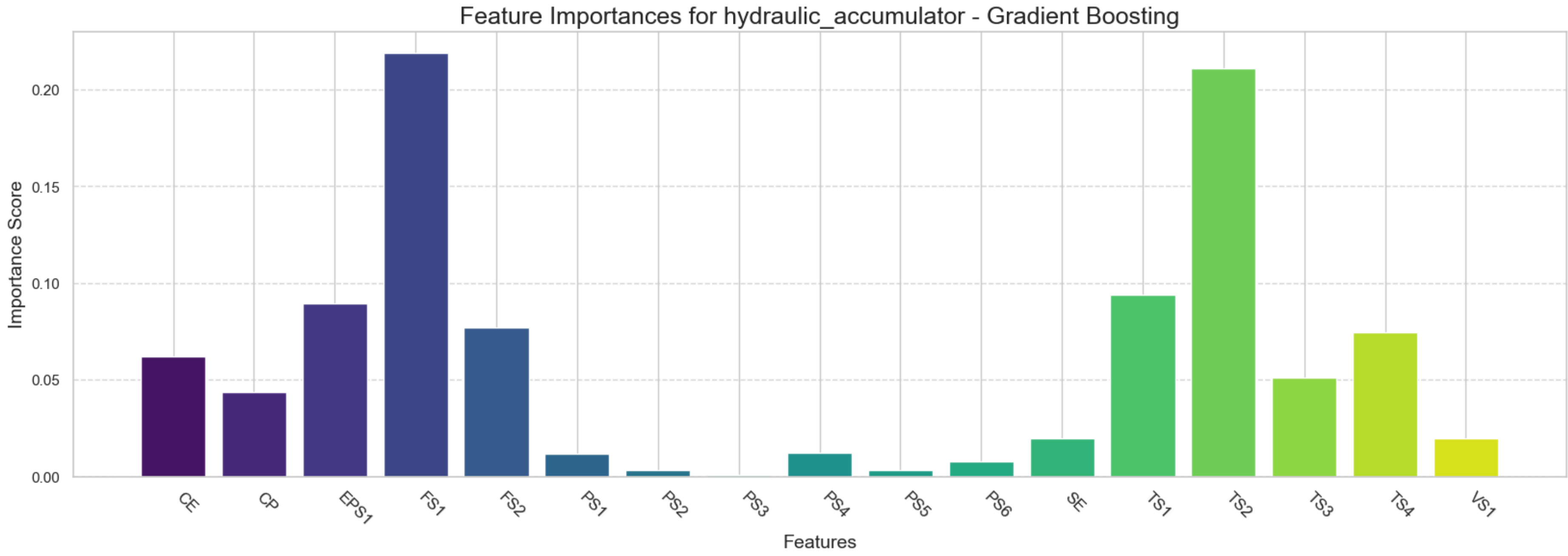

Feature Importances for internal_pump_leakage - Decision Tree
Importance Score
0.0
0.2
0.4
0.6
0.8
CE
CP
EPS1
FS1
FS2
PS1
PS2
PS3
PS4
PS5
PS6
SE
TS1
TS2
TS3
TS4
VS1
Features


Feature Importances for internal_pump_leakage - Random Forest
Importance Score
0.0
0.1
0.2
0.3
0.4
0.5
CE
CP
EPS1
FS1
FS2
PS1
PS2
PS3
PS4
PS5
PS6
SE
TS1
TS2
TS3
TS4
VS1
Features


Feature Importances for internal_pump_leakage - Gradient Boosting
Importance Score
0.0
0.2
0.4
0.6
0.8
CE
CP
EPS1
FS1
FS2
PS1
PS2
PS3
PS4
PS5
PS6
SE
TS1
TS2
TS3
TS4
VS1
Features


Feature Importances for hydraulic_accumulator - Decision Tree
Importance Score
0.00
0.05
0.10
0.15
0.20
Features
CE
CP
EPS1
FS1
FS2
PS1
PS2
PS3
PS4
PS5
PS6
SE
TS1
TS2
TS3
TS4
VS1


Feature Importances for hydraulic_accumulator - Random Forest
Importance Score
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
Features
CE
CP
EPS1
FS1
FS2
PS1
PS2
PS3
PS4
PS5
PS6
SE
TS1
TS2
TS3
TS4
VS1



In [37]:
```
X_full_train, X_test = train_test_split(hydraulic_systemdf, test_size=0.2, random_state=1)
X_train, X_val = train_test_split(X_full_train, test_size=0.25, random_state=1)
```

In [38]:
```
def highlight_cell(value):
    # Check if the value is a string and ends with 'weighted avg'
    if isinstance(value, str) and value.endswith('weighted avg'):
        # Apply background color light blue and text color dark blue
        return 'background-color: #ADD8E6; color: #00008B'

    # Check if the value is a float and equal to the 'recall' value in the 'weighted avg' row
    if isinstance(value, float) and value == classification_rep_df.loc['weighted avg', 'recall']:
        # Apply background color light blue and text color dark blue
        return 'background-color: #ADD8E6; color: #00008B'

    # If none of the conditions match, no styling is applied
    return ''
```

In [41]:
```
import os
from joblib import dump

# Define the directory path
```

```
directory = 'models/'

# Create the directory if it doesn't exist
if not os.path.exists(directory):
    os.makedirs(directory)

target_var = ['cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_accumulator']

for target in target_var:
    # Prepare the training and validation sets
    X = X_train.drop(columns=['Date', 'cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_accumulator', 's
    y = X_train[target]

    X_val_transformed = X_val.drop(columns=['Date', 'cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_ac
    y_val = X_val[target]

    # Create instances of your selected scaler and classifiers
    classifiers = {
        'KNN': KNeighborsClassifier(),
        'SVM': SVC(),
        'Decision Tree': DecisionTreeClassifier(),
        'Random Forest': RandomForestClassifier(),
        'Gradient Boosting': GradientBoostingClassifier()
    }

    for name, classifier in classifiers.items():
        # Create an instance of SMOTE inside the loop
        sm = SMOTE(random_state=1)

        # Create a pipeline with SMOTE, scaler, and classifier
        pipeline = ImbPipeline([('smote', sm), ('scaler', QuantileTransformer(output_distribution='normal', random_state=1)), ('c

        # Fit the pipeline on the training set
        X_resampled, y_resampled = sm.fit_resample(X, y)
        pipeline.fit(X_resampled, y_resampled)

        # Use the pipeline to make predictions on the validation set
        y_val_pred = pipeline.predict(X_val_transformed)

        # Compute precision, recall, and F1 score and output as a dictionary
        classification_rep = classification_report(y_val, y_val_pred, output_dict=True)

        # Convert the classification report dictionary to a DataFrame
        classification_rep_df = pd.DataFrame(classification_rep).transpose().drop(index=['accuracy', 'macro avg'])
```

```
        # Apply the custom styling function
        styled_df = classification_rep_df.style.applymap(highlight_cell).set_table_styles([
            {'selector': 'th', 'props': [('background-color', '#424242'), ('color', '#f0f0f0')]},
            {'selector': 'tr:nth-child(odd)', 'props': [('background-color', '#424242'), ('color', '#f0f0f0')]},
            {'selector': 'tr:nth-child(even)', 'props': [('background-color', '#303030'), ('color', '#f0f0f0')]}
        ])

        # Save the model to a file with a unique name
        dump(pipeline, f'{directory}{target}_{name}_model.joblib')

        # Display for checking
        print(f"Classification Report for {target} - {name}")
        display(styled_df)
```

```
Classification Report for cooler_condition - KNN
```

	precision	recall	f1-score	support
3	1.000000	0.992593	0.996283	135.000000
20	0.992424	1.000000	0.996198	131.000000
100	0.994286	0.994286	0.994286	175.000000
weighted avg	0.995482	0.995465	0.995465	441.000000

```
Classification Report for cooler_condition - SVM
```

	precision	recall	f1-score	support
3	1.000000	1.000000	1.000000	135.000000
20	0.992424	1.000000	0.996198	131.000000
100	1.000000	0.994286	0.997135	175.000000
weighted avg	0.997750	0.997732	0.997733	441.000000

```
Classification Report for cooler_condition - Decision Tree
```

	precision	recall	f1-score	support
3	0.992593	0.992593	0.992593	135.000000
20	0.977612	1.000000	0.988679	131.000000
100	0.994186	0.977143	0.985591	175.000000
weighted avg	0.988775	0.988662	0.988652	441.000000

```
Classification Report for cooler_condition - Random Forest
```

	precision	recall	f1-score	support
3	1.000000	0.992593	0.996283	135.000000
20	0.977612	1.000000	0.988679	131.000000
100	0.994220	0.982857	0.988506	175.000000
weighted avg	0.991056	0.990930	0.990938	441.000000

```
Classification Report for cooler_condition - Gradient Boosting
```

	precision	recall	f1-score	support
3	1.000000	1.000000	1.000000	135.000000
20	0.977612	1.000000	0.988679	131.000000
100	1.000000	0.982857	0.991354	175.000000
weighted avg	0.993350	0.993197	0.993206	441.000000

```
Classification Report for valve_condition - KNN
```

	precision	recall	f1-score	support
73	0.392405	0.462687	0.424658	67.000000
80	0.233010	0.387097	0.290909	62.000000
90	0.228571	0.347826	0.275862	69.000000
100	0.954545	0.604938	0.740554	243.000000
weighted avg	0.654113	0.512472	0.556638	441.000000

```
Classification Report for valve_condition - SVM
```

	precision	recall	f1-score	support
73	0.291667	0.313433	0.302158	67.000000
80	0.163934	0.161290	0.162602	62.000000
90	0.194444	0.405797	0.262911	69.000000
100	0.920732	0.621399	0.742015	243.000000
weighted avg	0.605125	0.476190	0.518767	441.000000

```
Classification Report for valve_condition - Decision Tree
```

	precision	recall	f1-score	support
73	0.391892	0.432836	0.411348	67.000000
80	0.270270	0.322581	0.294118	62.000000
90	0.354430	0.405797	0.378378	69.000000
100	0.822430	0.724280	0.770241	243.000000
weighted avg	0.606167	0.573696	0.587465	441.000000

```
Classification Report for valve_condition - Random Forest
```

	precision	recall	f1-score	support
73	0.518072	0.641791	0.573333	67.000000
80	0.328125	0.338710	0.333333	62.000000
90	0.387755	0.550725	0.455090	69.000000
100	0.943878	0.761317	0.842825	243.000000
weighted avg	0.705605	0.650794	0.669586	441.000000

```
Classification Report for valve_condition - Gradient Boosting
```

	precision	recall	f1-score	support
73	0.402439	0.492537	0.442953	67.000000
80	0.250000	0.306452	0.275362	62.000000
90	0.320000	0.463768	0.378698	69.000000
100	0.918033	0.691358	0.788732	243.000000
weighted avg	0.652212	0.571429	0.599870	441.000000

```
Classification Report for internal_pump_leakage - KNN
```

	precision	recall	f1-score	support
0	0.987234	0.974790	0.980973	238.000000
1	0.925234	0.961165	0.942857	103.000000
2	0.959596	0.950000	0.954774	100.000000
weighted avg	0.966486	0.965986	0.966130	441.000000

```
Classification Report for internal_pump_leakage - SVM
```

	precision	recall	f1-score	support
0	0.987288	0.978992	0.983122	238.000000
1	0.952381	0.970874	0.961538	103.000000
2	0.970000	0.970000	0.970000	100.000000
weighted avg	0.975215	0.975057	0.975106	441.000000

```
Classification Report for internal_pump_leakage - Decision Tree
```

	precision	recall	f1-score	support
0	0.982759	0.957983	0.970213	238.000000
1	0.899083	0.951456	0.924528	103.000000
2	0.910000	0.910000	0.910000	100.000000
weighted avg	0.946717	0.945578	0.945889	441.000000

```
Classification Report for internal_pump_leakage - Random Forest
```

	precision	recall	f1-score	support
0	0.983122	0.978992	0.981053	238.000000
1	0.951923	0.961165	0.956522	103.000000
2	0.970000	0.970000	0.970000	100.000000
weighted avg	0.972860	0.972789	0.972817	441.000000

```
Classification Report for internal_pump_leakage - Gradient Boosting
```

	precision	recall	f1-score	support
0	0.983122	0.978992	0.981053	238.000000
1	0.925234	0.961165	0.942857	103.000000
2	0.969072	0.940000	0.954315	100.000000
weighted avg	0.966416	0.965986	0.966069	441.000000

```
Classification Report for hydraulic_accumulator - KNN
```

	precision	recall	f1-score	support
90	0.930556	0.842767	0.884488	159.000000
100	0.602410	0.757576	0.671141	66.000000
115	0.741176	0.797468	0.768293	79.000000
130	0.992248	0.934307	0.962406	137.000000
weighted avg	0.866685	0.850340	0.855949	441.000000

```
Classification Report for hydraulic_accumulator - SVM
```

	precision	recall	f1-score	support
90	0.905109	0.779874	0.837838	159.000000
100	0.567308	0.893939	0.694118	66.000000
115	0.828571	0.734177	0.778523	79.000000
130	0.976923	0.927007	0.951311	137.000000
weighted avg	0.863153	0.834467	0.840954	441.000000

```
Classification Report for hydraulic_accumulator - Decision Tree
```

	precision	recall	f1-score	support
90	0.909677	0.886792	0.898089	159.000000
100	0.680556	0.742424	0.710145	66.000000
115	0.818182	0.797468	0.807692	79.000000
130	0.934307	0.934307	0.934307	137.000000
weighted avg	0.866648	0.863946	0.865019	441.000000

```
Classification Report for hydraulic_accumulator - Random Forest
```

	precision	recall	f1-score	support
90	0.974026	0.943396	0.958466	159.000000
100	0.797297	0.893939	0.842857	66.000000
115	0.873418	0.873418	0.873418	79.000000
130	1.000000	0.978102	0.988930	137.000000
weighted avg	0.937623	0.934240	0.935393	441.000000

```
Classification Report for hydraulic_accumulator - Gradient Boosting
```

	precision	recall	f1-score	support
90	0.967320	0.930818	0.948718	159.000000
100	0.760000	0.863636	0.808511	66.000000
115	0.881579	0.848101	0.864516	79.000000
130	0.978102	0.978102	0.978102	137.000000
weighted avg	0.924283	0.920635	0.921779	441.000000

In [42]:

```
from sklearn.preprocessing import OneHotEncoder, QuantileTransformer
from sklearn.compose import ColumnTransformer
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report
from joblib import dump
import pandas as pd
#import dfi
```

```python
# Define the columns to be one-hot encoded and the columns for the quantile transformer
categorical_features = ['cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_accumulator']
quant_features = ['CE', 'CP', 'EPS1', 'FS1', 'FS2', 'PS1', 'PS2', 'PS3', 'PS4', 'PS5','PS6', 'SE', 'TS1', 'TS2', 'TS3', 'TS4', 'V

# Create a preprocessor that applies the OneHotEncoder and the QuantileTransformer to the specified columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', QuantileTransformer(output_distribution='normal', random_state=1), quant_features),
        ('cat', OneHotEncoder(), categorical_features)])

# Create an instance of SMOTE
sm = SMOTE(random_state=1)

# Create a pipeline with SMOTE, preprocessor, and classifier
pipeline = ImbPipeline([('smote', sm), ('preprocessor', preprocessor), ('classifier', classifier)])

# Drop unnecessary columns from X_train and X_val
X = X_train.drop(columns=['Date', 'stable_flag'])
y = X_train['stable_flag']

X_val_transformed = X_val.drop(columns=['Date', 'stable_flag'])
y_val = X_val['stable_flag']

# Fit the pipeline on the training set
pipeline.fit(X, y)

# Transform X_val using the preprocessor
X_val_transformed = pipeline.named_steps['preprocessor'].transform(X_val_transformed)

# Use the pipeline to make predictions on the validation set
y_val_pred = pipeline.named_steps['classifier'].predict(X_val_transformed)

# Compute precision, recall, and F1 score and output as a dictionary
classification_rep = classification_report(y_val, y_val_pred, output_dict=True)

# Convert the classification report dictionary to a DataFrame
classification_rep_df = pd.DataFrame(classification_rep).transpose().drop(index=['accuracy', 'macro avg'])

# Apply the custom styling function
styled_df = classification_rep_df.style.applymap(highlight_cell).set_table_styles([
    {'selector': 'th', 'props': [('background-color', '#424242'), ('color', '#f0f0f0')]},
    {'selector': 'tr:nth-child(odd)', 'props': [('background-color', '#424242'), ('color', '#f0f0f0')]},
    {'selector': 'tr:nth-child(even)', 'props': [('background-color', '#303030'), ('color', '#f0f0f0')]}
```

```
])

# Save the styled DataFrame as a png image
#dfi.export(styled_df, 'images/stable_flag_class_rep.png')

# Save the model to a file
dump(pipeline, 'models/stable_flag_model.joblib')

# Display for checking
print(f"Classification Report: stable_flag")
display(styled_df)
```

```
Classification Report: stable_flag
```

	precision	recall	f1-score	support
0	0.992537	0.960289	0.976147	277.000000
1	0.936416	0.987805	0.961424	164.000000
weighted avg	0.971667	0.970522	0.970672	441.000000

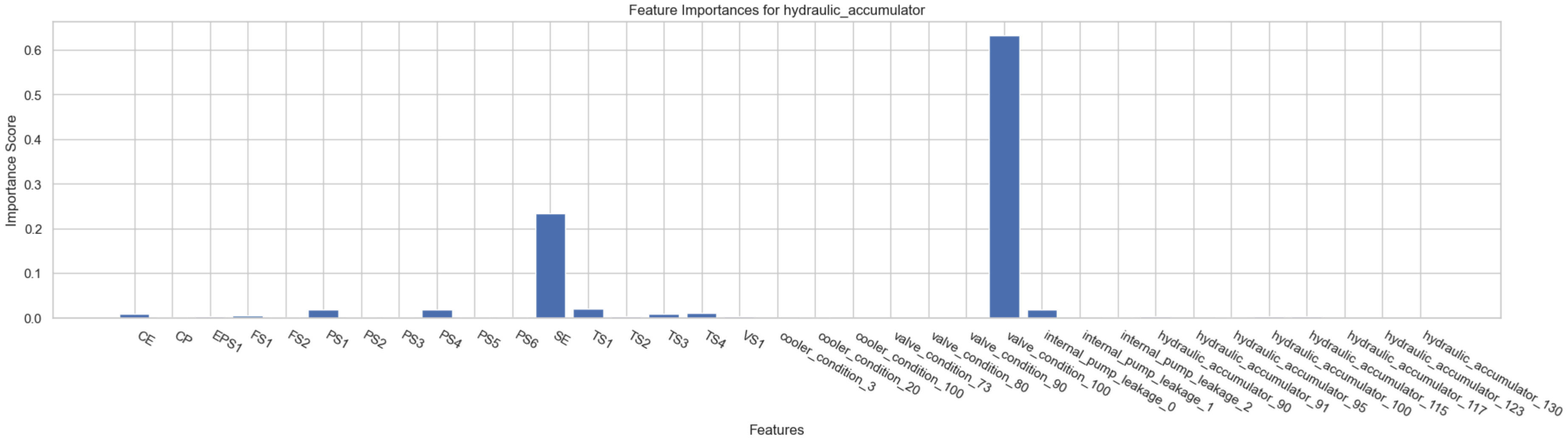
In [43]:

```
X_train_drop = X_train.drop(columns=['Date','cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_accumulato
X_val_drop = X_val.drop(columns=['Date','cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_accumulator',
X_test_drop = X_test.drop(columns=['Date','cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_accumulator'
```

In [55]:

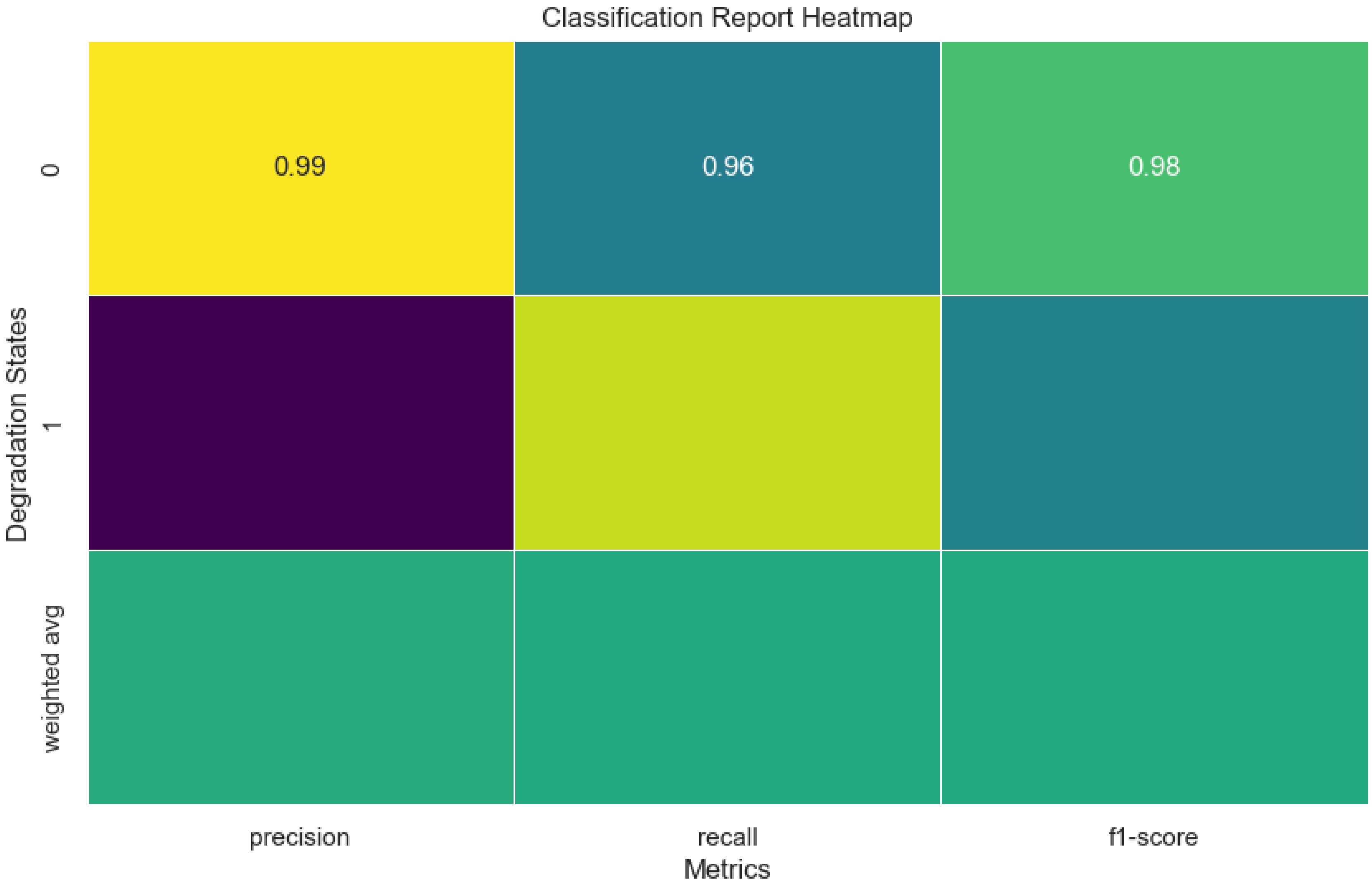
```
# Get feature names from the preprocessor
num_features = ['CE','CP','EPS1','FS1','FS2','PS1','PS2','PS3','PS4','PS5','PS6','SE','TS1','TS2','TS3','TS4', 'VS1']
cat_features = ['cooler_condition','valve_condition','internal_pump_leakage', 'hydraulic_accumulator']

# Numeric features remain the same
feature_names = num_features.copy()

# Get the feature importances from the trained model
important = stable_flag_model.named_steps['classifier'].feature_importances_

# For categorical features, we need to append the encoded feature names
ohe = stable_flag_model.named_steps['preprocessor'].named_transformers_['cat']
feature_names.extend(ohe.get_feature_names_out(cat_features))

# At this point, feature_names contains original numeric feature names and the transformed categorical feature names

# Now you can use this in your plot
fig, ax = plt.subplots(figsize=(22,6))
```

```
ax.bar(feature_names, important)
ax.set_xlabel('Features')
ax.set_ylabel('Importance Score')
ax.set_title(f"Feature Importances for {target}")
plt.xticks(rotation=-30, ha='left')
plt.subplots_adjust(bottom=0.3)
plt.savefig('test_feature_importance.png', format='png')
plt.show()
```



In [56]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline
from joblib import load
import pandas as pd

# Assuming you have the classification report stored in classification_rep_df
# Replace this with the actual classification report DataFrame
classification_rep_df = pd.DataFrame(classification_rep).transpose().drop(index=['accuracy', 'macro avg'])

# Create a heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(classification_rep_df.iloc[:, :3], annot=True, cmap='viridis', fmt='.2f', linewidths=.5, cbar=False)
plt.title("Classification Report Heatmap")
plt.xlabel("Metrics")
plt.ylabel("Degradation States")
plt.show()
```

```
# Extract metrics for plotting
metrics_to_plot = ['precision', 'recall', 'f1-score']
metrics_df = classification_rep_df[metrics_to_plot]

# Plot the bar plot
metrics_df.plot(kind='bar', figsize=(12, 6), colormap='viridis')
plt.title("Classification Report Metrics for Stable Flag Prediction")
plt.xlabel("Degradation States")
plt.ylabel("Score")
plt.legend(title="Metrics", loc='upper right')
plt.xticks(rotation=45, ha='right')
plt.show()
```


Classification Report Heatmap
Degradation States
0
1
weighted avg
0.99
0.96
0.98
precision
recall
f1-score
Metrics

In [57]:

```
# Drop unnecessary columns from X_train and X_val
X = X_train.drop(columns=['Date','cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_accumulator', 'stable
y = X_train['valve_condition']

X_val_transformed = X_val.drop(columns=['Date','cooler_condition', 'valve_condition', 'internal_pump_leakage', 'hydraulic_accumul
y_val = X_val['valve_condition']

# Create an instance of SMOTE
```

```
sm = SMOTE(random_state=1)

# Create instances of your selected scaler and classifier
quant = QuantileTransformer(output_distribution='normal', random_state=1)
classifier = RandomForestClassifier(random_state=1)

# Create a pipeline with SMOTE, scaler, and classifier
pipeline = ImbPipeline([('smote', sm), ('scaler', quant), ('classifier', classifier)])

# Fit the pipeline on the training set
pipeline.fit(X, y)

# Transform X_val using the scaler
X_val_transformed = pipeline.named_steps['scaler'].transform(X_val_transformed)

# Use the pipeline to make predictions on the validation set
y_val_pred = pipeline.named_steps['classifier'].predict(X_val_transformed)

# Compute precision, recall, and F1 score
classification_rep = classification_report(y_val, y_val_pred)
print("Classification Report:\n", classification_rep)

# Define the parameter grid
param_grid = {
    'classifier__n_estimators': [100, 200, 300, 500],
    'classifier__max_depth': [None, 5, 10, 20],
    'classifier__min_samples_split': [2, 5, 10],
    'classifier__min_samples_leaf': [1, 2, 4],
    'classifier__max_features': ['auto', 'sqrt', 'log2'],
    'classifier__criterion': ['gini', 'entropy'],
    'classifier__class_weight': [None, 'balanced', 'balanced_subsample']
}

# Create RandomizedSearchCV object
random_search = RandomizedSearchCV(pipeline, param_distributions=param_grid, scoring='precision_weighted', verbose=1, n_iter=10,

# Fit the random search to the training data
random_search.fit(X, y)

# Print the best parameters
print("Best parameters: ", random_search.best_params_)
```

```
Classification Report:
              precision    recall  f1-score   support

          73       0.54      0.67      0.60        67
          80       0.37      0.37      0.37        62
          90       0.39      0.57      0.46        69
         100       0.95      0.76      0.85       243

    accuracy                           0.66       441
   macro avg       0.56      0.59      0.57       441
weighted avg       0.72      0.66      0.68       441

Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecate
d in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter a
s it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecate
d in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter a
s it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecate
d in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter a
s it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecate
d in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter a
s it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecate
d in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter a
s it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecate
d in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter a
s it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecate
d in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter a
s it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecate
d in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter a
s it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecate
d in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter a
s it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecate
d in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter a
s it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
Best parameters:  {'classifier__n_estimators': 100, 'classifier__min_samples_split': 10, 'classifier__min_samples_leaf': 1, 'cla
ssifier__max_features': 'sqrt', 'classifier__max_depth': 10, 'classifier__criterion': 'entropy', 'classifier__class_weight': Non
e}
```

In [61]:
```
# Use the best model to make predictions on X_val
best_model = random_search.best_estimator_
y_val_pred = best_model.predict(X_val_transformed)

# Compute precision, recall, and F1 score
classification_rep = classification_report(y_val, y_val_pred)
print("Classification Report:\n", classification_rep)
```

```
Classification Report:
               precision    recall  f1-score   support

          73       0.00      0.00      0.00        67
          80       0.00      0.00      0.00        62
          90       0.00      0.00      0.00        69
         100       0.55      1.00      0.71       243

    accuracy                           0.55       441
   macro avg       0.14      0.25      0.18       441
weighted avg       0.30      0.55      0.39       441
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but QuantileTrans
former was fitted with feature names
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score a
re ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score a
re ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score a
re ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

CONCLUSION

The stability of a hydraulic system is influenced significantly by Valve Condition, making regular inspections and maintenance imperative to uphold system integrity. The System Efficiency Sensor (SE) follows closely as a crucial stability indicator, highlighting the need to utilize SE data for predicting potential system failures. Implementing an alert system based on SE readings allows for proactive notifications, particularly when values fall below a set threshold, indicating a possible decline in stability. Internal Pump Leakage emerges as another critical determinant of system stability, emphasizing the importance of vigilant monitoring. Timely identification and mitigation of internal pump leakages are crucial to sustaining stability in the hydraulic system. Interestingly, Cooler Condition does not exert a substantial influence on system stability. This insight provides valuable guidance for personnel to strategically allocate maintenance efforts. The ability to prioritize tasks related to Valve Condition and Internal Pump Leakages over Cooler Conditions enhances the effectiveness of stability management. In essence, understanding the hierarchy of factors influencing system stability allows for a targeted and efficient approach to maintenance efforts. By focusing on Valve Condition, SE data, and mitigating Internal Pump Leakages, the company can proactively manage and ensure the stability of the hydraulic system, optimizing the allocation of resources for maintenance tasks.

In [ ]: